Conception d'une caméra intelligente multi-vues à base de FPGA

Jonathan Bonnard

THÈSE DE DOCTORAT

Conception d'une caméra intelligente multi-vues à base de FPGA

Thèse présentée par **Mr Jonathan BONNARD**

Pour obtenir le grade de **Docteur de l'université Clermont Auvergne**

Spécialité : **Électronique et systèmes**

Thèse soutenue le 7 Juillet 2021 devant le jury composé de :

Président	Mr Dominique GINHAC
Directeur de thèse	Mr François BERRY
Co-directeur de thèse	Mr Maxime PELCAT
Rapporteurs	Mme Cecile BELLEUDY Mr Gilles SASSATELLI
Examinateurs	Mr Dominique GINHAC Mr Herve CHANAL

DREAM

Institut Pascal

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

Résumé

Conception d'une caméra intelligente multi-vues à base de FPGA

par Jonathan BONNARD

Les systèmes de vision font partie intégrante de notre société et continuent d'alimenter de nombreux axes de recherche et développement. Les systèmes multi-vues permettent d'étendre le champ des applications de vision par leur capacité à capturer une scène sous différents angles de vue. La vision stéréoscopique et la reconstruction 3D ou panoramique sont des exemples concrets d'applications multi-vues et sont déjà massivement employés dans l'industrie ou auprès du grand public. Par exemple, certaines productions cinématographiques utilisent des effets spéciaux à partir de dizaines de caméras synchronisées. Ou encore les satellites d'observation privilégient des matrices de caméras pour créer des images en haute-résolution. Cette versatilité, la diversité ou la redondance d'informations visuelles sont exploitables dans de nombreux domaines et en particuliers pour l'intelligence artificielle appliquée à la vision. Les dernières recherches ont démontré que les algorithmes de vision classiques sont devenus obsolètes face aux réseaux de convolutions (CNNs). Cependant, les CNNs restent extrêmement coûteux en calculs et complexes à mettre en oeuvre sur des électroniques basse consommation. En théorie, les circuits reprogrammables (FPGA) possèdent un potentiel significatif pour inférer un réseau de convolutions au plus près d'un capteur d'image avec un modèle de calcul orienté flots de données, garantissant à la fois une consommation électrique limitée et une grande vitesse d'exécution. En pratique, ils restent limités par leur faible densité d'éléments logiques et par d'autres facteurs inhérents à cette technologie. Par conséquent, un FPGA ne peut supporter à lui seul qu'une faible partie d'un réseau. Dans cette thèse, nous proposons une architecture matérielle multi-vues pour augmenter le facteur d'intégration d'un réseau sur ce type de cible matérielle. En effet, même dans sa forme la plus simple, une architecture de réseau de convolutions multi-vues possède une meilleure capacité

de reconnaissance que sa contrepartie mono-vue. Ce principe fondamental pose la question de l'équilibre entre les facteurs quantitatif et qualitatif d'un réseau, c'est-à-dire, la densité de paramètres définissant la complexité d'un CNN et son niveau de performance en terme de reconnaissance. Ce travail démontre de manière expérimentale que la nature multi-vues permet d'abaisser la quantité de paramètres d'un réseau et par conséquent, sa complexité. Un prototype de caméra embarquée est proposée pour tester cette hypothèse dans un contexte réaliste. Celle-ci est scindée en deux parties. La première est composée de plusieurs têtes de caméras reprogrammables capables d'inférer une partie d'un CNN en supportant le débit d'un capteur d'image. Celles-ci sont pilotées et alimentées par la seconde partie composée d'un FPGA central qui agrège les flots de données issues de chaque capteur et exécuter la suite du réseau de neurones. Entre autres, cette caméra peut synchroniser ou ajouter des délais pour chaque prise de vue avec une base de temps de l'ordre de la dizaine de nano secondes.

Mot-clés : Caméras intelligentes, FPGA, CNN, Compression, Multi-vues, Architectures matérielles

Je tiens à remercier tout particulièrement Laurent, Samuel, Nicolas, Richard, Pierre-Etienne, le directeur du LPC et l'université Clermont Auvergne de m'avoir donné la chance de continuer mon parcours universitaire dans le cadre de mon travail. Milles merci ne sauraient être suffisants pour exprimer ma gratitude.

A El-Medhi, Lobna, Abiel, Seyf, Walther et Nyando pour avoir supporté mes complaintes excessives mais fortement nécessaires.

Je n'oublie pas Kamel pour avoir guidé mes pas durant la thèse ainsi qu'à Maxime pour sa gentillesse et pour m'avoir bousculé aux moments opportuns.

A François sans qui je ne serais pas là où j'en suis aujourd'hui.

A ceux qui m'ont le plus soutenu durant ces 4 ans : mon père Guy, mon frère Cédric, Manue, Christophe, Lucile, ma grand-mère Marie et mes amis.

A Juju et maman.

A Julie.

Table des matières

Table des figures

Liste des tableaux

Glossaire

AGP Automatic Gradual Pruning. 84

ALM Adaptative Logic Module. 151

ARP Address Resolution Protocol. 144

ASIC Application Specific Integrated Circuit. 118, 126

CCA analyse de corrélation croisée. 8

CMOS Complementary Metal-Oxyde Semi-Conductor. 162

CNN Réseaux de Neurones Convolutifs. xvii, 4, 8–12, 15, 17, 19, 29, 39, 42, 43, 49, 51, 52, 54, 57, 60, 62, 65–67, 69–71, 80, 83, 84, 86, 98, 107, 109–111, 113, 116, 118, 119, 121, 122, 125–127, 129, 131, 133, 138, 147, 148, 157, 161

COO Coordinates. 125

CPU Central Processor Unit. 44, 109, 116, 120, 125, 137

CSC Compressed Sparse Column. 125

CSR Compressed Sparse Row. 125

DCT Discrete Cosine Transform. 57

DDR Double Data Rate. 147

DFT Discrete Fourier Transform. 119

DHM Direct Hardware Mapping. ix, xiv, xvii, 46, 66, 86, 110, 111, 113, 116, 120, 121, 124, 126, 129, 130, 132, 137, 157, 161–164

DMA Direct Memory Access. 116, 147

DNN Deep Neural Network. 122

DSP Digital Signal Processor. xvii, 41, 109, 110, 116, 121, 122, 139, 140

FC Fully Connected. 30, 34, 37, 41, 44, 52, 56, 60, 89, 112, 113, 126, 129

FCN réseau de convolutions. xvii, 30, 34, 38, 39, 41, 52, 69, 78, 80, 82, 109, 111, 112, 129

FIFO First-In First-Out. 161–164

FPGA Field Programmable Gate Array. ix, xiv, xv, xvii, 12, 41, 42, 44, 65–67, 86–88, 107, 109–111, 113, 116, 118, 119, 121–125, 127–132, 134, 135, 137–148, 150, 151, 153, 157, 161

GEMM General matrix matrix multplication. xiv, 119

GMII Gigabit Media Independant Interface. 147

GOPs Giga Opérations Par Seconde. 60, 119, 121

GPU Graphic Processor Unit. 4, 17, 26, 42, 44, 47, 109, 110, 118, 121, 125, 137, 138, 140

HLS High Level Synthesis. 118

HOG Histogram of Gradients. 16

IA Intelligence Artificielle. 153

ICMP Internet Control Message Protocol. 144

IVR Inverted residual layer. 67, 68, 84, 86–88, 159, 163

KNN K plus proches voisins. 31

LASSO Least Absolute Shrinkage and Selection Operator. 48

LUT Look Up Table. 139

MAC mutliplication accumulation. 12, 24, 44, 46, 68, 82, 86, 87, 109, 112, 113, 115, 121, 127, 137, 143, 157, 161–163

MACOPs opérations de multiplication-addition. 33, 119

MAE machine à états. 116

MLP Perceptron multi-couches. 19, 23, 33, 34, 36, 38, 41, 46, 112

MVCNN Multi View Convolutional Neural Network. 133, 157

NCC Normalized Cross Correlation. 10

NNZ Number of Non-Zero. 68, 76, 78, 81, 82

PE Processing Elements. xiv, 57, 109, 116, 120, 124, 126, 135, 157, 163, 164

POE Power over Ethernet. 147

PTP Precision Time Protocol. 141

RAM Random Access Memory. 150, 163

RELU Rectified Linear Unit. 24, 113

RGMII Reduced Gigabit Media Independant Interface. 147

RL Reinforcement Learning. 158

RNN Réseaux de Neurones Récurrents. 19

ROM Read-Only Memory. 161

RTL Register Transfer Level. 118, 127

SAD Somme des différences absolue. 10

SFP+ Small Form Factor Pluggable. 151

SGMII Serial Gigabit Media Independant Interface. 147

SIFT Scale Invariant Feature Transform. 16

SIMD Single Instruction Multiple Data. 52, 57, 111, 123, 137

SoC système sur puce. 137, 138

SSD Somme des différences au carré. 10

SVD Décomposition en valeurs singulières. 35

SVM Machine à Vecteurs de support. 31

TOF Time of flight. 7

TOPs Téra Opérations Par Seconde. 119

TPU unité de calcul tensorielle. 42

VCO Voltage Controlled Oscillator. 142

Chapitre 1

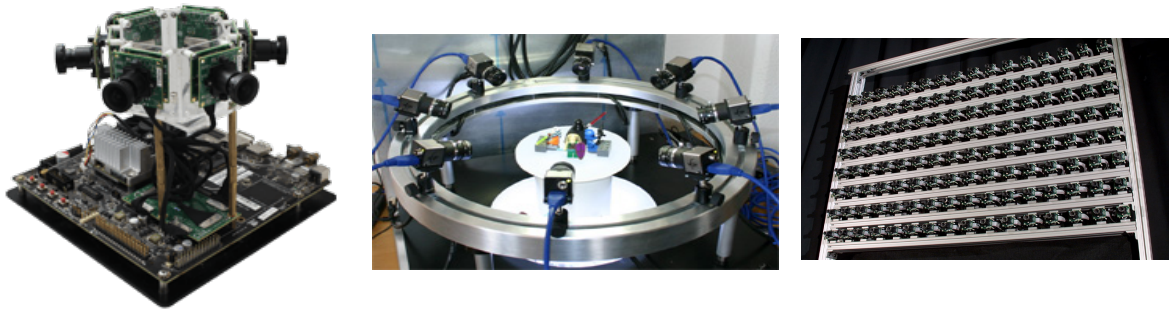
Systèmes de vision embarqués multi-vues

1.1 Introduction

Résumé : *Ce premier chapitre introduit l'idée principale de la thèse. En premier lieu sont présentés quelques uns des domaines applicatifs des systèmes multi-caméras. Ces mêmes applications ont connu le raz-de-marée de l'apprentissage machine dédiée à la vision avec les réseaux de neurones convolutifs. Le concept y est rapidement évoqué puisqu'il sera plus détaillé lors du deuxième chapitre. La trajectoire de ce chapitre est guidée par les résultats de la thèse de Kamel Abdelouahab effectuée dans l'équipe DREAM. Cette thèse a émis l'hypothèse d'inférer ce type de réseau sur des architectures reprogrammables lesquelles offrent une grande liberté d'optimisation matérielle du calcul tout en étant reprogrammables. La troisième section en redonnera les raisons. Après avoir situé le contexte général, la dernière section visera à expliquer aux lecteurs les tenants et les aboutissants de l'idée qui sera véhiculée tout au long du manuscrit, à savoir prendre avantage d'un contexte multi-vues pour alléger la charge matérielle d'un réseaux de convolutions sur FPGA.*

1.1.1 Multi-vues et domaines applicatifs

Les algorithmes et systèmes de visions sont devenus des acteurs majeurs dans différents secteurs industriels ou publiques. La maîtrise des concepts mathématiques [Har97, HZ04] et physiques de ces systèmes a permis de démultiplier leurs usages en surveillance, robotique ou encore en conduite autonome. Les axes de recherches en vision sont pléthoriques à commencer par le domaine du traitement d'images puis par la reconstruction de



(a) Convexe (b) Concave (c) Matrice

FIGURE 1.1 – Les 3 types de géométries possibles pour une caméra multi-vues : (a) convexe, (b) concave et (c) matricielle

FIGURE 1.2 – Exemple de collage multi-vues. Source : [BL07]

scène par analyse de la géométrie. Les systèmes multi-vues sont particulièrement adaptés dans ce dernier cas puisqu'ils permettent de visualiser une scène donnée sous différents points de vues de manière simultanée. La géométrie d'un système multi-caméras influe fortement sur la fonction finale. Ces arrangements de capteurs se divisent généralement en trois catégories distinctes, convexe, concave et coplanaire. Par exemple, une géométrie convexe, c'est à dire avec les capteurs tournés vers l'extérieur, augmente considérablement le champ de vision de la prise de vue comme illustré par la (figure 1.1(a). C'est dans ce contexte que la reconstruction panoramique (figure 1.2) apparaît comme l'application de référence [BL07]. Elle consiste à trouver suffisamment de zones similaires entre plusieurs points de vue pour les fusionner et synthétiser une image grand angle. Les paires ou matrices de caméras de la figure 1.1(c) sont aussi bien connus du grand publique puisque cette topologie a notamment permis l'émergence du cinéma dit "3D". Une caméra traditionnelle projette une scène 3D sur le repère capteur en 2D et perd nécessairement l'information selon le 3ème axe $\vec{z}$. Une caméra stéréoscopique peut retrouver cette composante par triangulation [SSZ01, BBH03] comme montré dans la figure 1.3. Une matrice de caméras peut aussi simuler un capteur de plus haute résolution par agrégation des pixels issus des différents capteurs. De manière similaire à la recherche de zone de recouvrement rencontrée en stéréo ou en génération d'images panoramiques,

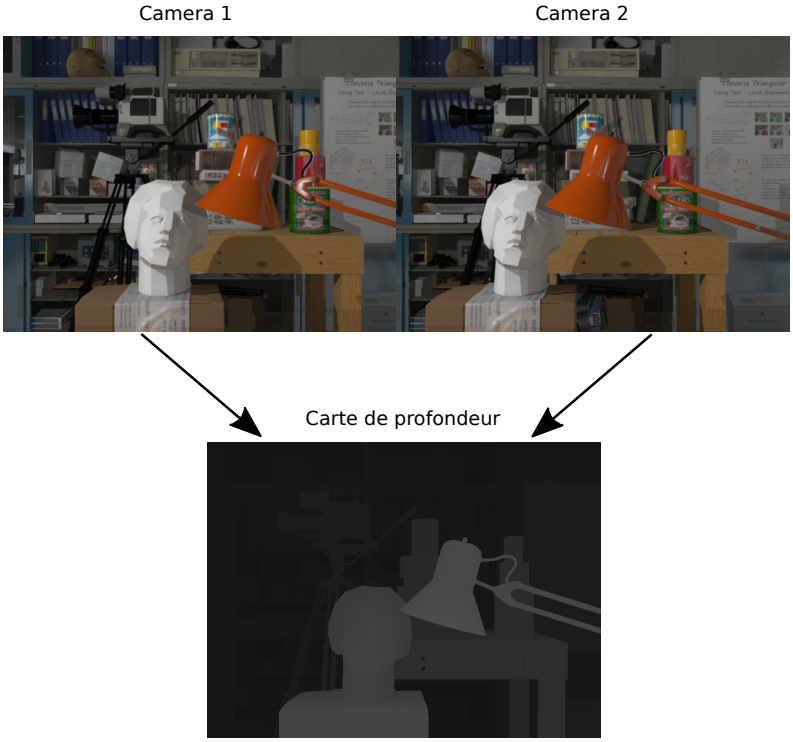
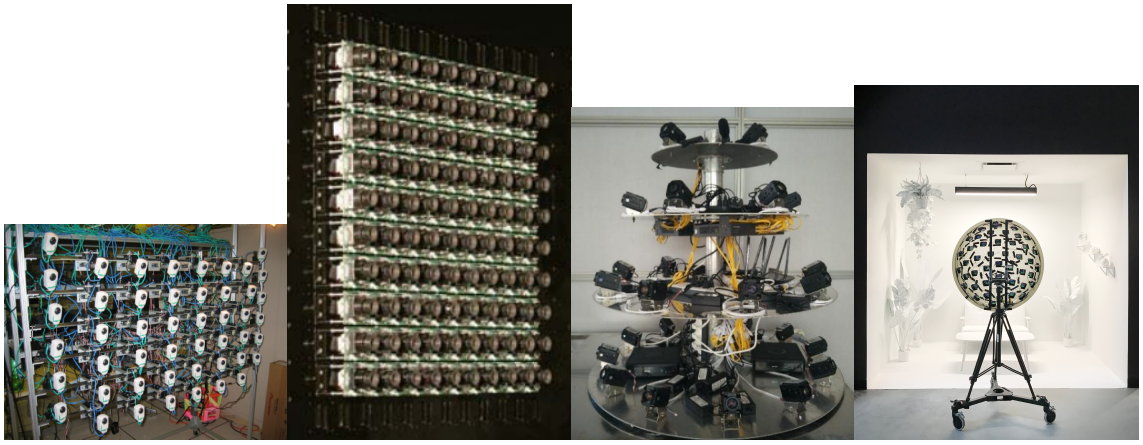


FIGURE 1.3 – Estimation de la carte de profondeur d'une scène.
Source : https://home.cvlab.cs.tsukuba.ac.jp/dataset

un algorithme recherche les déplacements sous-pixelliques entre images [PPK03]. Enfin la figure 1.1(b) illustre une géométrie multi-vues concave utilisée pour reconstruire un objet en 3 dimensions. Ce type de disposition prend avantage de la complémentarité des informations 2D pour les regrouper sous une forme tridimensionnelle [FLG14]. Cette complémentarité peut potentiellement fournir des informations supplémentaires dans le cas de la reconnaissance ou de la détection d'objet. Les exemples cités démontrent qu'un dispositif de caméra multi-vues offrent de nombreuses opportunités d'applications comparé aux systèmes mono-vues standard. A un instant donné, une caméra classique ne peut capturer plus d'informations visuels que ce que son champ de vision lui permet. Au contraire, la démultiplication du nombre de caméras allège la contrainte spatiale mais implique une capacité de traitement plus importante. Cette contrainte est souvent ignorée par la majorité des recherches menées dans ce domaine qui s'attachent surtout à en démontrer les capacités. L'essentiel des systèmes multi-scopiques de la littérature sont construit autour d'imageurs standards et connectés à un ou plusieurs coeurs de



(A) (B) (C) (D)

FIGURE 1.4 – Exemples d'architectures de caméras multi-vues : (A) [ZC05], (B) [Wil04], (C) [LQL+18], (D) [BBD+19]

calculs centraux pensés pour vérifier la fonctionnalité d'un algorithme. Même si les algorithmes et les domaines d'application consacrés à la perception multi-vues sont nombreux, le nombre de réalisation matérielles dédiées reste limitée. Le coût spatial, matériel et les temps de calcul mis en jeu, notamment pour les systèmes combinant plusieurs dizaines de caméras, rendent la conception difficile. Parmi les quelques systèmes réalistes disponibles dans la littérature [ZC05, Wil04, LQL+18] (figures 1.4), les solutions proposées sont difficilement qualifiables de *"systèmes embarqués"*. Certains travaux ont abordé la question avec des architectures électroniques parfaitement adaptées aux applications évoquées ci-dessus. Le dôme de caméra de *Afshari et Popovic* [AJB+13, PASL13] apparaît comme étant l'un des exemples les plus aboutis. Cependant, les algorithmes de vision mono et multi-vues ont connu une véritable révolution grâce aux efforts conjoints menés en apprentissage automatique et sur les structures de réseaux de neurones. Les réseaux de neurones convolutifs CNN ont supplantés la quasi totalité des algorithmes standards pour diverses applications de vision tels que la reconnaissance [DDS+09], la détection [GDDM14, RDGF16] ou la segmentations [LSD15, Gir15] d'objets à partir d'images brutes. Mais cette évolution a un coût et n'a pu voir le jour que grâce aux performances des nouvelles plateformes matérielles à base de processeurs graphiques GPU et leurs architectures massivement parallèles taillées pour exécuter des millions d'opérations de convolutions par seconde, souvent au prix d'une d'une consommation électrique élevée. Les applications multi-vues ont naturellement suivi cette tendance avec des performances toujours plus élevées pour des domaines applicatifs extrêmement variés comme

celles qui ont été citées au début de ce paragraphe. Cette thèse étend les travaux existants en vision multi-vues en considérant conjointement le coût matériel embarqué et les performances obtenues en termes de vision. Connaissant les difficultés inhérentes à l'implémentation de réseaux de neurones sur cibles embarquées [Abd19], l'idée consiste ici à engager une structure multi-vues convexe pour diminuer le coût matériel d'un réseau et par conséquent, le temps d'exécution.

1.1.2 Apprentissage machine pour la vision : La reconnaissance d'objets

Analyser une image pour en classifier son contenu est une tâche complexe et constitue une des plus grandes difficultés en vision par ordinateur. La dimension des données issue d'une simple image couleur en est la raison principale. En effet, comment expliquer à une machine qu'un ensemble de pixels extrêmement variables de par leur position, leur intensité et leur distribution évoque un animal, un vélo ou une chaise. Ce type de tâche nécessite un algorithme capable d'identifier des motifs récurrents pour une classe d'objets donnée. Ces motifs peuvent être par exemple une bouche, des yeux ou des oreilles si l'on veut reconnaître un mammifère. Le problème de la détection de véhicules peut se ramener à repérer des roues, des phares ou bien la silhouette bien particulière d'une voiture. Ces motifs sont autant d'éléments distinctifs qui permettent aux êtres humains de comprendre leur environnement mais qui restent difficiles à appréhender et à formuler en un algorithme robuste. Cette robustesse se définit notamment par une invariance face aux diverses conditions de luminosité ou de position de l'objet vis-à-vis de l'objectif d'une caméra. Intuitivement, si une caméra classique dotée d'un unique capteur possède déjà les attributs pour différencier telle ou telle classe d'objets alors une caméra multi-vues le sera à plus forte raison. En effet, une machine multi-vues peut s'affranchir de certaines de ces contraintes en capturant la scène sous diverses positions mais ne résous pas le problème de la méthode d'extraction et de manipulation des données. Les premières propositions pour identifier les points d'intérêts d'une scène usaient de fonctions reposant sur des descripteurs comme SIFT, SURF ou HOG [D.99, BTG06, DT05] lesquels sont des métriques figées, conçues à partir de décisions humaines. Ces dernières années ont été marquées par l'expansion de l'apprentissage automatique et particulièrement aux réseaux de convolutions dédiés à la vision, au traitement du langage ou à la robotique. Si ce n'est l'architecture du réseau lui-même qui est prédéfinie, ce type d'algorithme ne se

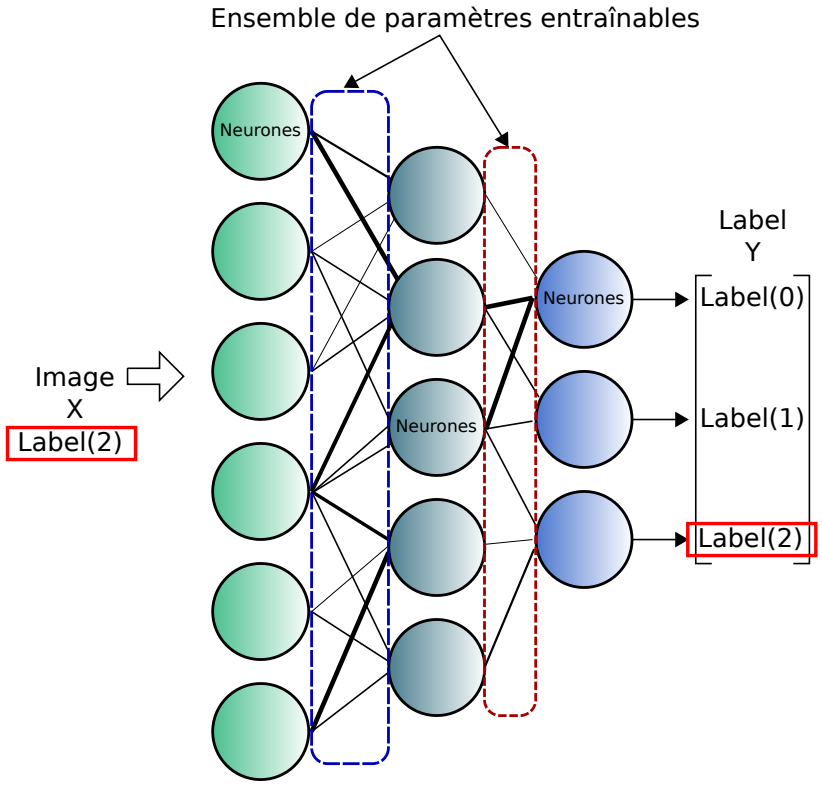


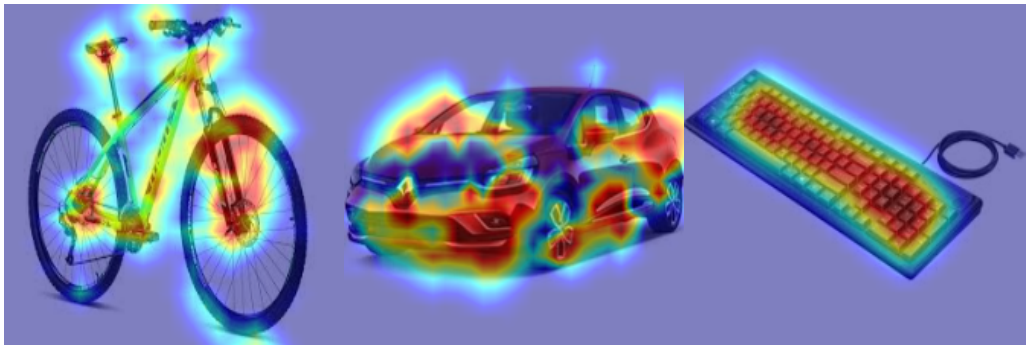
FIGURE 1.5 – Schéma simplifié d'un réseau de neurones

base sur aucune supposition en termes de caractéristiques d'image et élimine en grande partie les biais humains pour extraire les données relatives à l'objectif recherché. La méthode d'apprentissage supervisée part d'un constat simple, c'est-à-dire que le but final consiste à minimiser l'erreur E entre une sortie souhaitée Y_{train} et l'entrée X_{train} comme décrit par l'équation simplifiée 1.1 :

$$\min_{F} \; E(F, X_{train}, Y_{train}) \tag{1.1}$$

où F est un terme pour désigner l'algorithme d'optimisation par lequel un réseau de neurones apprend à corriger ses paramètres internes représentés par les connexions entre les neurones du réseau de la figure 1.1 et la sortie étant un vecteur condensant l'information utile. Par conséquent, un réseau possède de nombreux degrés de libertés pour porter son attention sur des zones spécifiques d'un objet. Les records de précision obtenus avec les réseaux de convolutions profonds ont rendu caduques toutes les anciennes approches. Les réseaux multi-vues ont suivi la même direction en proposant des résultats bien plus avancés en termes de précision mais aussi dans des domaines applicatifs plus variés.

(A) Vélo (B) Voiture (C) clavier



(D) Heatmap vélo (E) Heatmap voiture (F) Heatmap clavier

FIGURE 1.6 – Cartes d'activations d'un réseau de convolutions sur divers types d'objets

1.1.3 Apprentissage avec les réseaux multi-vues

L'apprentissage multi-vues apparaît comme une méthode complémentaire à la méthode monoculaire courante. L'idée principale consiste à réunir plusieurs types de données décrivant un objet ou plus généralement un contexte. Le terme "multi-vues" consacre plusieurs types de données dans la littérature [XTX13, Sun13, ZXXS17, LYZ19]. Dans ce cas, un réseau de neurones est dit "multi-vues" ou "multi-modal" s'il peut intégrer plusieurs données comme entrées et de natures différentes telles que :

- Un texte et une image associée
- Un texte traduit en plusieurs langues
- Une séquence vidéo et sonore
- Un histogramme, la silhouette et le squelette d'une personne dans une image
- une photo et un croquis
- Un objet capturé par un ensemble hétérogène de capteurs (TOF, RGB, stéréo)
- Plusieurs projections 2D d'un objet depuis l'espace en 3D

Dans notre cas, nous considérons un système de captation 2D comprenant N vues de même nature mais nos résultats se prêtent à être étendus à des cas hétérogènes. L'un des premiers travaux sur l'apprentissage multi-vues [BM98] étudie déjà le concept d'apprentissage avec un réseau bayesien naïf (équation 1.2) pour classifier des pages web selon les mots contenus dans une page et les mots contenus dans les liens hyper-textes associés.

$$p(C|X_0,\ X_1, ...,\ X_{n-1}) = \frac{p(C)p(X_0,\ X_1, ...,\ X_{n-1}|C)}{p(X_0,\ X_1, ...,\ X_{n-1})} \tag{1.2}$$

où C désigne la classe recherchée et $(X_0,\ X_1, ...,\ X_{n-1}) \in \mathbb{R}^n$ désignent les variables d'entrées correspondants aux différentes vues du système. Cette même idée est reprise puis améliorée dans [HSST04] avec un algorithme de recherche de corrélation CCA. La CCA cherche un couple de vecteurs $(a, b) \in \mathbb{R}^{n\times m}$ qui maximisent la corrélation ρ entre deux vecteurs de données $X = (x_0, x_1, ..., x_{n-1})^T$ et $Y = (y_0, y_1, ..., y_{m-1})^T$ tel que :

$$\rho = \underset{(a,b)}{\arg\max}(a^T X, b^T Y) \tag{1.3}$$

Le réseau de *Yu et al.* [YWT12] propose de mesurer les similarités entre des milliers d'images en tenant compte de plusieurs vues hétérogènes à savoir l'histogramme RGB, la silhouette et le squelette d'un personnage. Deux études notables datant de 2013 [XTX13, Sun13] dressent un état de l'art exhaustif pré-CNN des techniques d'apprentissage multi-vues notamment celle concernant le co-apprentissage multi-vues faisant appel au concept d'apprentissage semi-supervisé. Dans ce cas, deux classifieurs sont entraînés sur un ensemble de données annotées mais limitées en nombre et cet ensemble de départ contient deux vues d'une même classe. Ces deux classifieurs sont ensuite inférés sur les données non-annotées et la meilleure prédiction est utilisée pour annoter la donnée. Ce processus est répété de manière itérative pour atteindre les performances maximales pour chacun des deux classifieurs. Un article de synthèse de 2017 [ZXXS17] ré-auditionne la littérature scientifique sur le même sujet en pointant particulièrement les CNN comme outil de substitution aux précédentes méthodes multi-vues. De manière générale, les CNN ont permis une nette progression dans tous les secteurs de l'apprentissage machine. Par exemple, le réseau proposé par *Elkahky et al.* [ESH15] est une variante apprentissage profond des algorithmes de classification de pages webs multi-vues conçue pour améliorer

l'expérience utilisateur. Pour une requête utilisateur notée y_U et N propositions suggérées par un moteur de recherche, leur algorithme infère $N+1$ réseaux, puis combine les descripteurs de sortie 2 à 2 avec la fonction *cosine similarity* [1] (1.4) pour trouver le résultat le plus pertinent (figure 1.7). Le réseau apprend par conséquent à s'adapter vis-à-vis de la taille des descripteurs finaux et de la fonction mesure de similarités.

$$cosine\_sim(y_U, y_N) = \frac{y_U . y_N}{||y_U||.||y_N||} \tag{1.4}$$

Comme l'illustrent ces exemples, la mesure de similarités est un axe de recherche pré-

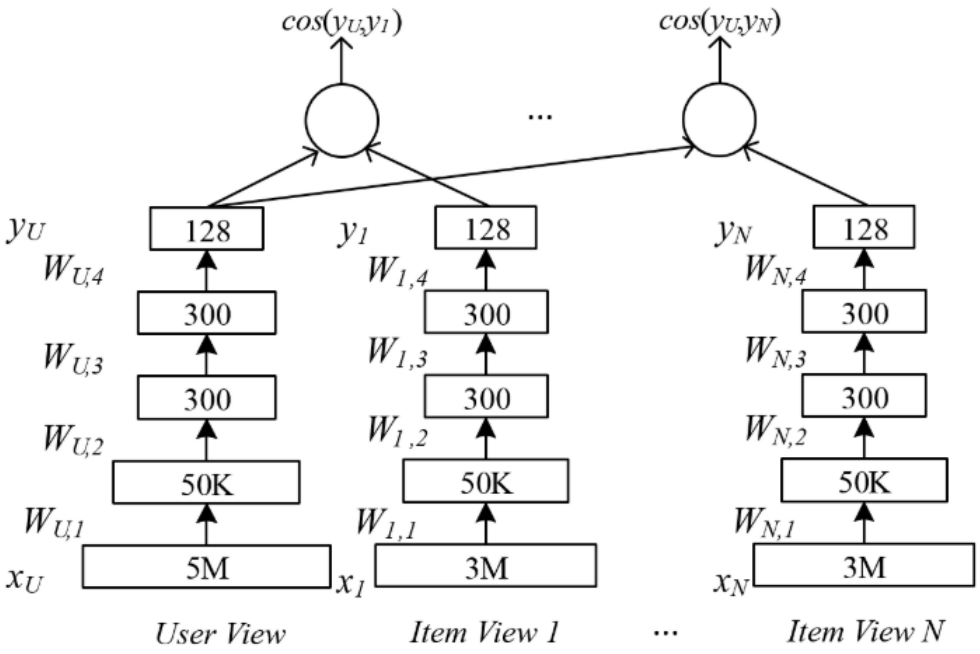


FIGURE 1.7 – Exemple de réseau multi-vues pour la requête de pages webs. Les nombres indiqués représentent le nombre de paramètres de chaque couche du réseau. Source : [ESH15]

pondérant en multi-vues puisqu'elle est au coeur de multiples algorithmes comme la reconstruction géométrique [Har97] ou la reconstruction de carte de profondeur [Hir07]. Ces applications possèdent désormais leurs variantes basées sur les réseaux de convolutions et l'apprentissage automatique. *Rocco et al.* [RAS17, RCA+18], *Choy et al.* [CGSC16], *Balntas et al.* [BJTM16], *Dusmanu et al.* [DRP+19], *Han et al.* [HLJS15] ou encore *Melekhov* [MTS+19] proposent des solutions à base de CNN pour détecter les meilleurs correspondances entre 2 images et démontrent la supériorité des modèles profond sur ce type d'exercice. Comme l'illustre la figure 1.8, un réseau de corrélation complètement

1. représentée par le produit scalaire de 2 vecteurs de même dimension

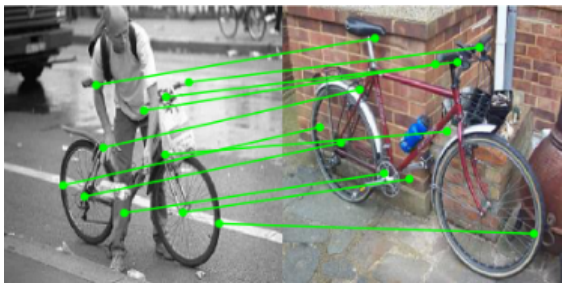


FIGURE 1.8 – Appariement de points d'intérêts par un réseau de neurones. Source : [RCA+18]

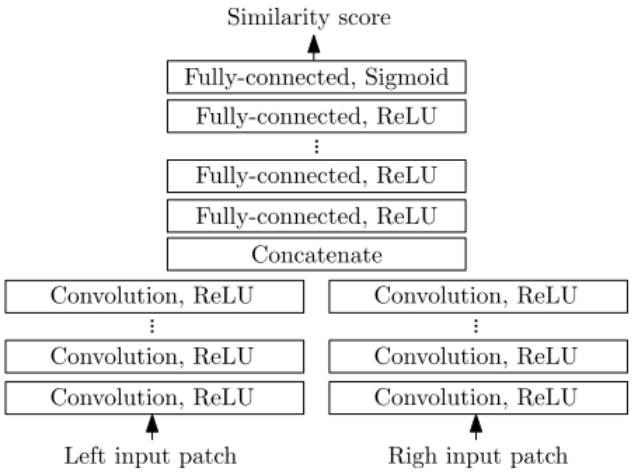


FIGURE 1.9 – Réseau de correspondances denses proposé par *Zbontar et Lecun*. Source : [ŽL16]

paramétrable peut reconnaître des similitudes entre des images extrêmement différentes. La reconstruction 2.5D (carte de profondeur) utilise ces mêmes principes mais avec une densité de correspondance plus élevée. Les algorithmes usuels considèrent par exemple diverses métriques pour mesurer les similarités, donc les disparités, telles que l'entropie [Hir07], le flot optique [BMK13], la NCC [HLL11], la SAD ou la SSD [SSZ01] pour identifier les meilleures correspondances entre 2 caméras voisines. Un CNN utilise sa propre métrique pour minimiser au mieux l'écart entre la vérité terrain et ce qui lui est fourni en entrée. La figure 1.9 montre un des premiers réseaux à introduire le concept d'apprentissage [ŽL16] pour la reconstruction 2.5D. La carte de profondeur générée par le réseau contient en moyenne 10% moins d'erreur que le précédent algorithme de l'état de l'art [Hir07]. Plusieurs améliorations ont réussi à surpasser ces résultats, notamment le réseau de *Lian et al* [LFG+18] qui amène une structure entièrement paramétrée et donc entraînable de bout en bout. Une autre méthode introduite par *FlowNet* [DFI+15] consiste à forcer un réseau à estimer le flot optique entre deux caméras. *Yao et al.* [YLL+18] parviennent même à construire un réseau capable d'estimer les disparités avec deux caméras

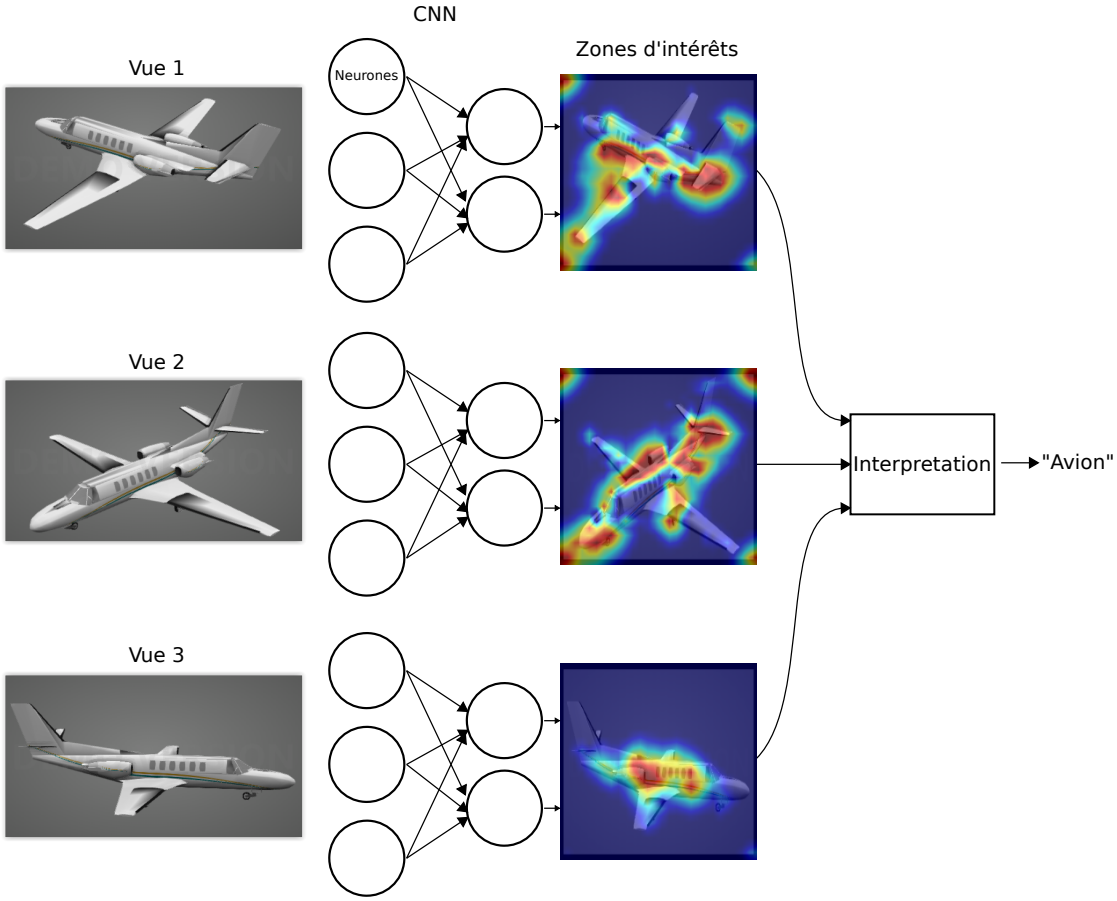


FIGURE 1.10 – Capture multi-vues et traitement par de multiples réseaux de convolutions. L'étape finale consiste à fusionner les sorties de chaque CNN.

non rectifiées, c'est à dire dont les positions n'ont pas été alignées et dont les aberrations optiques n'ont pas été compensées. Enfin, *Poggi et al.* [PPTM19] propose une méthode permettant de reconstruire une carte de profondeur à partir d'une seule et unique caméra.

1.1.4 Classification d'images avec les réseaux multi-vues

A la place de mesurer des similarités inter-caméras, les systèmes de reconnaissance multi-vues suggèrent au contraire d'étudier un objet sous plusieurs angles de vues pour en extraire le plus de détails possibles. C'est dans ce contexte qu'un CNN multi-vues peut se montrer plus robuste notamment face aux variations de positions de l'objet observé. La figure 1.10 simplifiée illustre un exemple où plusieurs caméras infèrent un réseau pour capter le maximum de détails. Même si cette logique est simple à concevoir, il n'en reste pas moins difficile de trouver les leviers avec lesquels un réseau peut regrouper plusieurs

sorties pour en extraire un unique résultat de classification. Un raisonnement simple voudrait pondérer ces résultats mais implique que chaque image soit traitée par un réseau qui lui sera propre. Cette logique n'est pas optimale dans un contexte d'électronique embarquée où chaque caméra devrait supporter plusieurs millions d'opérations MAC par seconde pour traiter les entrées en temps-réel. Plutôt que de promouvoir un gain en terme de performances d'utilisation, un CNN pourrait tout à la fois se distribuer en terme de perception et d'inférence sur un matériel adapté. Par matériel adapté s'entend ici une électronique capable de capturer de multiples images de manières synchronisées et d'en extraire l'information d'intérêt par un CNN potentiellement moins coûteux en termes de calculs. Cette dernière phrase résume parfaitement le fruit du travail effectué dans cette thèse, à savoir adapter un réseau à un contexte multi-vues pour le faire fonctionner de façon efficace sur une cible matérielle dédiée aux opérations de convolutions.

1.1.5 Structure du manuscrit

Comme expliqué dans la section 1.1.4, l'idée qui sera développée tout au long de cette thèse consistera à embarquer un réseau d'origine plus performant grâce au multi-vues puis de dégrader ses résultats en diminuant sa quantité de paramètres via une méthode de compression. Comme cela sera montré dans les chapitres suivants, ceci se traduira automatiquement par une meilleure intégration du modèle flots de données sur FPGA. La suite du manuscrit est partitionnée en quatre chapitres :

- Le chapitre 2 montre analyse les réseaux de l'état de l'art mono et multi-vues dédiés à la reconnaissance d'objets.
- Le chapitre 3 est scindé en deux parties. La première analyse des méthodes de compression des réseaux de convolutions. La deuxième est la contribution principale de la thèse, c'est à dire combiner le principe de compression au principe multi-vues pour démontrer que le nombre de paramètres restant est inférieur à celui d'un réseau mono-vue.
- Le chapitre 4 oriente la discussion vers les modèles d'accélération matériel connus sur FPGA. Un modèle d'accélération hybride plus performant est obtenue à partir d'une architecture multi-caméras.

- Le chapitre 5 montre les étapes du développement technique de la caméra multi-vues issue des précédentes démonstrations.
- Une annexe est aussi proposée sur un modèle d'accélération particulier destiné au réseau MobileNetv2.

Chapitre 2

Les réseaux de neurones convolutifs mono et multi-vues

Résumé : *Ce chapitre fournit un état de l'art des réseaux convolutifss mono et multi-vues. L'idée générale d'apprentissage automatique pour la reconnaissance d'objets est présentée dans un premier temps. La deuxième partie est consacrée aux évolutions architecturales qui ont permis aux CNN de s'imposer comme la méthode de référence pour résoudre le problème de classification d'images. La dernière section présente les réseaux multi-vues et leurs atouts comparés aux systèmes de classification mono-vues malgré leur coût plus élevé en terme d'opérations de calculs.*

Les méthodes d'apprentissage profond (*deep learning*) connaissent un immense succès depuis les résultats de classification du réseau AlexNet [KSH12] au challenge ILSVRC'12. Le but de cette compétition consistait à proposer un algorithme capable de classifier le dataset ImageNet [DDS+09] contenant 1000 catégories d'objets distincts. Les algorithmes de reconnaissance d'objets étaient jusqu'alors construits autour de programmes conçus manuellement [D.99, BTG06, DT05] capables de condenser puis de réorganiser les informations contenues dans une image en de multiples vecteurs de données structurées. Les données contenues dans ces vecteurs sont donc issues d'un traitement expert et ne garantissent pas les conditions suffisantes pour séparer efficacement chacune des classes contenues dans le *dataset*. Par séparation s'entend une distance inter-classes Dans ces conditions, il est difficile d'appréhender ce type de problématique et de créer un algorithme spécifique dédié à une tâche aussi complexe que la classification d'images multi-classes.

Les réseaux de neurones artificiels, et en particulier les CNN, ont été modélisés en

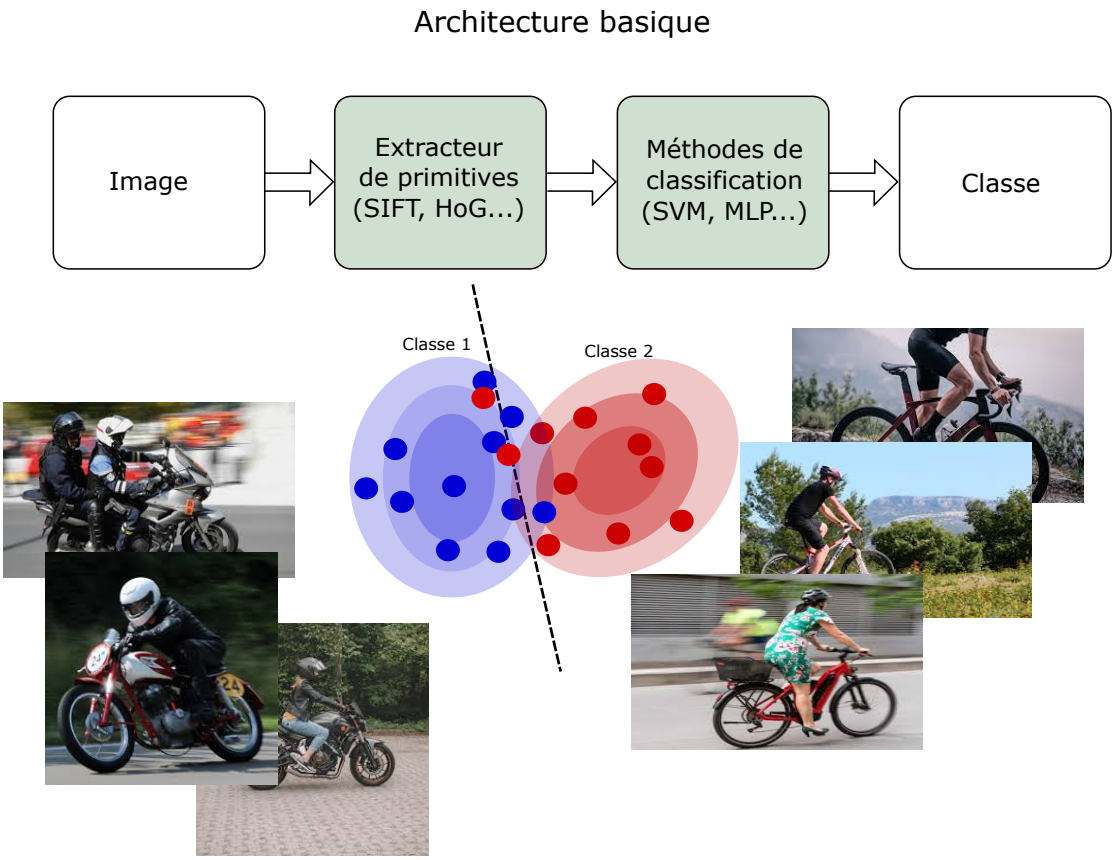


FIGURE 2.1 – Algorithmes traditionnels de classification d'image : L'étape d'extraction notée f peut être considérée comme une transformation d'un espace de grande dimension (ici la taille de l'image notée $\mathcal{D}$) vers un espace de dimensions plus réduites (ici représenté par un espace en deux dimensions), $f : \mathcal{D} \rightarrow 2$. Les primitives sont par exemple l'histogramme des gradients HOG, un jeu de vecteurs SIFT de l'image d'entrée.

s'inspirant de la structure interne neurones/synapses du cerveau et ont démontré des niveaux de performance plus élevées que ceux obtenus avec les méthodes *expertes* de classification d'images. L'apprentissage machine permet de supprimer l'intervention humaine dans le processus d'extraction d'informations. Bien que ce type d'algorithme généraliste existe depuis des décennies [LBD+90], leur exploitation n'a été possible que grâce à la disponibilité d'accélérateurs à base de processeurs graphiques (GPU) suffisamment rapides sur des tâches à parallélisme de données avancées et capables d'entraîner ce type de réseaux dans une fenêtre de temps relativement courte de quelques jours contrairement aux processeurs standards nécessitant plusieurs semaines. La facilité de prise en main des outils de *deep learning* tels que *Caffe* [JSD+14], *TensorFlow* [ABC+16] ou *Pytorch* [SDC+19] ont largement contribué à la démocratisation de l'apprentissage profond. Au delà de la simple classification, l'apprentissage profond pour le traitement d'images est aussi employé en détection [GDDM14, RF16] ou en segmentation d'objets [HGDG20], pour l'interpolation d'image haute résolution [YZT+19], l'estimation de pose [KMT+17], la compression vidéo [GPY+20] ou encore l'analyse de corrélation de points d'intérêts inter-images [CGSC16]. Les méthodes d'apprentissage multi-vues permettent de renforcer la qualité de prédiction des réseaux mono-vues classiques. Il paraît intuitivement logique que la géométrie de la scène et la position de la caméra vis à vis d'un objet, influe sur le score de classification. Par exemple, il est difficile à une machine de différencier un vélo d'une moto si la caméra ne perçoit qu'une roue, ou une tasse à café d'un verre si la caméra ne perçoit pas l'oreille de la tasse. Ce type d'exemple souligne l'avantage d'un système de caméras multi-vues par rapport à un système mono-vue classique. Une marge de progression est donc exploitable avec plusieurs vues d'un même objet. L'objectif de ce chapitre est de démontrer la supériorité des architectures multi-vues, en particulier pour la classification d'objets, avec un état de l'art des travaux traitant de ce type de systèmes. La première partie de ce chapitre introduit les notions d'apprentissages profonds (*deep learning*) et les ramifications architecturales autour des réseaux de l'état de l'art. La deuxième partie présente les CNN multi-vues pour la classification ainsi que divers domaines d'application. La suite de la thèse est construite sur une structure de CNN multi-caméras permettant à la fois de **préserver** la précision de classification comparé à sa contre-partie mono-vue, sans nécessiter plus de ressources logiques.

2.1 Modèle de neurone et CNN mono-vue

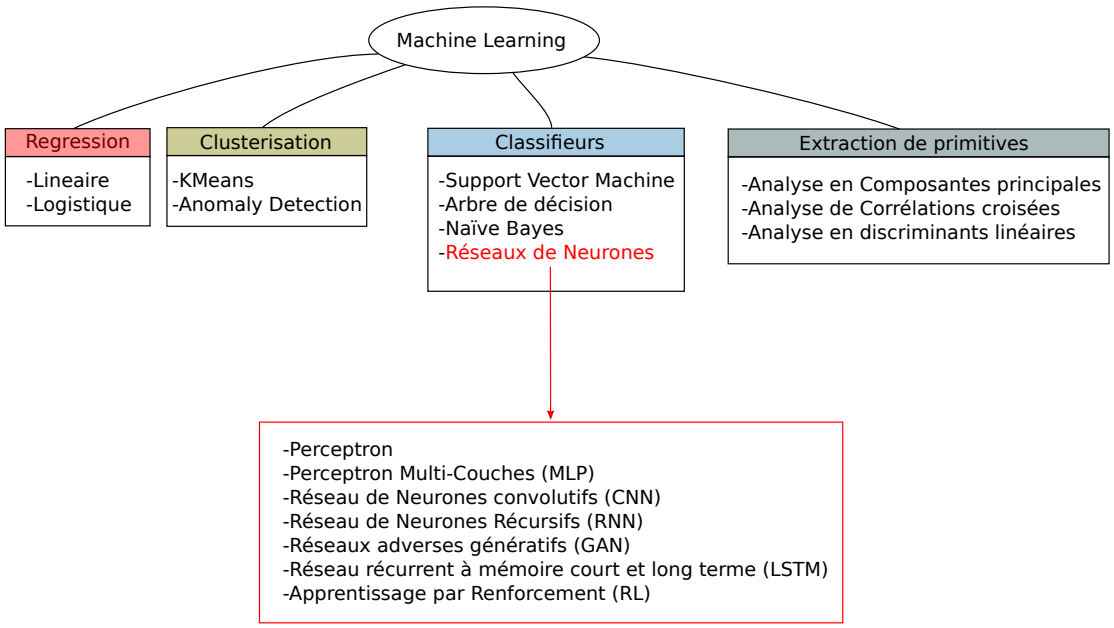


FIGURE 2.2 – Liste non exhaustive d'algorithmes d'apprentissage machines.

En mathématiques statistiques, les algorithmes d'apprentissage sont largement utilisés pour produire des approximations de modèles (régression), de partitionnement de données (*clusterisation*), pour classifier des données ou pour compresser les dimensions d'une donnée (extraction de primitives) comme montré dans la figure 2.2. L'ensemble des algorithmes d'apprentissage profond constitue une sous-catégorie des algorithmes d'apprentissage. Ces architectures s'inspirent de la structure neurones/synapses du cerveau où plusieurs milliards de neurones artificiels sont interconnectés. Un neurone est représenté par une fonction d'activation des sommes pondérées des états des neurones précédents comme exprimé dans l'équation 2.1 et montré par la figure 2.3.

$$Y = f(W^T.X + b), \forall (X, W) \in \mathbb{R}^N, \forall (Y, b) \in \mathbb{R}^2. \tag{2.1}$$

où X sont les entrées du neurones disposées en un vecteur colonne de dimension N, W est un vecteur colonne de N paramètres, b est un paramètre de biais d'apprentissage, f est une fonction d'activation et Y est un scalaire représentant la sortie du neurone.

Ces modèles sont conçus pour apprendre à reconnaître automatiquement des motifs dans la structure des données mais doivent être aussi capables de généraliser sur des

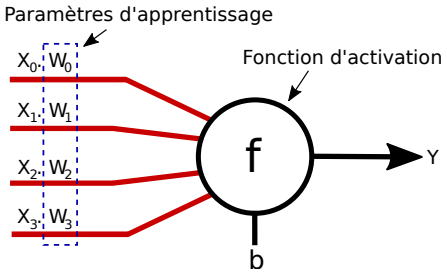


FIGURE 2.3 – Modèle de neurone

entrées inconnues. L'entraînement reste une étape cruciale et nécessite généralement plusieurs milliers d'échantillons qui sont soit annotés par des humains (on parle d'apprentissage supervisé) ou non, auquel cas un réseau est chargé d'extraire de lui-même les informations utiles pour catégoriser les données (apprentissage non supervisé). L'étape finale consiste à embarquer ces architectures sur des ordinateurs, robots ou caméra intelligentes pour traiter les données brutes émises par des capteurs et pour prédire l'état du système grâce au réseau pré-entraîné. Parmis cette famille, les Perceptron multi-couches (MLP), les Réseaux de Neurones Convolutifs (CNN) ou les Réseaux de Neurones Récurrents (RNN) sont largement considérés pour leurs performances en classification d'images, de la parole et de texte. A partir de ce principe général de fonctionnement, des innovations successives ont permis de réduire le nombre de paramètres en répétant des noyaux de convolutions identiques sur toutes les données d'entrée, d'employer des fonctions d'activations efficaces et d'apprendre des paramètres adaptés aux données.

2.1.1 Premier réseau appliqué à la reconnaissance d'image : le perceptron multi-couches (MLP)

Les MLPs (figure 2.4) sont une des formes les plus anciennes de réseaux de neurones artificiels [RHW86] et permettent de résoudre des problèmes de classification non-linéaires tel que la séparation de groupes de données en C classes ($C \geq 2$) contrairement aux perceptrons mono-couches limités aux problèmes de classification séparables linéairement. En général, les réseaux sont considérés comme profonds lorsqu'il existe plus d'une couche entre l'entrée et la sortie. En considérant le cas de la classification d'images en niveau de gris, l'entrée d'un MLP est un vecteur X de dimension $U \times V$ dans le cas de classification d'images de largeur U et de hauteur V. L'image est donc "rasterisée" et

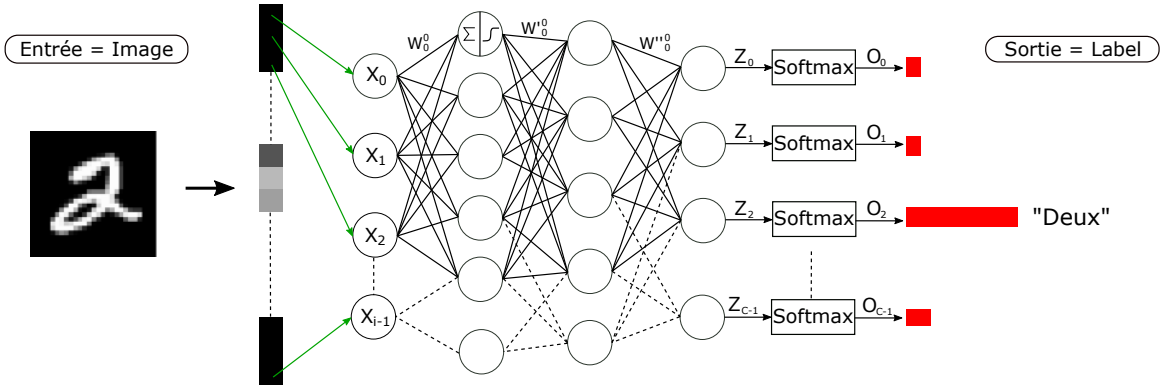


FIGURE 2.4 – Graphe d'un classifieur MLP.

l'information liée à la localité des pixels est perdue. La première couche possède N neurones où chacun d'eux est lié aux entrées par $U \times V$ paramètres. La sortie (ou activation) du neurone i ($0 \leq i \leq N-1$) est donc la somme pondérée de toutes les entrées et d'une fonction d'activation σ telle que :

$$Y_i = \sigma(W_i^T.X_i + b_i), i < N \tag{2.2}$$

où un neurone est lié à W_i, un vecteur colonne de paramètres d'apprentissage de dimension $U \times V$, b_i est un biais d'apprentissage et σ décide de l'état de la sortie par une fonction de non-linéarité, typiquement représentée par une fonction tanh, sigmoïd ou ReLU. Cette fonction décide si la sortie doit être activée (pour les grandes valeurs) ou annulée (pour les petites valeurs). La profondeur du réseau D permet d'augmenter le niveau d'absraction des informations extraites à partir des données d'entrée. La sortie du MLP est un vecteur z de C éléments (ou logits) où chaque élément est normalisé par la fonction $softmax$ (équation 2.3). L'élément de plus forte magnitude correspond au label prédit par le réseau.

$$O_i = \frac{e^{z_i}}{\sum_{k=0}^{k=C-1} e^{z_k}}, i < C \tag{2.3}$$

Ce modèle de réseau est dit à propagation directe (feed-forward) dans le sens où les données traversent la structure une seule et unique fois pour obtenir le résultat de classification. Tous les paramètres (W_i, b_i) du réseau sont appris pendant l'étape d'entraînement grâce à un algorithme d'optimisation sur un set de P données. Un set d'entraînement

$((I_0, O_0), (I_1, O_1), ..., (I_{P-1}, O_{P-1}))$ est divisé en plusieurs sous parties (*mini-batch*) successivement propagées dans le réseau. A chaque *mini-batch*, l'algorithme calcule une erreur *Loss* en mesurant la distance entre la vérité terrain et la valeur prédite par le réseau. Dans le cas d'un classifieur multi-classes, la distance est généralement mesurée avec une fonction de coût entropie croisée (*Cross Entropy Loss*) donnée par l'équation 2.4, ou par une erreur quadratique moyenne (*Mean Square Error*).

$$Loss = -\frac{1}{C}\sum_{i=0}^{C-1} p_i \log(q_i) \tag{2.4}$$

avec p_i la densité de probabilité de l'image connue, q_i la densité de probabilité de l'objet donnée par le réseau MLP.

Un algorithme de descente de gradient tel que ADAM [KB14], optimise les paramètres W_i et b_i (équations 2.5 et 2.6) par rétro-propagation [RHW86] de l'erreur dans les couches du réseau.

$$W_i \leftarrow W_i - \eta \times \frac{\partial Loss}{W_i} \tag{2.5}$$

$$b_i \leftarrow b_i - \eta \times \frac{\partial Loss}{b_i} \tag{2.6}$$

où η ($\eta \in \mathbb{R}$) est un hyper-paramètre (taux d'apprentissage) choisi manuellement.

D'une manière générale, le cycle d'entraînement nécessite plusieurs milliers d'itérations pour qu'un MLP converge vers une solution stable telle que l'erreur de prédiction soit minimale. Cependant ce type d'architecture souffre de plusieurs problèmes concernant la classification d'images :

- le nombre d'entrées dépend directement du nombre de dimensions et de pixels dans une image, limitant l'application à des images de petites dimensions [LC10]
- le réseau est sujet au phénomène de surapprentissage car chaque pixel d'entrée possède un paramètre qui lui est propre.
- Enfin, le réseau ne perçoit pas les relations spatiales entre pixels.

2.1.2 Notations

Pour le reste du manuscrit les notations suivantes on été adoptées :

- Soit D le nombre de couches d'un réseau
- Soit N le nombre de neurones dans une couche
- Soit k la taille d'un noyau de convolution
- Soit s le pas de sous-échantillonage
- Soit p le nombre de pixels de remplissage

2.1.3 Les réseaux de neurones convolutifs : CNNs

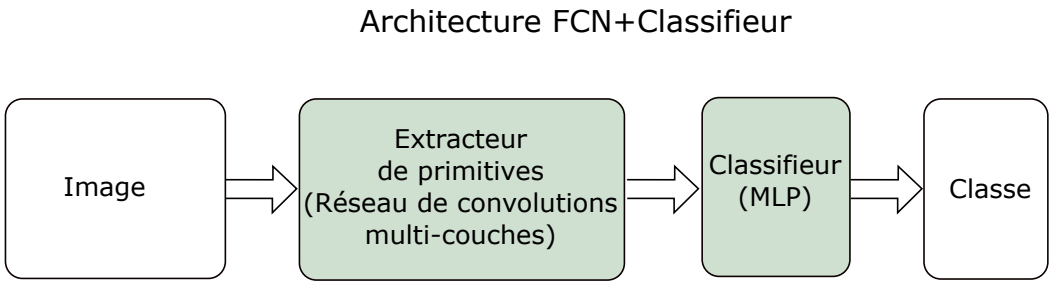


FIGURE 2.5 – Schéma simplifié d'un CNN. L'extracteur de primitives FCN (*fully convolutional network*) prétraite une image d'entrée par un succession de couche de convolutions et de fonctions de non-linéarité. Chaque paramètre de convolution est une variable optimisée par un algorithme de descente de gradient. Le vecteur de sortie est ensuite interprété par un réseau MLP qui attribue un label à l'image d'entrée.

Les architectures CNNs sont devenus l'état de l'art pour diverses tâches comme la reconnaissance, la détection ou la segmentation d'objets. Depuis la première étude publiée par Lecun et al. [LBD+90], les structures ont fortement évoluées ces dernières années pour augmenter les taux de reconnaissance sur des datasets beaucoup plus difficiles à classifier que MNIST tel que CIFAR10 [Kri09], ImageNet [DDS+09], PascalVOC12 [JZL+] ou COCO [LMB+14]. Ces jeux de données regroupent jusqu'à un millier de catégories d'objets différents et nécessitent des architectures plus complexes que le simple classifieur évoqué dans la section précédente. En particulier, le dataset ImageNet contient 1.2M d'images RGB réparties entre 1000 catégories d'objets pour le set d'entraînement et 50K pour la partie validation, ce qui en fait un des datasets le plus difficile à classifier.
Dégrouper autant de classes d'objets nécessite un algorithme capable d'isoler plusieurs milliers de dimensions différentes à partir des images sources afin de discriminer chaque

catégorie d'objets puis d'être capable généraliser les prédictions sur des images en dehors du dataset. Un CNN répond à ce problème en intégrant un pipeline de couches de convolutions en amont d'un MLP(figure 2.5) où chaque étage du pipeline produit des données avec un niveau de détails (ou niveau d'abstraction) de plus en plus fin suivant la profondeur du réseau.

L'idée générale consiste donc à produire un vecteur de données qui condense toutes les informations d'une image. Autrement dit, les données de l'image sont projetées sur chacun des axes d'un espace à F dimensions où F représente la taille du vecteur final. Pour donner un exemple concret, la figure 2.6 représente une architecture type avec $D = 3$ couches de convolutions. Une couche de convolutions d'indice i ($1 \leq i \leq D$) est organi-

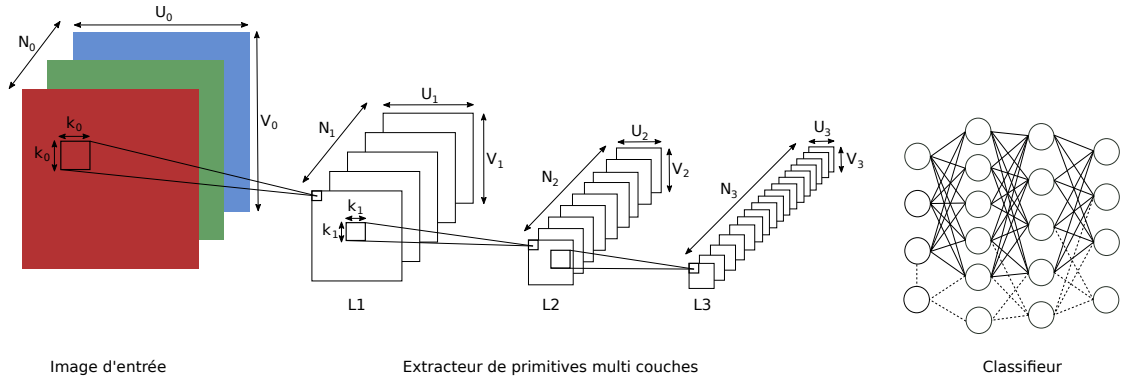


FIGURE 2.6 – Architecture générique d'un CNN. L'image source subit plusieurs transformations suivant les couches L1, L2 et L3. La sortie de L3 génère un vecteur de taille $N_3 \times U_3 \times V_3$ utilisé comme donnée d'entrée d'un classifieur.

sée en N_i canaux de N_{i-1} filtres convolutions 2D de taille $k_i \times k_i$. L'opération s'effectue sur un volume de données X (ou tenseur) de dimensions $[N_{i-1}, U_i, V_i]$. La relation F_i entre le tenseur d'entrée X_i de N_{i-1} primitives et la sortie du n-ème canal de convolution $Y_i(n)$ est donnée dans l'équation suivante :

$$Y_i(n) = F_i(X_i), \text{ avec } X_i = Y_{i-1} \text{ et } n < N_i \tag{2.7}$$

$$= \sigma(\sum_{j=1}^{N_{i-1}} W_j(n) * X_i + b(n)) \tag{2.8}$$

où $W(n)$ est un volume de convolutions 2D de dimensions (N_{i-1}, k_i, k_i), b_n est un biais d'apprentissage propre au canal de convolution. Chaque canal possède donc N_{i-1} filtres de convolutions 2D dont les poids sont des paramètres entraînés par descente de

gradient. σ est une fonction d'activation définie sur $\mathbb{R}$ essentiellement représentée par une fonction RELU, une fonction tangente hyperbolique ou sigmoïde comme vu dans la figure 2.7 :

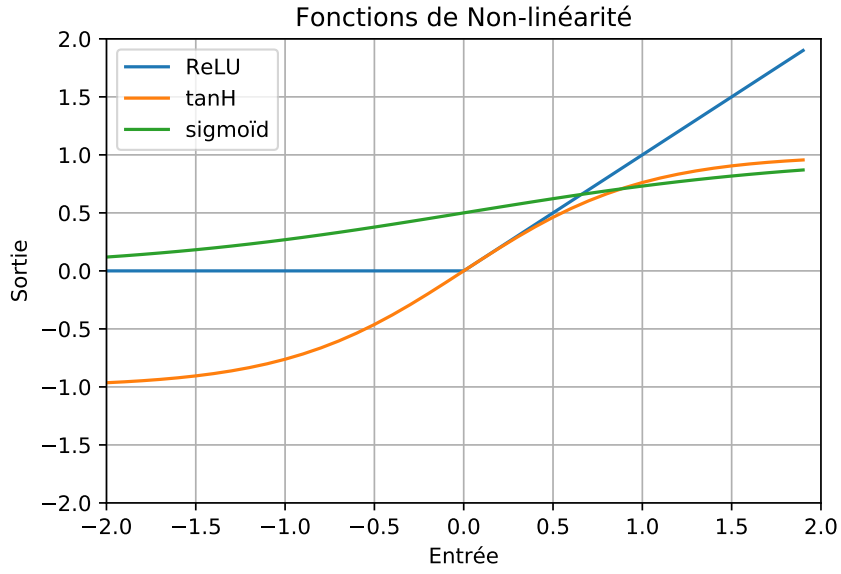


FIGURE 2.7 – Exemples de fonction de non-linéarité

Le nombre d'opérations de mutliplication accumulation (MAC) par canal pour l'inférence est proportionnel à la taille des kernels de convolutions, aux dimensions et aux nombre de primitives d'entrées tel que :

$$MACops(N_i) = \underbrace{N_{i-1} \times k_i^2}_{Nombre\ de\ parametres} \times \frac{(L_{i-1}+2p)(H_{i-1}+2p)}{s^2} \tag{2.9}$$

où (L_{i-1}, H_{i-1}) représentent la largeur et la hauteur des primitives en entrée de la couche L_i et (p,s) sont respectivement des hyper-paramètres de *padding* **p** et de *stride* **s**. Ces deux paramètres sont illustrés par la figure 2.8. Le *stride* permet de sous-échantillonner les données en entrée et donc de réduire le nombre d'opérations par primitives. Le padding rajoute des pixels nuls au bord des activations afin que les fenêtres de convolutions puissent s'appliquer au pixels au bord de chaque primitives. La figure 2.9 permet de visualiser concrètement le type de données certaines primitives à différents niveaux d'abstraction. Les récents progrès des CNNs sont largement imputables aux nouvelles micro-architectures de couches de convolutions et à la profondeur des réseaux. Le paragraphe ci-dessous regroupe les évolutions architecturales notables.

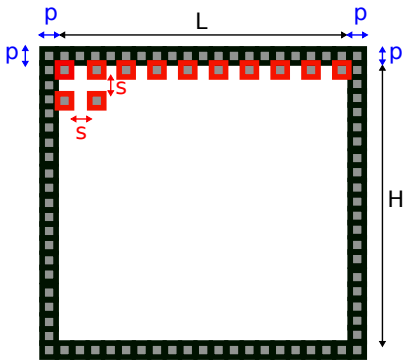


FIGURE 2.8 – Illustration du padding et du sous-échantillonage

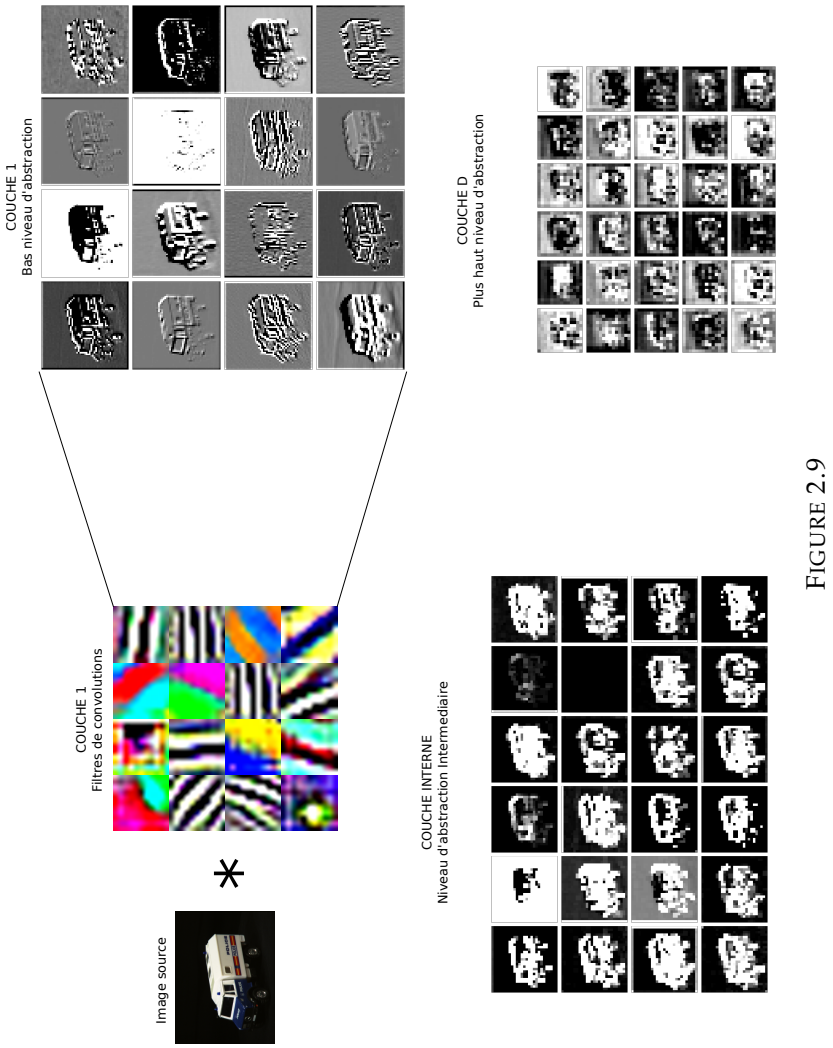


FIGURE 2.9

2.1.3.1 ALEXNET et VGG

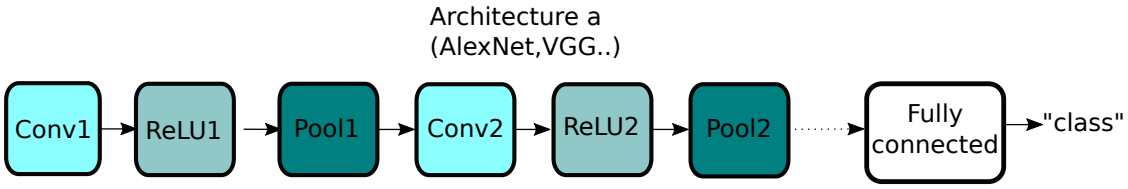


FIGURE 2.10 – Architecture de type AlexNet

Le réseau AlexNet a démontré la supériorité des réseaux de neurones convolutifs lors la compétition ILSVRC2012. Ce CNN fut le premier algorithme entièrement basé sur le concept d'apprentissage automatique à classifier le dataset ImageNet et à atteindre un taux d'erreur top-5 de l'ordre de 15%, surclassant ainsi le meilleur algorithme basé sur un concept d'extraction de données purement arbitraire [SP11]. Le temps d'entraînement représentait une des contraintes majeure mais *Krizhevsky et al.* ont proposé l'idée d'exploiter la puissance de calcul des processeurs graphiques GPU ou encore de remplacer la fonction d'activation tanH ($\mathbb{R} \rightarrow [-1;1]$) par une fonction ReLU ($\mathbb{R} \rightarrow \mathbb{R}^+$) (Rectified Linear Unit) plus simple à calculer. Ce CNN est composé de 5 couches de convolutions et d'un MLP à 3 étages pour un total de 0.66 GMAC opérations pour classifier une image RGB de dimensions 227x227. Le CNN VGG [SZ14] proposé par *Simonyan et al.* démontre expérimentalement que la profondeur d'un réseau a un impact significatif sur la précision de classification. En augmentant le nombre de couches, le niveau d'abstraction à chaque étape du réseau augmente aussi et permet de capturer plus d'informations à partir de l'image source. Malgré des performances supérieures à AlexNet, le nombre d'opérations constitue le problème majeur de ce réseau puisqu'il requiert 55x plus d'opérations (40 GMACs pour VGG-19) pour classifier une image.

2.1.3.2 GoogleNet

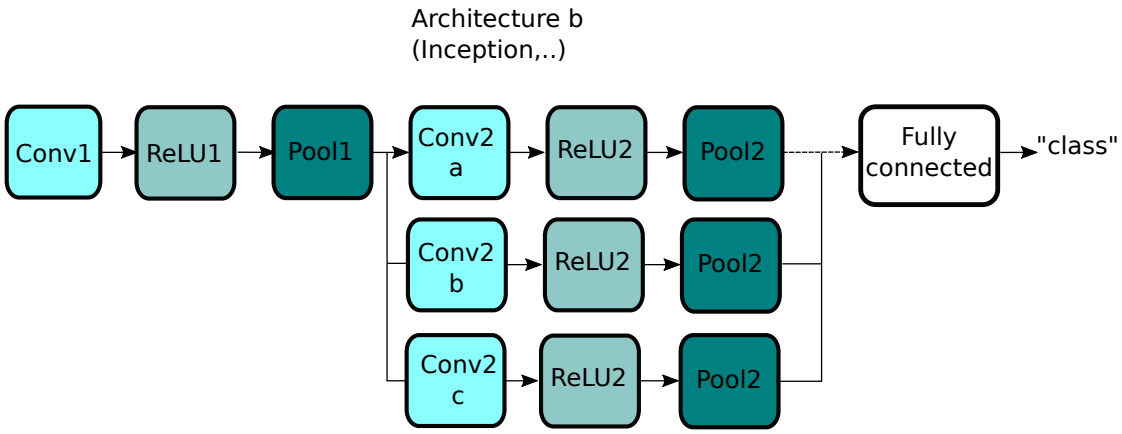


FIGURE 2.11 – Architecture de type GoogleNet.

GoogleNet (figure 2.11) introduit les modules *Inception* basés sur le principe de mini réseaux imbriqués dans le réseau précédemment introduit en [LCY13]. Ces modules consistent à concaténer des canaux contenant des fenêtres de convolutions différentes ($k = 1, k = 3, k = 5$) afin de réduire les redondances structurelles et théoriquement augmenter la richesse des informations captées à chaque noeud du réseau.

2.1.3.3 RESNET, DENSENET

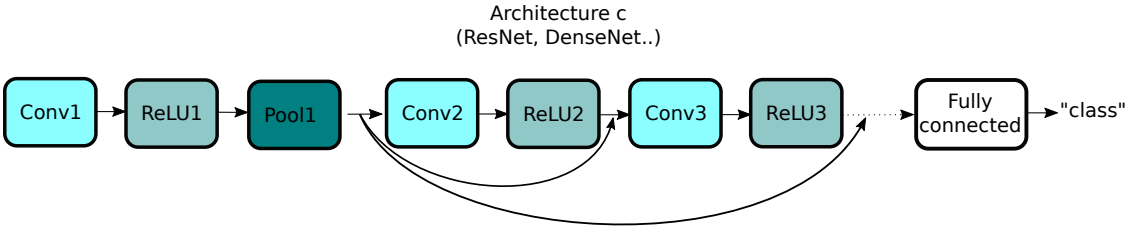


FIGURE 2.12 – Architecture à résidus

Les expériences menées par *He et al.* avec le réseau ResNet [HZRS16a] ont mis en évidence les limitations des architectures très profondes. Lorsque le nombre de couches devient trop important ($D \geq 20$), la norme du gradient d'erreur tend à diminuer de façon drastique pendant la rétro-propagation [GB10] et pousse les paramètres de convolutions à des valeurs infinitésimales. Ce problème mène à des taux d'erreur de classification supérieures à des réseaux moins profonds. La solution proposée par les auteurs consiste à court-circuiter certaines convolutions $\mathcal{F}_i$ en additionant le gradient de la i-ème couche avec celui de la $i+1$-ème comme montré dans la figure 2.12. En propagation directe, la

sortie Y_i de la couche L_i devient :

$$Y_i = \mathcal{F}_i(Y_{i-1}) + Y_{i-1} \text{ si et seulement si } [N_{i-1}, L_{i-1}, H_{i-1}] = [N_i, L_i, H_i] \tag{2.10}$$

Avec cette nouvelle structure de convolutions, le CNN ResNet parvient à converger vers une solution optimale même avec 1001 couches [HZRS16b] et à un taux d'erreur de classification top-5 inférieur à 5%. Ce seuil est symbolique dans le sens où il définit la limite de reconnaissance de l'être humain sur ce même *dataset*. Ce même concept est repris dans le réseau DenseNet [HLVDMW17] avec une densité de connexions résiduelles supérieure (figure 2.12). Un bloc DenseNet consiste en plusieurs couches de convolutions dont les sorties sont connectées à la dernière telle que :

$$Y_i = \mathcal{F}_i(Y_{i-1}) + Y_{i-1} + Y_{i-2} + ... + Y_{i-R} \tag{2.11}$$

R correspondant à la profondeur du bloc. Les auteurs montrent qu'un réseau de ce type avec 100 couches et 800K paramètres de convolutions offre une précision similaire à une structure ResNet à 1001 couches et 10M paramètres.

A tous ces modèles d'architectures peut s'ajouter un utilitaire aujourd'hui massivement déployé dans les réseaux actuels, le module *batchnorm* [IS15]. Le but de ce module est de réduire la variance et de recentrer les valeurs des activations autour de zéro grâce à un filtre γ de même dimensions que les kernels de convolutions et un biais b. En reprenant l'équation 2.8, une activation se réécrit :

$$Y'_i(n) = \gamma_i(n) * Y_i(n) + b_i(n), \text{ où } n < N_i \tag{2.12}$$

Cette rétrospective non-exhaustive met en évidence l'importance de la complexité d'un CNN, non pas en termes de paramètres mais plutôt en termes de niveaux d'abstraction, comme l'atteste la figure 2.13 proposé par *Caziani et al.* [CPC16]. Parmi tous les exemples d'architectures cités, l'emploi de connexions résiduelles (ResNet) et la diversité des informations (GoogleNet) permettent clairement de diminuer le taux d'erreur de classification par rapport à une structure uniquement basée sur la démultiplication du nombre de canaux par couches. Existent-t-ils des leviers autres que la profondeur et la densité d'un réseau CNN pour augmenter les performances de classification ? La section

suivante présente une clé supplémentaire pour diminuer le taux d'erreur d'un classifieur à base de CNN grâce aux réseaux multi-vues.

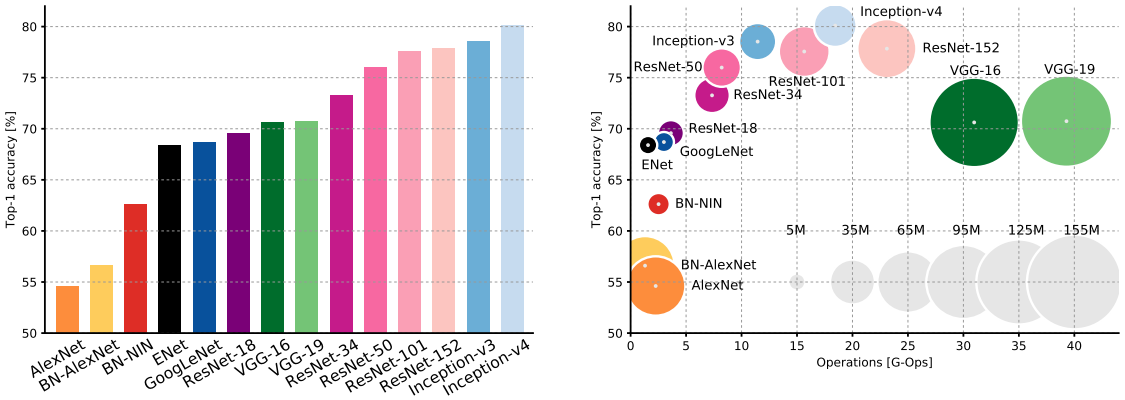


FIGURE 2.13 – Taux de classification des CNNs de l'état de l'art à gauche. La figure adjacente rapporte la précision top-1 en fonction du nombre de MAC opérations. Source : [CPC16]

2.2 Classification d'images avec les CNNs multi-vues

2.2.1 CNNs multi-vues pour la reconnaissance d'objets

La reconnaissance d'objets 3D rigides et non-rigides avec des réseaux CNNs est un sujet récemment abordé dans la littérature. Plusieurs travaux ont déjà proposé des solutions à base de réseaux de convolutions qualifiés pour la classification, la détection, la segmentation, l'estimation de pose, la reconstruction 3D et la mesure de similarité entre plusieurs vues. Dans une étude récente (2019) [ASS+18], *Amhed et al.* dressent un bilan représentatif des différentes structures de réseaux CNNs 3D et multi-vues. Les auteurs de l'enquête pointent plus particulièrement 2 principaux points de divergence entre les réseaux existants, c'est à dire, **1)** la manière d'aborder les problèmes avec différents types de données d'entrées (dualité entre la représentation 3D et la représentation multi-vues) et **2)** la façon dont sont construits les réseaux pour répondre à une ou plusieurs problématiques. En effet, les types d'entrées conditionnent la façon dont le réseau doit traiter les données. Par exemple, un objet rendu par un nuage de points [Qi, RSP19] ou par un ensemble de voxels [MS15, Sha, SZAB17] appelle un CNN capable de gérer les 3 dimensions (x, y, z) tandis qu'un objet capturé sous différents angles de vues ou par carte de profondeur ne nécessite pas d'architecture fondamentalement différente de celles déjà existantes dans les CNN (2.1.3) comme cela sera montré plus loin dans cette même section. D'un point

de vue de l'utilisation pratique, les systèmes RGB classiques sont plus accessibles que les capteurs Lidar ou 2.5D. D'autre part, les architectures de CNN 3D sont consommatrices en termes de volume d'opérations et de mémoires puisque l'objet est défini dans $\mathbb{N}^3$. Puisque le thème de la thèse est centré sur la conception d'une caméra multi-vues à capteurs conventionnels, nous ne nous intéresserons qu'aux cas de réseaux multi-vues RGB. L'idée de ce type d'architecture consiste à capturer des informations habituellement inaccessibles à de simples imageurs mono-vues généralement bruités par des occlusions ou sujet à des prédictions tributaires de la géométrie de la scène. Une architecture de CNN pour le multi-vues doit pouvoir bénéficier de l'orientation des caméras pour booster le taux de reconnaissance ou résoudre un autre type de problème comme l'estimation de pose. Les réseaux multi-vues proposés dans l'état de l'art reposent majoritairement sur 3 topologies illustrées dans la figure 2.14 :

- Une topologie du réseau avec un CNN multi-vues et mono branche : Le CNN prend V images puis les compresse en un unique vecteur de dimension D où D représente le nombre d'éléments total en sortie de la dernière couche de convolution d'un réseau.
- Un CNN multi-vue et multi branches : La partie FCN est dupliquée selon le nombre de vues V, donnant ainsi un jeu de V vecteurs de dimensions D.
- Ou un CNN complet où chacune des vues infère un réseau, partie FC comprise.

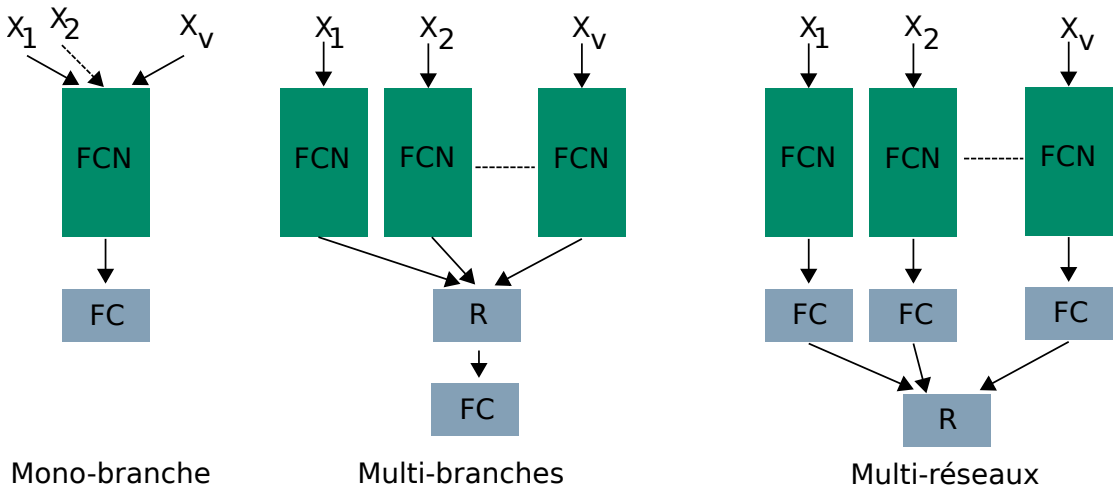


FIGURE 2.14 – Principales architectures multi-vues, mono-branche, multi-branches et multi-réseaux. Une branche est une succession de couches de convolutions FCN et R est une fonction de regroupement.

2.2.1.1 Topologie mono-branche

LeCun et al. [LHB04] proposent de comparer les performances de classification d'un CNN avec des méthodes plus conventionnelles K plus proches voisins (KNN) et Machine à Vecteurs de support (SVM) sur un dataset (*NORB*) composé de 5 catégories d'objets. Chaque objet est capturé par une paire de caméras sous diverses conditions de luminosité, d'élévation et d'occlusions. Le CNN utilisé ne comporte qu'une seule et même branche où les 2 vues sont mixées à l'entrée du réseau. Plus précisément, la première couche du réseau CNN comporte 8 canaux de convolution où les 2 premiers sont attribués à l'image de gauche, les 2 suivants à l'image de droite et enfin, les 4 restants à des images combinant les 2 images [1]. Les auteurs démontrent expérimentalement qu'un réseau binoculaire réduit l'erreur de détection de 20% par rapport à sa contre-partie mono-vue. *DeePano* [SMBM15] est un réseau similaire à AlexNet avec une seule entrée mais peut être classé dans la catégorie multi-vues. L'idée principale des auteurs consiste à pré-traiter un modèle 3D par une projection cylindrique selon un axe de révolution (2.15). Ce type de rendu panoramique permet d'éliminer les occlusions et d'exprimer tous les aspects géométriques d'un objet. Cependant, cette méthode nécessite un pré-traitement spécifique comparé à un système multi-vues classique.

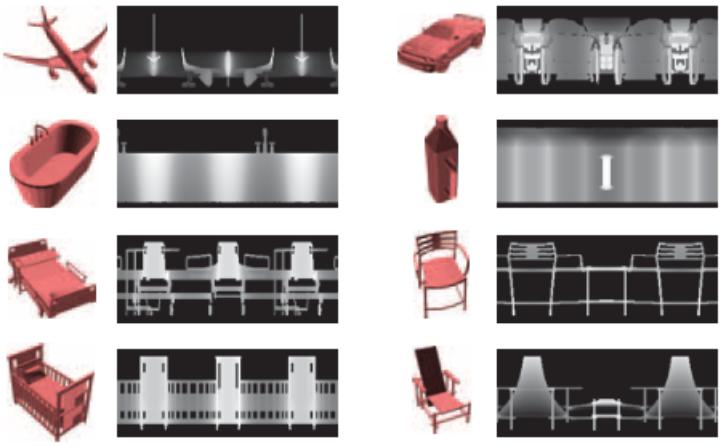


FIGURE 2.15 – Le rendu panoramique 2D à partir du rendu 3D d'un objet du dataset ModelNet40 (source : [SMBM15])

1. Les images sont en niveaux de gris avec une résolution de 96×96

2.2.1.2 Topologie multi-branches

La topologie multi-descripteurs est la plus répandue dans les travaux sur les CNNs multi-vues. La première section f appelle la partie convolutive d'un CNN pour transformer l'espace d'entrée (image) vers un espace de plus faible dimension. Les contributions s'articulent principalement autour d'une fonction intermédiaire spécialement conçue pour manipuler les V descripteurs (cf. 2.16) et prédire le label correspondant à la classe de l'objet, la pose d'une des caméras ou un score de similarité entre les V projections.

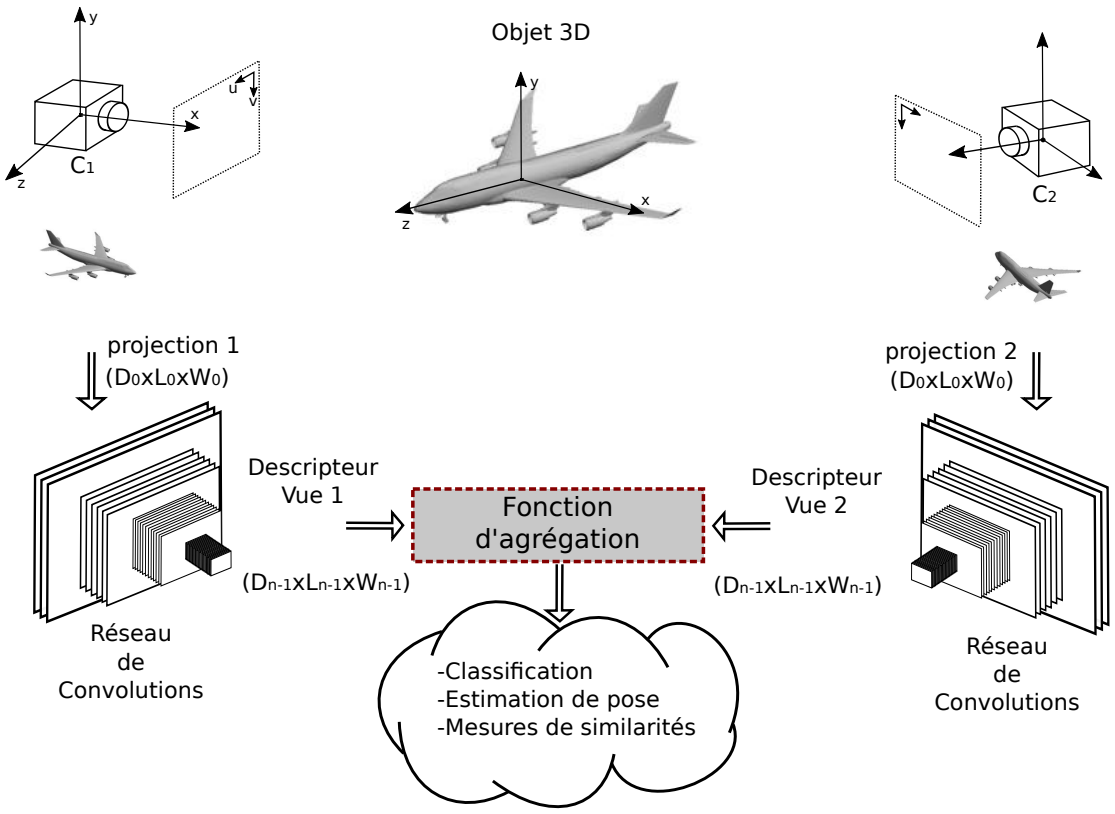


FIGURE 2.16 – Exemple d'architecture de CNN pour 2 vues d'un même objet. Ici, la fonction f est représentée par les couches de convolution et une seconde partie g traite ces données pour extraire des informations relatives à la fonction voulue.

Su et al. [SMKLm15a] ont proposé le premier CNN multi-vues basé sur ce principe (figure 2.17). *MVCNN* repose sur la même architecture que le réseau VGG où chaque vue $(x_1, x_2, ..., x_v)$ infère la même partie convolutive f, paramètres de convolutions compris, puis combine le jeu de vecteurs $(f(x_1), f(x_2)..., f(x_v))$ par une fonction *view-pooling*. Cette fonction prend la valeur maximum un à un des éléments de chaque vecteur et produit un nouveau vecteur de même dimension comme exprimée par l'équation 2.13.

$$Y_i = max(f(X_1), f(X_2), ..., f(X_v)), \forall i \in [0, D-1] \quad (2.13)$$

D représentant la taille du vecteur (descripteur) final en sortie du réseau de convolution de chaque vue.

Cette nouvelle variable latente[2] est interprétée par un MLP associé à une fonction softmax pour estimer le label de l'objet. Les tests sur le dataset ModelNet40 [WS] montrent qu'un setup de 12 caméras obtient des performances de classification (top-1) 4 à 5%[3] supérieure à un CNN mono-vue (85%). A noter que dans leur cadre expérimental, les 12 caméras sont disposées selon l'axe de révolution $\vec{y}$ de chaque objet 3D du dataset comme montré dans la figure 2.17. Les auteurs reportent également des résultats similaires avec un réseau entraîné sur 80 vues. Cependant, le nombre de MACOPs représente un défaut majeur dans ce cas de figure. Le réseau doit produire un descripteur à travers les N couches de convolutions pour chacune des 12 vues. De plus, aucune analyse n'est effectuée sur la pertinence de certains des descripteurs locaux. Autrement dit, est-il nécessaire de capturer autant d'informations (nombre de vues) pour parvenir à de telles performances de classification ? Cette discussion sera menée dans le prochain chapitre sur la compression de réseau.

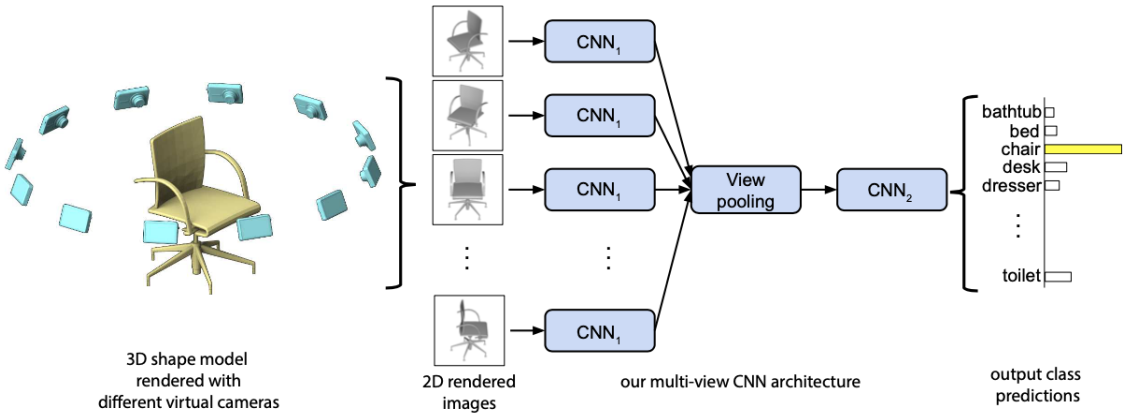


FIGURE 2.17 – Le réseau MVCNN proposé par *Su et al.*

Yang et al. présente un réseau [YTZW18] similaire à *MVCNN* à la différence près que leur approche introduit un autre type de réseau de neurones en lieu et place du couple

2. variable cachée dans le réseau
3. La partie convolutive de VGG-M est pré-entraînée sur le dataset ImageNet.

{MLP, softmax} usuel. Les descripteurs $f(x_1), f(x_2)..., f(x_v)$ sont concaténés pour donner un nouveau un vecteur X de dimension $V \times D$. Comme remarqué par *Su et al.*, cette mécanique n'est pas viable avec un MLP classique car le nombre de paramètres connectant la sortie de la FCN et le MLP est multiplié par le nombre de vues, c'est à dire $V \times D \times N$. Une telle architecture tombe dans le cas typique du sur-apprentissage où il existe un trop grand nombre de paramètres à affiner pour en dériver un modèle suffisamment généraliste. Le réseau ELM-AE propose d'encoder le vecteur de primitives X en un vecteur Y compressé avec une solution couplant un algorithme d'apprentissage extrême *ELM* [HZS06, KYHZ16] et un algorithme d'auto-encodage *AE*. L'apprentissage extrême consiste en un unique vecteur de neurones cachées β_{AE} dont les paramètres sont choisis de manière aléatoire mais figés de sorte qu'ils ne sont pas optimisés durant la desciente de gradient. Les informations en sortie de cette première couche de neurones sont encodées dans une variable latente Y de dimension égale au nombre de classes du datasets.

L'équation de ce type de classifieur s'exprime par :

$$Y = X\beta_{AE}^{T} \tag{2.14}$$

avec une matrice β_{AE} de dimensions ($V \times D$, C) et $X = (f(x_0(i), f(x_1(i), ..., f(x_{N-1}(i)))$.

Les paramètres de la matrice β_{AE} étant figés, l'algorithme d'optimisation (SGD ou Adam) n'a pas à affiner autant de paramètres que dans le cas d'un réseau *fully-connected* et accélère donc le processus d'entraînement. Au final, les performances de classification de leur réseau sont similaires à celui de *Su et al.* pour une quantité de paramètres moindre pour la partie FC. Cette même approche est d'ailleurs reprise dans [XXS+15] pour la classification et la reconstruction d'image 3D à partir d'un jeu d'images 2.5D.

Le réseau **MBHN** intercale une fonction de normalisation entre les primitives et le réseau *fully-connected*. Cette fonction spécifique, *pooling* bilinéaire, est héritée des précédents travaux sur la réduction des variations intra-classes de *Lin et al.* [LRM15]. Le pipeline (*bilinear harmonized pooling*) à la sortie des fonctions FCN produit un nouveau vecteur plus représentatif de l'objet observé. La figure 2.19 montre la succession d'opérations appliquées aux descripteurs de dimension $D \times H \times W$ en sortie des V réseaux de convolutions du CNN VGG-M. Les vecteurs en sortie des V réseaux de convolutions sont

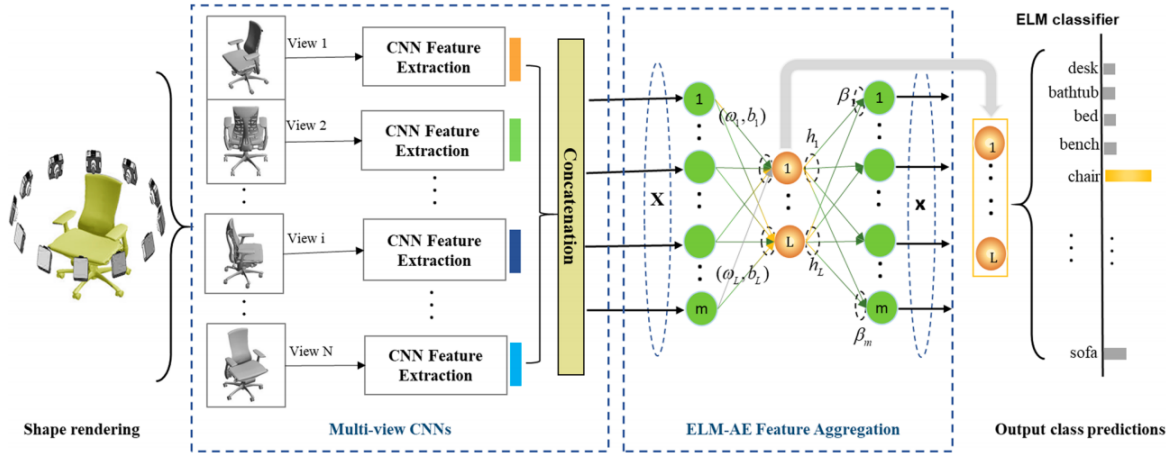


FIGURE 2.18 – Le réseau multi-vues avec classification par Auto-Encoder proposé par *Yang et al.*(source : [YTZW18])

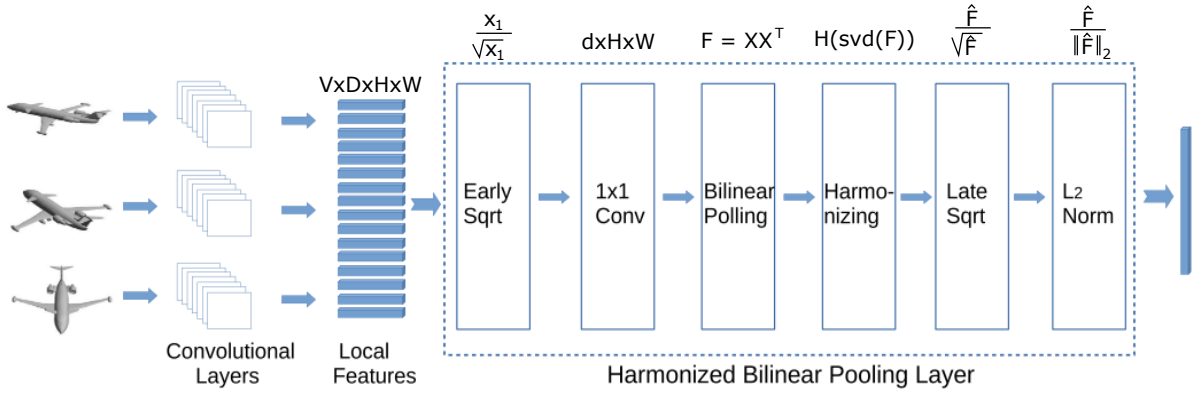


FIGURE 2.19 – la fonction g utilisée dans MHBN embarque une succession d'opérations différentiables. Le volume de primitives $V \times D \times H \times W$ est transformé en un nouveau descripteur de taille $d \times D \times H \times W$.

normalisés dans un premier temps puis un second réseau de convolution est chargé de réduire la dimension $D \times H \times W$ de chaque vecteur d'entrée en de nouveaux vecteurs X_v de dimension $d \times H \times W$ où $d << D$. Les V données sont regroupées dans une matrice M par une fonction de *pooling* bilinéaire [LRM15] telle que :

$$M = X.X^T \tag{2.15}$$

Les valeurs singulières de la matrice symétrique M de dimension $d \times d$ sont calculées par une SVD telles que :

$$M = U.\Sigma.V^T,\ \Sigma = \begin{bmatrix} \frac{\sigma_1 - 1}{\lambda_1} & 0 & 0 & .. & 0 \\ 0 & \frac{\sigma_2 - 1}{\lambda_2} & 0 & .. & 0 \\ 0 & 0 & \frac{\sigma_3 - 1}{\lambda_3} & .. & 0 \\ .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & \frac{\sigma_n - 1}{\lambda_n} \end{bmatrix} \tag{2.16}$$

Enfin, les valeurs de la matrice diagonale Σ sont harmonisées par un paramètre entraînable λ_i avec $i \in [1; d \times H \times W]$. Le réseau se termine par un classifieur classique (*mlp*) avec une fonction *softmax* associée. Les auteurs reportent un taux de classification de 94% avec seulement 6 vues sur ModelNet40 mais la complexité du pipeline augmente nettement le temps de calcul comparé à MVCNN en particulier à cause de la décomposition en valeur singulière de la matrice M.

Les réseaux **DSCNET** [WPS19] et **GVCNN** [FZZ+18] visent aussi à compresser les descripteurs finaux par des algorithmes plus complexes que ceux utilisés par *Su et al.* ou *Yang et al.*. Par exemple, l'architecture **DSCNET** introduit un graphe de discrimination inséré entre les V réseaux de convolutions et le classifieur pour atteindre un taux de compression final de V :1. L'objectif du graphe est d'apparier 2 à 2 les vues présentant des similitudes et d'éliminer les doublons pour compresser le vecteur final. Pour converger vers un unique vecteur de représentation, l'algorithme nécessite plusieurs itérations, brisant le modèle *feed-forward* d'un CNN classique, constituant le point faible de cette approche.

GVCNN est conçu pour regrouper les projections présentant des similarités et attribuer un poids relatif à chaque sous-groupe suivant leur réelle contribution à la classification. L'intuition derrière ce réseau repose sur l'aspect géométrique d'un objet et plus particulièrement sur les similarités existantes selon les axes ou plans de symétrie. Comme présenté dans la figure 2.20, le réseau est scindé en 2 parties où la première extrait les descripteurs intermédiaires pour chaque vue avec les premières couches de convolutions de GoogleNet. La deuxième partie exploite les primitives intermédiaires (couche 5) pour estimer les similarités entre les V vues grâce à un réseau MLP. Le score en sortie agit comme un multiplexeur permettant de classer les descripteurs finaux dans un des M groupes, où M[4] est un entier tel que $1 \leq M \leq V$. Le descripteur final d'un groupe particulier est le

4. A noter que le nombre de groupe M peut varier selon le type d'objet à classifier.

résultat de la moyenne des descripteurs groupés. Les M primitives sont ensuite pondérées par un poids λ_m destiné à attribuer un score de contribution aux groupes et sont propagés dans le classifieur, troisième et dernière section du réseau.

$$f_d = g(\lambda_1 f_1, \lambda_2 f_2, ..., \lambda_M f_M) \tag{2.17}$$

Avec seulement 8 vues, cette architecture surpasse les résultats de classification de *MVCNN*(12 vues) de 3% et possède la capacité de grouper les vues similaires. Le nombre de paramètres est cependant plus élevé puisque chaque vue nécessite un réseau FC intermédiaire pour calculer le score d'indexation.

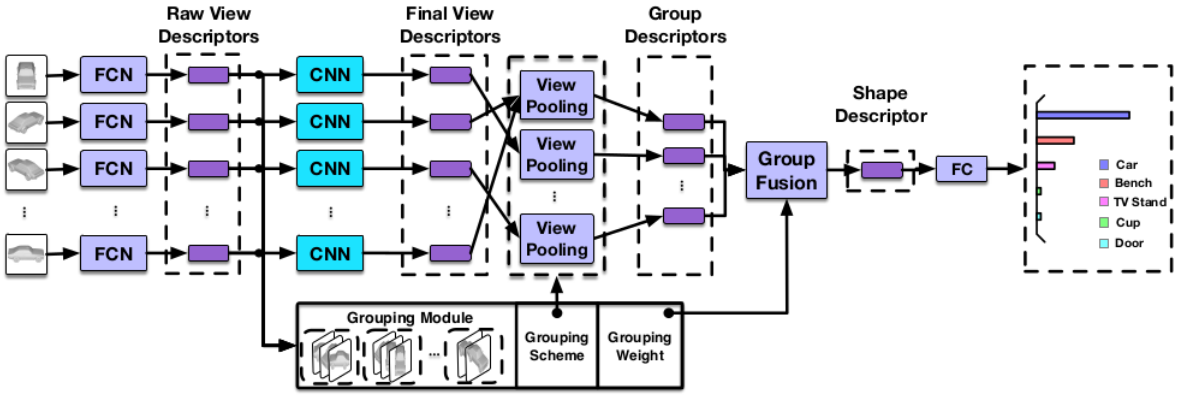


FIGURE 2.20 – Réseau multi-vues de *Feng et al.*(source : [FZZ+18])

Une autre technique consiste à analyser les combinaisons géométriques de caméras selon le nombre de vues disponibles dans le dataset. Le système de *johns et al.* [JLD16a] propose un modèle stéréo actif dans le sens où les caméras visualisent une scène dynamiquement en suivant une trajectoire prédéfinie, par exemple une révolution autour de l'axe Z lié au centre de gravité d'un objet. La paire de caméras suit cette trajectoire pendant la phase d'entraînement pour découvrir une séquence optimale avec laquelle le taux de classification sera maximal. Au total ce sont $\frac{V(V-1)}{2}$ paires de vues $(x_1(i), x_2(i))$ qui sont utilisées durant l'apprentissage où chaque pose ϕ_i se voit attribuer un score de contribution λ_i. Ce poids agit comme un terme de régularisation qui pénalise les couples de vues les moins pertinents. Ce modèle de classification peut être modélisé par :

$$f(y|v_1...v_i) = \sum_{i=1}^{i=N} \lambda_i p(y|w_i) \tag{2.18}$$

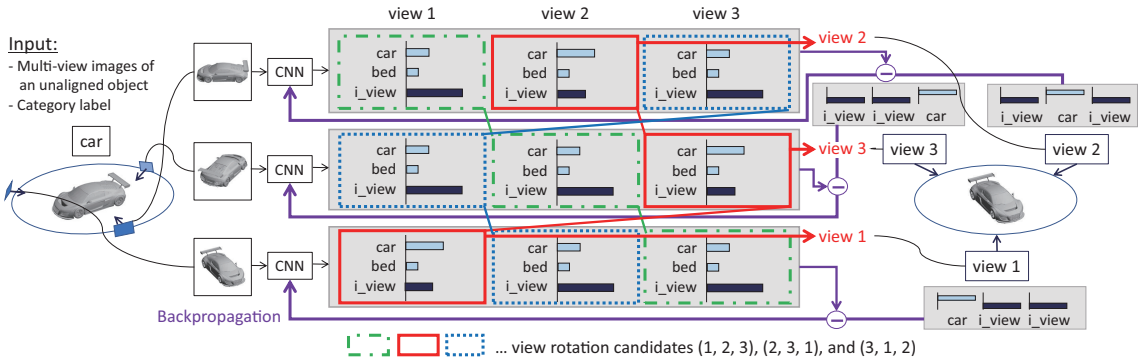


FIGURE 2.21 – Processus d'entraînement du réseau RotationNet où chaque caméra produit son propre histogramme de classification. La dernière valeur de l'histogramme correspond à une estimation de la pose de l'objet.

avec $w_i = \{x_1(i), x_2(i), \phi_i\}$. Pour chaque nouvelle position de la séquence de prise de vue, les primitives de chaque paire sont concaténées en un nouveau vecteur de dimension $2 \times D$. Les valeurs *softmax* sont ensuite accumulées au fur et à mesure que les 2 caméras couvrent la trajectoire formée par une séquence de V vues. Pour $V = 6$ et $V = 12$, les auteurs rapportent respectivement un gain 1.6% et 0.6% comparé à *MVCNN*[5].

2.2.1.3 Topologie multi-réseaux

La dernière catégorie de ce classement concerne les architectures où chacune des caméras du systèmes possède un CNN complet, c'est à dire FCN, MLP et fonction softmax compris.

Le réseau de *Kanezaki et al.* [KMN16] est basé sur cette idée où un réseau complet est dupliqué V fois pour classifier les objets et estimer les positions de chaque caméra simultanément. La différence majeure avec un réseau classique se situe au niveau de la fonction *softmax* où, en plus de produire un histogramme de C valeurs (C est égal au nombre de classes), une $C+1$-ième valeur retranscrit la position de la caméra vis-à-vis de l'objet. En effet, selon la pose v_i de l'objet devant la caméra, le résultat de classification sera différent, c'est pourquoi il est intéressant de mesurer la corrélation entre la pose et l'estimation du réseau. Ici, la pose v_i est considérée comme une variable à optimiser durant tout le processus d'entraînement au même titre que les paramètres des convolutions afin de minimiser conjointement l'erreur de reconnaissance de l'objet et de la pose

5. à nombre de caméras équivalent

de la caméra. Le réseau propose de résoudre le problème d'optimisation formulé dans l'équation 2.19 :

$$MAX \prod_{i=0}^{C-1} P(\hat{y} = y \mid x_i, v_i) = MAX \sum_{i=0}^{C-1} \left(p_{v_i,y} + p_{j,C} \right) \tag{2.19}$$

où P est la vraisemblance de la catégorie de l'objet x_i, $p_{v_i,y}$ la sortie de la fonction softmax relative à la pose de la caméra v_i en considérant la classe d'objet y et $p_{j,C}$ est une fonction de bernoulli égale à 1 si $j = v_i$, c'est à dire, si la pose de la caméra est correctement résolue comme montré dans la figure 2.21. De toutes les architectures multi-vues citées précédemment, la structure de *Kanezaki et al.* obtient les meilleurs résultats de classifications. Par exemple, la triplication du même réseau entraîné sur un set de 12 vues classifie le dataset ModelNet40 avec le même niveau de performance que MVCNN, lequel est entraîné avec 80 poses de caméras différentes. Le taux maximal de classification reporté est de 97% avec 12 vues. Comme expliqué au début de la section, ce réseau nécessite d'embarquer la totalité d'un CNN sur chaque vue et requiert un moteur d'inférence capable de contenir tous les paramètres du réseau et la totalité des opérations de multiplication-accumulation.

Pour simplifier ce classement et appuyer le contexte de cette thèse, les CNN multi-vues de l'état de l'art présentés sont répertoriés dans le tableau 2.1 avec, respectivement, leur charge de calcul sur la partie FCN, la partie intermédiaire de regroupement des vues et la partie classification.

D'un point de vue matériel, le réseau MVCNN [SMKLM15b] présente le meilleur compromis de par la faible complexité qu'implique la fonction de regroupement des primitives G. Ici, G se traduit par un pipeline de comparateurs dont la profondeur varie en fonction du nombre de vues. Cette profondeur P est exprimée par l'équation 2.20 :

$$P(V) = sup(\log_2 V)),\ V \in \mathbb{N}^{+*},\ P(V) \in \mathbb{N}^{+*} \tag{2.20}$$

et le nombre de comparateurs s'exprime par :

$$\mathcal{R}(comp) = V - 1, V \in \mathbb{N}^{+*} \tag{2.21}$$

Hormis le réseau *MV ELM-AE*, les autres CNN multi-vues se distinguent par des

TABLE 2.1 – Récapitulatif des réseaux multi-vues

classification (ModelNet40)	Prétraitement	charge de calcul FCN [a]	Charge de calcul intermédiaire	Charge de calcul classification [b]	Similarités	Estimation de pose	Gain max/ architecture mono-vue
MVCNN	-	Vx	-	1x	-	-	+5%
MV ELM-AE	-	Vx	VxDxHxW	1x	-	-	+5%
MHBN	-	Vx	(VxDxHxW)+VxDxdxHxW+svd(dxd)	1x	✓	-	+8%
Deepano	projection cylindrique	1x	-	1x	-	-	-
Lecun	-	1x	-	1x	-	-	+20%(détection)
GVCNN	-	Vx	VxDx1xHxW	1x	✓	-	+8%
Johns	-	2x(simultanées) x3 séquences	3x2xDxHxW	3x	-	-	+8%
Rotationet	-	Vx	-	Vx	-	✓	+12%

a. en nombre de paramètres
b. Dx4096x4096xC pour AlexNet ou VGG

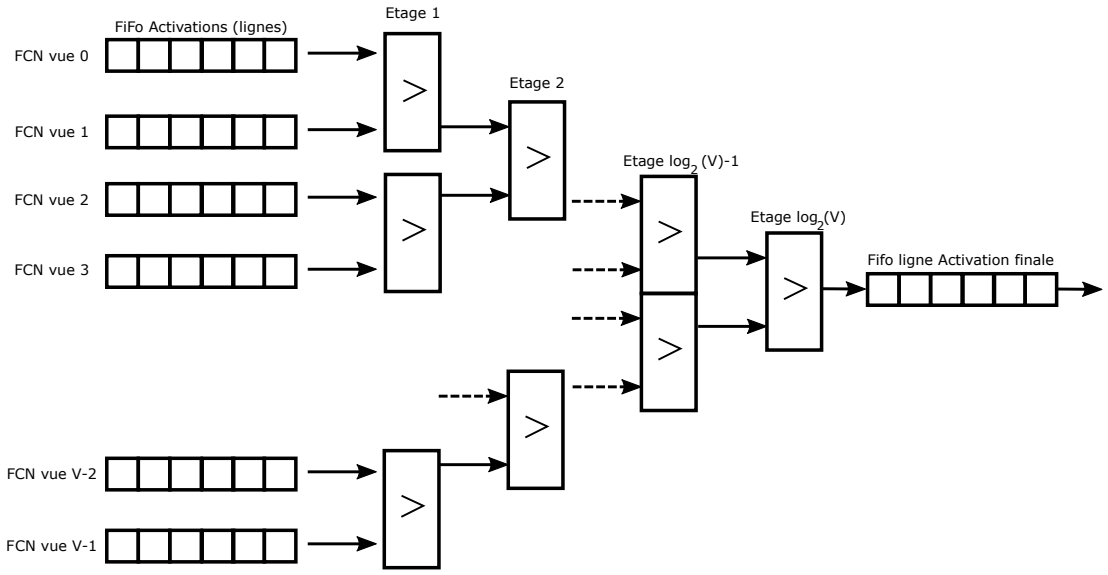


FIGURE 2.22 – Arbre de comparateurs pour la fonction view pooling de *MVCNN*. La complexité matérielle de la fonction *ViewPooling* est de l'ordre de $\mathcal{O}(\log_2 V)$.

fonctions de regroupement plus complexes en rajoutant, en plus des réseaux complet FCN, des fonctions difficilement implémentables sur cible FPGA comme la décomposition en valeurs singulières dans le cas du réseau *MHBN*, ou encore la fonction de regroupement par similarités de *GVCNN* où un réseau MLP doit être inféré pour chacune des vues en plus du MLP pour la classification. Dans le cas du CNN ELM-AE, la partie convolution FCN est dupliquée $V\times$ mais le nombre de paramètres à l'entrée du la couche FC est également $V\times$ plus important. Le réseau *MHBN* appelle des opérations complexes telles que la décomposition en valeurs singulières. Les rotations de *Jacobi* [BJM+06] et la trigonalisation *HouseHolder* [Roa11] sont les 2 principaux algorithmes existants sur cible FPGA permettant d'extraire les valeurs singulières d'une matrice symétrique mais sont toutes deux des méthodes itératives coûteuses de complexité en $\mathcal{O}(n^2)$ avec n la taille de la matrice. Notons que les 2 algorithmes s'appliquent sur des opérandes codées sur 32 bits flottants et nécessitent par conséquent, des unités DSP dédiées. L'architecture de *Johns et al.* requiert seulement 2 caméras puis une séquence de 3 prises de vues. Pour dérouler la structure de leur méthode et s'affranchir de la séquence, il serait nécessaire de connecter les 6 vecteurs finaux à 3 MLP puis de pondérer les résultats des fonctions *softmax* par les 3 poids de contribution λ_i (i<3). La suite de cette thèse s'articule donc autour du réseau *MVCNN* pour toutes les raisons évoquées plus haut.

2.3 Conclusions

Ce chapitre a proposé un état de l'art sur les CNNs mono puis multi-vues. En raisonnant sur plusieurs variables distinctes, un réseau multi-vues étend les possibilités des CNNs classiques en apportant une ou plusieurs dimensions supplémentaires. Pour exploiter efficacement ce surplus d'informations, la clé réside dans une fonction de regroupement dans le cas de la classification ou dans l'élaboration d'une nouvelle fonction objectif spécifique. Ces réseaux multi-vues apportent un gain non négligeable en termes de performance de classification comparé à un système mono-vue classique. En contrepartie, ces CNN appellent plus de calculs que les architectures mono-vues, elles-mêmes déjà extrêmement consommatrices en termes de ressources de calculs et de mémorisation. Ces limitations verrouillent le spectre d'utilisation à des GPU, TPU ou des modèles de FPGA haut de gamme. Le prochain chapitre introduit les outils de compression de réseau conçus pour répondre à cette problématique.

Chapitre 3

Compression et dégradation de réseaux multi-vues

Résumé : *Ce troisième chapitre introduit le principe de dégradation de réseaux de convolutions multi-vues. La suite du chapitre évoquera la compression de réseaux au sens large, allant des méthodes de régularisation basiques puis finalement à celles dites de "pruning" plus avancées. Ces deux variantes de compression sont essentielles pour embarquer un CNN sur un système embarqué. Seulement, chaque réseau possède un seuil de compression au delà duquel les performances sont nettement dégradées. La contribution se situe dans la dernière partie de ce chapitre où nous démontrons qu'un réseau multi-vues peut entrer dans cette région de dégradation pour atteindre un niveau de compression inexploitable dans le cas standard mono-vue.*

3.1 Etat de l'art de la compression de CNN

Tout type d'application a des contraintes de latence plus ou moins strictes. Les architectures de CNN actuelles sont particulièrement coûteuses à stocker et à inférer. Cette densité pose la question de l'utilité de certains paramètres ou de primitives. Des travaux ont mis en évidence le sur-paramétrage des réseaux de l'état de l'art en pointant par exemple des redondances d'activations à l'intérieur d'une même couche de convolutions ou des poids de convolutions inutiles. Ce type de problème est en grande partie résolue par de nouvelles architectures mobiles et diverses méthodes de compression parmi lesquelles figurent la régularisation, le *pruning*, la quantification des paramètres ou encore

la distillation de connaissance [HVD15]. La compression de réseaux de neurones fut initialement proposée par *Lecun et al.* [Yan97] dans l'article "*Optimal Brain Damage*", où les auteurs analysent la plasticité d'un réseau FC et sa capacité à récupérer ses facultés de reconnaissance après avoir sectionné une partie des connexions inter-neuronales. Cette section propose une analyse de l'état de l'art sur la compression de réseau de convolutions et les solutions retenues pour minimiser le nombre de paramètres d'un réseau multi-vues mais adapté à un schéma d'inférence flot de données sur FPGA. La section suivante reprendra cette étude pour analyser l'efficacité de la compression dans un contexte multi-vues.

3.1.1 Architectures de CNN pour plateformes mobiles

Les architectures pour plateformes mobiles (figure 3.1) sont conçues pour amener l'inférence directement sur des CPU ou GPU mobiles, c'est-à-dire d'une consommation électrique de l'ordre du Watts, en minimisant le nombre de paramètres et d'opérations MAC. *Howard et al.* [HZC+17] ont suggéré dans leurs travaux sur MobileNetv1, de remplacer les canaux de convolutions usuels par une version nativement moins paramétrée séparée en 2 parties (équation 3.1). La première partie d'un canal est constitué d'une convolution 2D et la deuxième sert à combiner les résultats de tous les canaux de convolutions d'une même couche. Autrement dit, un canal de convolution ne contient plus qu'un filtre 2D de taille $k \times k$ (k=3) suivi de N_{i-1} filtres 1D (k=1). La sortie d'un canal séparable d'indice m (m$\leq$ N$_i$) est décrit dans l'équation 3.1 :

$$Y(m) = \delta(\sum_{n=1}^{N_{i-1}} W^{1D}(m,n).\delta(\sum_{u=1}^{k}\sum_{v=1}^{k} W^{2D}(m,u,v).X(m,k-u,k-v))) \tag{3.1}$$

Comme décrit dans l'équation 3.2, les canaux de convolutions sont factorisés en $\alpha \times N_{i-1}$ canaux *depthwise* de filtres 2D suivis de $\alpha \times N_i$ canaux *pointwise* de filtres 1D, où l'hyperparamètre α est un réel apportant une flexibilité supplémentaire sur le nombre total de paramètres et de multiplications-additions.

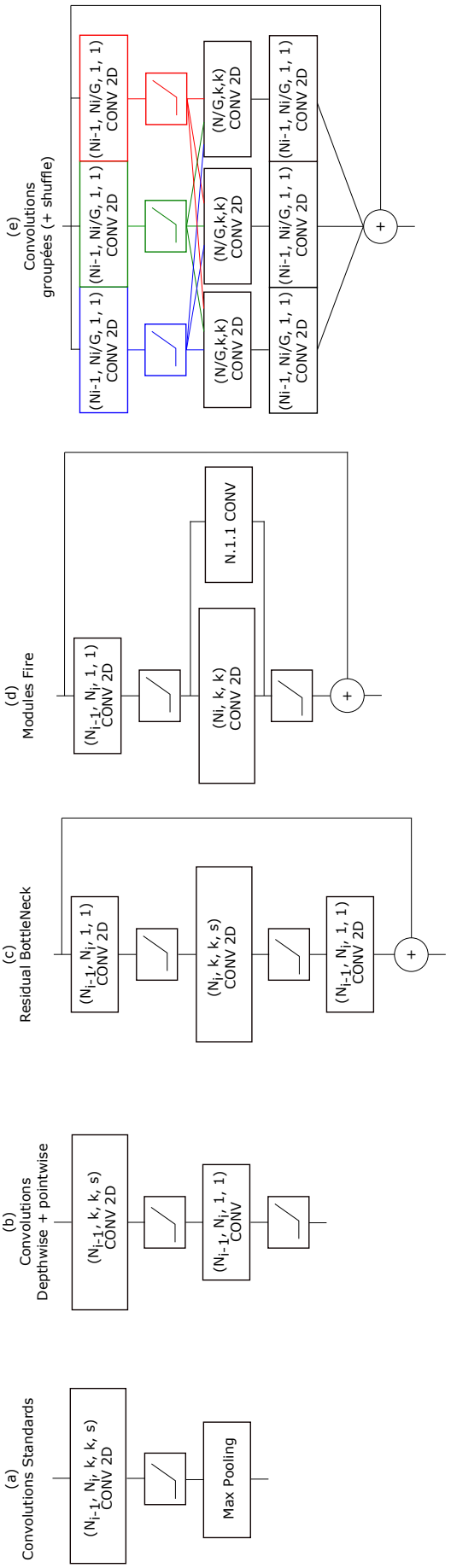


FIGURE 3.1 – Différents types d'architecture de réseau mobiles : le schéma (a) représente une couche de convolutions standard (AlexNet, VGG ou GoogleNet) comme un volume d'opérations. En (b), une couche de convolution est factorisée en un plan 2D de convolutions 3x3 et par ensemble de convolutions 1x1 notée "pointwise". La version avancée (c) contient des résidus similairement à ResNet. Les couche "Fire" (d) de SqueezeNet empruntent l'idée de GoogleNet et des résidus. En (e), une couche de ShuffleNet où l'entrée est dispatchée sur plusieurs canaux de convolutions afin de réduire les interconnexions entre neurones.

Les auteurs démontrent expérimentalement un écart marginal (< 1%) du taux de classification entre la version dense, c'est à dire un réseau uniquement composé de convolutions 2D 3×3 et leur version optimisée. Le facteur de compression γ du nombre de paramètres est donné par l'équation suivante :

$$\begin{aligned} \gamma &= \frac{nb\_parametres\_couche\_depthwise}{nb\_parametres\_standard} \\ &= \frac{N_i \times k_i \times k_i + N_i \times N_{i-1} \times 1 \times 1}{N_{i-1} \times N_i \times k_i \times k_i} \\ &= \frac{1}{N_{i-1}} + \frac{1}{k_i^2} \end{aligned} \tag{3.2}$$

Une extension logique à cette architecture est proposée dans la deuxième version de MobileNet [SHZ+18] où les auteurs introduisent plus de couches de convolutions pour augmenter le niveau de détail dans l'extraction de données d'un côté et utilisent la technique de propagation des résidus de l'autre. Le CNN final nécessite moins de paramètres et moins de calculs que la première version avec un taux de classification supérieur de 2%. En combinant ces optimisations, MobileNetV2 peut classifier ImageNet [DDS+09] avec une précision Top-5 supérieure à 90%. [1]. *Iolanda et al.* [IMA+16] émettent plusieurs idées pour construire un réseau peu paramétré dont une consiste à sous-échantillonner les cartes d'activations le plus tard possible dans le réseau. Plus précisément, la qualité informative des couches dépend en partie de la résolution (U_i, V_i) des primitives. Cette proposition est adaptée au modèle de calcul DHM puisque le nombre d'opérateurs MAC est indépendant de la taille (U_i, V_i). La deuxième contribution concerne la couche *fully connected* remplacée par $C-1$ canaux de convolutions afin de réduire l'empreinte mémoire du réseau. Pour rappel (section 2.1.1), un perceptron multi-couche (MLP) nécessite de stocker un nombre important de paramètres comparé aux réseaux de convolutions 2D. Avec leur solution, les dernières $C-1$ primitives du réseau, de taille 7×7 sont traitées par une fonction moyenne de manière à générer un unique vecteur colonne de $C-1$ éléments. Le dégroupage de canaux convolutions fait partie des solutions retenues dans l'architecture ShuffleNetv1 [ZZLS18] et ShuffleNetv2 [MZZS18]. Les canaux sont dégroupées en G (typiquement $G = 3$) sections indépendantes, autrement dit, certaines

1. Ce travail emploie ce réseau comme une base de démonstration sur la dégradation multi-vues et sera plus détaillé dans la section

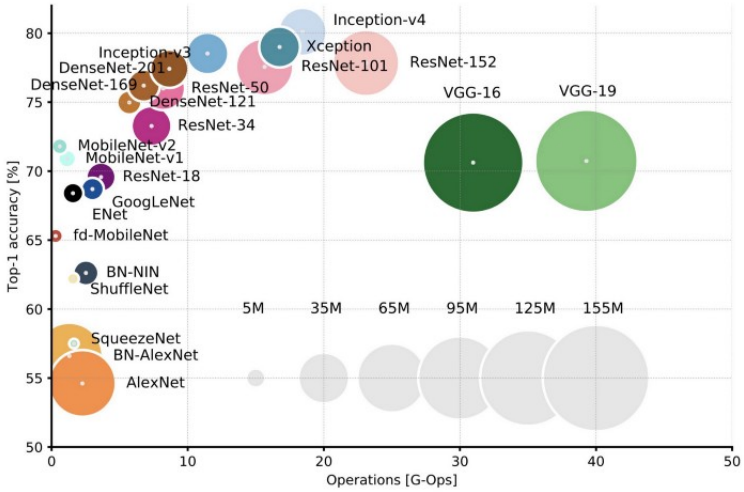


FIGURE 3.2 – Performances des derniers réseaux de l'état de l'art, architectures mobiles inclues. Source : [CPC16]

des interconnexions entre les canaux sont supprimées. Cette technique était initialement utilisée par *Krizhevsky et al.* afin de partager les opérations de convolutions entre 2 GPU et afin de pouvoir contenir les primitives et les paramètres en mémoire. Chaque couche de convolution utilisant le dégroupage d'informations est $G\times$ moins paramétrée que son équivalent standard, sans dégroupage. L'équation décrit cette transformation en termes de paramètres.

$$(N_{i-1}, N_i, k, k) \rightarrow (\frac{N_{i-1}}{G}, \frac{N_i}{G}, k, k)3.1.1 \tag{3.3}$$

EfficientNet [TL19] et MnasNet [TCP+18] regroupe toutes ces approches par une exploration d'architectures. Le premier introduit de nouveaux hyper-paramètres définissant la largeur des couches w, leur nombre N et la résolution des primitives (H_i, W_i). Trouver le meilleur compromis nécessite soit une recherche exhaustive ou une méthode pour diriger la recherche d'architecture selon tous les hyper-paramètres que sont (N_i, k_i, s_i) et D.

Les réseaux mobiles représentent de solides alternatives aux réseaux plus denses. Les écarts de performance de classification sont très faibles tandis que les charges de calculs deviennent plus supportables sur des architectures basse consommation possédant moins de ressources de calculs. Le graphique 3.2 affiche les taux de classification top-1 des réseaux de l'état de l'art cités dans le chapitre précédent et les réseaux bas-coût sur ImageNet en fonction du nombre d'opérations. En particulier, MobileNetv2 servira de

base de recherche pour la suite de cette thèse, c'est-à-dire démontrer qu'une combinaison de méthodes de compression avec un contexte multi-vues est significativement plus efficace qu'un réseau mono-vue en termes de nombre de paramètres.

3.1.2 Régularisation L1 et L2

La régularisation de fonctions cible premièrement le problème d'apprentissage des modèles sur-paramétrés, c'est-à-dire pour les problèmes mal définis à priori. Il est difficile de connaître par avance le nombre de paramètres nécessaires pour dériver un modèle optimal sur une problématique aussi complexe que la reconnaissance d'objet. Concrètement, un modèle sur-paramétré fonctionnera mal pour généraliser sur des données en dehors du set d'entraînement alors qu'un modèle optimal aura une variance minimale sur tout type de données. La régularisation [KH92] apporte une solution efficace pour contrer ce phénomène en intégrant une pénalité λ sur l'ensemble des paramètres d'un modèle si celui-ci devient trop complexe. La place donnée à cette régularisation est elle-même paramétrée sous la forme d'un scalaire noté λ ($\lambda \in \mathbb{R}^+$). De manière générale, cette pénalité s'applique avec deux formes de régularisation, à savoir, la régularisation au sens L_1 (LASSO) ou L_2 (Tikhonov). Elles permettent d'atténuer l'effet des paramètres de convolutions W dont la valeur excède la variance du groupe de poids. Ces termes peuvent être additionnés à la fonction coût finale afin de réduire l'influence de certains paramètres. Pour une fonction coût donnée, la régularisation ajoute le facteur λ aux paramètres du modèle tel que :

$$\text{Loss}_{reg} = \underbrace{L(W,x,y)}_{\textit{fonction coût usuelle}} + \lambda \begin{cases} ||W||_2^2,\ L_2\ reg \\ |W|,\ L_1\ reg \end{cases} \tag{3.4}$$

où x est la donnée d'entrée, y la sortie du réseau et W désigne l'ensemble des paramètres du réseau. L'algorithme de descente de gradient inclut ce terme de pénalité, par conséquent les paramètres W sont mis à jour comme indiqué par l'équation 3.5 :

$$W \leftarrow W - \eta(\frac{\partial L(W,x,y)}{\partial W} + 2\lambda W^T W) \tag{3.5}$$

avec η le taux d'apprentissage.

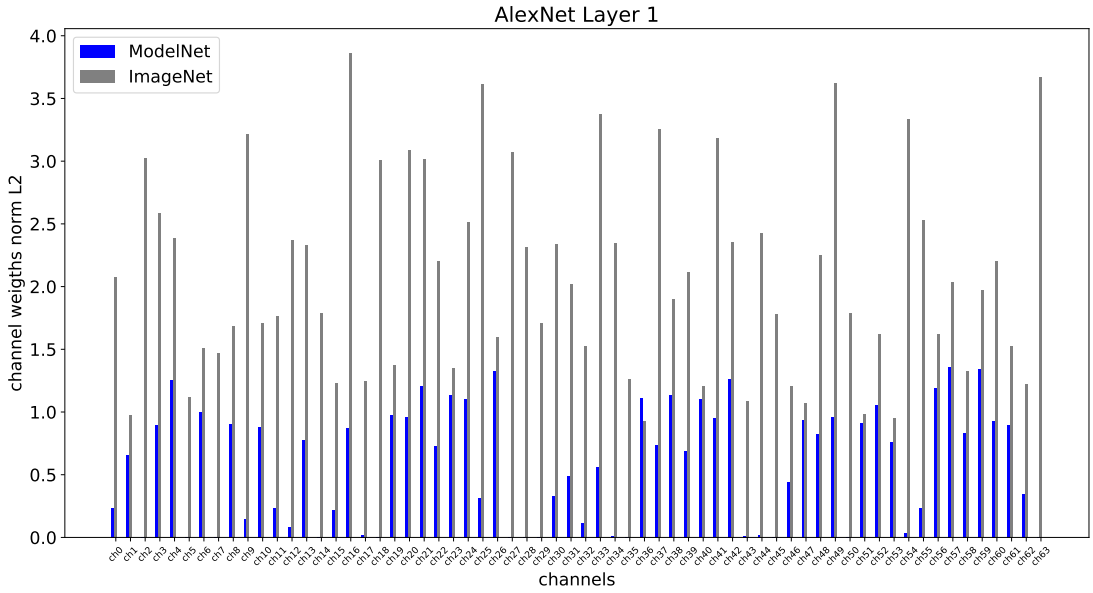


FIGURE 3.3 – Régularisation L_2 de la couche 1 d'AlexNet sur ImageNet et ModelNet40. Les normes L_2 de chaque canal sont reportées sur la figure. Pour un dataset moins complexe, la régularisation élimine une partie des paramètres du réseau.

Pour $\lambda = 0$, la fonction coût revient à celle de départ, mais pour $\lambda > 0$, généralement choisi de telle sorte que $0 < \lambda < 10^{-3}$, le terme $2\lambda W^T W$) ramènent les paramètres vers 0 dans le cas L_2 ou peuvent être exactement égaux à 0 dans le cas L_1. Dans un CNN, la régularisation se traduit par la minimisation voire l'élimination d'un ou plusieurs canaux de convolution. Cette affirmation est vérifiable dans le cas du réseau AlexNet mono-vue entraîné sur la banque d'images ImageNet puis ModelNet40 [WS].

Pour le réseau Alexnet entraîné sur ModelNet40, 25% des canaux de convolutions ont été supprimés par régularisation comme l'atteste la figure 3.3. Ceci signifie que le CNN de départ est surparamétré pour cette banque d'images ne comportant que 40 classes d'objets. Aucun des canaux de convolutions du même CNN ne sont supprimés lorsqu La question posée ici est de savoir si un contexte multi-vues a un effet sur la régularisation d'un réseau. Dans le cas du multi-vues et du CNN AlexNet modifié par la tactique de regroupement employé par *MVCNN*, la régularisation du réseau avec une contrainte $\lambda = 10^{-4}$ reste similaire à celle du réseau mono-vue comme montré par les figures 3.4, 3.5 et 3.6.

Une simple régularisation appliquée à un réseau multi-vues n'apporte donc pas de

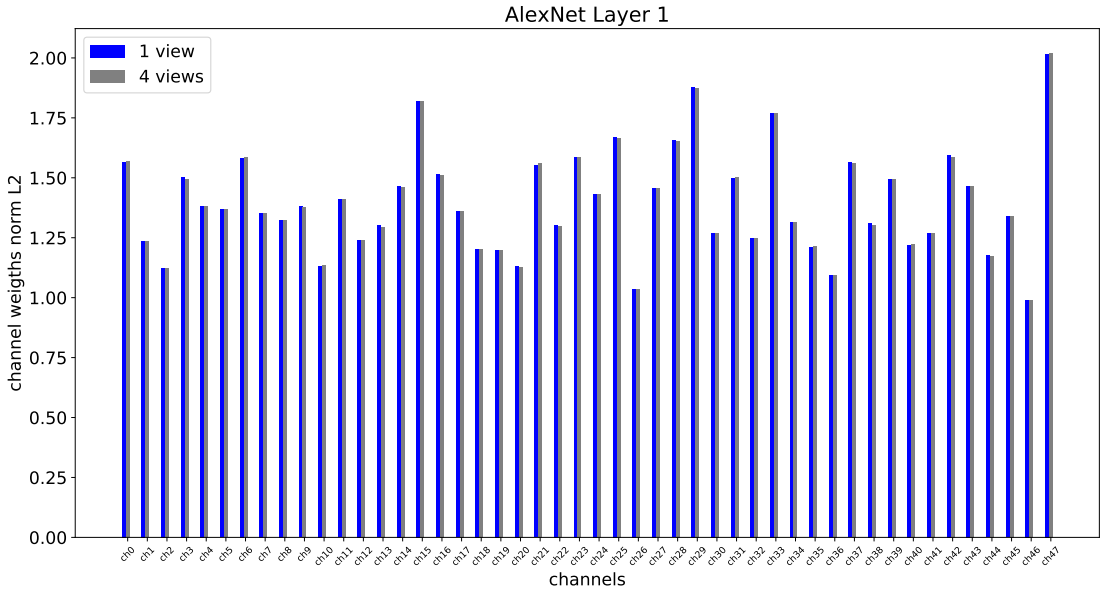


FIGURE 3.4 – Régularisation L_2 d'AlexNet sur le dataset ModelNet40 en mono puis multi-vues. Aucune différence notable n'existe entre les normes des paramètres des deux modèles.

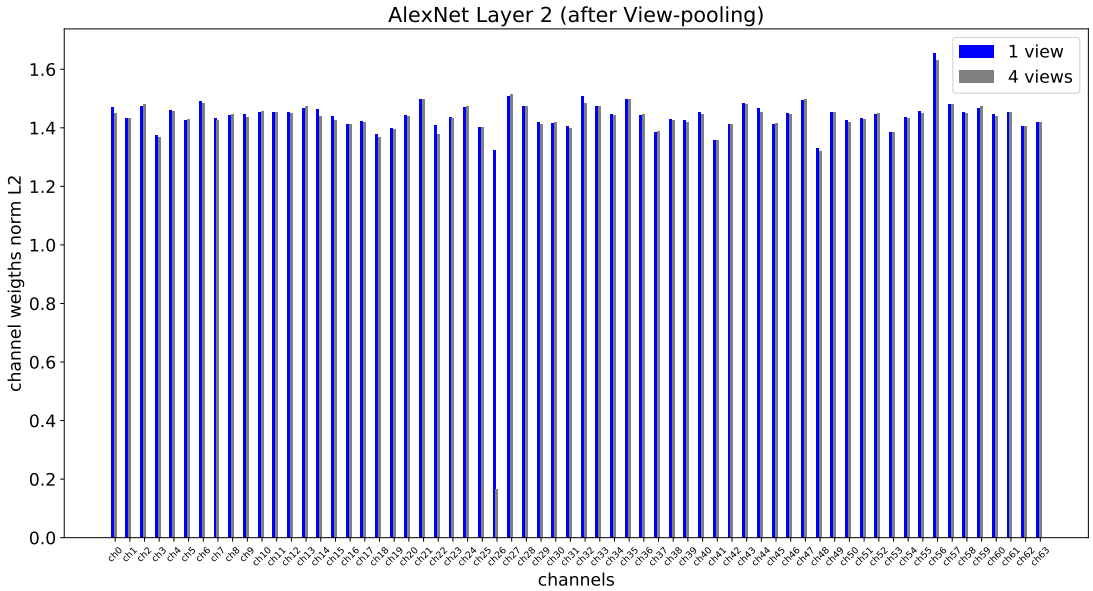


FIGURE 3.5 – Régularisation L_2 d'AlexNet sur le dataset ModelNet40 en mono puis multi-vues (4 vues). Hormis le 25-ème canal de convolution, les normes des poids de convolutions restent similaires entre les deux réseaux.

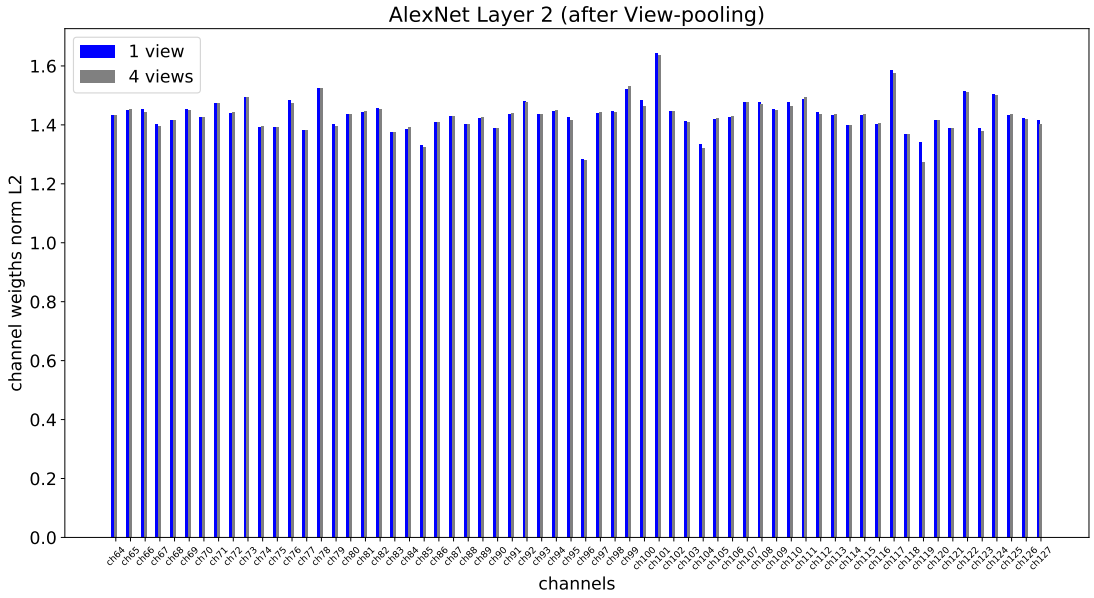


FIGURE 3.6 – Suite de la figure **??** pour les canaux 64 à 127 du modèle AlexNet.

gains supplémentaires comparé à sa variante mono-vue. Les données en surplus à l'entrée du CNN n'implique pas moins de paramètres. Ici, il apparaît que le nombre de paramètres non nuls est plutôt tributaire du nombre de classes du dataset, c'est-à-dire, de la difficulté du problème à résoudre.

3.1.3 Pruning

Le *pruning* représente la majorité des efforts de recherche en compression de réseau de neurones, que ce soit au niveau des couches de convolutions FCN ou des réseaux pleinement connectés FC. L'objectif du *pruning* est de trouver les neurones (activations) et les poids associés qui ont le moins d'impact sur le taux de reconnaissance d'un réseau déjà entraîné sur un dataset. Divers travaux engagent des métriques variées pour quantifier l'impact des hyper-paramètres d'un CNN sur la qualité de la classification. Ces métriques s'étendent de la valeur de poids de convolution en passant par la variance des activations et jusqu'au traçage de l'importance des neurones le long d'un réseau. Il existe deux catégories principales de pruning, de granularités différentes, avec d'un côté les méthodes qui jouent sur la structure même d'un réseau, et celles qui s'appuient sur une analyse fine des paramètres comme montré dans la figure 3.7. Cette segmentation des méthodes de pruning a son importance non seulement sur le taux de compression final mais aussi sur le modèle de calcul. En effet, une méthode structurée permet de relaxer la charge de calcul sur un modèle d'architecture de type SIMD, et donc de diminuer le temps d'inférence, en éliminant des groupes de paramètres. L'autre méthode est peu exploitable en l'état car elle n'agit que sur les paramètres individuellement sans se préoccuper de l'ordre des opérations. Les deux sous-sections suivantes proposent un aperçu des algorithmes de compression vus dans la littérature.

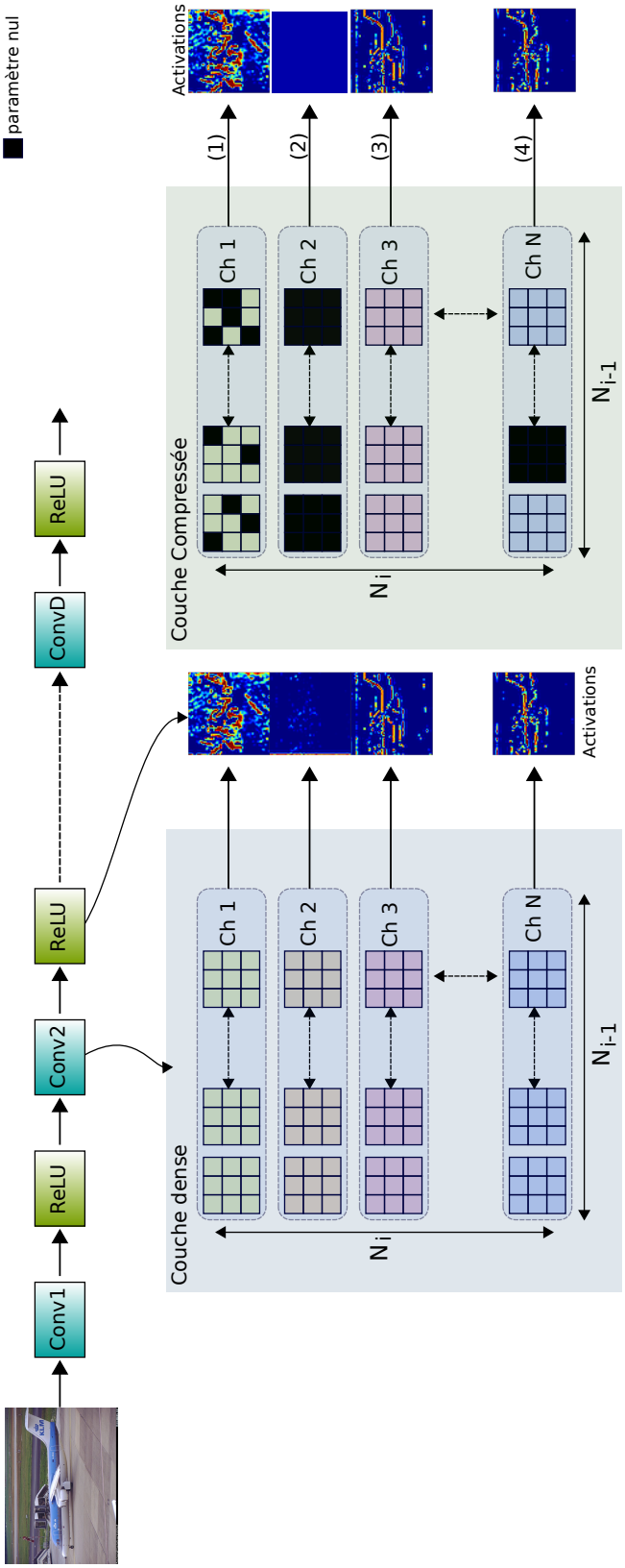


FIGURE 3.7 – Les 2 types de pruning et leurs effet sont représentés dans cette figure. Le pruning non structuré en (1) annule les paramètres qui n'ont pas d'impact sur la capacité de reconnaissance du réseau. En règle générale, l'amplitude de chaque poids est comparée à la valeur maximale trouvée dans une couche de convolutions. L'exemple en (2) représente le pruning structuré où un canal de convolution est totalement supprimé de la couche. En (4), cette fois un kernel est supprimé du canal de convolutions. Il peut être assimilé au pruning structuré même s'il n'amène pas un changement structurel régulier au réseau.

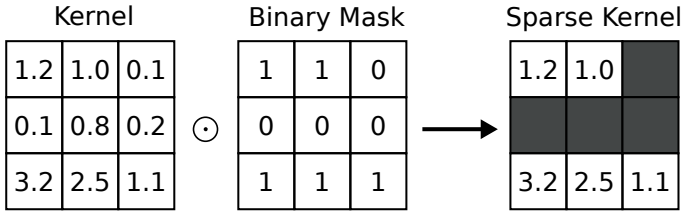


FIGURE 3.8 – L'algorithme de pruning élimine les plus petites valeurs de la matrice de convolution. En contrepartie, un masque binaire doit contenir les coordonnées des éléments nuls pour chaque kernel.

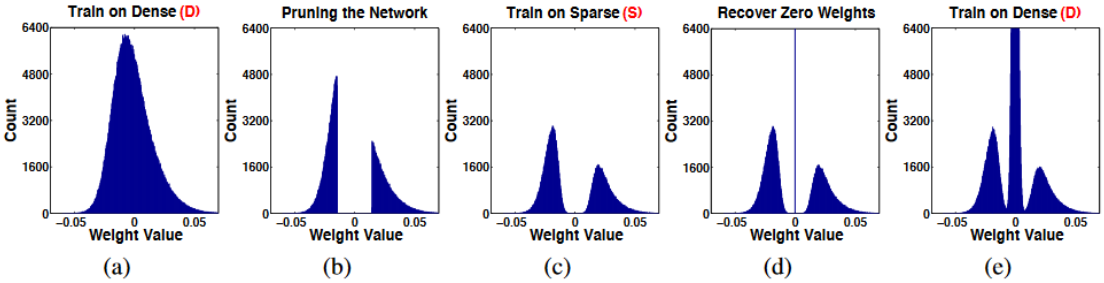


FIGURE 3.9 – Méthode non structurée **Dense-Sparse-Dense** proposée par *Han et al.* L'histogramme (a) représente la distribution initiale des poids de convolutions d'une même couche. La figure (b) illustre l'étape de seuillage puis de ré-entraînement en (c) avec un masque binaire. Les figures (c) et (d) illustrent le comportement du réseau lorsque le masque est supprimé. [HPN+16]

3.1.3.1 Pruning Non-structuré

Les méthodes de compression dites non structurées font partie des premières mises en avant sur les CNN. Les études [HPTD15, HPN+16] présentées par *Han et al.* exploitent le fait qu'une majorité des paramètres de convolutions ont une distribution gaussienne 3.9 au sein d'une même couche de convolution. L'idée véhiculée par le pruning non structuré consiste à imposer manuellement la suppression de ces poids soit par un taux de compression arbitraire λ. L'algorithme, décrit en 1, ajoute un réseau d'éléments binaires B^d_{mask} au réseau intiale noté NET_{dense} pour masquer les plus petites valeurs de W^d, où d est l'index de la couche, tel qu'exprimé par l'équation et illustré par la figure 3.8.

$$W^d \leftarrow W^d \odot B^d_{mask},\ B^d \in \{0;1\}^{N_{d-1} \times N_d \times k \times k} \tag{3.6}$$

Algorithme 1 : Pseudo-code de pruning non-structuré

Data : NET, sparsity = $\{\lambda_1, ..., \lambda_D\}$,

Result : NET_s

// Initialisation des masques binaires

$B_{mask} \leftarrow B_1$;

// pour chaque couche du réseau

for *(d=1 ; d=D ; d++)* **do**

  // Tri et indexation des paramètres dans les vecteur V et I

  V,I = Sorted(flatten($|W^d|$));

  // Suppression des λ_d% plus petits éléments

  V $\leftarrow$ Remove(λ_d%, V);

  // Mise à jour du masque binaire pour chaque kernel W

  $B^d_{mask}(I|_{V=0}) = 0$;

  // réorganisation des masques binaires selon la forme du tenseur

  B^d_{mask} = reshape(B^d_{mask},W^d);

end

// Renforcement du CNN NET pendant E epochs

while *epoch* < *E* **do**

  W$\leftarrow$ (W$\odot$ B_{mask}) - η $\frac{\partial \mathcal{L}_{softmax}}{\partial W}$

end

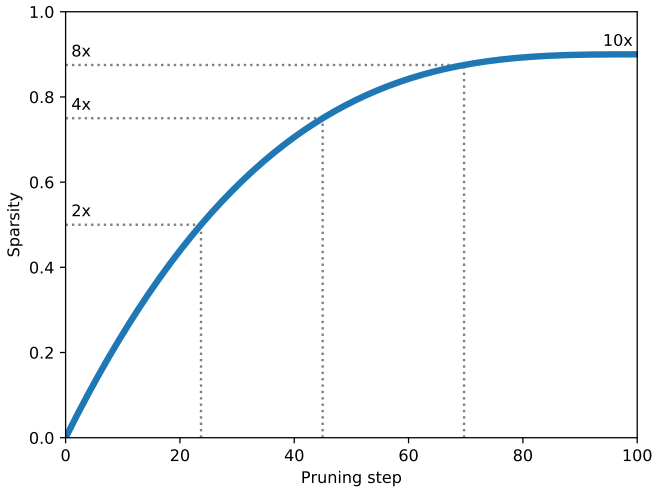


FIGURE 3.10 – Compression graduelle des poids d'une même couche de convolutions [MZ18]

Le réseau est donc arbitrairement compressé d'un facteur λ puis ré-entraîné sur plusieurs itérations pour essayer de retrouver les performances du réseau d'origine à l'image des travaux de *Lecun et al.*. *Han et al.* montrent par ailleurs que cette méthode permet non seulement de réduire de 30 % le nombre de paramètres de la couche FC sur VGG16 mais qu'elle permet aussi d'améliorer le taux de reconnaissance de 4% sur ImageNet. Les auteurs proposeront plus tard d'étendre leurs travaux [HMD15] à la compression par une quantification non-linéaire et par un codage de *Huffman* des poids de convolutions restants. Cette nouvelle optimisation permet de réduire la taille du modèle AlexNet de 47x. Néanmoins, l'utilité de cette méthode reste limitée et réside surtout au niveau du stockage des paramètres, c'est à dire leur résolution en bits. L'idée de *Zhu et al.* [MZ18] consiste à supprimer les paramètres graduellement plutôt que d'imposer un taux de compression dès le départ. Cette gradation est exprimée par l'équation 3.7 et peut être visualisée avec la figure 3.10. Concrètement, l'algorithme va rapidement supprimer les paramètres de faible importance au sein d'une **même couche**. Arrivé à un certain seuil, le gradient du facteur de compression se rapprochant de zéro, les paramètres importants sont ajustés pendant de nouvelles itérations et permet au réseau d'affiner les poids de convolutions importants. La métrique symbolisant l'importance d'un poids est similaire à celle employée par *Han et al.*, c'est-à-dire la valeur absolue du maximum des poids dans

une même couche $MAX_{N_i}|W_i|$.

$$\lambda_t = \lambda_f + (\lambda_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad \text{for} \quad t \in \{t_0,\ t_0 + \Delta t,\ \ldots,\ t_0 + n\Delta t\} \tag{3.7}$$

où λ_t, λ_i et λ_f représentent les taux de compression actuel, initial et final. t est l'epoch [2] d'entraînement actuelle , t_0 l'epoch intiale, n est le pas de compression et Δt la fréquence de compression. Cette approche parvient à compresser 75 % des paramètres de MobileNetv1 sur ImageNet, sans accuser de perte de précision de classification, alors que le réseau est déjà initialement peu dense. Une autre approche consiste à supprimer les redondances en travaillant dans le domaine fréquentiel [LXPX18] et en supprimant les coefficients de la DCT de plus faible amplitude. Toutefois, il est nécessaire de convertir les opérations de non-linéarité et de *pooling* dans le domaine transformé.

Concrètement, un algorithme de compression non-structurée agit sur les paramètres individuellement. L'algorithme n'a pas le contrôle sur l'ordre avec lequel les poids de convolutions sont supprimés, par conséquent, il est difficile pour un moteur d'inférence de type SIMD d'ordonnancer les opérations plus rapidement comparé à un réseau dense. Le seul gain potentiellement exploitable concerne l'espace mémoire dans lequel les paramètres à zéro ne sont pas stockés, mais un index est nécessaire pour pointer les positions des éléments non-nuls. Au contraire, ces index surchargent encore plus la mémoire puisqu'il est nécessaire de les contenir dans un espace dédié. Comme il sera montré dans le chapitre suivant 4.1.3, certains moteurs d'inférence gèrent tout de même ce type de compression mais dont l'objectif consiste à diminuer l'utilisation des PE et donc d'abaisser la consommation.

3.1.3.2 Pruning Structuré

Le pruning structuré est la méthode de compression la plus exploitée dans la littérature car elle permet d'extraire facilement une accélération sur les architectures SIMD en maintenant l'ordre des calculs et des accès aux données. En effet, ce schéma de compression est axé sur la suppression d'un ensemble de paramètres, allant de la suppression des vecteurs que composent un filtre de convolutions, celle des filtres entiers, des canaux, voire de la totalité d'une couche de convolution si l'architecture du CNN en question le

2. Pour rappel, une epoch signifie que tout le set d'entraînement a été propagée une fois dans le réseau

permet[3]. La procédure est cependant fastidieuse car pour chaque groupe de paramètres supprimés, il est nécessaire d'évaluer l'impact sur le taux de classification et engager une nouvelle étape de réentraînement si nécessaire. Comme remarqué dans la section précédente, l'amplitude des poids d'une même couche constitue la métrique usuelle concernant le pruning irrégulier. Bien qu'elle soit extrêmement simple, cette méthode reste très efficace sur la plupart des réseaux de l'état de l'art. Les métriques de décision de pruning structuré sont plus diverses et plus complexes. Cette diversité découle des différents motifs de compression applicables à une couche de convolution et rend l'exploration très coûteuse en temps de développement.

Parmi les critères retenus dans la littérature, la variance des activations [HPTT16], évaluée sur un ensemble de données d'évaluation, donne une indication sur la pertinence de la sortie d'un neurone noté $Z(n_i)$ avec $n_i \leq N_i$. *Hu et al.* proposent d'annuler les canaux dont les variances sont minimales en mesurant leur déviation $\sigma(Z(n_i))$ face aux données de l'ensemble des images test d'un dataset. Le seuil σ_{tol} pour lequel une activation est considérée comme non significative est décidée en amont par le designer. Pour les canaux dont la distribution n'est pas centrée autour de 0 mais dont la déviation est plus faible que le seuil de déviation, l'algorithme infère un biais correspondant b_{n_i}. Cette mesure démontre expérimentalement que 47% des canaux de la première et 93% de la dernière couche de VGG16 sont potentiellement inutiles.

$$Z(n_i) = \begin{cases} Z(n_i) \text{ si } \sigma > \sigma_{tol} \\ b_{n_i} \text{ sinon} \end{cases} \tag{3.8}$$

[AAY17] propose une solution simple où les canaux de convolutions sont classés en fonction de leur contribution sur le taux de classification. C'est-à-dire qu'un ensemble de filtre de convolution, noté F, est indexé par un score de contribution, noté CAR, fort si il est susceptible de dégrader les performances et au contraire faible si il n'a pas ou peu d'influence. Cet index est calculé en 3.9 et mesure la différence du taux de classification entre la version dense du réseau NET et sa version amputée du filtre F à la couche d'indice i.

$$CAR_{index} = ACC(NET) - ACC(NET, (-F, i)),\ \forall i \in D \tag{3.9}$$

3. uniquement sur les couches résiduelles

Dans [LKD+17], les filtres des N_i canaux d'une couche de convolutions sont triés puis supprimés selon la somme de leur norme L_1, décrite par l'équation 3.10. Par ailleurs, cette étude montre que l'intuition derrière le tri des normes L_1 est plus efficace comparée à une suppression aléatoire des filtres à taux de compression égal ou similaire à l'idée précédente évoquée précédemment en termes de taux de compression. Les auteurs parviennent à réduire le coût d'inférence de 34% pour VGG16 sur CIFAR-10 et 24% pour ResNet-34 sur ImageNet sans dégradation du taux de classification.

$$\sum^{N_{i-1}} |F_i|_1^1 = \sum_{n}^{N_{i-1}} \sum_{c}^{k} \sum_{r}^{k} |W_i(c,r)| \tag{3.10}$$

Luo et al. [LW17] utilise l'entropie H_{N_i} pour mesurer la distribution des paramètres au sein d'un canal de convolution. Leur intuition se base sur le fait qu'un canal contenant une majorité de paramètres autour de 0 aura une faible entropie et par conséquent, une faible contribution dans la construction du vecteur d'activations final.

Les précédentes méthodes requièrent généralement un nombre considérable d'étapes de réentraînement puisque les paramètres sont sélectionnés manuellement par le designer. Des algorithmes plus rapides répondent à cette problématique en automatisant la sélection des groupes de paramètres à supprimer à la manière de la régularisation présentée dans la sous section 3.1.2. *Liu et al.* [LLS+17] utilisent une forme régularisation L_1 en ajoutant un scalaire $\gamma(\gamma \geq 0)$ devant les activations et après les modules de *batch-normalization* sur chaque canal d'une couche tel que :

$$Z_{out}(n_i) = \gamma Z_{in}(n_i) + \beta(n_i),\ \gamma \in \mathbb{R}^{+2},\ n_i \leq N_i \tag{3.11}$$

où Z_{in} est la primitive de sortie d'un canal de convolution n_i et Z_{out} la nouvelle valeur après multiplication du facteur d'échelle γ. Le réseau apprend cette valeur pour chaque couche conjointement avec la fonction de coût standard. Dans ce cas, si γ est un scalaire nul, le canal est inutile et peut être supprimé ou remplacé par la constante β si $\beta > 0$. L'algorithme de *Torfi et al.* [Tec18] copie les techniques de régularisation vues en 3.1.2 avec une pénalité L_1 sur les activations à faible variance λ_{gv} ainsi qu'une pénalité sur le nombre d'éléments non-nuls λ_{gs}. L'équation 3.12 présente la nouvelle fonction de coût qui consiste à minimiser l'erreur *softmax* tout en minimisant le nombre le nombre de

canaux, noté $G(W_i)$, de la i-ème couche suivant leur variance.

$$\text{Loss} = \underbrace{L_{softmax}(W)}_{Cout\ usuel} + \lambda_{L_2}(W) + \frac{1}{\sqrt{G(W_i)}}(\lambda_{gs}\sum^{D} L_{gs}(G(W_i)) + \lambda_{gv}\sum^{D} L_{gv}G(W_i)) \quad (3.12)$$

He et al. [HZS17] s'appuient aussi sur le principe de régularisation avec une pénalité L_1 sur les N_i canaux en minimisant l'erreur de reconstruction des activations de la couche suivante comme exprimé par l'équation 3.13.

$$\min(\lambda, W)\frac{1}{2M}||Z(i+1) - \sum^{N_i} \lambda_i X_i W_i^T||^2 \quad (3.13)$$

où X_i est le volume d'activations de la couche précédente et λ_i est un terme de compression masquant un ou plusieurs canaux de convolutions. Encore une fois, ce travail montre qu'il est plus efficace d'appliquer ce principe sur un CNN déjà entraîné sur le dataset à résoudre que d'entraîner un réseau vierge de complexité équivalente en termes de GOPs.

Un CNN est globalement difficile à compresser car, malgré tous les indicateurs proposés par les précédentes études, la suppression d'un seule neurone peut avoir une incidence sur le reste du réseau. De plus, toutes les métriques arbitraires décidées en amont par l'architecte système sont sujettes à mauvaise interprétation, dans le sens où elles ne constituent pas nécessairement les méthodes les plus otimales pour éliminer une variable d'un réseau. *Huynh et al.* [HLB18] introduisent un sous-réseau FC couplé à une fonction *softmax* insérée dans chaque couche de convolutions du réseau cible pour apprendre de manière autonome les activations les moins utiles. Ce sous-réseau apprend les contributions de chaque filtre et masque celles qui n'apportent pas d'informations utiles. En d'autres termes la fonction *softmax* possède $N_{i-1} \times N_i$ sorties et le designer choisit manuellement le taux de compression final λ. Les λ% plus petites valeurs de la fonction *softmax* modifient le masque afin d'annuler les poids de convolutions des filtres correspondant. [YLC+18] émet l'idée de baser la recherche des neurones insignifiants en partant de la dernière couche de convolutions. Cette dernière couche est la plus importante puisqu'elle produit le vecteur final contenant l'essentiel des informations avant la partie FC d'un réseau. La procédé de *NISP* est similaire à celle évoquée par [HZS17] ou par [LZZ+18]. Les auteurs démontrent qu'il est plus efficace de compresser les couches

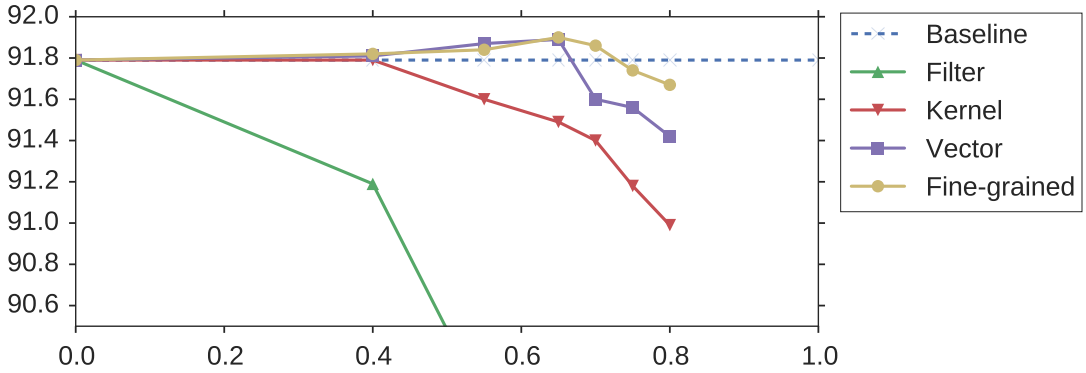


FIGURE 3.11 – Niveaux de compression possibles sur le réseau AlexNet avec les méthodes allant de la granularité la plus forte (canal de convolution) à la plus fine (paramétrique). Source : [MHP+17]

conjointement en essayant de minimiser l'erreur de reconstruction de la **dernière couche** plutôt que chaque couche indépendamment l'une de l'autre. En partant du D-ième vecteur de primitives, l'algorithme supprime les canaux qui n'ont pas d'incidence sur la valeur du vecteur final.

Le taux de compression de ce type d'approche est généralement plus faible qu'une compression grain-fin comme le remarque *Mao et al.* [MHP+17] et illustré dans la figure 3.11. Toutefois, ces algorithmes jouent sur la structure d'une couche pour, idéalement, délester le système d'inférence d'autant d'opérations d'un côté et pour accélérer le processus en diminuant le nombre de requêtes d'entrée/sortie vers la mémoire externe de l'autre. Le tableau 3.1 liste les résultats des méthodes de pruning de l'état de l'art et souligne les plus efficaces en termes de réduction de paramètres et d'opérations.

(A=CIFAR10, B=CIFAR100, C=IMAGENET) *Taux de compression à capacité de reconnaissance identique	AlexNet		VGG-Net		GoogleNet		ResNet (R18, R34, R50, R56, R101,R110, R152)		MobileNetv1		DenseNet-40	
	Params	Flops	Params	Flops	Params	Flops	Params	Flops	Params	Flops	params	Flops
Greedy Pruning [AAY17]	C(layer 1) : -60% C(layer 2) : -60%	C : N/A										
An Entropy-based Pruning Method for CNN Compression [LW17]			C : -5%	C : -70%			C,R50 : -32%	C, R50 : -55%				
Dynamic Network surgery [GYC16]	C : -94.3%(FC inclus)	C : N/A										
Network Trimming [HPTT16]			C(Conv5-3+FC6) : -62%	C : N/A								
Netwok Slimming ([LLS+17])			A : -88.5% B : -75.1% C : -82.5%(FC inclus)	A : -51% B : -37.1% C : -30.4%							A : -65.2% B : -54.6% C : N/A	A :-55% B :-47% C : N/A
Pruning Filter for Efficient ConvNet 1 [LKD+17]			A : -64%	A : -34.2%			A,R34 : -10.8% A,R56 : -13.7% A,R110 : -38.6%	A,R34 : -24.2% A,R56 : -27.6% A,R110 : -32.4%				
Channel pruning for accelerating very deep neural network [HZS17]							A,R56 : -50.6%	A,R56 : N/A				
"Rethinking the smaller-norm-less-informative" [YLLW18]							C,R101 : -38%	C,R101 : -47.3%				
"Deep Compression" [HMD16] et "Learning both weights and connections" [HPTD15]	**C : -89% (FC inclus)**	C : N/A	**C : -92.5% (FC inclus)**	C : N/A								
"Frequency-Domain Pruning" [LXPX18]	**C : -95% (FC inclus)**	C : N/A					**A,R20 : -85%** **A,R110 : -89.2%**	**A,R20 : -86.4%** **A,R110 : -84%**				
"To Prune or Not to Prune" [MZ18]					**C : -87.5%**	C : N/A			C : -75%	C : N/A		
Attention Based Guided Strutured Sparsity of Deep Neural Networks [Tec18]	C : N/A	C :-75%										
ThiNet : Pruning CNN Filters for a Thinner Net [LZZ+18]			C :-84%	**C :-70%**								
Neuron Importance Score [YLC+18]	C : -50%	C : -75%					A,R56 : -35%	A,R56 : -42.4%				
Learning structured sparsity in deep neural networks [WWW+16]	C : -84% (Sans FC)	**C : -81%(CPU)** **C : -66%(GPU)**										

TABLE 3.1 – Revues des taux de compression des méthodes de pruning de l'état de l'art sur CIFAR10, CIFAR100 et ImageNet. Les chiffres en gras dénotent les algorithmes les plus efficaces selon le modèle de CNN. Les valeurs reportées sont celles affichés par leurs auteurs respectifs. Les méthodes non-structurées atteignent les plus fort taux de compression. A noter que les réductions de calculs sont théoriques et nécessitent une étape de post-traitement ou un accélérateur adapté.

3.1.3.3 Sensibilité du réseau suivant les méthodes de pruning

Pour corroborer l'intuition derrière les taux maximum de compression typiquement atteignables selon le modèle de pruning choisi, les figures en 3.12 propose une exploration succincte [4] sur le réseau MobileNetv2 entraîné sur ModelNet40. La figure du haut expose une compression avec la granularité la plus fine par suppression des paramètres de plus faible valeur, c'est-à-dire non structurée. La deuxième méthode supprime les canaux d'activations par ordre croissant de leur norme L_1. Ces deux figures démontrent qu'une méthode non-structurée est significativement plus efficace pour réduire la quantité de paramètres du réseau. Par conséquent, la suite de la thèse sera centrée sur la méthode de compression non-structurée et ce, afin de démontrer qu'un réseau multi-vues est moins sensible à la compression.

4. Seules les 10 premières couches sont représentées sur les graphes

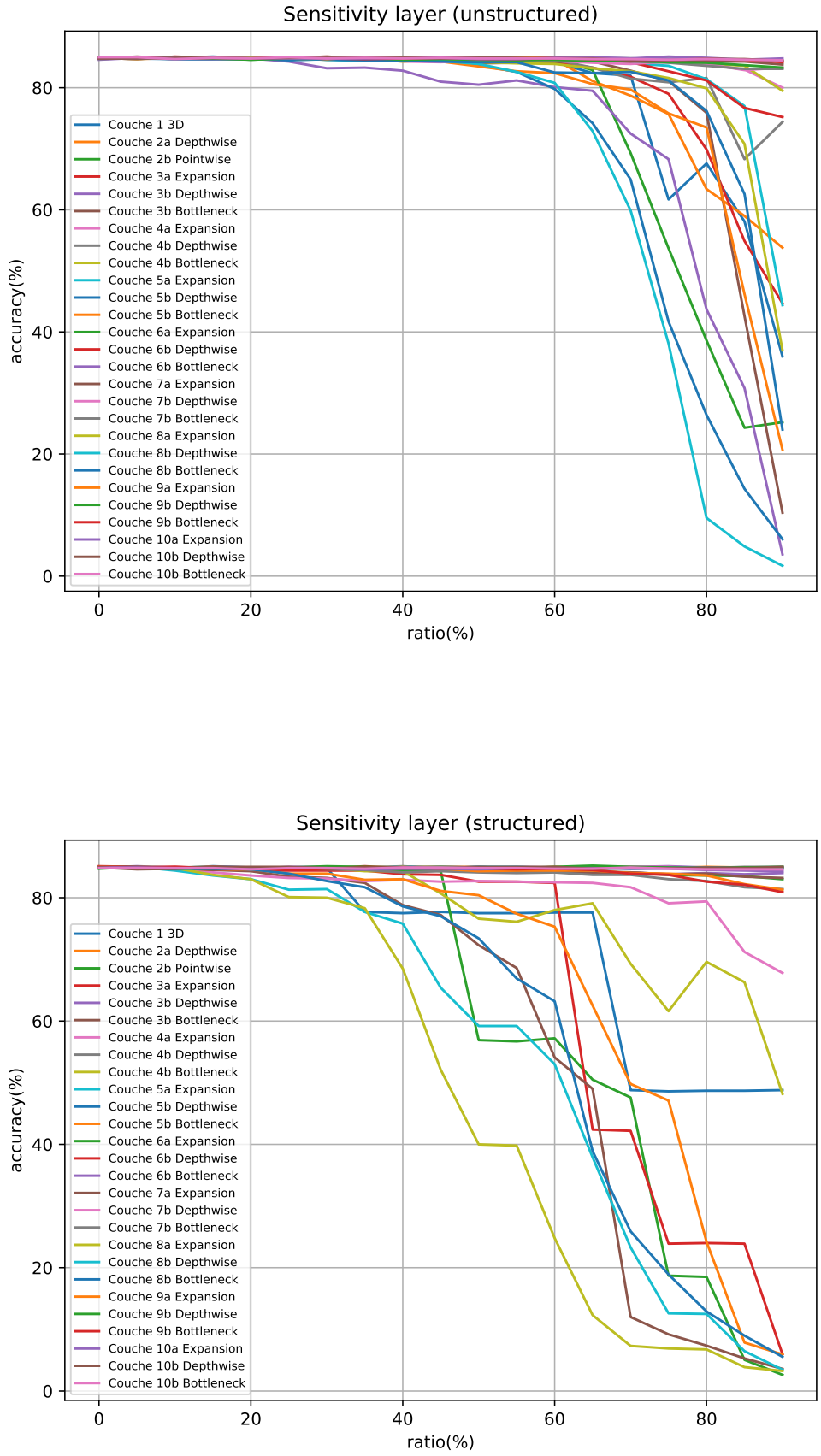


FIGURE 3.12 – La figure du haut correspond à une méthode non-structurée. La figure du bas correspond à une suppression des canaux de couches sur MobileNetv2. Les deux courbes considèrent les couches 1 à 10 de MobileNetV2 sur ModelNet40

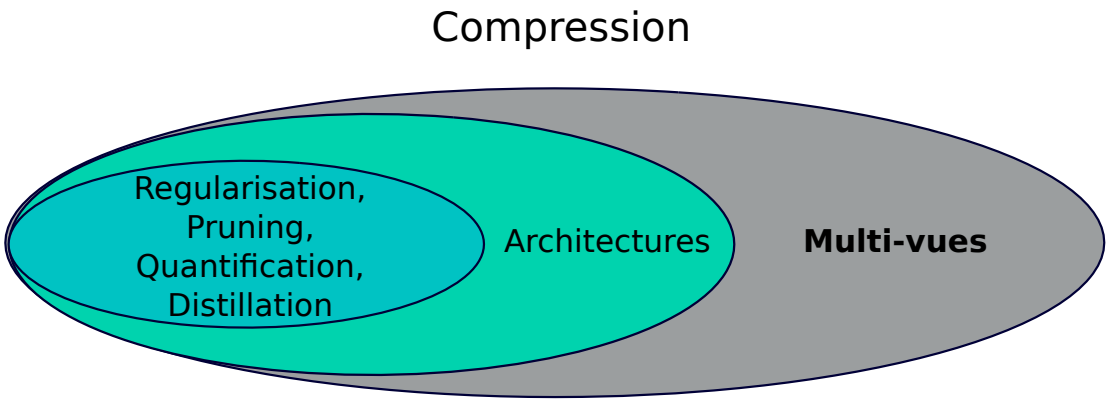


FIGURE 3.13 – L'idée véhiculée dans la prochaine section consiste à appliquer la technique du multi-vues sur architecture mobile laquelle est soumise à un algorithme de compression. La technique de pruning non structurée est choisie parmi les 4 grandes classes de compression existantes, ceci afin de démontrer qu'un réseaux multi-vues requiert moins de paramètres à taux de classification égal à un réseau mono-vue

3.1.4 Discussion

Cet état de l'art sur la compression de réseau de convolutions a mis en lumière un point intéressant, le fait que la majorité des réseaux de l'état de l'art sont sur-paramétrés. Évidemment, il reste difficile d'évaluer à priori le nombre de paramètres (nombre de couches, nombre de canaux par couches, nombre de noyaux de convolutions par canal ou encore la taille des noyaux de convolution) nécessaires pour classifier un ensemble de données particulier. La régularisation et la compression par pruning permettent d'identifier et de supprimer les paramètres redondants ou inutiles d'un CNN. Par exemple, les réseaux pensés pour la classification d'ImageNet atteignent de forts taux de compression s'ils sont réutilisés sur des problèmes moins complexes tels que CIFAR10 ou ModelNet40. Cependant, aucune de ces études n'établit la relation entre la quantité de données utilisée lors de l'inférence et le facteur de compression, autrement dit le rapport qualitatif-quantitatif d'un réseau de convolutions. La prochaine section propose d'établir ce lien d'inter-dépendance et de démontrer qu'un réglage multi-vues pour une scène donnée, est plus apte à la compression et donc reste plus adapté à un schéma d'inférence flot de données sur cible FPGA. Cette régularisation multi-vues est nouvelle dans le sens où aucun des travaux cités ne considèrent la diversité des données pour dégrader un réseau de convolutions. Cette idée est illustrée par la figure 3.13 où une architecture multi-vues, multi-données incorpore les optimisations pré-existantes.

3.2 Contribution : Dégradations non-structurées multi-vues

Désormais, nous cherchons à connaître le comportement d'un réseau de convolutions multi-vues face à de fortes dégradations en termes de nombre de paramètres et par conséquent, en termes de taux de classification. Comme il sera montré dans ce chapitre puis le chapitre4, le nombre de paramètres dicte en partie la taille de l'architecture sur la surface d'un FPGA, il est donc plus intéressant de focaliser la recherche avec un méthode non-structurée. Par conséquent, nous cherchons à savoir si, à taux de classification égal, un CNN multi-vues nécessite moins de paramètres que son équivalent mono-vues. La finalité de ce travail est de comprendre comment un réseau de convolutions standard et son homologue multi-vues se comportent à des taux de compression élevés et enfin démontrer qu'un système un système multi-vue peut résoudre le problème d'intégration du modèle flots de données DHM introduit par *Abdelouahab et al.* [APS+17a].

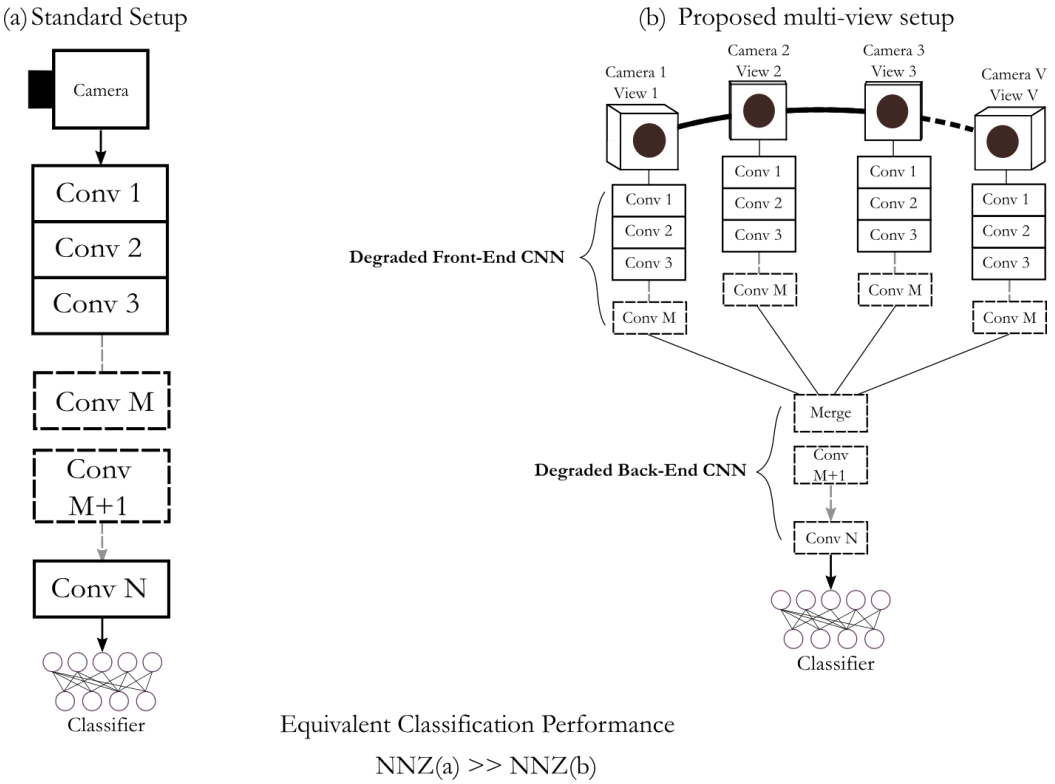


FIGURE 3.14 – Le classifieur de gauche (a) représente une configuration conventionnelle avec une caméra reliée à une succession de couches de convolutions. Le setup (b) représente son équivalent en terme de performances de classification où plusieurs caméras intelligentes embarquent les M premières couches du réseau compressées. Dans le cas où une telle solution existe, la configuration multi-vues est considérée efficace si et seulement si le nombre de paramètres combiné est au plus équivalent à celui du réseau mono-vue compressé.

Contrairement aux accélérateurs à architecture figée, un schéma d'accélération flot de

données sur FPGA est peut profiter de toutes les méthodes de compression. Par conséquent, la méthode qui permet de supprimer le plus de paramètres est de facto la plus efficace pour réduire l'empreinte matérielle à son minimum. Dans cette section, nous proposons d'explorer l'aptitude d'un CNN multi-vues face à ce type d'optimisation à granularité fine. Nous recherchons un taux de compression maximal à partir duquel le taux de reconnaissance commence à chuter de façon importante. Comme décrit dans la section 4.2, un réseau est représenté par un pipeline de N couches de convolutions F_i ($1 \leq i \leq N$) tel que :

$$\text{CNN} = \underbrace{FC \circ F_N \circ F_{N-1} \circ ... \circ F_2 \circ F_1(X_1)}_{reseau\ mono-vue} \tag{3.14}$$

où X_1 représente l'image d'entrée au format RGB et FC est le nom donné au réseau pleinement connecté. Comme montré précédemment dans la sous-section 3.1.1, les réseaux les plus récents [HZC+17, SHZ+18, TCP+18, IMA+16] sont nativement peu paramétrés mais affichent des performances proches des réseaux de l'état de l'art nettement plus denses. Pour cette raison, la méthode de compression multi-vues de cette section repose sur l'architecture MobileNetv2 [SHZ+18], bien qu'elle reste généralisable à d'autres modèles.

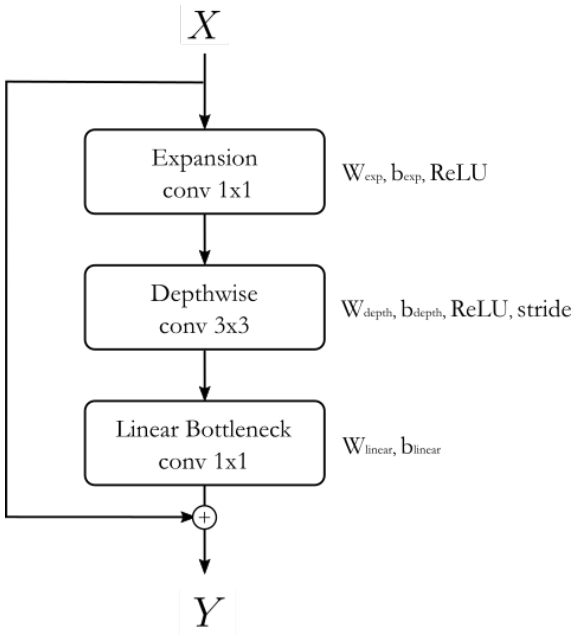


FIGURE 3.15 – La figure ci-dessus représente une couche IVR consistuée des 3 sous-modules *expansion*, *depthwise* et *linear-bottleneck*.

L'architecture des couches de MobileNetv2 combine l'idée de résidus introduits par

ResNet [WZL15] et celle de la factorisation des convolutions de MobileNetv1 [HZC+17]. En considérant les modules de normalisation ("*batchnorm*") directement intégrés dans les opérations de convolutions, la relation entre l'entrée et la sortie d'une couche s'exprime par :

$$Y = W_{linear}.[\delta[\delta(W_{exp}.X + b_{exp}) * W_{depth} + b_{depth}]] + b_{linear} + Res \tag{3.15}$$

où :

- X correspond au volume de primitives d'entrée de dimensions $(C_{exp}, U_{in}, V_{in})$.
- Y est le volume de primitives en sortie de dimensions $(C_{linear}, U_{out}, V_{out})$ de la couche de convolutions.
- W_{exp} et W_{linear} sont les matrices de paramètres de convolutions et leurs biais associés b_{exp},b_{linear}.
- W_{depth} est un tenseur de matrices de convolutions 3x3 de taille C_{depth}, et b_{depth} est son vecteur colonne de biais associé.
- δ représente la fonction d'activation
- Res est le terme résiduel tel que $Res = X$ si et seulement si le facteur de sous-échantillonage $stride = 1$ et le nombre de canaux de sorties est identique au nombre de canaux d'entrée $C_{exp} = C_{linear}$.

Le coût matériel du modèle flot de données d'une couche IVR est donné par l'équation 3.16 :

$$\mathcal{R}_n^{sv} = \mathcal{R}_{exp} + \mathcal{R}_{depth} + \mathcal{R}_{linear} \tag{3.16}$$

Ici, nous considérons que le coût d'une couche de convolutions est définie par le nombre de paramètres non-nuls, notés NNZ. Puisque chaque paramètre non-nul invoque un opérateur MAC, le coût $\mathcal{R}_n^{sv}$ de la n-ième couche de convolution IVR peut être exprimée par l'équation :

$$\mathcal{R}_n^{sv} = \text{NNZ}(W_{exp}^{(n)}) + \sum_{i=1}^{C_{depth}} \text{NNZ}(W_{depth}^{(n)}) + \text{NNZ}(W_{linear}^{(n)}) \tag{3.17}$$

Les couches du réseau étant traversées une seule fois pendant l'étape d'inférence, le nombre total de ressources nécessaires est donc la somme de tous les paramètres de la partie FCN du CNN tel que 3.18.

$$\mathcal{R}^{sv} = \sum_{n=1}^{n=N} \mathcal{R}_n^{sv} \tag{3.18}$$

Un réseau multi-vues est construit de la même manière que celui introduit dans la section 4.2, c'est-à-dire le modèle de *Su et al.* [SMKLM15b] avec la fonction de regroupement *view-pooling* ($G = \max$) en se basant sur l'architecture de MobileNetV2($\rho = 0.5$). Ce réseau, noté MVNET(V,M), est retranscrit par le modèle en 3.19 formulé en 3.19 et décrit dans la figure 3.16.

$$\text{MVNET(V,M)} = \underbrace{[FC \circ B_N \circ ... \circ B_{M+1} \circ G]}_{Back-end} \circ \underbrace{[(F_M \circ ... \circ F_1(X_v))]}_{Front-end}, \forall\, v \in [2, V] \tag{3.19}$$

où :

— M représente l'index de la dernière couche de convolutions du réseau *front-end*.

— $(F_M \circ ... \circ F_1(X_v))$ sont les M premières couches du réseau *front-end*.

— G est la fonction de regroupent des V primitives issues.

— $(B_N \circ ... \circ B_{M+1})$ sont les $N - M$ dernières couches du réseau formant la partie *back-end*.

— $(X_1, ..., X_v)$ sont les images du même objet spécifiques à chaque *front-end*.

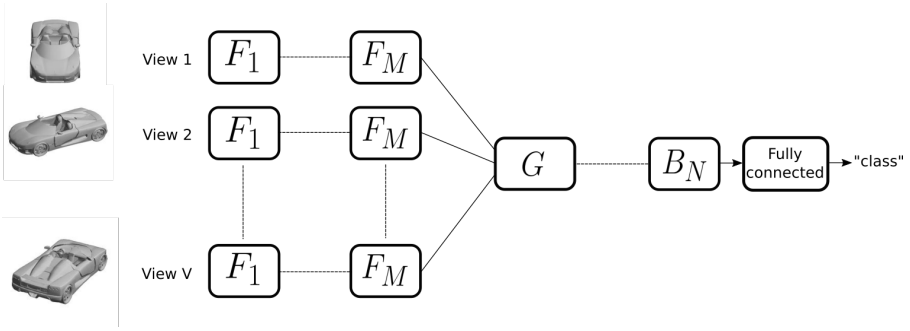


FIGURE 3.16 – Présentation du système multi-vues où de multiples *front-end* embarquent les M premières couches de convolutions. La fonction G regroupe V les primitives et les renvoie à la seconde partie du CNN.

Dans ce modèle, les parties *front-end* du réseau sont dupliquées autant de fois qu'il y a de vues et partagent toujours les mêmes paramètres. Cette configuration est par conséquent invariante à la pose de l'objet, c'est-à-dire à la configuration géométrique des caméras. Le coût matériel de la partie *front-end* ($\mathcal{R}_{FE}^{mv}$) est donc multipliée par V fois comparé à un réseau mono-vue. De l'autre côté du CNN multi-vues, c'est-à-dire la partie *back-end*, les ressources notées $\mathcal{R}_{BE}^{mv}$ restent similaires à celles rencontrées au réseau d'origine. Ces deux coût sont exprimés par les équations 3.20 et 3.21.

$$\mathcal{R}_{FE}^{mv} = v \times \sum_{n=1}^{n=M} \mathcal{R}_n^{sv} = v \times \mathcal{R}_{FE}^{sv} \tag{3.20}$$

$$\mathcal{R}_{BE}^{mv} = \sum_{n=N}^{n=M+1} \mathcal{R}_n^{sv} = \mathcal{R}_{BE}^{sv} \tag{3.21}$$

3.2.1 Formulation du problème

Pour réduire le besoin en ressources matérielles avec un CNN multi-vues, les conditions suivantes sont fixées :

- Le nombre de paramètres non-nuls du réseau multi-vues doit être nécessairement inférieur à celui du réseau de base tel que : $\mathcal{R}^{mv} < \mathcal{R}^{sv}$
- Le taux de classification doit être au minimum égal à celui du réseau mono-vue tel que : $\mathcal{A}_{mv}(M, V) \geq \mathcal{A}_{sv}$

De plus, dans notre cas d'étude, nous fixons les paramètre suivant :

- La position du point de regroupement des primitives M est défini par $M \leq N \ \forall \ (M, N) \in \mathbb{N}_+^*$ et $N \leq 18$ dans le cas de MobileNetV2
- Le nombre de vues v varie entre 2 et 8 tel que : $v \in \{2, 3, 4, 5, 6, 7, 8\}$

Toutes ces conditions réunies peuvent être exprimées par :

$$\mathcal{R}^{mv} < \mathcal{R}^{sv} \iff \mathcal{R}_{FE}^{mv} + \mathcal{R}_{BE}^{mv} < \mathcal{R}^{sv} \tag{3.22}$$

$$\iff v \times \left(\sum_{n=1}^{n=M} \mathcal{R}_n^{sv} \right) + \sum_{n=M+1}^{n=N} \mathcal{R}_n^{sv} < \sum_{n=1}^{n=N} \mathcal{R}_n^{sv} \tag{3.23}$$

Cette inégalité posée en 3.23 n'a pas de solution en l'état puisque $v \in [2,8]$. Une ou plusieurs solutions peuvent être envisagées en ajoutant les termes de compression $\alpha^{mv} < 1$ et $\beta^{mv} < 1$ au réseau multi-vues. Le CNN mono-vue incorpore aussi un une variable de dégradation pour que la comparaison reste équitable. Par conséquent, la relation posée en 3.23 devient :

$$\alpha^{mv} \times \mathcal{R}_{FE}^{mv} + \beta^{mv} \times \mathcal{R}_{BE}^{mv} < \alpha^{sv} \times \mathcal{R}^{sv} \tag{3.24}$$

où (α^{mv} et β^{mv} sont respectivement les facteurs de compression appliqués à la partie *front-end* et *back-end*. Si une ou plusieurs solutions existent, alors elles peuvent être traduites par :

$$\alpha^{mv} < \frac{1}{v} + \frac{(\alpha^{sv} - \beta^{mv})}{v} \times \frac{\mathcal{R}_{BE}^{sv}}{\mathcal{R}_{FE}^{sv}} \tag{3.25}$$

ou,

$$\beta^{mv} < \alpha^{sv} + (\alpha^{sv} - v \times \alpha^{mv}) \times \frac{\mathcal{R}_{FE}^{sv}}{\mathcal{R}_{BE}^{sv}} \tag{3.26}$$

Puisque le nombre de paramètres $mathcal R_{FE}^{sv}$ et $\mathcal{R}_{BE}^{sv}$ du réseau mono-vue sont connues ainsi que le taux de compression maximum α^{sv}, l'étude se focalisera sur le nombre de vues v et les facteurs de compression α^{mv}, β^{mv}.

3.2.2 Dégradations

Ici nous proposons de définir la **région de dégradation** comme un seuil de compression au-delà duquel le taux de classification chute de plus d'une déviation standard [5] comparé au taux de classification nominal, c'est-à-dire sans compression. La figure 3.17 illustre cette définition avec le réseau MobileNetv2($\rho = 0.5$) pré-entraîné sur le dataset ImageNet puis affiné sur ModelNet40 avec les méthodes de *pruning* à seuillage fixe et graduelle [MZ18]. Les deux méthodes n'ont aucun impact en dessous du seuil des 50% de compression mais des dégradations apparaissent au-delà. Les performances chutent drastiquement lorsque 80% des paramètres de plus faible valeurs sont supprimés. Ces trois principales régions sont définies dans le tableau 3.2. Dans le cas d'étude présent, un modèle multi-vues doit être en mesure d'échanger son excès de taux de classification avec

5. σ est mesurée sur 100 validations du set d'images de test

un certain nombre de paramètres en entrant dans une des deux régions de dégradation. Cette hypothèse sera démontrée expérimentalement dans la section 3.2.4.3 et 3.2.4.4.

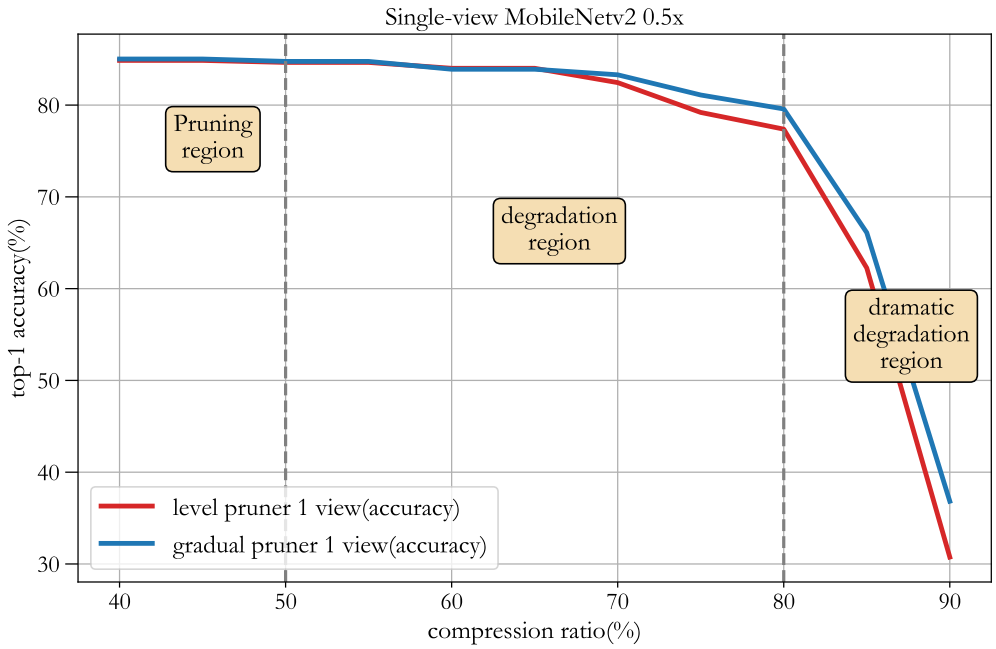


FIGURE 3.17 – Le réseau MobileNetV2 compressé selon les méthodes proposées par *Han et al.* (en bleu) et par *Zhu et al.* (en rouge). La partie gauche de la figure est considéré comme la région où la compression est effective. La dégradation (partie du milieu) commence au moment où la précision de classification dévie de plus d'un sigma de sa précision intiale, $\mathcal{A}_{max} - \sigma$. Une forte dégradation apparaît au delà de 7σ (zone de droite).

region	pruning(1)	degradation(2)	dramatic degradation(3)
Accuracy($\mathcal{A}$) (σ = 0.6)	$\mathcal{A} \geq \mathcal{A}_{sv}$ - σ	$\mathcal{A}_{sv}$ - 7*$\sigma \leq \mathcal{A} <$ $\mathcal{A}_{sv}$ - σ	$\mathcal{A} <$ $\mathcal{A}_{sv}$ - 7*σ

TABLE 3.2 – Définition des 3 régions de compression de MobileNetv2 sur ModelNet40. σ est la déviation standard du réseau original mono-vue.

Les métriques de précision/rappel et la matrice de confusion donnent des indications supplémentaires sur les performances d'un classifieur. La précision et le rappel sont définis tels que :

$$Precision = \frac{T_p}{T_p + F_p} \tag{3.27}$$

$$Recall = \frac{T_p}{T_p + F_n} \tag{3.28}$$

où pour une classe donnée, T_p (Vrai positif) est le nombre d'objets correctement classifiés, F_p (Faux positif)est le nombre d'objets en dehors de cette classe labellisés comme

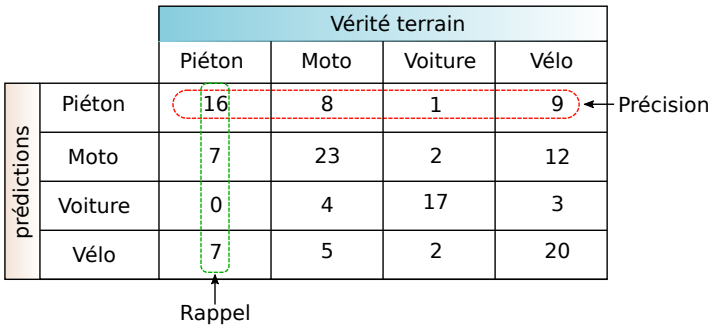
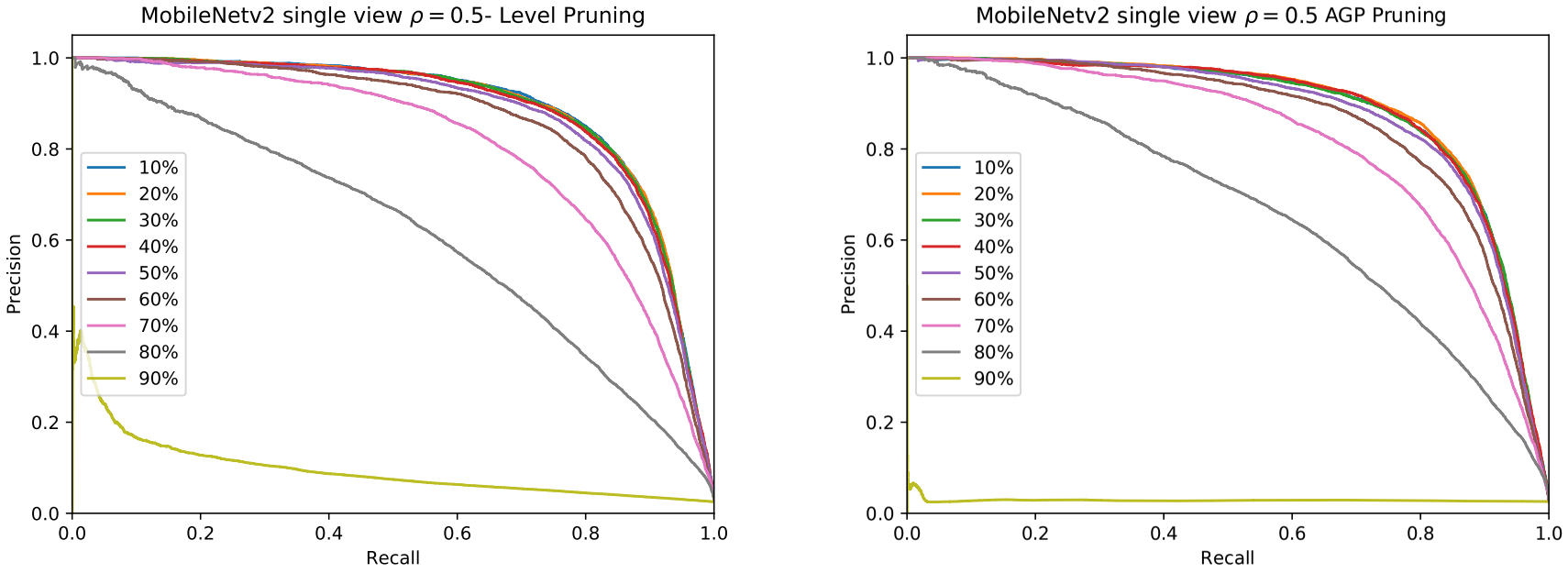


FIGURE 3.18 – Exemple de calcul de précision rappel multi-classes.

faisant partie de la classe recherchée. F_n représente le nombre de faux négatifs c'est-à-dire le nombre d'objets faisant parti de la classe mais non reconnus comme tel. Cette métrique, initialement conçue pour une classification binaire peut être étendue à une classification multi-classes. La figure 3.18 illustre un exemple où, pour un panel de 143 images, la précision du classifieur pour la classe piétons est la proportion d'images correctement classifiées en tant que piétons (16) sur le nombre total d'images classées comme piétons (16+8+1+9, en rouge). Le rappel est la proportion d'images de piétons classifiées comme piétons sur le nombre total d'images de piétons. Une courbe de précision/rappel convexe (proche de 1.0, ou "haute") est typiquement décrite par un classifieur performant. La figure 3.19 illustre les spécificités en terme de précision du réseau mono-vue MobileNetv2 sur ModelNet40 et dénote une nette dégradation pour un taux de compression supérieur à 80%. La matrice de confusion permet de visualiser concrètement les classes d'objets du dataset pour lesquelles le classifieur commet des erreurs. Pour le réseau mono-vue, la matrice de confusion montre l'effet de dégradation sur la plupart des classes de l'ensemble ModelNet40. Il est intéressant d'observer l'effet de régularisation entre les classe "Bureaux" et "Tables", très similaires, pour une compression de 80%. En d'autres termes, la diminution du nombre de paramètres permet au réseau de mieux distinguer les différences inter-classes dans ce cas particuliers.



(A) (B)

FIGURE 3.19 – Courbes de Précision/Rappel pour MobileNetv2(ρ=0.5) sur ModelNet40 avec une compression à niveaux fixes(A) et graduelle(B), de 10% à 90%. Les deux méthodes de compression présentent des dégradations similaires.

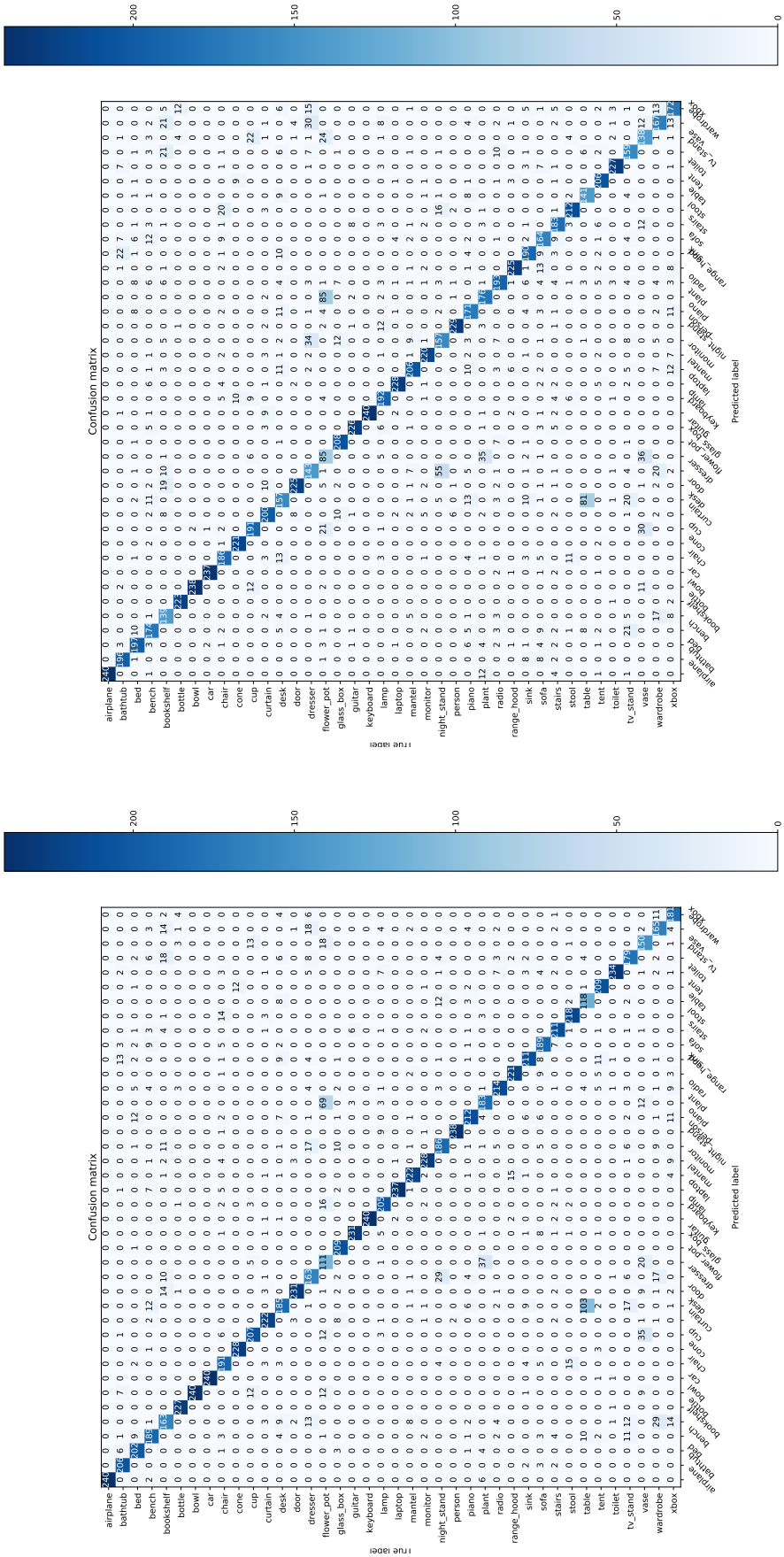


FIGURE 3.20 – Matrice de confusion pour MobileNetv2(ρ=0.5) sur ModelNet40 avec (a) un taux de compression de 50% et (b) 80%.

3.2.3 Espace d'exploration

Le problème formulé par l'équation 3.23 exige une exploration sur les variables v, α_{mv} et β_{mv} afin de trouver une voire plusieurs solutions. Cependant, l'espace des solutions est trop vaste puisqu'il requiert $M \times V$ nouveaux entrainements sur le dataset ModelNet40 et ce, pour chaque valeur α^{mv} puis β^{mv} au regard des conditions listées précédemment 3.2.1. Pour cette raison les valeurs de départ de α^{mv} et β^{mv} sont fixées tels que $\alpha^{mv} = \frac{1}{v}$ et $\beta^{mv} = 1$, ce qui garantie au minimum que le nombre de paramètres NNZ du modèle multi-vues est au moins égal à celui du réseau mono-vue non compressé. Une tendance globale de dégradation peut être extraite en suivant cette procédure. Ensuite, les modèles multi-vues valides sont de nouveau entraînés, cette fois avec un facteur de compression β^{mv} variable. De plus, le meilleur taux de compression du réseau MobileNetv2($\rho = 0.5$) étant déjà connu ($\alpha_{sv} = 0.5$), l'équation 3.26 est donc défnie par :

$$\beta^{mv} < \frac{1}{2} \times (1 - \frac{\mathcal{R}_{FE}^{sv}}{\mathcal{R}_{BE}^{sv}}) \tag{3.29}$$

3.2.4 Expériences

3.2.4.1 Ensemble d'entraînement multi-vues ModelNet40

Les expériences (entraînements et validations) sont menées sur le dataset ModelNet40 [WS]. Ce dataset est composé de 12K images de 40 classes d'objets où chaque objet est décrit par sa silhouette enregistrée dans un fichier contenant les coordonnées 3D des sommets (vertices). Chaque objet peut être affiché comme un ensemble de voxels pour un modèle 3D grossier, ou par l'intermédiaire d'un moteur de rendu, Phong-Shading [Pho75], ray-tracing [App68] ou celui intégré dans des suites logiciels 3D Blender, pour obtenir un rendu visuel réaliste (figure 3.21). Le dataset d'origine ne contient pas un nombre équi-

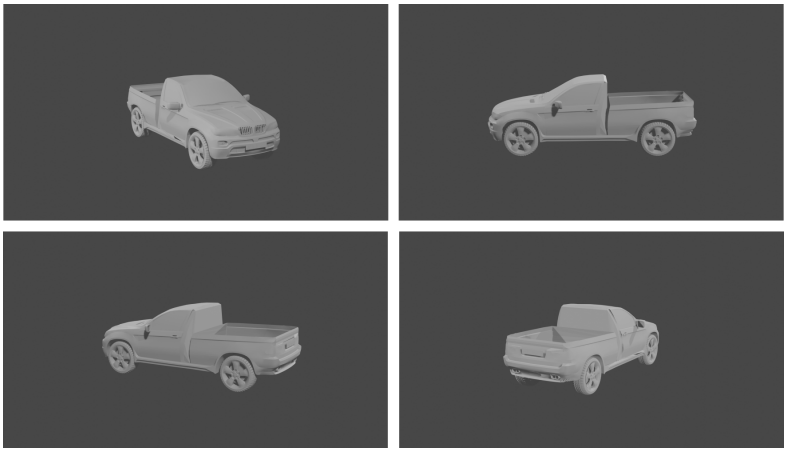


FIGURE 3.21 – Les projections 2D d'un objet sont rendues avec le logiciel de conception 3D Blender.

valent d'exemples pour chaque catégories, mais celui utilisé ici est une version modifiée avec 3200 objets pour la partie entraînement et 800 pour la validation.

3.2.4.2 MobileNetv2 multi-vues sur ModelNet40

Comme évoqué précédemment, une recherche exhaustive du meilleur point de regroupement M impliquerait de ré-entraîner M fois le même réseau pour chaque taux de compression. C'est pourquoi l'espace d'exploration est restreint à M = [2,7,12,17,N] pour un maximum de 8 vues c'est-à-dire $V \in [2;8]$. Cette solution couvre les 2 extremums du réseau MobileNetv2 concernant le point M d'une part. Les autres points $M = 7$ et $M = 12$ sont choisis de telle sorte qu'il n'existe pas de connexions résiduelles à la sortie de la dernière couche de la partie *front-end* du réseau. Une analyse de la sensibilité aux couches 6 et 11 du réseau (figures 3.23, 3.24 et 3.25) fait apparaître peu de variations du

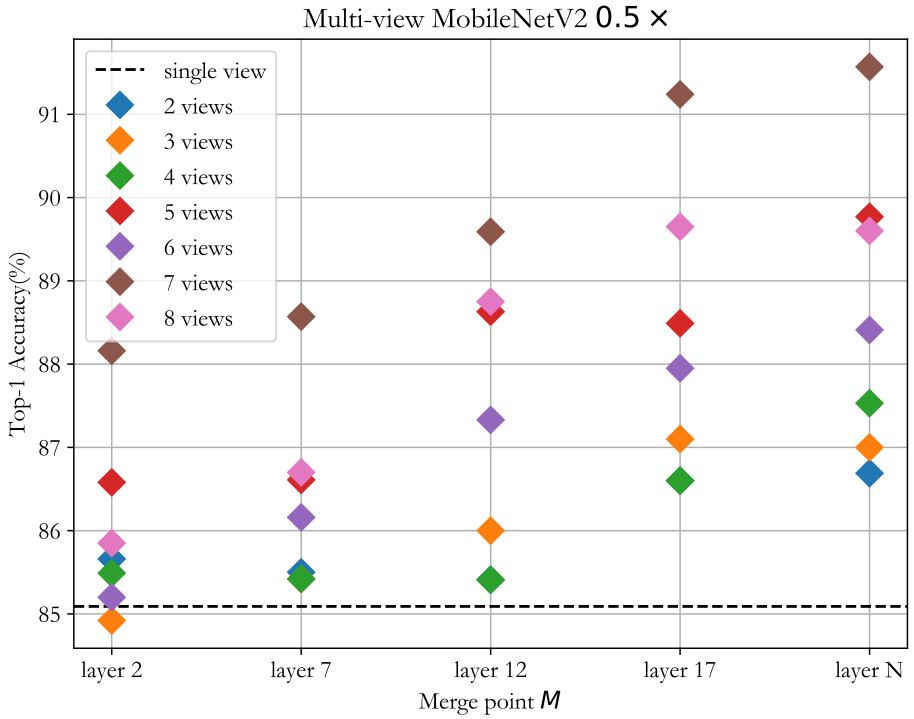
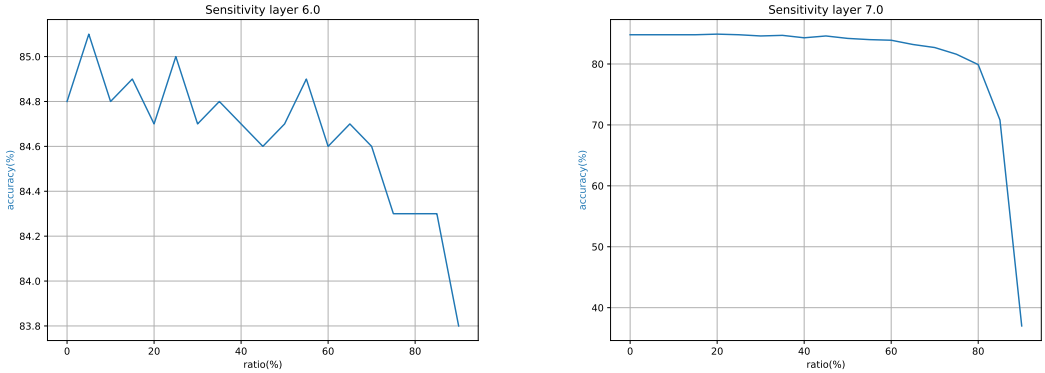


FIGURE 3.22 – Résultats de classification avec un ensemble multi-vues basé sur l'architecture MobileNetV2. Ici, $M \in [2, 7, 12, 17, N]$, $V \in [2; 8]$ où N est l'index de la dernière couche de convolution.

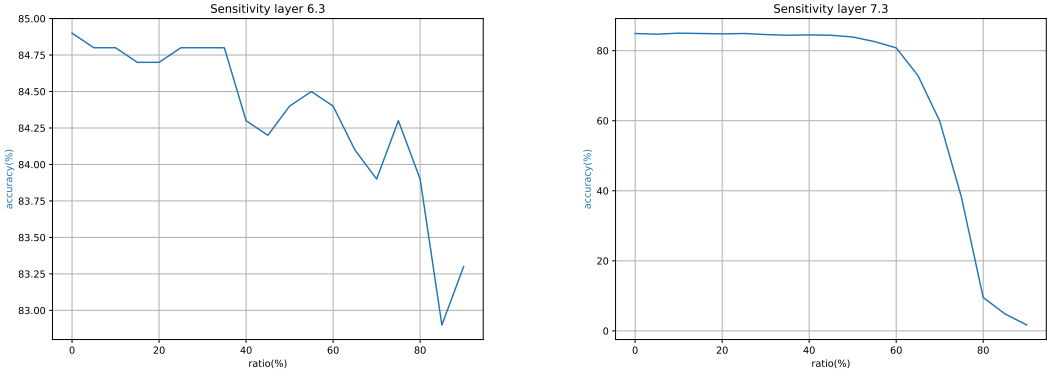
taux de classification en fonction du taux de pruning. Ceci peut s'expliquer par le fait que le réseau n'utilise pas ces couches pour apprendre de nouvelles fonctions sur la banque de données ModelNet40. Par conséquent, il est inutile de départager les parties *front-end* et *back-end* directement après les couches résiduelles. En effet, les taux de compression $\alpha_{mv}(M-1)$ et $\alpha_{MV}(M)$ pour M=6 ou M=12 présenteraient des similitudes.

Les réseaux sont entraînés sans régularisation L_2, c'est à dire que l'hyper-paramètre *weigth-decay* est réglé à zéro pour garder un contrôle total sur le nombre final de NNZ. Les résultats, illustrés par la figure 3.22, dénotent une nette sensibilité du taux de classification en fonction de l'index du point de convergence des primitives. Inférer un réseau la totalité du réseau FCN pour chacune des vues permet d'obtenir une meilleure précision et le meilleur réglage est celui pour lequel le nombre de vues $V = 7$ quelle que ce soit la valeur de M. Les deux algorithmes de compression sont directement accessibles à partir du programme *Nervana Distiller* [ZJZ+19] de *Zmora et al.* modifié pour cette étude afin d'intégrer le dataset ModelNet40 et plusieurs vues de chaque objet.



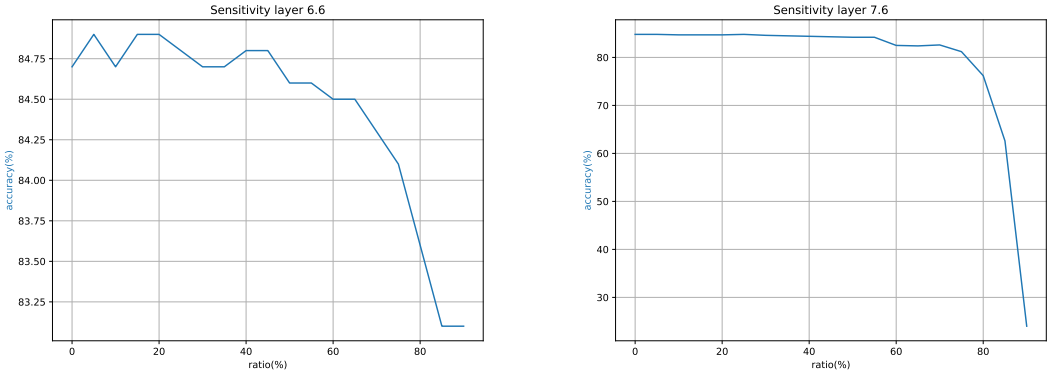
(A) (B)

FIGURE 3.23 – Sensibilité à la suppression de paramètres entre les modules *expansion* sur la couche 6 (résiduelle) (a) et 7 (non résiduelle) (b). La 6-ème couche accepte un plus haut taux de compression que la suivante. (Note : les échelles diffèrent.)



(A) (B)

FIGURE 3.24 – Différence de compression sur les modules *depthwise* de la couche 6(A) et 7(B)



(A) (B)

FIGURE 3.25 – Différence sur les modules *linear* de la couche 6(A) et 7(B)

3.2.4.3 Tendance globale de compression

Une compression globale ($\alpha^{mv} = \beta^{mv}$) contraint notamment la partie FCN des CNN telle que $(1 - \alpha^{mv})\%$ des plus petits paramètres sont supprimés sur chaque couche où :

$$\alpha^{mv} = \frac{1}{v} \tag{3.30}$$

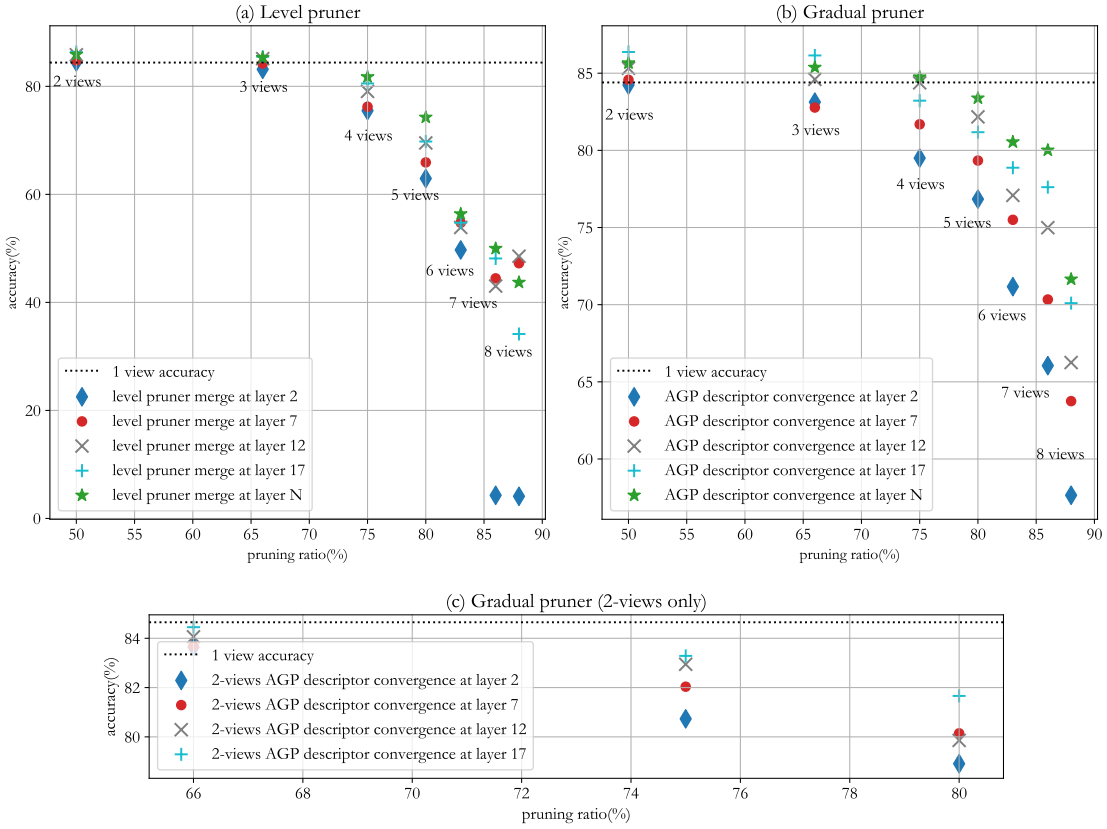


FIGURE 3.26 – Expériences de compression globale en $\frac{1}{v}$ sur les réseaux multi-vues pré-entraînés avec la méthode fixe et la méthode graduelle

La figure 3.26(a) illustre les résultats d'entraînement en appliquant la méthode de *Han et al.* avec un taux de compression fixe. La seconde 3.26(b) illustre celles avec la compression graduelle de *Zhu et al* [ZG17]. A l'issue de 30 epochs, le dernier algorithme affiche une moindre sensibilité pour le même taux de compression et plus spécialement lorsque de fortes dégradations apparaissent au delà de 5 vues ou plus. Le modèle de dégradation peut être évalué avec la relation suivante :

$$\mathcal{A}(\alpha_G) = \underbrace{-e^{a_{(M,v)}.\alpha_G^{mv}+b_{(M,v)}}}_{\text{terme de degradation}} + \mathcal{A}_{(M,\,v)}|_{\alpha_G=1},\ \alpha_G = \frac{1}{v} \tag{3.31}$$

où $a_{(m,v)}$ et $b_{(m,v)}$ sont des termes constants propre à chaque réseau multi-vues dont les valeurs dépendent du point de convergence des primitives M et du nombre de vues V. $a_{(m,v)}$ est strictement positif sur $\mathbb{R}$ et possède un effet de retard sur la dégradation du taux de classification tout comme $b_{(m,v)}$(<0). $\mathcal{A}_{(M,\,v)}|_{\alpha_G=1}$ retranscrit le taux de classification initial d'un réseau multi-vues non compressé. La figure 3.27 montre l'impact du triplet (M, V, α_G) sur les performances de classification des réseaux multi-vues. Comme énoncé plus haut, M et V ont un impact significatif sur la dégradation.

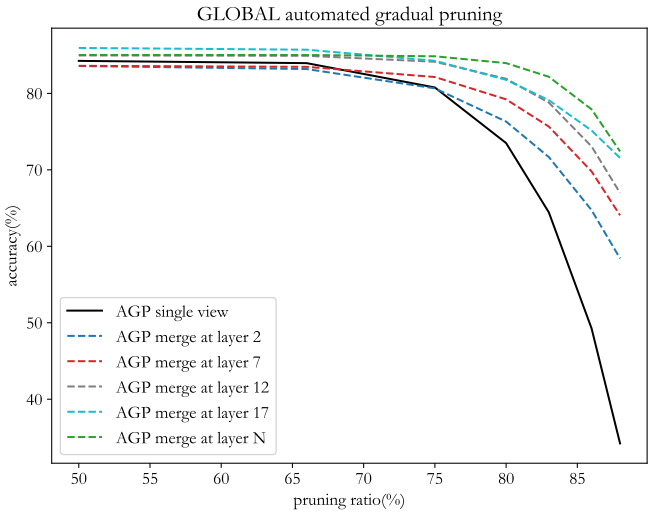


FIGURE 3.27 – Modélisation de la compression globale. Le taux de pruning est proportionnel au nombre de vues (sauf pour le modèle mono-vue).

La position $M = 2$ est la première extrémité de l'espace d'exploration où les primitives sont regroupées très tôt dans le réseau, c'est-à-dire après la deuxième couche du réseau. Cette architecture apporte peu d'informations supplémentaires comparé au réseau mono-vue mais parvient à retarder les effets de dégradations dans le cas de fortes dégradations. Dans ce cas, la partie *back-end* du système inclus 99% des NNZ. La dernière position ($M = N$) correspond au réseau MVCNN classique où toute l'extraction de primitives repose sur les V *front-end* du système. Quelque soit le taux de compression α_G^{mv}, le

M	7	12	17
$\max(\beta^{mv})$	0.486	0.429	-

TABLE 3.3 – Valeur maximum pour β_{mv} lorsque le taux de compression du *front-end* FCN est égal à $\alpha_{mv} = \frac{1}{v}$

réseau résultant aura toujours autant de paramètres que celui sur lequel il repose. Le réglage avec 2 vues n'a que peu d'intérêts car le taux de compression de 50% est équivalent à la limite haute du réseau mono-vue mais reste incapable de compenser des dégradations supérieures ajustées sur celles des modèles à 3,4 et 5 vues (figure 3.26(c)). La figure 3.26(b) démontre que certaines configurations pour $v \in \{3, 4, 5\}$ reste valides pour de plus amples explorations.

3.2.4.4 Compression du Front-end et du back-end

Les prochaines expériences étudient le comportement d'un réseau équilibré entre *front-end* et *back-end* es réseaux pour lequel le taux de classification restent au dessus des 80%, et dans la région de dégradation du modèle mono-vue, c'est-à-dire avec des points de regroupement de réseaux *front-end* tels que $7 \leq M \leq 12$. Le taux de compression de la section *front-end* reste à $\alpha^{mv} = \frac{1}{v}$ tandis que le taux appliqué au *back-end* varie entre 60% et 90% pour respecter la condition énoncée par l'équation pour $\alpha_G^{sv} = \frac{1}{2}$. La figure 3.28 montre les résultats finaux pour $v \in \{3, 4, 5\}$ et $M \in \{7, 12\}$ avec la méthode proposée. Le réseau mono-vue (en gris) est ré-entraîné dans les mêmes conditions, c'est-à-dire, la partie *front-end* puis *back-end* pour souligner les différences avec les réseaux de convolutions multi-vues compressés. Le premier setup ($v = 3, M = 7$) ne peut pas récupérer les dégradations précédentes quelque soient les valeurs de β^{mv} signifiant qu'il faille ou plus d'informations à traiter ou plus de paramètres. La première intuition est confirmée avec le réseau ($v = 4, M = 7$). Les meilleures configurations trouvées correspondent à celles dont le terme $1 - \beta^{mv}$ est le plus élevé pour un taux de reconnaissance similaire à celui du réseau mono-vue. Les points optimaux sont annotés sur la figure 3.28 et leurs projections sur la droite $y = v \times \alpha^{mv}\mathcal{R}_{FE}^{sv} + \beta^{mv} \times \mathcal{R}_{BE}^{sv}$ donnent le nombre de paramètres non-nuls correspondants.

Les taux maximum (β^{max}) reportés pour $M = 7$ et $M = 12$ sont listés dans le tableau 3.3. Le tableau 3.4 rassemble les réglages multi-vues optimaux et leurs gains en termes de NNZ ainsi que leurs gains concernant le nombre d'opérations MAC.

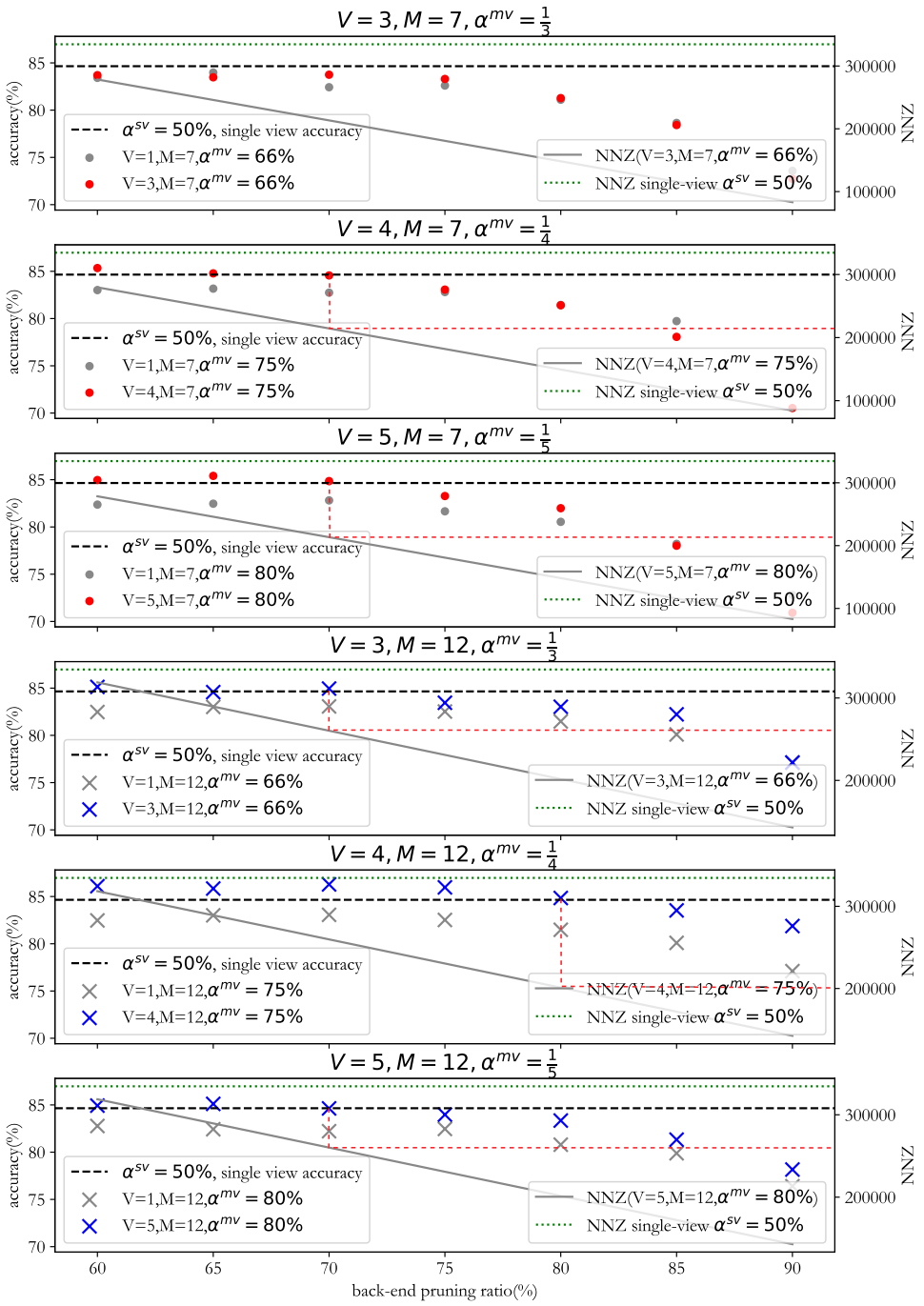


FIGURE 3.28 – Expériences menées sur les réseaux multi-vues dégradés de la section 3.2.4.3. La partie *back-end* est à nouveau entraînée avec un facteur de compression variable tel que $\beta^{mv} \in [60\%;90\%]$. Le graphe en rouge représente le réseau avec $M = 7$. Celui en bleu représente le réseau avec $M = 12$. La configuration $v = 4$ et $M = 12$ obtient le meilleur facteur de compression ($\beta^{mv} = 0.2$) à taux de reconnaissance égal avec le modèle de CNN mono-vue.

Multi-view setup	(v=3,M=7)	(v=4, M=7)	(v=5, M=7)	(v=3, M=12)	(v=4, M=12)	(v=5, M=12)
NNZ gain	-	-35.8%	-36.2%	-22.2%	-39.9%	-42.46%

TABLE 3.4 – Résumé des meilleurs taux de compression à reconnaissance équivalente. Les comparaisons sont effectuées avec le taux maximum obtenu par le CNN original.

3.2.4.5 Mise à jour des résultats

Le logiciel utilisé pour la compression des réseaux ne donne pas les facteurs de *pruning* réels dans le cas de MobileNetv2 car il ne prend pas en compte les spécificités de l'architecture du réseau. Comme expliqué dans la section 3.1.3.1, l'algorithme AGP trie les poids de convolutions de toute une couche et supprime ceux dont les valeurs sont les plus faibles. Or certains filtres 2D (W_{depth}) sont entièrement supprimés par l'algorithme alors que leurs filtres 1D (W_{exp}) associés ne le sont pas nécessairement. L'équation 3.15 stipule que le terme $\delta(W_{exp}.X + b_{exp})$ peut être nul sous cette condition. Il est donc possible de supprimer manuellement tout le canal de filtres 1D associé au filtre 2D à la manière d'un algorithme de pruning structuré. La figure 3.29 illustre cette analyse.

Le tableau 3.5 présente les nouveaux de taux de compression obtenus. Contrairement

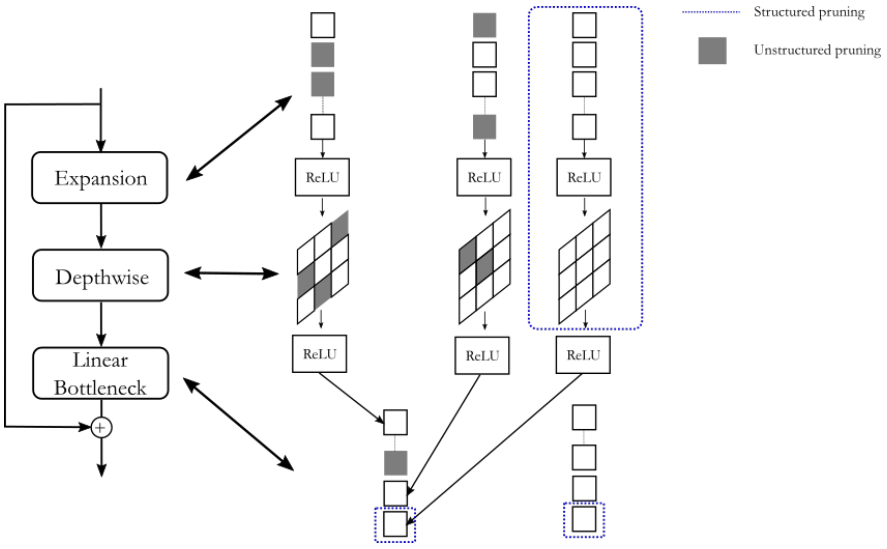


FIGURE 3.29 – La figure illustre les connexions entre les sous-blocs d'une couche IVR. Supprimer un filtre *depthwise* revient aussi à supprimer le canal *expansion* associé (traits bleus) et un paramètre de chaque bloc *bottleneck*.

aux CNN multi-vues, le réseau mono-vue compressé ne comporte aucun canal de convolutions nuls.

Le réseau contenant 4 *front-end* avec $M = 12$ confirme l'intuition formulée en 3.23 avec 41% de paramètres non-nuls de moins que le CNN mono-vue compressé à 50% pour le même taux de reconnaissance. Le réglage à 5 vues et un regroupement des activations au

Multi-view setup	(v=3,M=7)	(v=4, M=7)	(v=5, M=7)	(v=3, M=12)	(v=4, M=12)	(v=5, M=12)
Updated NNZ gain	-	-36.2%	-36.7%	-22.6%	-41.2%	-42.98%
Updated OPs gain (théorique)	-	-36.4%	-48.4%	-29.7%	-41.97%	-46.03%

TABLE 3.5 – Mis à jour des meilleurs taux de compression à taux de reconnaissance équivalente.

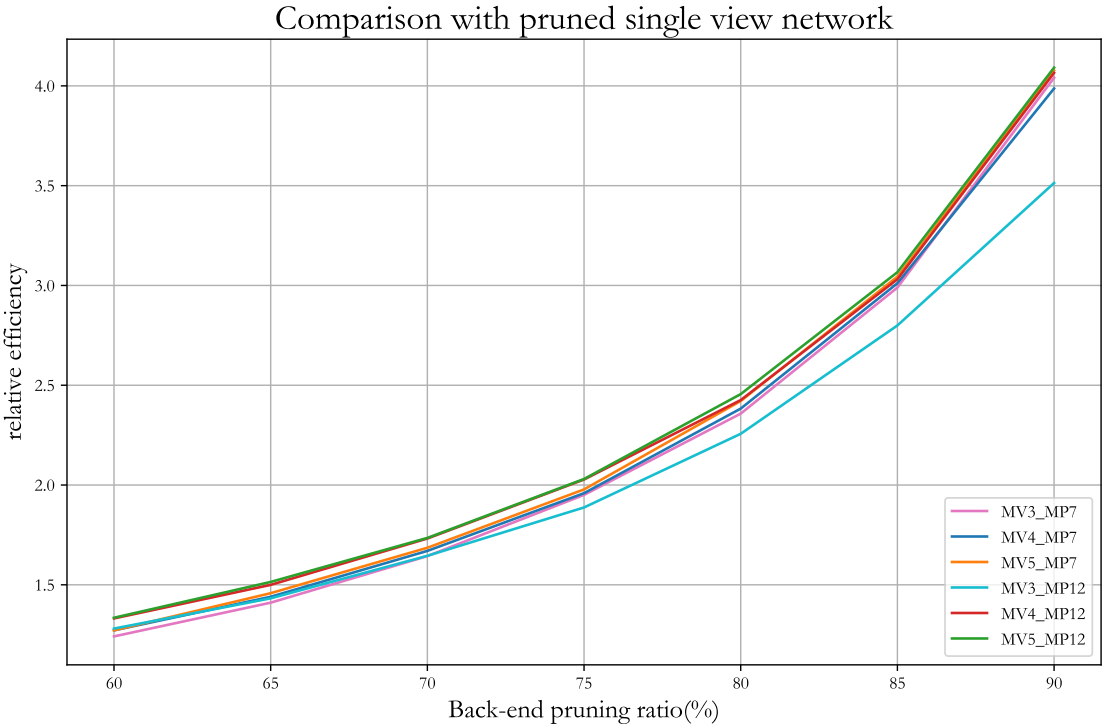


FIGURE 3.30 – Visualisation de la métrique d'efficacité des réseaux en fonction du nombre total de paramètres. Le nombre de paramètres est multiplié par le nombre de vues sur les parties *front-end*.

point $M = 12$ est le plus efficace avec 42.98% de paramètres en moins et appelle, théoriquement, 46% d'opérations en moins en considérant un accélérateur idéal capable de ne traiter que les opérations à paramètres non-nuls. Une nouvelle métrique peut être utilisée pour visualiser l'efficacité des réseaux multi-vues en fonction du nombre de paramètres comparativement à celle du réseau mono-vue. Cette fois, aucune considération n'est faite sur le taux de classification. Cette métrique est décrite par l'équation 3.32 telle que :

$$\mathrm{MV}_{relative\ eff}(\alpha_{mv}, \beta mv) = \frac{Eff(V,M)}{Eff(SV)} = \frac{Acc(V,M)}{V \times N_{fe}(V,M) + N_{be}(V,M)} \times (\frac{Acc_{sv}}{N_{sv}})^{-1} \tag{3.32}$$

avec N_{fe} et N_{be} le nombre de paramètres non-nuls sur les parties *front-end* et *back-end*. La figure 3.30 démontre plutôt que les réseaux *back-end* sont moins sensibles à un fort taux de dégradation dans un mode multi-vues.

3.2.4.6 De l'intérêt de la compression multi-vues dans un contexte FPGA

L'intérêt de la méthode explorée dans la section précédente est exclusif au contexte d'inférence DHM sur FPGA puisqu'un paramètre supprimé équivaut à un opérateur MAC en moins sur la matrice d'un FPGA. Dans le cas d'une architecture de CNN à résidu, la liste des ressources matérielles est plus difficile à appréhender qu'une architecture uniquement composée de convolutions standards. C'est pourquoi les ressources d'une couche IVR sont listées et précisément décrites ci dessous par l'équation 2.10 :

$$(\mathcal{R}+\mathcal{M})_{total} = \mathcal{R}_{exp} + \mathcal{R}_{depth} + \mathcal{R}_{residual} + \mathcal{M}_{kernel} + M_{residual} \tag{3.33}$$

et chaque sous bloc par les équations suivantes :

$$\mathcal{R}_{exp} \simeq [C_{exp}.\mathcal{R}_{\times}(m,n) + ((C_{exp}-1).\mathcal{R}_{+}(m+n+ceil(\log_2(C_{exp})))].C_{depth} \tag{3.34}$$

$$\mathcal{M}_{kernel} = (2.U_{in}+2).n.C_{depth} \tag{3.35}$$

$$\mathcal{R}_{depth} \simeq [k^2.\mathcal{R}_{\times}(n,n) + (k^2-1).\mathcal{R}_{+}(2n+ceil(\log_2(k)))].C_{depth} \tag{3.36}$$

$$\mathcal{R}_{linear} \simeq [C_{depth}.\mathcal{R}_{\times}(n,n) + (C_{depth}-1).\mathcal{R}_{+}(2n+ceil(\log_2(C_{depth})))] \times C_{linear} \tag{3.37}$$

$$\mathcal{R}_{residual} = C_{in} \times \mathcal{R}_{+}(m+n, m+n) \tag{3.38}$$

$$\mathcal{M}_{residual} = [2.U_{in} + 2 + LAT_{exp} + LAT_{depth} + LAT_{linear}].C_{exp}.m \tag{3.39}$$

où :

- $\mathcal{R}_{\times}$ et $\mathcal{R}_{+}$ correspondent au nombre de multiplieurs puis additionneurs à 2 entrées.
- n est la résolution en bits des paramètres de convolutions et des données d'activations.
- m est la résolution en bits des données de la couche précédente (pas de fonction d'activation ReLU).
- C_{exp}, C_{depth} et C_{linear} représentent le nombre de canaux pour les sous-modules d'*expansion*, *depthwise* et *linear bottleneck*.
- M_{kernel} est la largeur en bits de la mémoire pour la fenêtre de convolutions glissante de taille $k \times k$
- $M_{residual}$ est la largeur en bits pour retenir les informations résiduelles

– et $LAT_{exp} + LAT_{depth} + LAT_{linear}$ correspond au nombre de cycle d'horloge du pipeline d'une couche résiduelle. Ici, $LAT_{exp} = LAT_{linear} = LAT_{depth=1}$.

Comme le même taux de compression s'applique à toutes les ressources MAC ($\mathcal{R}$) du modèle, le nombre d'éléments logiques devrait être une fonction linéaire du taux de compression. La figure 3.31 regroupe les résultats de synthèse à partir de la couche 2 et jusqu'à la couche 7. Ces résultats montrent que le taux de compression n'est pas l'unique facteur permettant d'expliquer la surface du modèle sur la matrice d'un FPGA. D'une part, les multiplieurs sont spécialisés à leur poids de convolutions et le nombre de canaux restants dépend de l'organisation de ces poids à l'intérieur même d'une couche d'autre part. Pour la première explication, les travaux de *Abdelouahab et al.* démontrent que les ressources allouées par un synthétiseur dépendent de leur distribution, car un poids codé comme une puissance de 2 consommera moins d'éléments logiques qu'un autre. Ce modèle de ressources $\mathcal{R}_{LE}$ est formulé dans [Abd19] tel que :

$$\mathcal{R}_{LE} = a_{q^2} \times w_{q^2} + a_{q^1} \times w_{q^1} + a_{qdyn} \times w_{qdyn} \tag{3.40}$$

où a_{q^2} est le coefficient du modèle pour les poids en puissance de 2, a_{q^1} pour un poids égal à 1 et a_{qdyn} pour tous les autres. Le deuxième élément de réponse est issue de l'explication montrée par la figure 3.29 puisqu'un canal de convolution supprimé réduit le nombre d'additionneurs en sortie de la couche IVR.

Le tableau 3.6 donne les résultats de la synthèse totale de la partie *front-end* pour les réseaux avec le points de regroupements des activations à la couche 7. Notons que le tableau n'est pas complet puisque seules les sept premières couches ont pu être synthétisées avec Quartus. Au delà de 8 couches, le synthétiseur est incapable de fournir un résultat et arrête le processus pendant l'élaboration de la netlist.

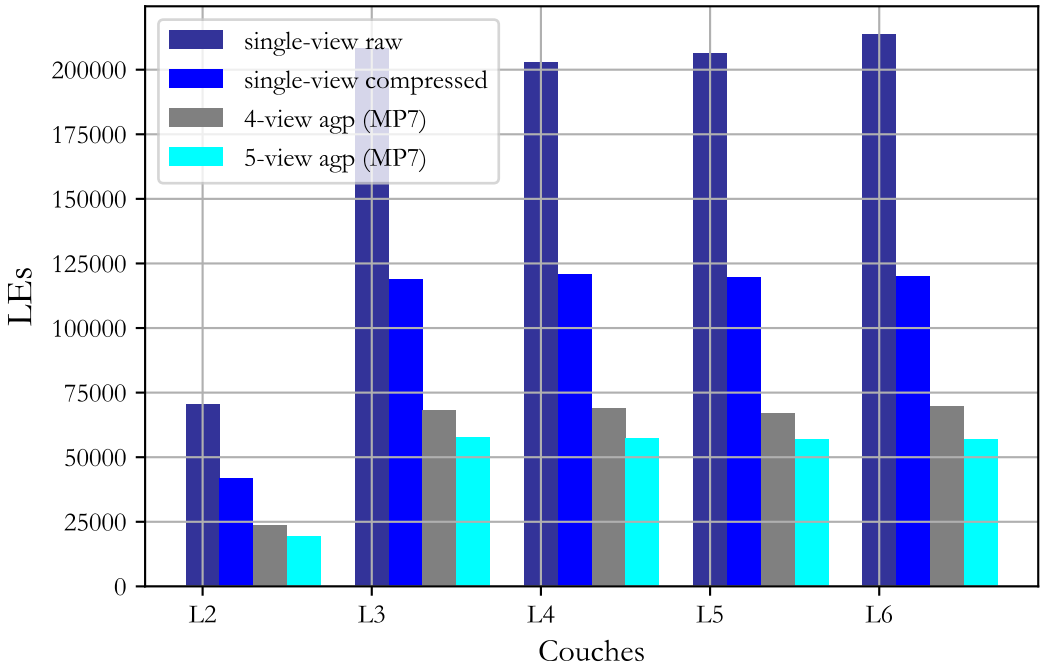


FIGURE 3.31 – Résultats de synthèse du modèle flots de données des couches IVR 2 à 6 du réseau Mobilenetv2. Les valeurs reportées correspondent au nombre d'éléments logiques intégrés dans le FPGA d'une tête de caméra.

MobileNetV2 (ρ=0.5)	Logic Elements	Memory(bits)	Gain(LEs)
single view raw (7 couches)[1]	915868	9885	-
single view agp (7 couches)[2] α = 50%	543966	8286	-41%[1]
4 views M7 (7 couches) α = 75%	325686	5674	-65%[1], -40.1%[2]
5 views M7 (7 couches) α = 80%	258231	5111	-71%[1], -52.5%[2]

TABLE 3.6 – Résultats de synthèse pour la partie front-end des réseaux multi-vues et un codage sur 8 bits. Les gains sont reportés suite à la comparaison avec le réseau non compressé[1] et compressé [2].

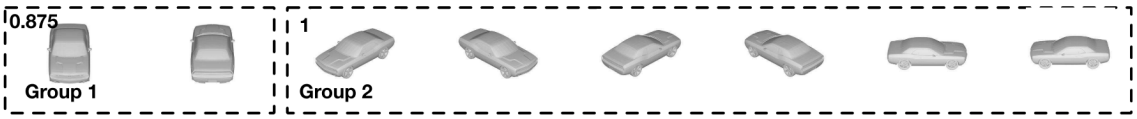


FIGURE 3.32 – Le réseau *GVCNN* regroupe les vues selon leur contribution et leur similarité. Source : [FZZ+18]

3.3 Dégradations extrêmes dans un contexte géométrique spécifique

Cette section complète les précédents résultats, cette fois en analysant la relation entre la disposition des caméras autour des objets du *dataset* et la dégradation du réseau avec un taux de compression plus élevé que précédemment. *Johns et al.* [JLD16b] ont démontré que la géométrie avait un impact sur le taux de classification en décomposant la trajectoire d'une paire stéréo en plusieurs sous-ensembles. Une deuxième méthode intimement liée à la géométrie est proposée par *Feng et al.* [FZZ+18]. Leur algorithme, décrit dans la section 2.2.1.2, catégorise les vues d'un objet et affecte un poids de contribution à chacun des sous-groupes créés. D'une certaine manière, cette méthode trouve d'elle même les poses les plus significatives pour chaque objet. Deux contre-parties sont à déplorer dans ce choix. D'une part l'algorithme duplique la totalité des couches de convolutions du réseau de base par un facteur correspondant au nombre de caméras et ajoute un réseau FC intermédiaire pour classer une vue dans un des sous-groupes. D'autre part, les expériences montrent qu'un minimum de 4 vues reste nécessaire pour atteindre un taux de classification acceptable sous peine de dégrader les performances de plus de 15%. L'idée de cette section consiste à explorer les sous-ensembles de géométries possible à partir des données de ModelNet40, lequel comporte 12 vues. Il paraît intuitif qu'une ou plusieurs géométries particulières sont plus aptes à la compression que d'autres. Par exemple, une voiture présente de fortes similarités en vue de face et en vue arrière, affirmation par ailleurs démontrée par le réseau *GVCNN* et illustrée par la figure 3.32. Qu'en est-il face à de forts taux de compression ? Quels réglages géométrique multi-vues sont les plus robustes face à la dégradation d'un réseau ? Pour répondre à ces questions, l'exploration menée dans la section précédente doit être étendue à cette nouvelle dimension.

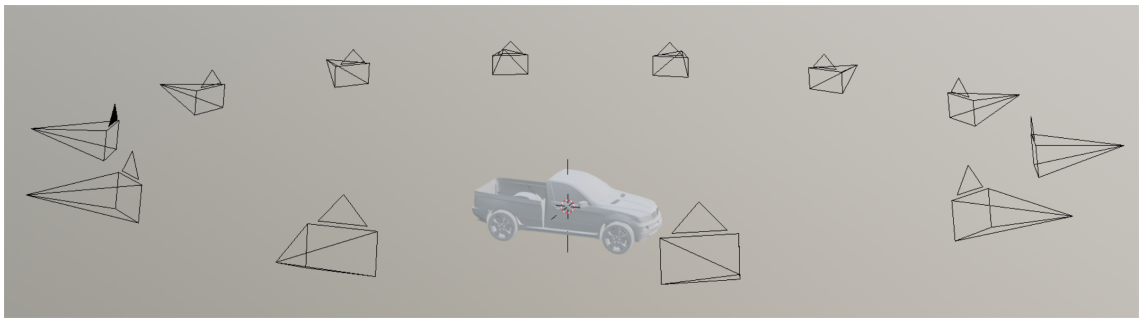


FIGURE 3.33 – Ensemble d'origine où 12 vues sont réparties autour du centre de gravité de chaque objet

3.3.1 compression et géométrie

Les expériences menées dans les sections précédentes s'appuient sur une géométrie générique. Tel qu'illustré par la figure 3.33, les V caméras sont équi-réparties selon une trajectoire circulaire autour de l'axe $(0,\vec{z})$ passant par le centre de gravité des objets. Le *dataset* original, constitué de 12 vues, peut servir de base d'exploration en n'utilisant que des sous-ensembles de v vues où v < 12. Cette idée peut être retranscrite en utilisant une formulation géométrique. Soit v caméras de même paramètres intrinsèques K et extrinsèques $[R_k|t_k]$ qui circulent autour de l'axe $\vec{z}$ telle que la projection P_k de l'objet sur le plan image de la caméra est :

$$P_k = K \times [R_k|t_k] = \begin{bmatrix} \alpha_u & 0 & C_u & 0 \\ 0 & \alpha_v & C_v & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} cos\ k\theta_s & -sin\ k\theta_s & 0 & 0 \\ sin\ k\theta_s & cos\ k\theta_s & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} cos\ \theta_r & 0 & sin\ \theta_r & 0 \\ -sin\ \theta_r & 0 & cos\ \theta_r & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.41}$$

où θ_s est un pas angulaire de résolution $\frac{2\pi}{V}$ et V représente le nombre total de vues disponibles pour un objet donné. La norme du vecteur de translation t_k est telle que $||\vec{t_k}|| = 1$ puisque les caméras circulent sur le même cercle de rayon r=1. α_u et α_v sont les constantes du capteur en pixels par mètres dans les 2 directions u et v du plan du capteur et C_u et C_v sont les coordonnées de l'origine du capteur. Le vecteur $(1,0,0,1)^T$ est la position de référence dans le plan $(O,\vec{x},\vec{y},\vec{z})$. Ici, on se propose d'explorer une nouvelle dimension relative à la géométrie des caméras dans le cas de fortes dégradations, c'est -à-dire avec un facteur de compression global $\alpha_G >> \frac{1}{v}$ appliqué à la totalité du réseau de convolutions. Les caméras sont placées autour des objets à partir du *dataset* original composé

de V = 12 vues et par conséquent $\theta_s = \frac{\pi}{6}$. Pour une caméra de référence notée C_0 dont la position initiale est donnée par le vecteur $t_0 = (1\ 0\ 0\ 1)^T$, la relation avec la k-ième caméra est donnée par l'équation 3.42 :

$$P_k = R_z(k\theta) \times P_0,\ \forall\, k\theta \in [\frac{\pi}{6};\ \frac{11\pi}{6}] \tag{3.42}$$

Pour un réglage de v caméras, toutes les géométries sont contenues dans un ensemble fini E(v) = $\{\theta_{1\to2}, \theta_{1\to3}, ..., \theta_{1\to v}\}$ de dimension égal au coefficient binomial $\binom{V}{v}$:

$$dim(E(v)) = \frac{V!}{v!(V-v)!},\ v < V \tag{3.43}$$

et la distance minimale entre 2 éléments $min(\theta_{n\to k})$ est égale au pas angulaire θ_s. Toutes les positions sont des sous-ensembles des combinaisons de v parmi les V vues disponibles dans le dataset. Par exemple, le premier ensemble de 3 vues parmi les 12 disponibles est un vecteur E_3 de 220 éléments tel que :

$$E_3 = \begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ [...] \\ G_{219} \end{bmatrix} = \begin{bmatrix} [1,2,3] \\ [1,2,4] \\ [1,2,5] \\ [...] \\ [10,11,12] \end{bmatrix} \tag{3.44}$$

Ici, nous supposons qu'un réseau doté d'un réglage géométrique initial $G_v = [\theta_1, \theta_2, \theta_3, .., \theta_v]$ particulier doit être invariant à la pose de l'objet, par conséquent le réglage G_v est augmenté de $V-1$ termes pour traduire les rotations de pas θ_s tel que :

$$G_v^{k\theta_s} = [\theta_1 + k\theta_s,\ \theta_2 + k\theta_s,\ \theta_3 + k\theta_s, ...,\ \theta_v + k\theta_s],\ k\ <\ V-1 \tag{3.45}$$

Cette rotation réduit la dimension de E_v qui devient :

$$dim(E'(v)) = \frac{\binom{V}{v}}{V} = \frac{(V-1)!}{v!(V-v)!},\ v < V \tag{3.46}$$

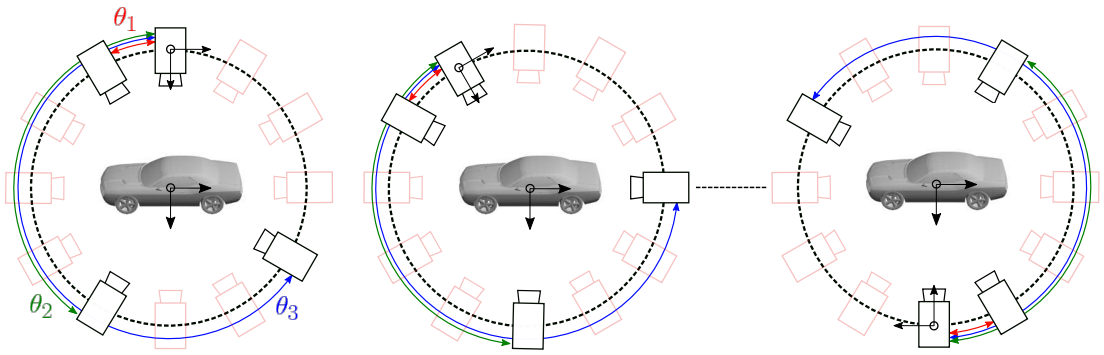
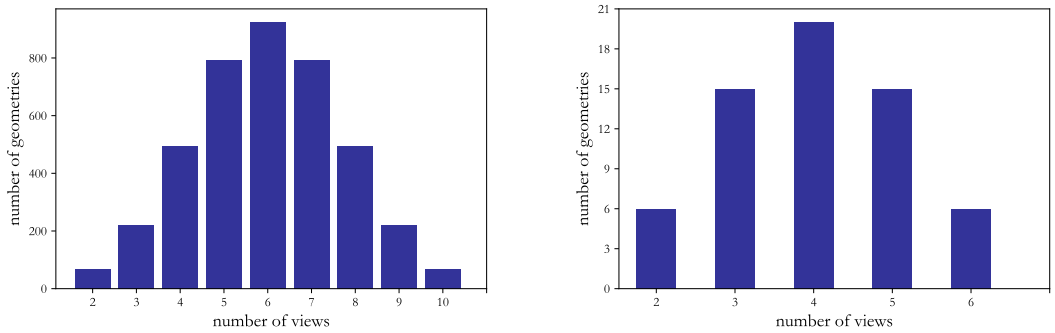


FIGURE 3.34 – Exemple de géométrie spécifique avec un réglage à 4 caméras. Une fois les angles angles $\{\theta_{1\to2}, \theta_{1\to3}, \theta_{1\to4}\}$ fixés, les caméras tournent autour de l'objet avec un pas $\theta_{step} = \frac{\pi}{6}$ pour couvrir la totalité des points de vues possibles.



(A) (B)

FIGURE 3.35 – Illustration (A) de la quantité d'entraînements requis pour évaluer l'impact de la géométrie à partir du dataset original. Cette section fait l'hypothèse d'une symétrie sur l'axe $\vec{y}$ pour ramener la recherche d'architecture à un minimum de 62 itérations pour un nombre de vues compris entre 2 et 6 (B).

La figure 3.34 permet de visualiser cette idée avec 4 caméras initialement placées autour d'un objet par l'ensemble $E_4^0 = [0, \frac{\pi}{6}, \frac{5\pi}{6}, \frac{8\pi}{6}]$. Au final, en plus du paramètre de dégradation α_G, la nouvelle fonction coût à minimiser prend cette fois en entrée un vecteur de $E(v)$ et peut être formulée de la manière suivante :

$$\underset{W, G_v^{k\theta_s}}{\text{Loss}}\,(\text{MVNET}) = L_{softmax}(W, M, \alpha_G, G_v^{k\theta_s}),\ \forall\, v < V \tag{3.47}$$

Le nombre d'entraînements nécessaires pour valider cette approche est donc $dim(E'(v))\times$ plus large que dans le cas étudié dans la section 3.2.4.3. La figure 3.35(A) montre la quantité de géométries distinctes en fonction du nombre de vues. L'espace d'exploration peut être réduit (figure 3.35(B)) en faisant l'hypothèse d'un plan de symétrie (y=0) (figures :) sur les classes du dataset. Autrement dit, le champ d'observation initial des caméras

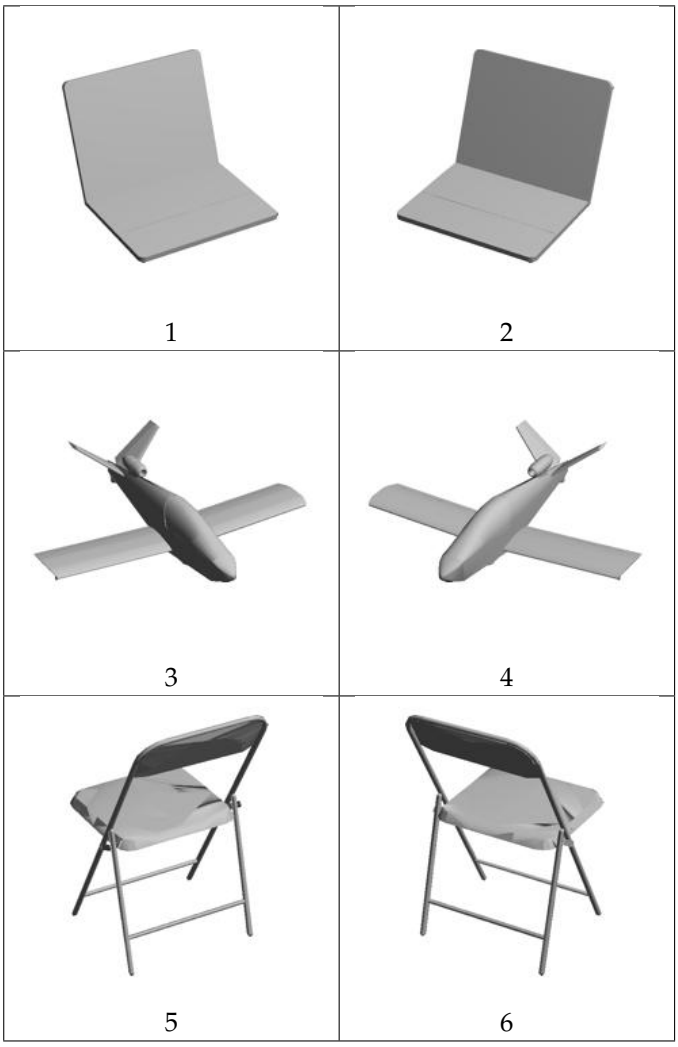


FIGURE 3.36 – Illustrations de la symétrie suivant un plan d'équation y = 0 sur les objets du dataset ModelNet40.

passe de $\theta_v \in [0, \frac{2\pi(V-1)}{V}]$ à $\theta_v \in [0, \frac{(\frac{2\pi V}{2}-1)}{\frac{V}{2}}]$ mais en maintenant le nombre de rotations $k\theta_s$ constant.

Le code python 3.3.1 complète le programme *distiller* afin d'initialiser chaque entraînement avec une géométrie prédéfinie puis en complétant l'ensemble de données par V-1 rotations autour de l'objet.

LISTING 3.1 – code de séléction des géométries

```
import numpy as np
from itertools import permutations, combinations


# ETAPE 1 : liste de toutes les geometries selon le nombre de vues
def geometry_list(num_views):
    # calcul des combinaisons geometriques
    comb_temp = combinations(idx, num_views)
    comb = list(comb_temp)
    nb_geom = len(list(comb)))
        # retour de la liste et du nombre de combinaisons
    return(list(comb), int(nb_geom))

# ETAPE 2 : Selection des geometries de depart
def starting_geometry(max_views, num_views):
    geom_array = geometry_list(num_views)
    starting_geometry_array = []
    for i in range(len(geom_array)):
        if (geom_array[i][0][0] == 1) and (geom_array[i][0][num_views-1] <= max_views/2)
    :
            starting_geometry_array.append(geom_array[i])
    return starting_geometry_array

# ETAPE 3 : Reconstruction du dataset
def geometry_select(comb, nb_geom, max_views, num_views):
    step_angle = 0 #
    temp = []
    geom_array = np.zeros((int(nb_geom), max_views, num_views))
    for g in range(int(nb_geom)):
        # Ne prendre que les geometries de depart comprises entre 0 et pi
        geom_array[g][0]=starting_geometry(max_views, num_views)
        for n in range(max_views-1):
            for v in range(num_views):
                temp.append("%03d" %((int(starting_geom[v])+n)%12+1))
            # rotation autour de l'objet suivant la geometrie choisie
            step_angle+=1
            # modulo le nombre maximum de vues
            step_angle%=12
            geom_array[g][n+1]=temp
            temp = []
        step_angle = 1
    # retour du tableau contenant les geometries
    return(geom_array)
```

```
g_list, number_of_geometry = geometry_list(num_views)
geom_array = geometry_select(g_list,number_of_geometry, max_views, num_views)
```

3.3.2 Geométrie à 2 vues

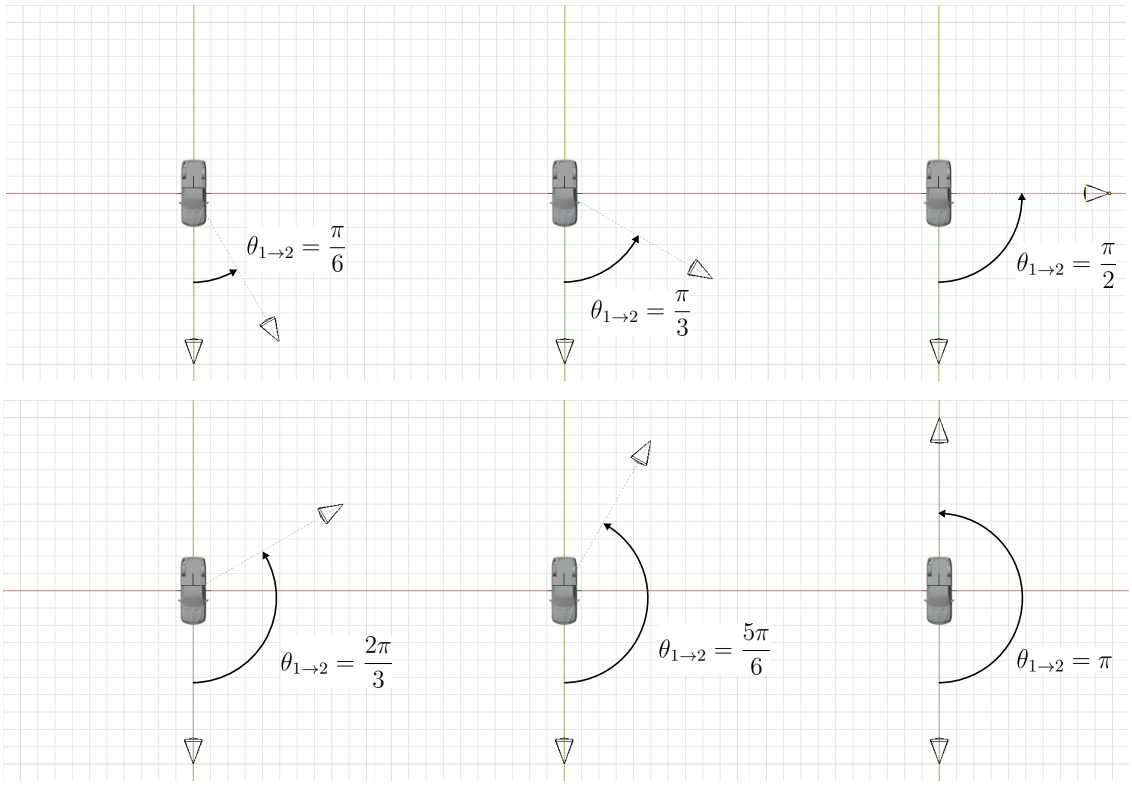


FIGURE 3.37 – Positionnements de départ dans un réglage à 2 caméras. L'espace d'exploration est limité aux 6 premières géométries tel que $\theta_{1\to2} = [\frac{\pi}{6}; \pi]$.

Cette section évalue l'impact de la géométrie dans une configuration à 2 caméras. Les caméras circulent autour de l'objet comme illustrée par la figure 3.37. Les courbes 3.38 et 3.39 montrent la réponse du réseau en fonction de la géométrie avec 2 vues. Les facteurs de dégradations compris entre $\alpha_G \in [66\%, 85\%]$ ne montrent que peu de sensibilité vis à vis de la géométrie alors que celui à 90% affecte invariablement le taux de classification. Parmi les combinaisons testées, une géométrie singulière $\theta = \frac{\pi}{2}$ reste moins sensible aux dégradations pour M=7. Dans le cas où les caméras embarquent les 12 premières couches du réseau, la réponse maximale est obtenue pour $\{\theta = \frac{5\pi}{6}\}$. Ce point de convergence n'a pas uniquement une conséquence sur le nombre de paramètres embarqués par les

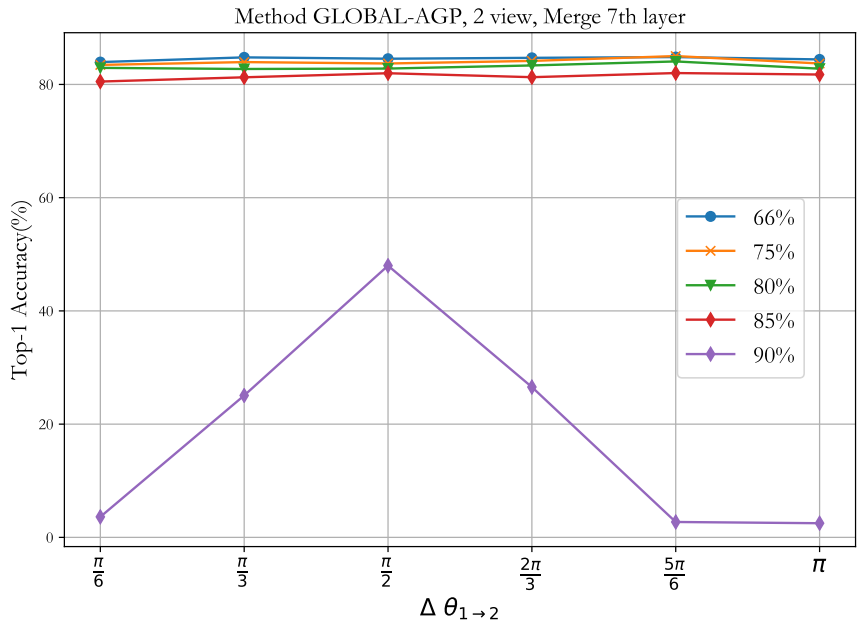


FIGURE 3.38 – Compression globale de mobilnetV2 avec M=7 et V=2

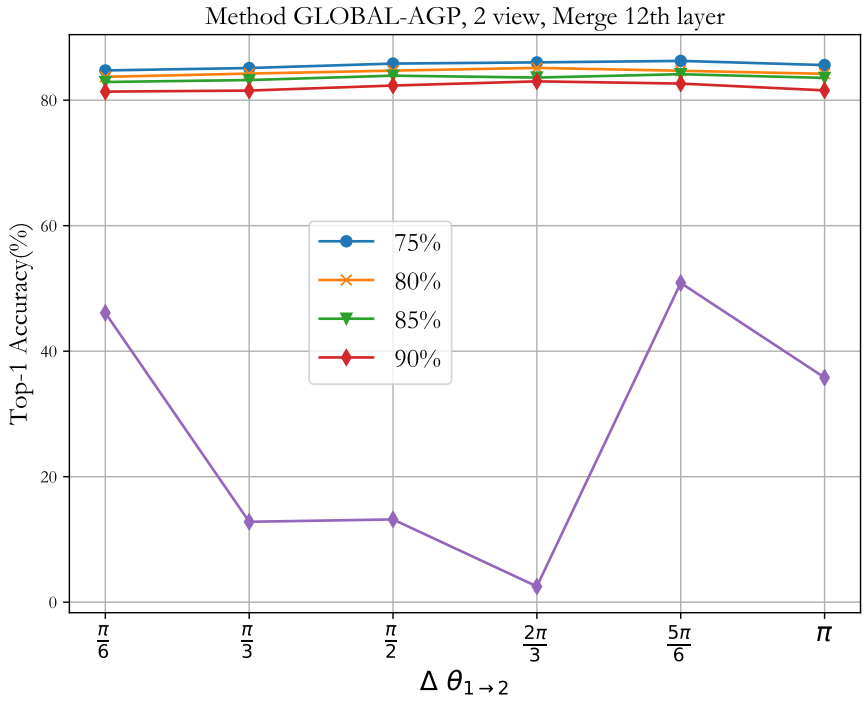


FIGURE 3.39 – Compression globale de mobilnetV2 avec M=12 et V=2

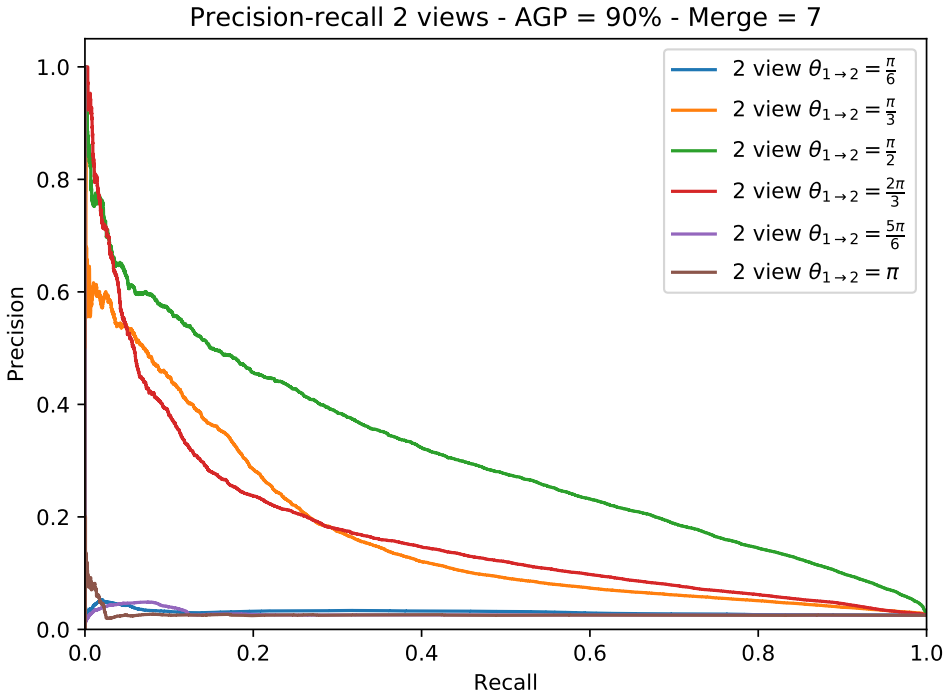


FIGURE 3.40 – Précision/Rappel pour 2 vues et M= 7

têtes de caméra du système multi-vues mais aussi sur la géométrie. Les courbes de précision/rappel et montrent que ces réseaux subissent de fortes dégradations avec un taux de pruning de 90%. Cependant, contrairement au réseau mono-vue (courbes 3.19) où la précision est nulle, la disposition des caméras a une influence positive sur les performances de classification.

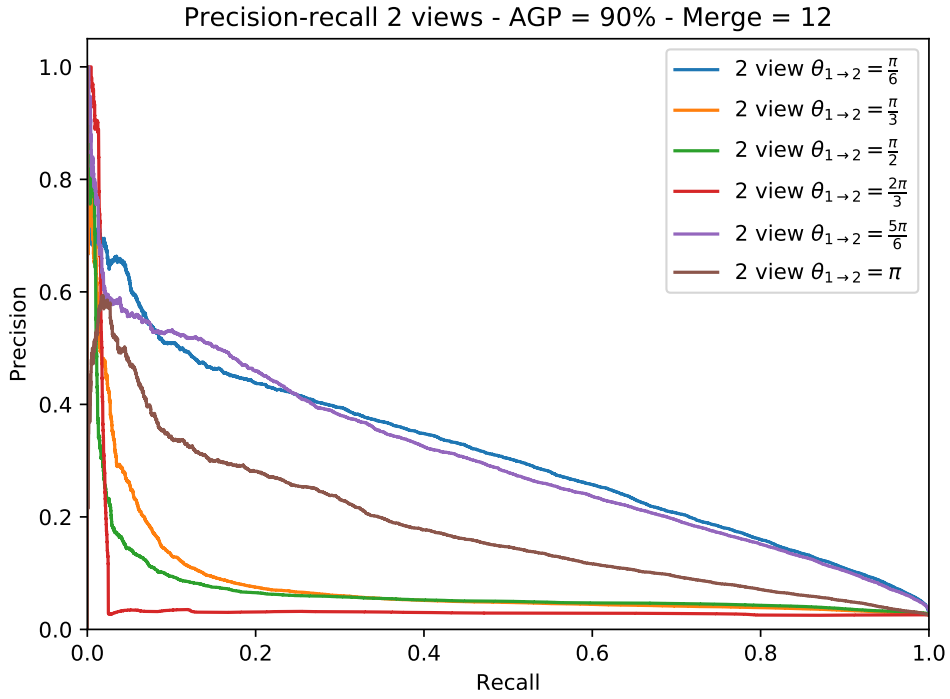


FIGURE 3.41 – Précision/Rappel pour 2 vues et M= 12

3.3.3 De 3 à 6 vues

Ici, le nombre de vues varie de 3 à 6 vues avec un nombre de géométries plus élevé. Par exemple, la dimension des sous-ensembles G_3 et G_5 sont de 15 géométries distinctes et 20 pour G_4. Le réseau à 3 vues dégradé avec les facteurs de compression $\alpha_G \in [70\%, 85\%]$ a un comportement similaire à celui utilisant 2 vues, c'est à dire que les variations du taux de classification sont minimales devant la géométrie comme montré par la figure 3.42. Par la suite, le terme de dégradation reste fixé à $\alpha_G = 90\%$ (figures 3.43, 3.44) pour visualiser uniquement les données intéressantes. Même si la limite des 80% de taux de classification est tout de même atteinte par les systèmes à 4 et 5 vues, un plateau est atteint dès l'ajout d'une troisième caméra avec plusieurs géométries quelque soient le point de regroupement des activations. La métrique de précision/rappel permet également de dégager les mêmes conclusions que celles déclarées au travers des taux de classification. Les résultats des figures 3.3.2, 3.41, 3.47, 3.48, 3.49, 3.50, 3.51, 3.52, 3.53 et 3.54 montrent qu'il est impossible de combler les dégradations et ce, quelque soit le type de géométrie appliquée au système multi-vues. L'étude de la géométrie montre une instabilité du système de reconnaissance même si, pour certaines configurations, le CNN est capable

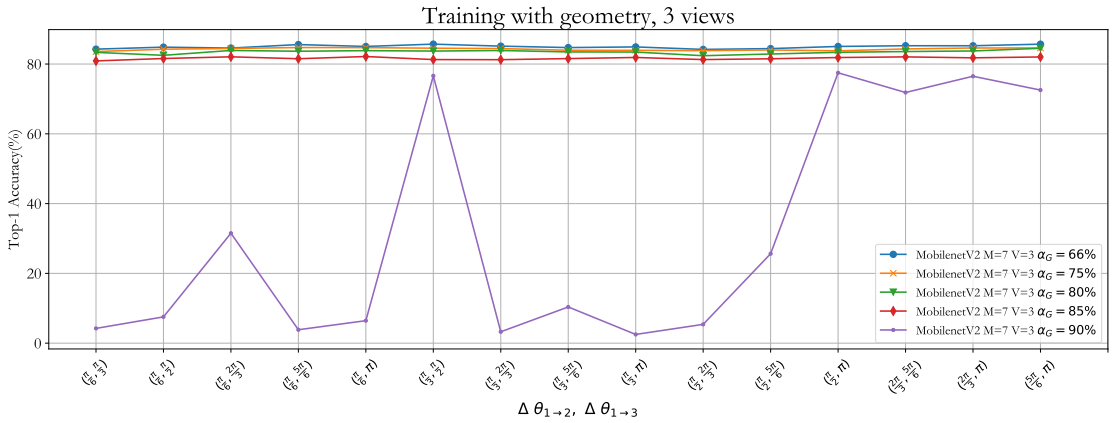


FIGURE 3.42 – Entraînements des réseaux à 3 vues avec des facteurs de dégradation compris entre $\alpha_G = 66\%$ et $\alpha_G = 90\%$. Ces réseaux affichent le même comportement que ceux à 2 vues, c'est à dire que les taux de classification varient peu devant devant la géométrie de la scène.

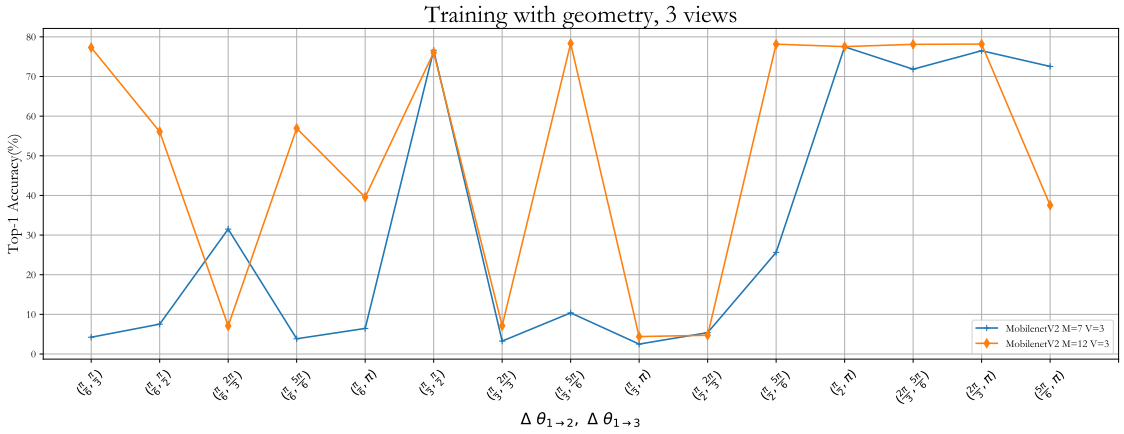


FIGURE 3.43 – Entraînements des réseaux à 3 vues et $\alpha_G = 90\%$

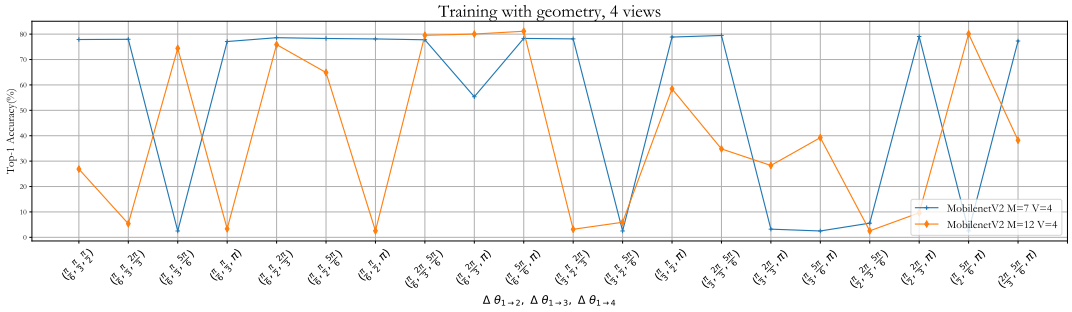


FIGURE 3.44 – Entraînements des réseaux à 4 vues et $\alpha_G = 90\%$

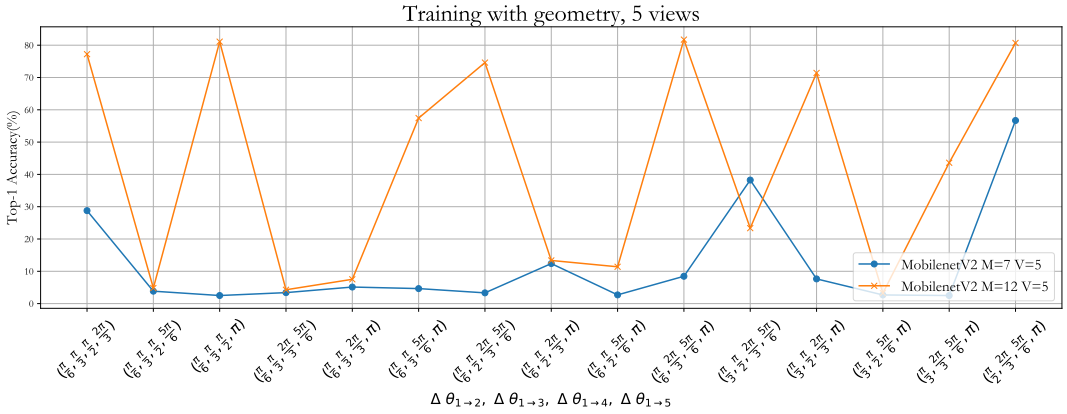


FIGURE 3.45 – Entraînements des réseaux à 5 vues et $\alpha_G = 90\%$

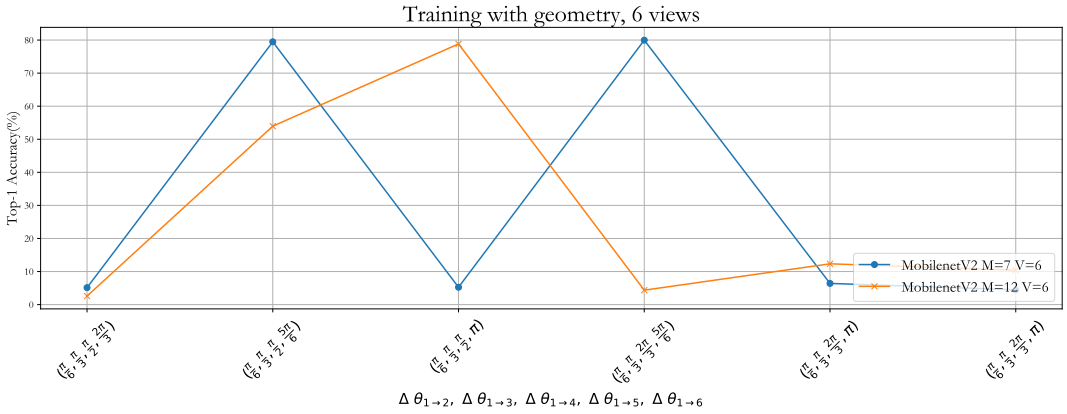


FIGURE 3.46 – Entraînements des réseaux à 6 vues et $\alpha_G = 90\%$

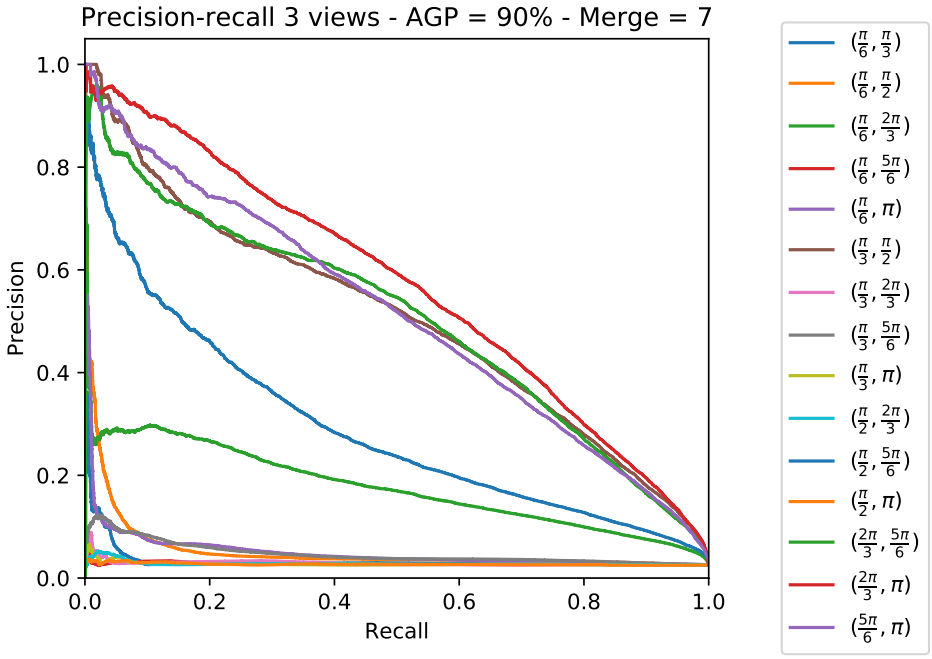


FIGURE 3.47 – Précision/Rappel pour 3 vues et M= 7

d'atteindre 80% de taux de classification. A cause de cette instabilité, il est difficile de modéliser le comportement du réseau en fonction de la géométrie. Néanmoins, une étude plus approfondie similaire à celle effectuée dans la section précédente pourrait permettre de dégager un tendance plus claire sur l'implication de la géométrie. Plus précisement, il serait nécessaire d'appliquer des taux de compression différents entre les parties *front-end* et *back-end*, et ce pour chaque géométrie.

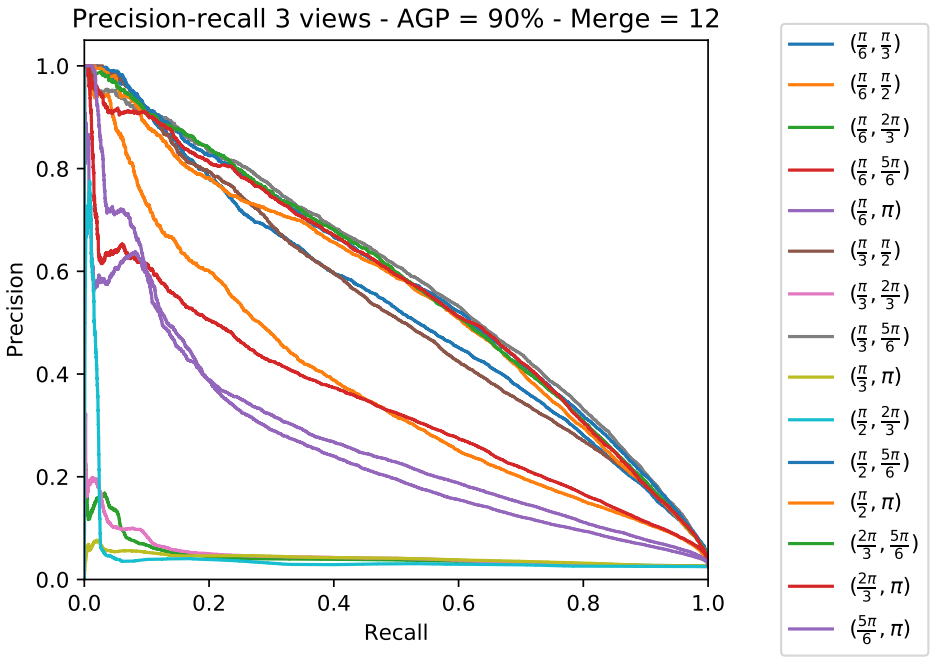


FIGURE 3.48 – Précision/Rappel pour 3 vues et M= 12

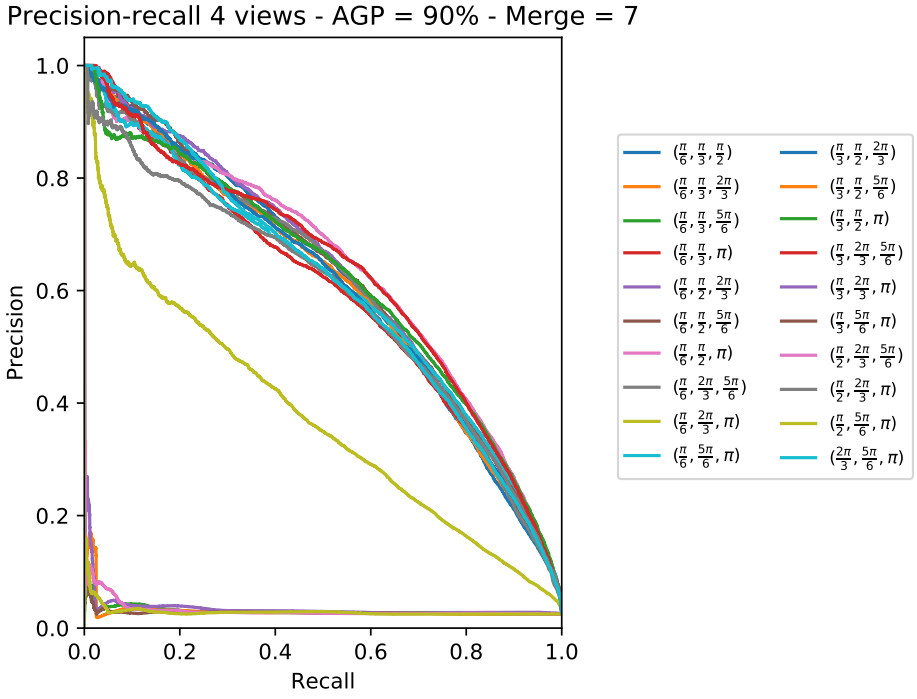


FIGURE 3.49 – Précision/Rappel pour 4 vues et M= 7

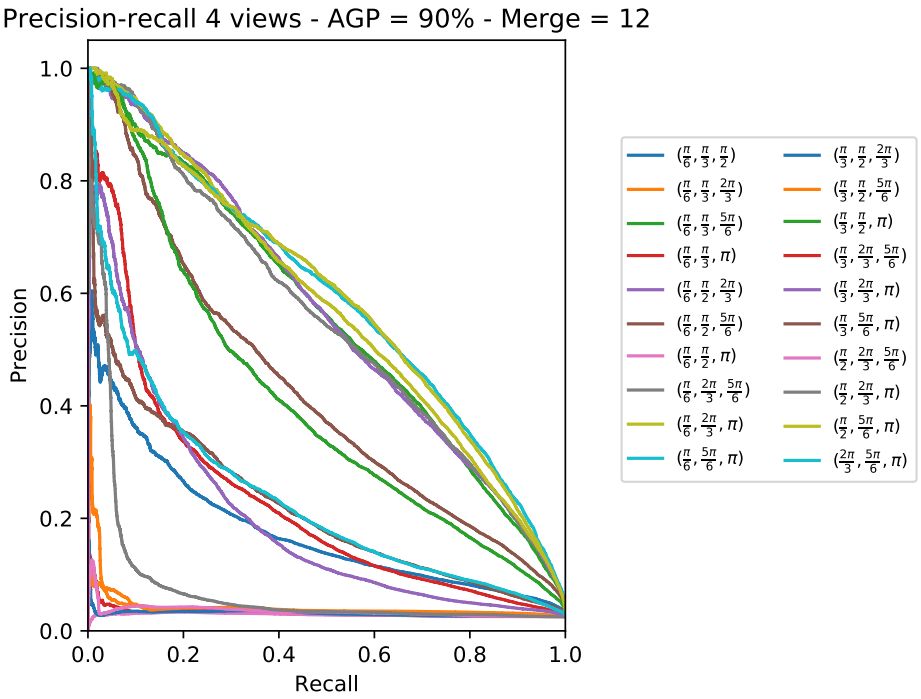


FIGURE 3.50 – Précision/Rappel pour 4 vues et M= 12

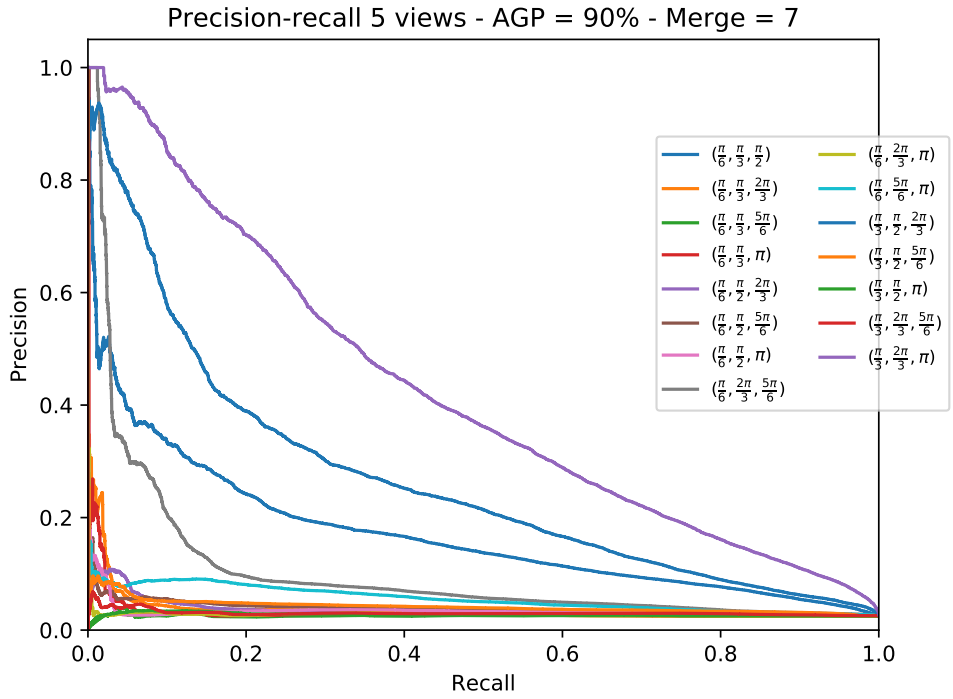


FIGURE 3.51 – Précision/Rappel pour 5 vues et M= 7

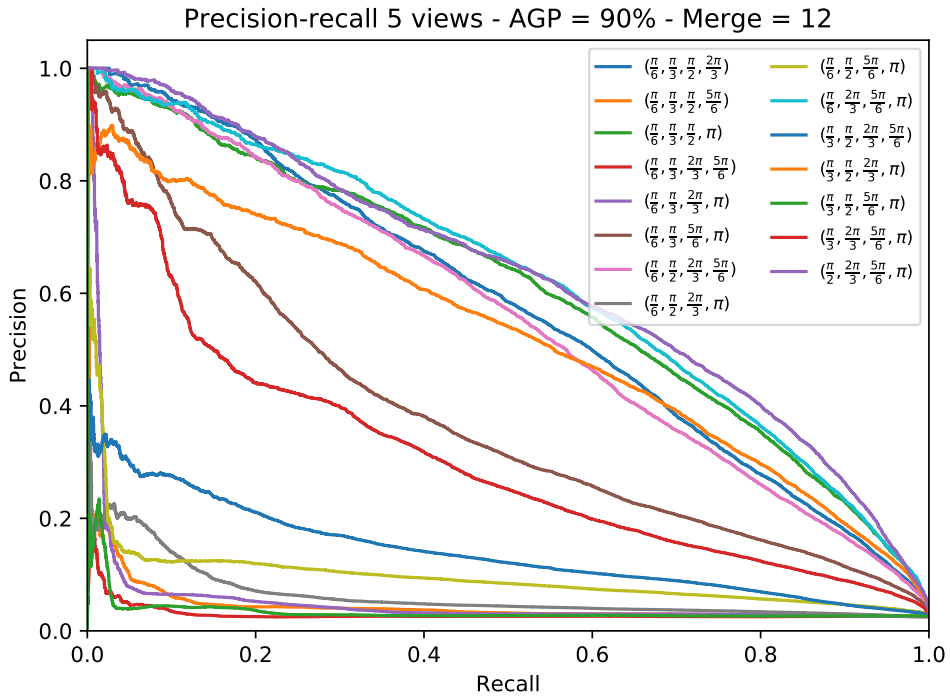


FIGURE 3.52 – Précision/Rappel pour 5 vues et M= 12

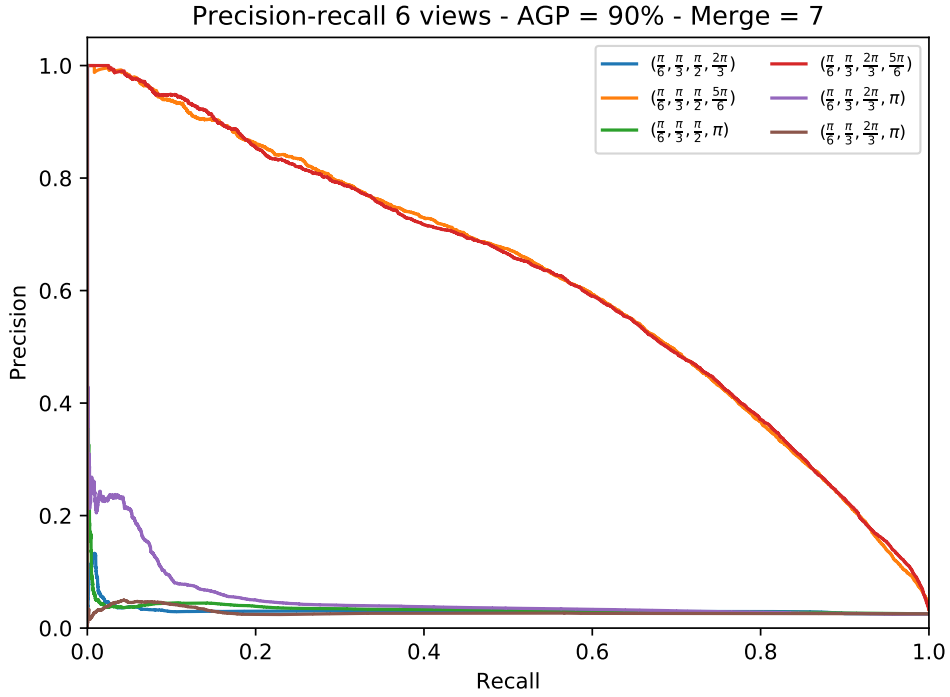


FIGURE 3.53 – Précision/Rappel pour 6 vues et M= 7

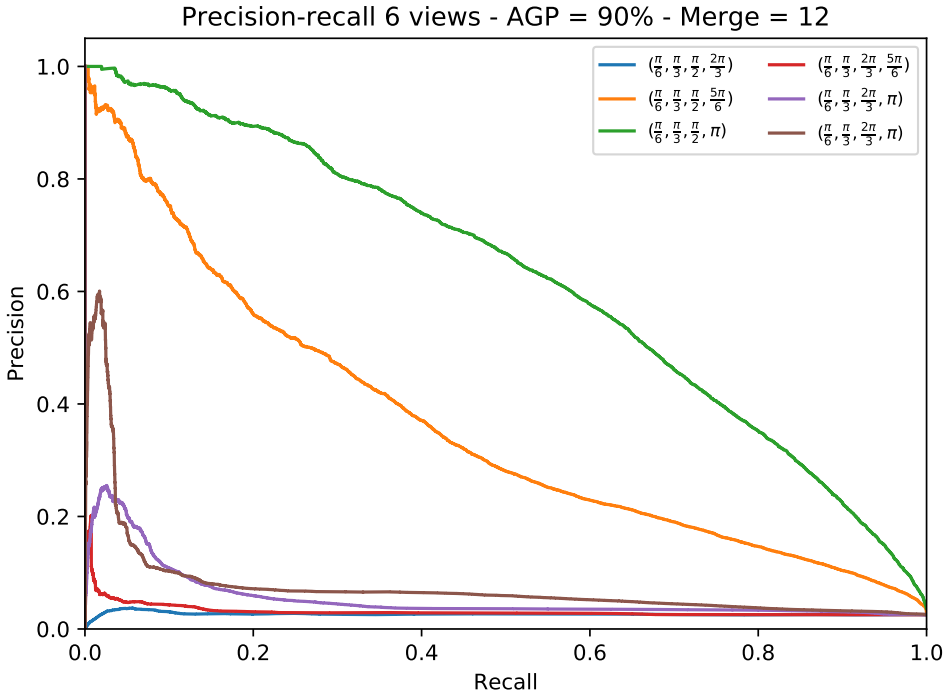


FIGURE 3.54 – Précision/Rappel pour 6 vues et M= 12

MobileNetV2 (ρ=0.5)	Logic Elements	Memory(bits)	Gain(LEs)
single view raw (7 couches)[1]	915868	9885	-
single view agp (7 couches)[2]	543966	8286	-41%[1]
best geometry (3 views M7)* α_G = 90%	119587	3159	-86.9%[1], -78%[2]
best geometry (4 views M7) α_G = 90%	119264	3159	-87%[1], -78.1%[2]

TABLE 3.7 – Synthèses des 7 premières couches avec un taux de compression de 90%. Ces résultats sont mis en perspective avec le réseau mono-vue d'origine[1] et le réseau mono-vue compressé[2].

3.3.4 Résultats de synthèse

Les résultats de synthèse pour les réseaux les plus performants sont inscrits dans le tableau 3.7 avec leurs taux de compression annotés. Avec une disposition optimale des caméras, le taux de compression a un impact notable sur le nombre de ressources matérielles nécessaires. Le réseau (M=7, V=3), fortement dégradé en termes de paramètres, avec une géométrie $\{\theta_{1\to2} = \frac{\pi}{6},\ \theta_{1\to3} = \frac{5\pi}{6}\}$ appelle près de 87% d'éléments logiques en moins que la version mono-vue non compressée pour une diminution de 6% du taux de classification. L'efficacité de ce réseau est maximale comparé aux autres ensembles multi-vues puisqu'il appelle le moins de caméras.

3.3.5 Conclusion

Ce chapitre a étudié l'optimisation non-structurée de réseaux de convolutions multi-vues. L'espace d'exploration initialement proposé concerne le nombre de vues (V), le point de regroupement (M) selon la méthode introduite par *Su et al.* puis les taux de compression (α_{fe}, β_{be} appliqués aux deux parties du système multi-vues. Le CNN de démonstration MobileNetv2 mono-vue perd en précision lorsque son taux de compression α_{sv} est supérieur à 50% puis atteint une région où la compression dégrade fortement les performances de reconnaissance au-delà d'un facteur de compression de 80%. Les réseaux multi-vues, même s'ils restent sensibles à la compression, permettent de retarder l'effet de dégradation comme illustré par la figure 3.27. Les résultats expérimentaux (figure 3.28) montre que les taux de compression sont fonctions du point de regroupement et du nombre de vues. Les performances sont maximales lorsque les parties *front-end* embarquent un maximum de couches de convolutions, revenant ainsi à la proposition faite par le modèle de réseau MVCNN. Le réseau à 5 vues et un regroupement des activations à M=12 reste le plus efficace en termes de paramètres parmi tous les réglages multi-vues explorés. Cependant, un point de regroupement aussi avancé dans le réseau ne permet pas d'embarquer autant de couches sur une tête de caméra de la partie *front-end* selon un modèle de calcul flot de données matériel. Les modèles à 4 et 5 vues pour un regroupement à M=7 représentent une contrepartie plus réaliste avec les technologies de FPGA actuels. La seconde partie proposait d'ajouter une variable liée à la géométrie de la scène à l'espace d'exploration. Le temps nécessaire pour entraîner chaque réseau étant prohibitif, des choix ont été effectués sur la méthode de compression et sur le nombre de géométries prises en compte. Par exemple, la méthode de compression est appliquée sur l'ensemble du réseau. Cette nouvelle étude a démontré expérimentalement que la géométrie pouvait avoir une influence uniquement dans les cas de compression extrêmes, mais aussi que les aptitudes de reconnaissance deviennent instables. Les meilleures configurations géométriques butent à 80% de taux de classification, soit 4.5% de moins que le réseau mono-vue compressé. Une nouvelle exploration, similaire à celle effectuée dans la partie 3.2.3, pourrait donner des indications supplémentaires sur la sensibilité des réseaux multi-vues face à la dégradation afin d'établir un modèle entre la nature des données, la structure du réseau et le taux de compression final.

Chapitre 4

Modèle de calcul mixte DHM/Multiplexage Temporel

Résumé : *Le chapitre suivant propose une manière d'exploiter de manière efficace les résultats du chapitre précédent. Limiter uniquement le nombre de paramètres est difficilement exploitable sur des architectures d'accélérateur conventionnelles. La plupart des architectures proposées dans la littérature s'appuient sur la distribution des calculs sur un ensemble de processeurs élémentaires PE orchestrés par une machine d'état centrale. Nous verrons dans la section 4.2 de ce chapitre comment le niveau de compression atteint par un modèle de réseau multi-vues permet de diminuer le temps d'inférence grâce à un mélange entre un modèle d'inférence classique et le modèle purement flot de données matériel situé au plus près des capteurs. Par manque de temps pour implémenter le réseau MobileNetv2 sur FPGA avec le modèle à multiplexage temporel (OpenCL), la majorité des résultats s'appuient sur des données issues du réseau AlexNet avec une méthode de dégradation naïve comparée à celle utilisée dans le chapitre précédent.*

Les CNN présentés dans la section 2.1.3 sont essentiellement étudiés dans le but de maximiser le taux de classification sans véritablement considérer les capacités de calcul des cibles matérielles (FPGA, CPU). Comme souligné dans la section 2.2, ce constat prévaut également pour la majorité des CNN multi-vues où la partie FCN est multipliée par le nombre de caméras. Les systèmes les plus adaptés aux étapes d'entraînement de CNN sont les processeurs graphiques GPU dotés de milliers de coeurs DSP pouvant exécuter des milliards d'opérations en virgules flottantes par seconde. Les opérations MAC d'un CNN sont parallélisées sur les coeurs de calculs du GPU le rendant au minimum 40 fois plus performant [HU16] qu'un CPU standard.

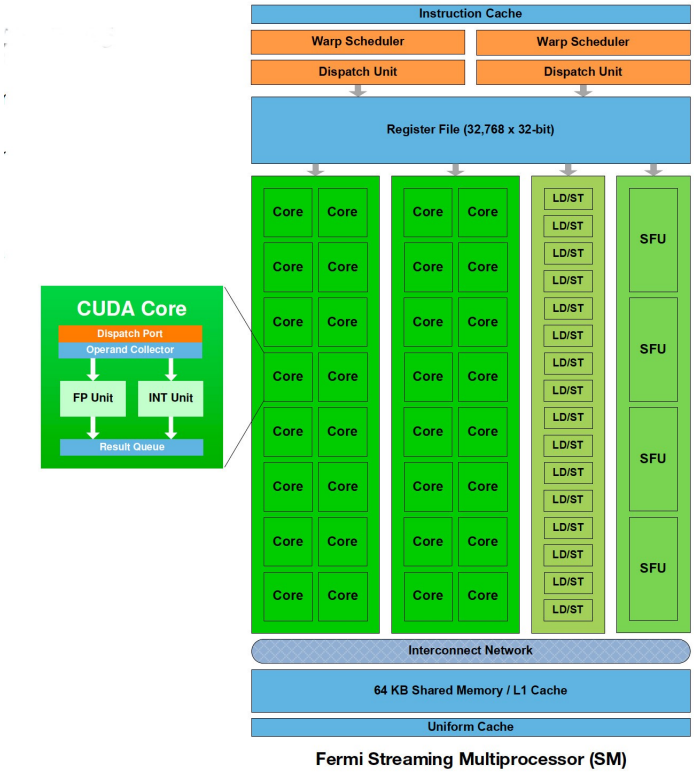
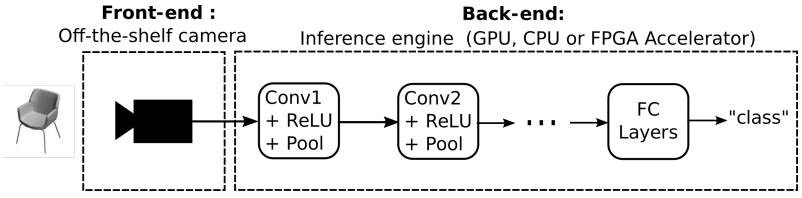


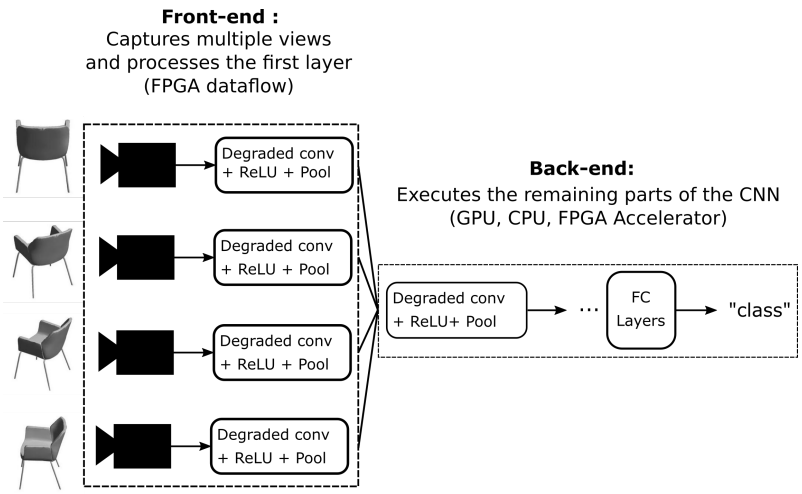
FIGURE 4.1 – Exemple d'architecture GPU composée de plusieurs unités de calculs DSP

En revanche, l'acquisition d'images se fait souvent sur des appareils externes mobiles construits autour d'électroniques de faible puissance et de plus faibles capacités de calcul incapables de traiter un nombre d'opérations aussi important en temps réel. Une solution triviale consiste à externaliser l'inférence sur des serveurs dotés de GPU au prix d'une latence avec l'appareil client. Hors, cette latence est un paramètre non négligeable pour des applications à contraintes temps réels telle que la détection d'obstacle pour la conduite autonome. Ce problème peut, en partie, être résolue par l'optimisation des chemins de données, par des structures de CNN orientées basse consommation et par la compression, au sens large, de tout type de réseau. Comme présenté dans le chapitre précédent, l'idée de compression est largement soutenue dans la littérature pour diminuer la taille de modèles denses et ce, par différentes approches telles que le *pruning*, la distillation [HVD15] ou encore la quantification des données pour limiter l'empreinte mémoire d'un côté, et pour amener les opérations de convolutions sur des unités arithmétiques moins complexes que les unités DSP en virgules flottantes. Ce chapitre montre que les FPGA représentent une alternative performante pour inférer un CNN. En effet, leur architecture générique permet d'intégrer le modèle flots de données matériel (DHM)

qui retranscrit directement la structure d'un CNN. Cependant, le nombre de ressources logiques disponibles limite grandement la possibilité d'embarquer un CNN suivant ce modèle, même pour les couches de convolutions les moins denses. Hors, comme vu dans le chapitre précédent, un CNN multi-vues dégradé requiert moins de ressources que son équivalent mono-vue à taux de classification équivalente. Au final, il devient possible de déporter un certain nombre de couches sur des FPGA embarqués comme illustré par la figure 4.2. La première section présente une revue des deux modèles de calculs de CNN tandis que la deuxième partie présente une solution mixte composée de FPGA embarqués illustrée par la figure 4.2.



(A) Architecture typique : Une caméra gère la partie acquisition puis un moteur d'inférence de type SIMD classifie l'image capturée.



(B) Architecture proposée : Plusieurs caméras acquièrent un objet sous différents points de vues et pré-traitent une partie de l'inférence sur FPGA avec le modèle DHM. La seconde partie de la caméra regroupe les primitives issues des caméras puis exécute le réseau de convolutions restant.

FIGURE 4.2 – La figure (a) décrit un système d'inférence classique constitué d'une caméra capturant une scène sous un unique point de vue et d'un accélérateur de CNN conventionnel. La figure (b) représente l'architecture du système proposé où plusieurs caméras embarquent un modèle de réseau dégradé tout en capturant un objet selon différents points de vues.

4.1 Modèles de calcul et optimisations sur FPGA

Les architectures de CNN actuelles sont très exigeantes en termes de ressources de calculs et de mémorisations. La première partie FCN d'un réseau appelle généralement

des millions d'opérations de convolutions tandis que la deuxième, FC, demande un espace de stockage de paramètres de plusieurs dizaines voire centaines de Méga-octets. L'inférence d'une couche de convolutions standard nécessite $W(i)$ opérations MAC comme décrit dans l'équation 4.2 :

$$\mathcal{W}(i) = N_{i-1} \times N_i \times k_i^2 \times U_i \times V_i \tag{4.1}$$

où N_{i-1} et N_i sont respectivement le nombre de primitives à l'entrée puis à la sortie de la i-ème couche de convolutions, U_i et V_i sont les dimensions des primitives en sortie et k_i la taille du noyau de convolution.À noter qu'entre chaque couche de convolutions, les cartes d'activations doivent être stockées dans un espace mémoire dédié M tel que :

$$\mathcal{M}(i) = N_i \times U_i \times V_i \tag{4.2}$$

La partie MLP est constituée de plusieurs couches FC et chacune d'elles peut être appréhendée comme une succession de multiplications de matrices telle que :

$$Y = W^T.X + b = \begin{pmatrix} w_1^1 & w_2^1 & ... & w_d^1 \\ w_1^2 & w_2^2 & ... & w_2^i \\ w_1^3 & w_2^3 & ... & w_3^i \\ ... & ... & ... & ... \\ w_1^i & w_2^i & ... & w_d^i \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_d \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ .. \\ b_d \end{pmatrix} \tag{4.3}$$

où d est la dimension du vecteur de primitives X de la dernière couche de convolution du réseau FCN et i le nombre de neurones de la couche FC. Le tableau 4.1 donne un aperçu du nombre d'opérations et de paramètres pour le réseau AlexNet [Kri12].

ALEXNET	Primitives d'entrée $[U_{i-1},V_{i-1},(N_{i-1},N_i),k_i,s_i,p_i]$	Charge de calcul (OPs)	nombre de paramètres
conv1 + ReLU	224,224,(3,64),11,4,2	70M	23.3K
Maxpool	55,55,64,64,3,2,0	-	-
conv2 + ReLU	27,27,(64,192),5,1,2	224M	307.2K
Maxpool	27,27,(192,192),3,2,0	-	-
conv3 + ReLU	13,13,(192,384),3,1,1	112M	663.9K
conv4 + ReLU	13,13,(384,256),3,1,1	149.5M	884K
conv5 + ReLU	13,13,(256,256),3,1,1	99.7M	590K
Maxpool	13,13,(256,256),3,2,0	-	-
FC1		37M	37M
FC2		16.7M	16.7M
FC3		4M	4M

TABLE 4.1 – Résumé du nombre d'opérations et de paramètres pour chaque couche d'AlexNet. Le réseau de convolutions ($1 \leq i \leq 5$) demande le plus d'opérations tandis que le réseau FC comporte le plus de paramètres. Les notations utilisées ici restent les mêmes que celles utilisées en 2.1.3. (U_{i-1}, V_{i-1}) est la taille des primitives d'entrées de la couche i, (N_{i-1},N_i) le nombre de canaux d'entrées puis de sorties, (k_i, s_i, p_i) représentent respectivement la taille des kernels de convolutions, le facteur de sous-échantillonage et le paramètre de remplissage.

A la vue du nombre d'opérations et de paramètres à stocker, le plus grand challenge reste de trouver des leviers efficaces pour déporter l'inférence de CNN directement sur des systèmes embarqués à base de FPGA. Deux principaux types d'architecture sont évoqués dans la littérature lorsque l'on parle d'inférence sur circuits à logique reprogrammable.

4.1.1 DHM

Ce schéma d'accélération permet d'exploiter le haut degré de parallélisme d'un FPGA par une instanciation directe de chacune des couches de convolutions à partir des cellules logiques [APS+17a]. La figure 4.3 illustre l'idée du DHM où une couche de convolution est entièrement déroulée sur les éléments logiques d'un FPGA, c'est à dire opérateurs MAC, fonctions d'activations RELU incluses.

Le nombre d'opérateurs MAC nécessaires , $\mathcal{N}_{MAC}$, est donné par l'équation 4.4.

$$\mathcal{N}_{MAC} = N_{i-1} \times N_i \times k_i \times k_i \tag{4.4}$$

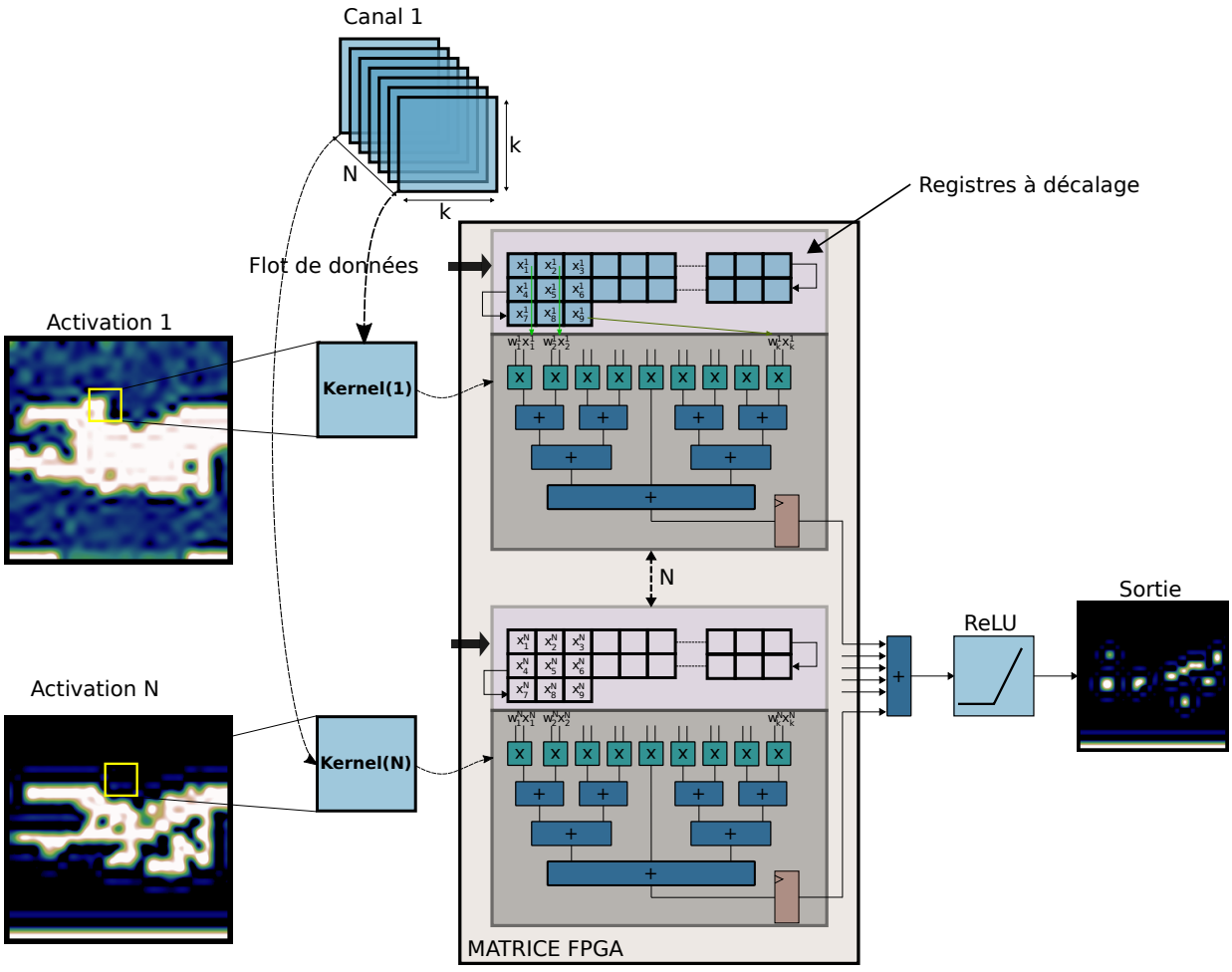


FIGURE 4.3 – La figure présente une architecture type flot de données matériel. Pour chaque couche et chaque canaux d'une même couche, les kernels de convolution sont synthétisés et placés sur la matrice du FPGA à partir d'éléments logiques. Les N_{i-1} activations remplissent les N_{i-1} registres à décalage jusqu'à former une fenêtre $k_i \times k_i$ (ici 3×3). Une donnée est calculée à chaque cycle d'horloge du système.

où i est l'indice de la couche du réseau de convolutions. Il est important de souligner que, dans ce cas précis, le nombre d'opérateurs est indépendant de la taille des primitives d'entrées. Le nombre de bits à mémoriser $\mathcal{N}_{MEM}$ est directement corrélé à la taille des noyaux de convolutions k_i, de la largeur des primitives (U_i) et du nombre N_i de primitives d'entrée (équation 4.5).

$$\mathcal{N}_{MEM}(i) = N_i \times [(k_i - 1) \times U_i + k_i] \tag{4.5}$$

Le temps, noté t_{conv}, requis par l'accélérateur de la couche d'indice i pour délivrer sa **première** donnée est donné par l'équation 4.6 :

$$t_{conv} = T_{clk} \times [(k_i - 1) \times U_{i-1} + k_i + \text{pipeline}_{stage} \tag{4.6}$$

où T_{clk} est la période d'horloge du système, U_{i-1} la largeur des données en entrée. Le terme pipeline_{stage} représente le délai en nombre de cycles d'horloge introduit par le nombre d'étage d'un opérateur MAC.

Cette méthode est la plus optimale dans le sens où :

- Les pixels peuvent être traités directement en sortie de capteur dès la première couche de convolution.
- Le temps d'inférence ne dépend que de la fréquence du capteur F_{sensor} (en général $F_{sensor} = F_{clk}$) plus la latence introduite par le pipeline de convolutions. Cette latence dépend de la profondeur du réseau, de la taille des noyaux de convolution et de la fréquence de fonctionnement F_{clk}
- Les primitives intermédiaires n'ont pas besoin d'être mémorisées dans une mémoire externe, supprimant ainsi les trajets entre la mémoire et les unités spécialisées d'une architecture où les ressources de calculs sont partagées.
- Les performances sont maximales en terme de densité de calcul (OPs/s). Pour un réseau de type AlexNet, cela représente la somme des débits de chaque couche de convolutions comme décrit par l'équation 4.7
- Chaque multiplication est dédiée à son propre paramètre de convolution. Le nombre de bits servant à coder le paramètre w_i^N et le nombre de bits codant une valeur de

la primitive d'entrée x_i^N définissent la taille du multiplieur logique sur la matrice du FPGA.

$$\text{Max}_{ops} = \sum_{i=1}^{D} \mathcal{N}_{MAC}(i) \times F_{clk} = \sum_{i=1}^{D} N_{i-1} \times N_i \times k_i^2 \times F_{clk} \tag{4.7}$$

Alexnet	L1	L2	L3	L4	L5
Multiplieurs (8bits)	23.3K	614K	885K	1327K	885K
Elements logiques (LEs)	800K	N.C	N.C	N.C	N.C

TABLE 4.2 – Synthèse des couches d'AlexNet suivant le modèle DHM

Ce type d'accélération purement matérielle est difficile à mettre en oeuvre sur les technologies de FPGA actuelles. La quantité de multiplieurs et d'éléments logiques nécessaires dépassent largement celles disponibles sur les FPGA destinés aux systèmes embarqués. Le tableau 4.2 illustre ce problème avec le réseau AlexNet. Dans cet exemple, il est important de noter que seule la première couche a pu être synthétisée par la suite Quartus avec un codage sur 8 bits pour les poids de convolutions et les données des primitives. Les implémentations des autres couches plus profondes n'ont tout simplement pas pu être synthétisées. A titre de comparaison, un FPGA embarqué **Cyclone III** dispose d'un maximum de 120K éléments logiques et jusqu'à 300K pour le modèle le plus dense (**Cyclone V GX**). Un système de vision embarqué ne peut donc pas supporter à lui seul un modèle de CNN avec ce schéma d'accélération à moins d'appliquer une méthode de compression adéquate comme celle vue dans le chapitre précédent 3.1.

4.1.2 Accélérateurs à multiplexage temporel

Ce type d'accélération consiste à distribuer toutes les opérations de convolutions sur un nombre limité de coeurs de calculs spécialisés notés PE. Les PE sont attachés à un bus de données sur lequel transitent les poids de convolutions et les données de chaque primitives comme présentés dans la figure 4.4. Un processeur ou plus généralement une machine à état (MAE) centrale séquence les tâches de convolutions avec ces unités spécialisées. Le CPU envoie des instructions à un DMA chargé de faire suivre les données et les paramètres de convolutions depuis la mémoire vers les PE. En général, le nombre de PE est dicté par la quantité de ressources logiques et de DSP disponibles sur un modèle de FPGA donné. Ce modèle de calcul est largement répandu dans le domaine du

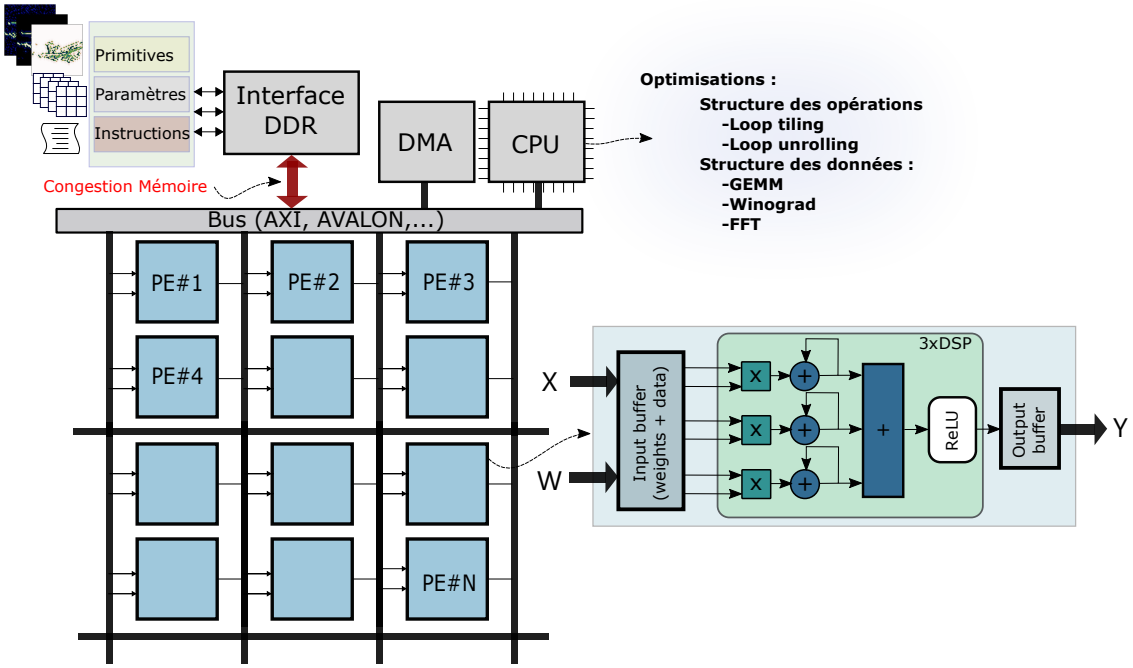


FIGURE 4.4 – Vue simplifiée d'une architecture à processeurs spécialisés. Un processeur central (**CPU**) et un moteur d'accès à la mémoire (**DMA**) suivent une séquence d'instructions pour alimenter les processeurs spécialisés (**PE**) avec les paramètres et les primitives intermédiaires propres à chaque couche d'un CNN. Les **PE** possèdent une ou plusieurs unités arithmétiques ainsi que des mémoires tampon d'entrées/sorties (Caches de niveau 1). Le flot basique consiste à alimenter les caches d'entrées avec les données situées dans la mémoire externe, procéder aux convolutions, enregistrer les données sur les caches de sortie puis de mémoriser ces résultats dans la mémoire externe.

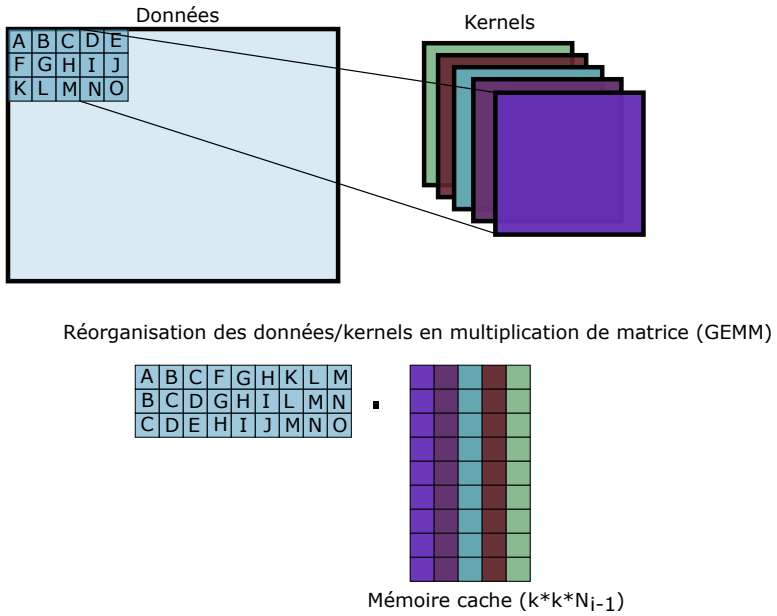


FIGURE 4.5 – Représentation de la réorganisation des opérations de convolution en multiplications de matrices. Cette technique permet de réduire le nombre de transferts avec la mémoire externe. Ici les paramètres d'un canal de convolutions sont stockés en mémoire cache (BRAM) au prix d'une redondance des primitives d'entrées.

Big Data [ZLS+15a, QWY+16, FOP+18], où les FPGA se démarquent des ASIC grâce à leur versatilité (via la reconfiguration), ou bien des GPU pour leur meilleur rapport opérations/watts [NSB+17] et une plus faible latence de calcul même sur les modèles de CNN les plus larges [YGW+19]. Les outils de conception rapides [VKB18a] et les nouveaux langages de synthèse haut niveau [KGL+, GLX+17, Wan17] (HLS ou OPENCL) dérivés du C++ permettent d'exploiter plus facilement les ressources des FPGA en diminuant le temps de développement et de déploiement comparé aux langages de synthèse bas niveau RTL. Même si ces accélérateurs sont spatialement optimaux dans le sens où ils exploitent toutes les ressources logiques du circuit, ils souffrent généralement d'une bande passante limitée et d'une latence non négligeable de l'interface mémoire [HZWK16, GZY+17]. Pour un ordre de lecture ou d'écriture, cette latence est typiquement de l'ordre de plusieurs dizaines de cycles d'horloge du bus de données. Des techniques d'optimisation du modèle de calcul prennent en compte les opportunités de parallélisme d'un réseau de convolutions en favorisant la localité des données afin

de réduire le nombre d'opérations d'entrée/sortie avec les mémoires externes. Les accélérateurs de CNN sur FPGA actuels atteignent des performances de l'ordre de plusieurs Téra-opérations par seconde [MCVS17, SFM17] sur les circuits les plus denses (familles Stratix ou Virtex). Ces optimisations sont pour la plupart listées dans plusieurs enquêtes [ABP18, SMC+, VKB18b, GZY+17, Mit18]. Les tactiques les plus courantes déployées sur cible FPGA pour ce modèle de calcul sont listées ci dessous.

- Les opérations de convolutions peuvent être réorganisées en multiplication de matrices GEMM. Certains travaux reportent une réduction du temps de calcul de 47% [CX14] sur les couches les plus denses d'un CNN de l'état de l'art. Pour un canal de convolution N_i, un FPGA doit être en mesure de contenir les N_{i-1} filtres de convolutions en mémoire cache pour implémenter cette technique comme décrit par la figure 4.5. L'autre problème vient du fait que les primitives d'entrées sont réorganisées en une matrice de Toepliz dans le sens où les données sont régulièrement dupliquées (modulo la taille du kernel de convolution) dans le cache d'entrée.
- La transformation de Winograd permet de réduire globalement le nombre de MACOPs jusqu'à un facteur de 2.25 [LG15] sur des filtres de convolutions de 3×3. Cette optimisation n'est possible que dans le cas où le facteur de sous-échantillonage *stride* est à 1. Cette technique est par exemple utilisée par l'accélérateur OpenCL *DLA* de *Aydonat et al.* [AOC+17] capable de supporter 1.3 TOPs de calculs.
- La DFT est une technique connue pour transformer une opération de convolution circulaire en une multiplication de matrice dans le domaine fréquentiel. La complexité d'une convolution 2D passe de $\mathcal{O}(k^2.U_i.V_i)$ à $\mathcal{O}(U_i.V_i.\log_2(U_i.V_i)$ [SMC+].
- La technique du *loop-tiling* divise les primitives en plusieurs sections pour maximiser la localité des données et donc diminuer la quantité d'opérations d'entrées/sorties avec la mémoire externe [ZLS+15b].
- Le *loop-unrolling* propose de dérouler partiellement une ou plusieurs des boucles de l'algorithme décrivant une couche de convolutions pour accélérer le processus d'inférence [SFM17]. Le déroulement peut se faire au niveau du noyau du filtre, d'un canal de convolution voire de plusieurs canaux en parallèle. *Ma et al.* [MCVS17] combine cette d'optimisation au *loop-tiling* pour atteindre plus 600 GOPs sur un FPGA Arria10.

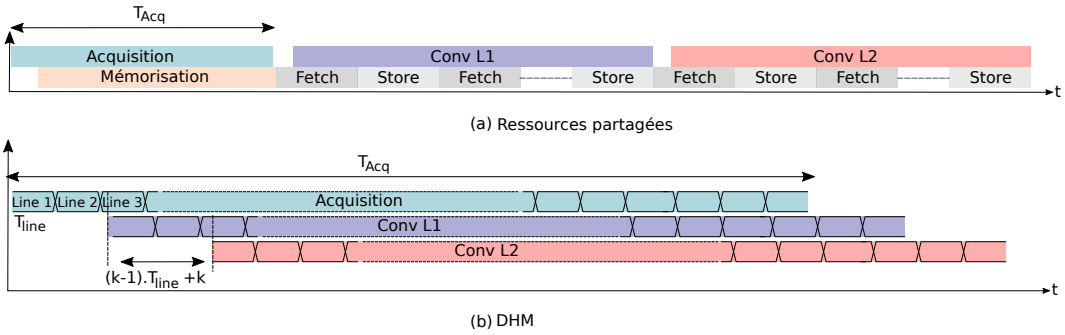


FIGURE 4.6 – Le chronogramme (a) montre un système d'inférence conventionnel porté sur une architecture à partage de ressources de calculs. Une caméra capture l'image dans un laps de temps T_{acq} qui est stockée dans la mémoire de l'accélérateur. Le CPU séquence la distribution des poids et des primitives aux PE. Le chronogramme du bas (b) décrit la séquence dans une architecture matérielle orientée flots de données. Tous les kernels de convolutions sont directement implémentés dans le circuit, par conséquent, il n'existe aucun temps mort entre la disponibilité des données et leurs traitements.

Toutes les optimisations listées ci-dessus restent limitées par les échanges de données entre la mémoire externe et les coeurs de calculs. Une tactique DHM n'est pas contrainte par ces mouvements puisque le données Les chronogrammes de la figure 4.6 illustrent les différences entre la tactique DHM et la tactique du partage de ressources.

D'autres optimisations peuvent améliorer les performances des deux techniques d'accélération évoquées. Le pruning et la quantification font partie des candidats les plus souvent évoqués dans littérature.

Intel FPGA	fp32	18x18	fmax (MHz)	LEs
Cyclone V (GX,E,GT)	342	684	287	300k
Cyclone 10 GX	192 (134GFLOPS)	384	456	220k
Arria 10 GX	1518 (1,366TFLOPS)	3036	635	1150k
Stratix 10	5760 (9.2TFLOPS)	11520	1000	10200k

TABLE 4.3 – Ressources (DSP) maximales disponibles sur les plateformes FPGA Intel. Les caractéristiques sont celles reportées par le constructeur.

La quantification des poids de convolutions et des valeurs de primitives représente un aspect important pour une accélération efficace des CNN. L'impact de l'encodage est encore plus proéminent pour une implémentation sur FPGA selon le modèle DHM. Dans un réseau de convolutions standard, les paramètres sont nativement codés avec une représentation de 32 bits flottants (IEEE 754) et nécessitent des DSP compatibles pour procéder aux opérations de convolutions ou de multiplications de matrices. Sur des architectures FPGA dédiés à l'embarqué, les DSP ne sont cependant disponibles qu'en faible quantité et sont relativement peu rapides (tableau 4.3) comparativement aux architectures conventionnelles de type GPU.

Un FPGA Cyclone10GX ne peut éxecuter que 134 GOPs en virgule flottante simple précision avec 192 DSP. A contrario, les circuits graphiques GPU sont parfaitement adaptés puisque leurs architectures sont basées sur un emploi massif de DSP (coeur CUDA) fonctionnant au delà du GHz pour, à l'origine, exécuter des transformations géométriques en temps réel et à grande échelle (images haute résolution). Par conséquent, une architecture GPU peut supporter plusieurs dizaines de Téra-Opérations par seconde en virgule flottante. La quantification des données est donc une solution idéale pour implémenter un moteur d'inférence sur cible FPGA puisque les éléments logiques peuvent être reconfigurés comme des opérateurs multiplieur-accumulateurs. Par exemple, pour une quantification d'une donnée sur $N_{q(w)}$ [1] bits, le nombre de transactions mémoires est théoriquement réduit d'un facteur $\frac{N_{q(w)}}{N_{f(w)}}$ [2] tandis que les opérateurs MAC peuvent être

1. q(w) est la version quantifiée des poids de convolution
2. f(w) est la version en virgule flottante des poids de convolution

implémentés à l'aide de multiplieurs logiques $N_{q(w)} \times N_{a(w)}$ [3], unités présentes en plus grand nombre sur la matrice d'un FPGA. Dans le cas d'une inférence sur FPGA, les *dsp* 32 bits sont divisibles en 2 modules *dsp* 16 bits. Les centres de données Microsoft *Azure* utilisent des FPGA Intel Arria 10 et Stratix 10 avec une représentation en 8 bits flottants pour supporter l'inférence de tout type de DNN [FOP+18] et sont capables de délivrer des performances temps réel aux utilisateurs. Par exemple, *Lai et al.* [LSC17] ont proposé de préserver les poids de convolution codés en 32 bits flottants mais en réduisant la précision des activations. Leurs résultats montrent qu'il est possible de compresser les données d'activation de 32 bits à 8 bits sur VGG-16, à 7 bits sur SqueezeNet voire 5 bits avec le réseau AlexNet pour un taux de reconnaissance équivalent. L'exploration de *Kouris et al.* [KVB18] démontre qu'un CNN comme AlexNet accepte une résolution de 8 bits sur les poids et les primitives (figure 4.7) sans accuser de dégradation sur la qualité de reconnaissance. *Banner et al.* démontrent cette même propriété en [BHHS18] et plus tard sur 4 bits [BNHS18]. Cette dernière méthode ne quantifie pas strictement chaque paramètre d'un réseau avec une même résolution mais analyse canaux par canaux la distribution des valeurs d'activation pour obtenir une moyenne de 4 bits. Cette attention portée sur les variations existantes entre les canaux d'une même couche diminue fortement les dégradations de performance de classification comparé à une implémentation naïve des poids et des données sur 4 bits (figure 4.8). La quantification peut être orientée pour une cible matérielle particulière. *McDanel et al.* [MZKD19] propose une architecture de CNN ne supportant que des poids encodés par des valeurs en puissance de 2. Pour des paramètres de convolutions codés sur 4 bits, leur réseau construit pour résoudre CIFAR-10 [Kri09] est seulement 3% moins précis que son équivalent invoquant des DSP 32 bits mais accuse 7% de dégradation de classification sur un sous-ensemble d'ImageNet.

Des schémas de quantification extrêmes sur 2 bits ou 1 bit montrent de fortes dégradations [CDB15, RORF16] en termes de performances de classification car elles réduisent la dynamique des activations. Une solution consiste à multiplier le nombre de ces primitives pour contrecarrer les pertes de dynamique comme expliqué dans [MNCM17] où des réseaux de l'état de l'art tels que AlexNet ou ResNet acceptent des paramètres codés sur 2 bits et 4 bits pour les primitives avec, toutefois 2× plus de canaux de convolutions pour

3. a(w) est la version quantifiée des primitives (activations)

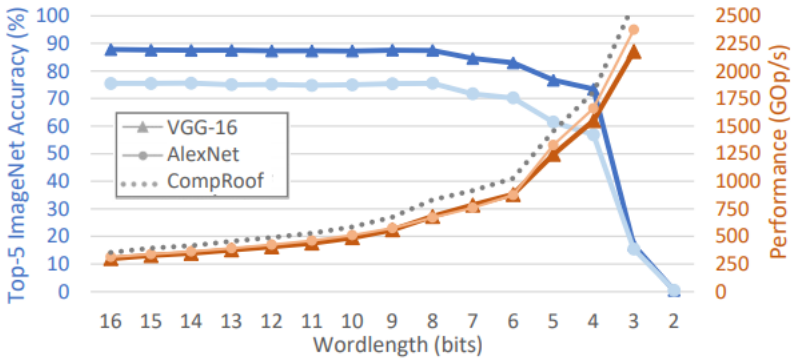


FIGURE 4.7 – Les données quantifiées sur 8 bits ne dégradent pas les performances d'un réseau comme AlexNet ou VGG. De plus, la vitesse d'exécution sur cible FPGA est multipliée par 3 avec une architecture SIMD. Source : [KVB18]

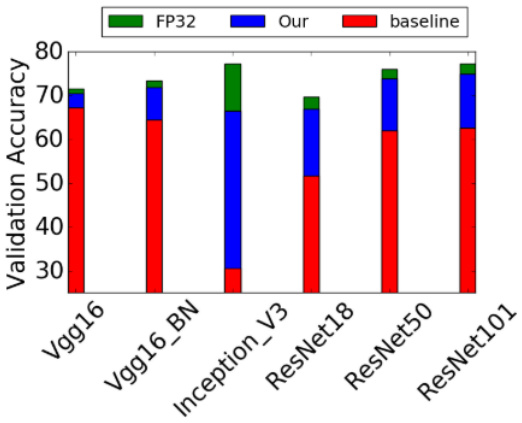


FIGURE 4.8 – Résultats de quantification sur 4 bits post-entraînement de *Banner et al.* source : [BNHS18]. Les dégradations de performances de classification sont largement compensées par la méthode analytique proposée par les auteurs.

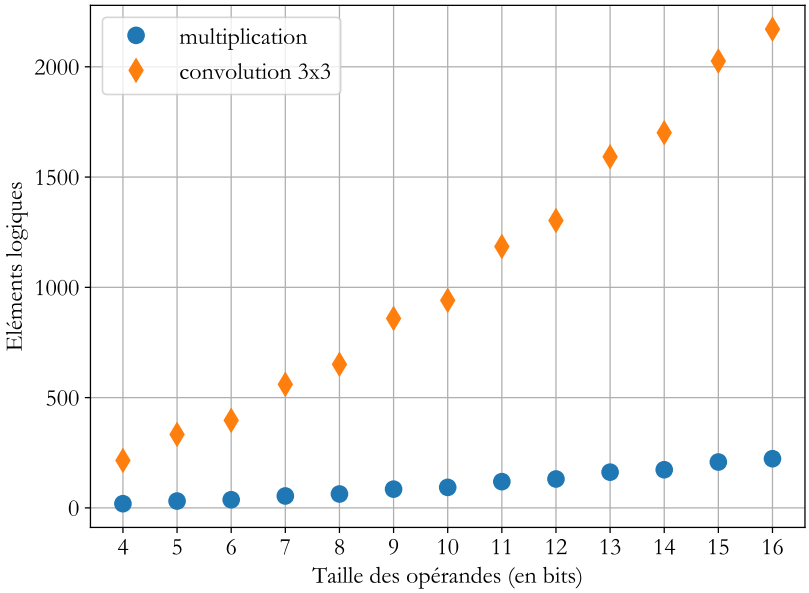


FIGURE 4.9 – Empreinte matérielle d'une multiplication NxN bits puis d'une convolution 3x3.

chacune des couche composant ces réseaux. Leurs expériences montrent que cette résolution ne dégrade que de 3% le taux de classification de ces deux réseaux en comparaison à leurs équivalents codés en 32 bits flottants.

Comme démontré dans la littérature, une inférence uniquement composée de valeurs codées sur 8 bits est directe dans le sens où il n'existe pas de pertes de taux de classification comparé à un codage 32 bits conventionnel à nombre de canaux de convolutions constant. Cette observation est essentielle sur FPGA et une architecture DHM. Comme illustré par la figure 4.9, la densité de convolueurs 3×3 8 bits est 3 fois supérieure à la densité de convolueurs 16 bits.

4.1.3 Adaptation entre pruning et systèmes d'inférence sur FPGA

Comme expliqué dans la section 4.1, les architectures à ressources partagées sont tributaires des temps d'accès mémoires, du nombre de PE disponibles et de l'efficacité du système d'ordonnancement des opérations. Ainsi, une compression structurée affiche un meilleur potentiel pour réduire les opérations d'entrée/sortie qu'un pruning non-structuré comme expliqué dans la figure 4.10. La nature aléatoire de l'organisation des paramètres du *pruning* non-structuré reste difficile à mettre en oeuvre dans le cadre d'une accélération matérielle. La plupart des travaux misent avant tout à réduire l'empreinte

mémoire d'un modèle de CNN. Cependant, les éléments non-nuls doivent être indexés dans des vecteurs spéciaux (COO, CSR et CSC) contenant leurs coordonnées et annulant potentiellement le gain de stockage recherché initialement. Même si des algorithmes optimisés [DBN+20] sont susceptibles d'améliorer les temps d'exécution sur des machines standards comme les CPU ou GPU, des architectures d'accélérateurs doivent être dédiées à l'une ou l'autre des méthodes de compression.

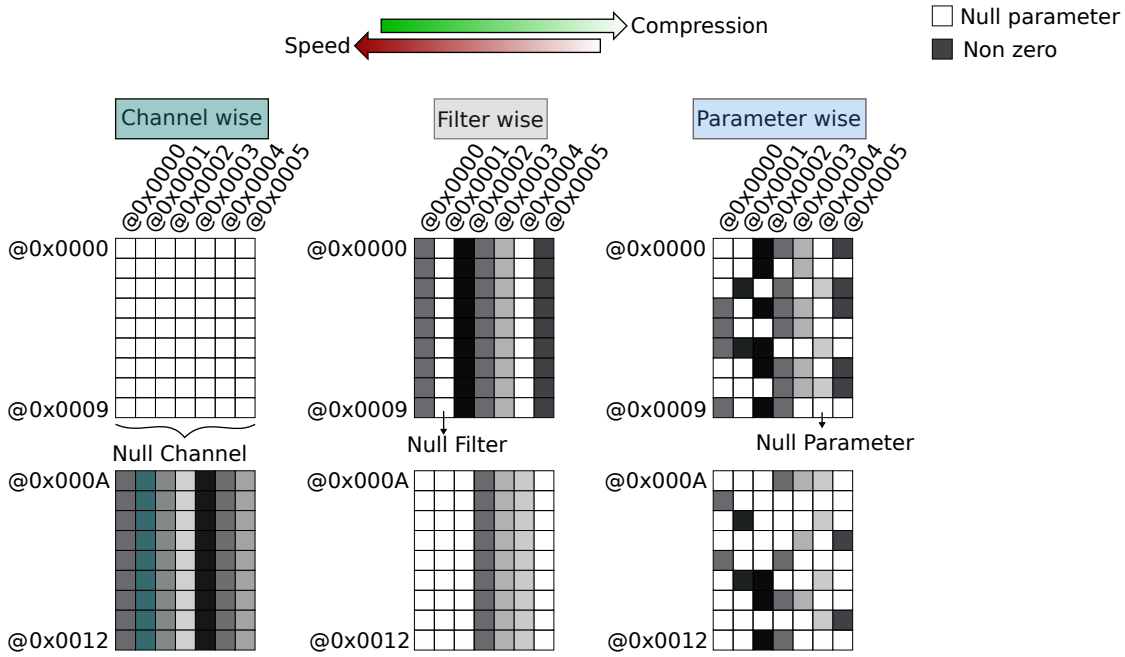


FIGURE 4.10 – En théorie, plus un algorithme de compression agit avec une granularité forte plus grande sera l'accélération. Au contraire, les matrices creuses (*sparse*) sont difficiles à exploiter puisque les paramètres non-nuls ne sont pas organisés selon un motif régulier. La figure ci-dessus illustre l'organisation en matrice des filtres 3×3 d'un canal de convolutions $N_{i-1} \times k_i \times k_i$=(6,3,3) depuis un modèle de compression structuré (à gauche) jusqu'à un modèle non-structuré (à droite).

4.1.4 Accélération et pruning structuré

Les moteurs d'inférence sur FPGA tirant bénéfice du pruning structuré occupent la majorité des efforts de recherches. L'accélérateur de *Lu et al.* [LXH+19] repose sur le principe présenté par d'*Abbasi et al.* [AAY17]. Les auteurs utilisent les statistiques sur les activations collectées en amont de l'inférence pour éviter les calculs inutiles sur la totalité d'un canal de convolution. Le système d'accélération *Cambricorn-S* [ZDG+18] est en réalité une suite combinant un algorithme de *pruning* structuré hors-ligne mais dont le motif de compression est spécialement adapté à leur moteur d'inférence sur FPGA. L'idée est

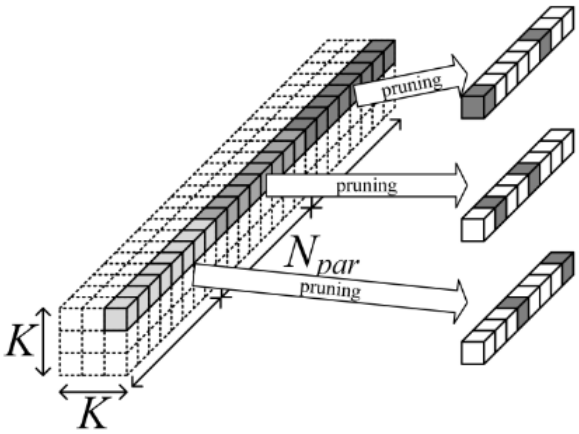


FIGURE 4.11 – Motif de compression imprimé le long d'un axe prédéfini, ici suivant la position (x,y) = (0,2) d'un paramètre des kernels de convolution. Source : [Kan19]

d'imposer un pas de compression, noté N_{par}, sur un ensemble des filtres 2D de chacun des canaux pour induire une répétition sur le motif de compression. *Kang et al.* [Kan19] utilise une technique similaire mais selon un grain plus fin. Cette fois, ce sont les paramètres individuels d'un filtre qui sont ciblés mais le long d'un canal de convolution comme illustré par la figure 4.11.

4.1.5 Accélération et pruning non-structuré

Les accélérateurs tirant parti du pruning non-structuré sont très peu représentés dans la littérature puisqu'il est difficile d'en tirer bénéfice en termes de vitesse d'exécution. *Han et al.* [HLM+16] ont dessiné un accélérateur sur ASIC spécialement conçu pour les produits de matrices creuses mais orienté sur la partie FC d'un CNN avec un débit de données 3× supérieur à un accélérateur conventionnel. Le moteur de *Zhu et al* [ZH19] prend avantage de l'irrégularité de la compression non-structuré en désactivant les PE lorsqu'un poids de convolution est nul afin d'optimiser la puissance consommée d'un circuit. Le grand avantage du modèle DHM réside dans son immunité aux motifs de compression. Quelque soit la méthodologie de *pruning* employée, la retranscription matérielle d'une couche de convolutions est uniquement dictée par le nombre de paramètres non-nuls, leur résolution en bits ainsi que la résolution en bits des activations. Par exemple,

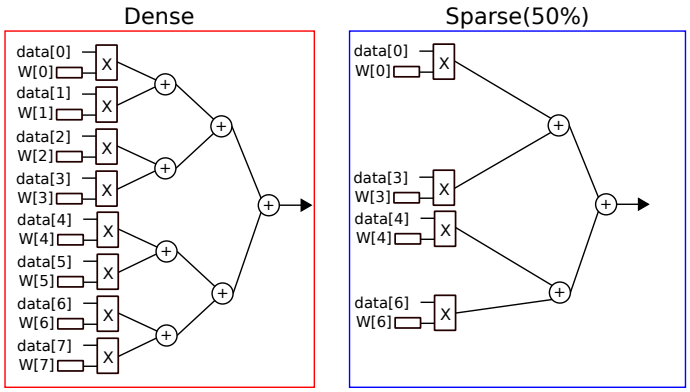


FIGURE 4.12 – Seul les paramètres non-nuls appellent un opérateur MAC spécialisé dans la matrice d'un FPGA. En d'autres termes, l'irrégularité du pruning non-structuré n'a aucune incidence sur le comportement de l'accélérateur.

les ressources matérielles d'un noyau de convolution noté $\mathcal{R}_F$ de taille k^2 appelle $\mathcal{R}_x$ ressources pour un multiplieur et $\mathcal{R}_+$ ressources matérielles pour un additionneur tel que :

$$\mathcal{R}_F = \alpha \times k \times k \times \mathcal{R}_\times(n,n) \tag{4.8}$$

$$\mathcal{R}_+ = ((\alpha \times (k \times k)) - 1)\mathcal{R}_+(2n,2n) \tag{4.9}$$

avec α ($0 < \alpha \leq 1$) un facteur de compression obtenu après *pruning* et n la résolution en bits des poids de convolution. La figure 4.12 illustre cet exemple avec $\alpha = 50\%$. Selon les résultats affichés du tableau 3.1, une méthode non-structurée est la mieux adaptée à une accélération purement matérielle puisque les taux de compression affichés sont supérieurs à ceux des méthodes structurées comme il a été montré dans le chapitre précédent.

Les FPGA sont de solides alternatives pour inférer un CNN. Les outils de conceptions haut niveau ont permis de populariser l'usage des FPGA auprès des développeurs en évinçant la complexité de l'étape de design RTL. Malgré ces avantages comparé aux autres plateformes, tous les modèles de calcul présentés sont inadaptés aux modèles de FPGA plus modestes. La densité de portes logiques, la fréquence de fonctionnement ou encore les performances des interfaces mémoires sur ce type de circuit ne permettent pas d'atteindre des performances temps réels. Dans le premier cas, les ressources de calculs sont partagées entre les différentes couches du réseau et la fréquence des accès mémoires est prohibitive en termes de temps et de consommation d'énergie. Le deuxième cas résout ces problèmes mais reste inexploitable sur les technologies actuelles. La prochaine section montre qu'un contexte multi-vues peut contribuer à la combinaison des deux

schémas d'inférence sur FPGA pour réduire le temps de calcul.

4.2 Contribution : Système d'inférence mixte distribué sur réseau de caméra intelligentes

Le chapitre présentait les CNN multi-vues comme des alternatives capable d'augmenter le taux de reconnaissance. Chacun de ces modèles impliquent une charge de calcul plus élevée dans le sens où le nombre de vues duplique V fois la charge initiale notée $\mathcal{W}_{NET_{sv}}$. Toutefois, le simple terme $\mathcal{W}_{NET_{sv}}$ implique déjà un nombre important de convolutions et reste impossible à mettre en oeuvre sur moteur d'inférence de type DHM comme expliqué précédemment à travers le tableau 4.2. Implémenter le modèle flots de données sur FPGA oblige à diminuer le nombre de convolutions avec pour conséquence de dégrader les performances de classification. La première idée consiste à exécuter un modèle multi-vues similaire à celui introduit en [SMKLm15a] dans un environnement matériel composé de plusieurs caméras intelligentes non pas pour augmenter le taux de reconnaissance du modèle d'origine mais pour augmenter le taux de reconnaissance d'un nouveau réseau basé sur NET_{sv} dégradé et optimisé pour intégrer le modèle flots de données. Un modèle de caméra peut être dérivé de cette proposition où une partie *front-end* composé d'un ensemble de V caméras embarquent une partie des convolutions en DHM tandis qu'une partie *back-end* concentrent ces données pour exécuter le reste de l'inférence par une accélération conventionnelle.

4.2.1 Modèle de calcul avec MVCNN

Un système d'inférence classique se décompose généralement en 2 parties, une caméra puis un accélérateur supportant un réseau FCN noté H et un réseau FC comme décrit dans l'équation 4.10.

$$NET_{sv} = \underbrace{FC \circ H}_{pe}(X) \tag{4.10}$$

Le modèle multi-vues de *Su et al.* étend ce modèle avec V vues d'un même objet où chaque point de vue est traité par son propre réseau H de manière concurrente et la section G représente la fonction *view-pooling* regroupant toutes les V vecteurs de primitives, comme montré dans la figure 4.13). Ce modèle peut être formulé par l'équation 4.11 :

$$NET_{MVCNN} = FC \circ G \circ H_v(\text{input}(X_v)), \qquad 1 < v < V \tag{4.11}$$

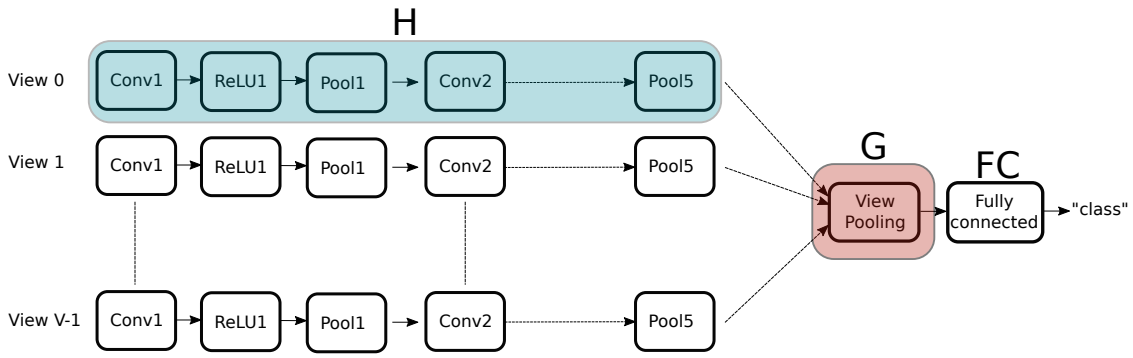


FIGURE 4.13 – La configuration multi-vue introduite par *Su et al.*

La méthode retenue ici, consiste à introduire un paramètre m ($m < M$)[4] qui divise le réseau multi-vues en 2 sous réseaux. La partie *front-end* F^m implémente les m premières couches de convolutions et la partie *back-end* H^{M-m+1} supporte les convolutions restantes. Ce modèle est décrit par la formule 4.12

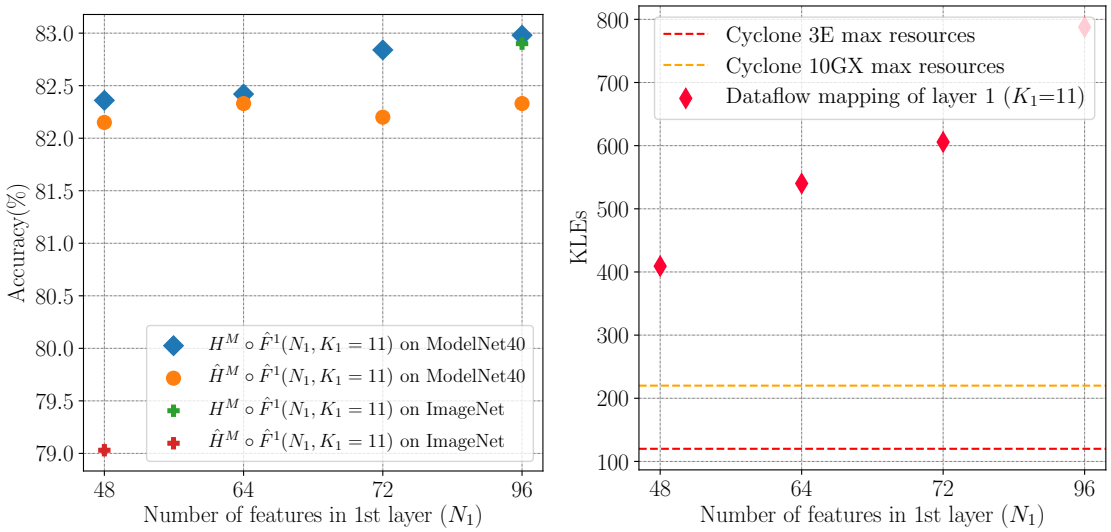
$$NET_{sv} = FC \circ H^{M-m+1} \circ F^m \tag{4.12}$$

En appliquant la solution de *Su et al.* et la fonction G (*view-pooling*), 4.12 devient :

$$NET_{MVCNN} = FC \circ H^{M-m+1} \circ G \circ F_v^m \quad \text{with} \quad 1 < v < V \tag{4.13}$$

La partie *front-end* F_v^m est distribuée sur les V caméras du système multi-vues avec un modèle de calcul DHM. Comme toutes les convolutions de chaque couche sont exécutées en parallèle, il n'existe aucun surcoût en termes de temps de calculs tandis que la partie *back-end* reste similaire à celle retrouvée dans un système mono-vue classique. Cependant, plus grand sera le paramètre m, plus cela aura d'impact sur la quantité de ressources matérielles pour les têtes de caméras embarquant le modèle flots de données. Les têtes de caméra sont basées sur des FPGA *low-cost* pour lesquelles une valeur de m telle que $m = 1$ est déjà trop importante en termes de ressources. Pour répondre à cette problématique, une solution consiste à réduire le nombre de paramètres en ajustant les termes N_i et k_i ($i \leq M$)[5]. Les sous-réseaux résultants, $\hat{F}^m$ et $\hat{H}^{M-m+1}$, comportent moins de paramètres mais nécessitent un levier de compensation pour ajuster le taux de classification à un niveau supérieur, typiquement celui du réseau mono-vue.

4. Ici, M représente le nombre de couches de convolutions
5. voir équation 4.4



(A) (B)

FIGURE 4.14 – (a) Les résultats de classification des 2 réseaux *back-end* (couche 2 à 5) et 4 réglages du *front-end* differents (N_1). (b) Les résultats de synthèse de la partie *front-end* selon le modèle flot de données.

Le reste de l'étude utilise le réseau AlexNet ($m = 1$) comme preuve de concept de la solution. Le nombre de paramètres du CNN est abaissé graduellement d'un facteur α pour intégrer la partie *front-end* du réseau dans les têtes de caméras à base de FPGA.Le coût de cette première couche de convolutions est donnée en 4.14.

$$\text{Cost}_{hw}(\hat{F}^1) = \alpha \times N_0 \times N_1 \times k_1^2 \ll \text{Cost}_{hw}(F^1), \forall \alpha \in]0;1[\tag{4.14}$$

avec N_0 le nombre de canaux pour une image d'entrée au format RGB.

4.2.2 Experiences sur le nombre de primitives

Comme pour le chapitre précédent, chaque objet de ModelNet40 est rendu par un ensemble de V caméras ($V \in [2;8]$) réparties uniformément autour de son centre de gravité et un roulis $\theta_r = \frac{\pi}{6}$. Toutes les variantes du CNN sont entraînées sur pytorch [PGC17] pendant 60 itérations. Le taux d'apprentissage est fixé à 10^{-4} sur les 30 premières itérations puis 5.10^{-5}. Les premières comparaisons sont effectuées avec plusieurs configurations entre le réseau dense NET_{sv}, une version dégradée NET_{sv} pour $m = 1$. Sur la figure 4.14(a), le nombre de canaux de la première couche de convolutions de la partie *front-end* est diminuée d'un facteur α ($\alpha \in [\frac{3}{4}; \frac{2}{3}; \frac{1}{2}]$). En parallèle, les paramètres du *back-end* sont réglés selon deux modes. Le premier, c'est-à-dire H, a le même nombre de canaux

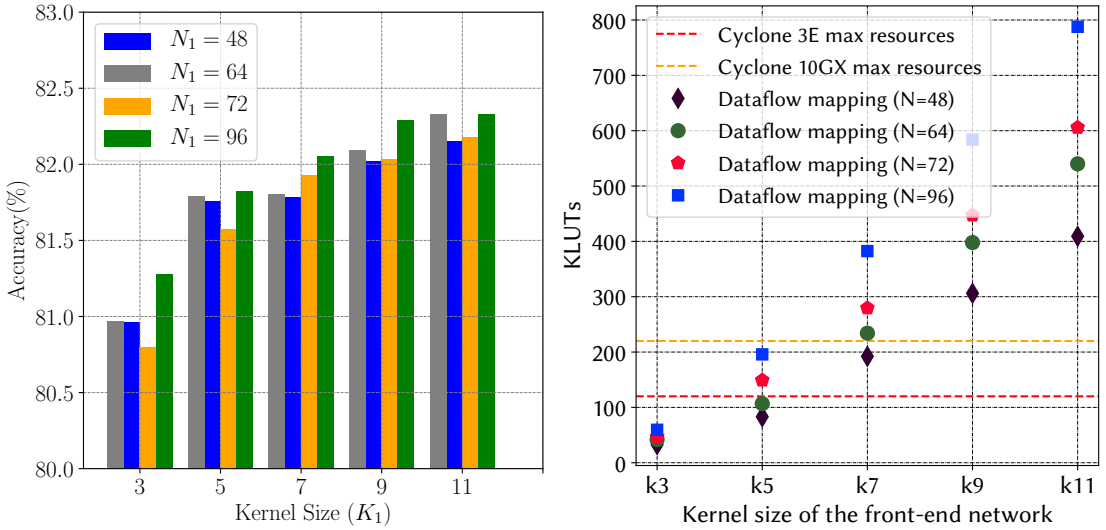
que celui du réseau de base tel que $(N_2, N_3, N_4, N_5) = (256, 384, 384, 256)$ et $\hat{H}$ $\hat{H}^D$ en possède la moitié $(128, 192, 192, 128)$. Les capacités de reconnaissance du deuxième modèle $\hat{H}$ sature à 82%. Ceci est du en partie au faible nombre de primitives haut niveau disponibles pour décrire l'objet. A titre de comparaison le modèle complet parvient à extraire les informations pour $k_1 > 64$. Le dernier modèle servira de base pour démontrer que la diversité des informations à l'entrée du système, c'est-à-dire celles apportées par un contexte multi-vues, peut servir de levier de compensation. La figure 4.14(a) soulève aussi l'importance du nombre de paramètres pour résoudre un dataset plus diversifié comme ImageNet. Ici l'écart de classification atteint 4% entre le réseau $H \circ F$ et $\hat{H} \circ \hat{F}$. Le réseau $\hat{F}^1$ est synthétisé avec Quartus et souligne la difficulté d'implémentation du modèle d'accélération DHM. Comme prévu, le nombre d'éléments logiques croît linéairement avec le nombre de primitives. Les poids et des primitives sont quantifiés sur 8 bits avec le logiciel *Delirium* [APS+17b]. Le nombre de multiplieurs synthétisés excède clairement les capacités du support d'inférence (figure 4.14(b)).

4.2.3 Experiences sur la taille des kernels

D'après les précédentes observations, le modèle de ressources décrit par l'équation 4.4 prévoit que le nombre d'éléments logiques varie en fonction de k_1^2. Le deuxième modèle de réseau est de nouveau entraîné avec un facteur de dégradation sur la taille des kernels de la première couche tel que $k_1 = [3; 5; 7; 9]$. Pour maintenir la compatibilité entre les primitives de sorties de $\hat{F}^1$ et le sous réseau $\hat{H}$, les images du dataset sont redimensionnées suivant l'équation 4.15 telles que $U_{in} = [219; 221; 223; 225]$.

$$U_{out} = \frac{U_{in} + 2 \times p - K}{s} + 1 \tag{4.15}$$

où p et s représentent les termes de remplissage de bord et de sous-échantillonage. Les résultats d'entraînement suivant différents couples $[k_1, N_1]$ sont illustrés dans la figure 4.15 (A). L'empreinte matérielle des réseaux $k \in [3; 5]$ est conforme aux capacités des technologies de FPGA embarqués actuels. Par exemple un FPGA Cyclone III ou Cyclone 10 contiennent assez d'éléments logiques pour embarquer la première couche du réseau optimisé. Les binômes retenues pour les explorations suivantes sont ceux dont le ratio entre la perte du taux de classification et le nombre de ressources matérielles appelées



(A) (B)

FIGURE 4.15 – (a) Exploration du couple (k_1, N_1) sur le taux de reconnaissance suivant la taille des kernels de convolution de la 1ere couche d'AlexNet pour $\hat{H}$(N = [128, 192, 192, 128]). La figure (b) donne les résultats de synthèse de 5 tailes de kernel de convolutions différentes selon le modèle flots de données matériel.

est minimale. Ce ratio est définit comme un index d'efficacité matériel HE_{index} décrit par l'équation 4.16. La dernière étape consiste à introduire le modèle MVCNN pour compenser les pertes de classification dues aux précédentes dégradations structurelles.

$$\text{HE}_{index} = \frac{\text{Cost}_{\text{hw}}(H^M, F^1) - \text{Cost}_{\text{hw}}(\hat{H}^M, \hat{F}^1)}{\text{Acc}(H^M, F^1) - \text{Acc}(\hat{H}^M, \hat{F}^1)} \times \frac{\text{Acc}(H^M, F^1)}{\text{Cost}_{\text{hw}}(H^M, F^1)} \tag{4.16}$$

4.2.4 Compensation multi-vues

Les réseaux $\hat{H}^M \circ \hat{F}^1$ ($k_1 \leq 7$,) sont ré-entrainés sur ModelNet40 avec les paramètres de convolutions des CNN multi-vues initialisés avec ceux entraînés de la section précédente. L'ensemble des réseaux $\hat{F}^1$ partagent les mêmes paramètres de convolutions. La fonction de regroupement *view-pooling* G regroupent les V sous réseaux $\hat{F}^1$ Les expériences sont menées avec 2 et jusqu'à 6 vues de chaque objet contenu dans le dataset. Dans ce cas, l'index de performance est redéfini en fonction du nombre de vues tel que :

$$\text{MVNHE}|_{\text{Acc}\simeq\text{Acc}_{AlexNet}} = \frac{\text{Cost}_{\text{hw}}(H^M, F^1) - \text{Cost}_{\text{hw}}(\hat{H}^M, \hat{F}^1)}{\text{Cost}_{\text{hw}}(H^M, F^1) \times V} \tag{4.17}$$

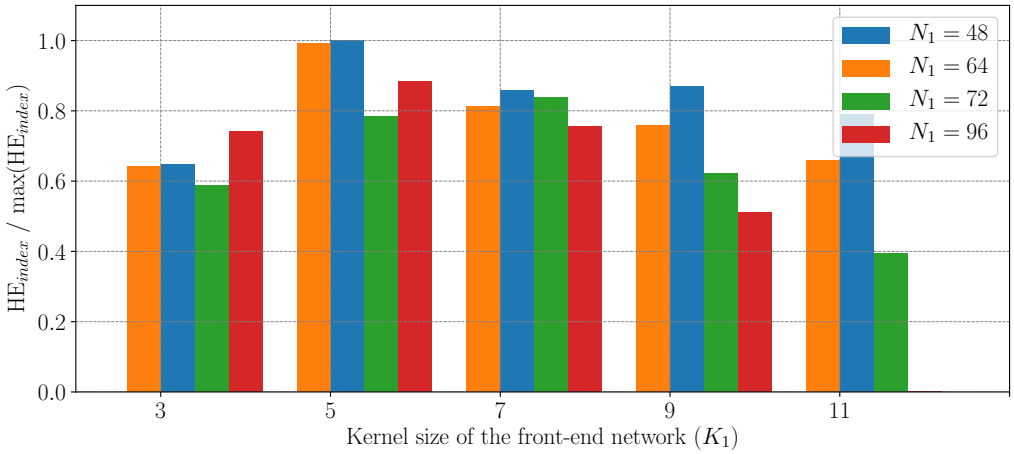


FIGURE 4.16 – L'index d'efficacité matérielle de la première couche du réseau $\hat{F}^1$. Toutes les valeurs de la figure sont normalisées par rapport au meilleur compromis trouvés expérimentalement ($N_1 = 48, K_1 = 5$). Un index HE élevé signifie que le réseau compense mieux les pertes de classification par rapport au coût matériel demandé

En d'autres termes, le réglage nécessitant le moins de vues aura un index élevé, tant que les performances de classifications restent semblables à celle du réseau mono-vue (équation 4.17). La majorité des systèmes multi-vues sont capables de compenser les pertes de classification dues aux différentes étapes de dégradations. La figure 4.17 montre que le système à 2 vues pour $[k_1, N_1] = [5, 48]$ est le plus efficace en terme de coût matériel, c'est-à-dire en nombre de FPGA (ou têtes de caméras), d'éléments logiques et de capacité de compensation.

4.2.5 Analyse d'implémentation

Un système de prétraitement sur caméra intelligente multi-vues doit contenir la gestion des communications, la gestion du capteur d'image et le réseau de convolutions $\hat{F}^1$ comme montré dans la figure 4.18. La partie communication est gérée par une interface Ethernet Gigabit intégrant une pile UDP/IP. La bande passante nécessaire (équation 4.18) pour transférer les données vers $\hat{H}$ dépend de la quantification des données, du nombre de primitives issues de $\hat{F}^1$ et du nombre d'images par seconde en sortie du capteur.

$$BW = N_i \times W_i \times H_i \times \text{bitwidth} \times \text{fps}\ Mb.s^{-1} \tag{4.18}$$

où, pour rappel, N_i est le nombre de primitives de taille (H_i, W_i) en sortie de la i-ème couche du réseau, le terme *bitwidth* est le nombre de bits servant à coder une valeur de

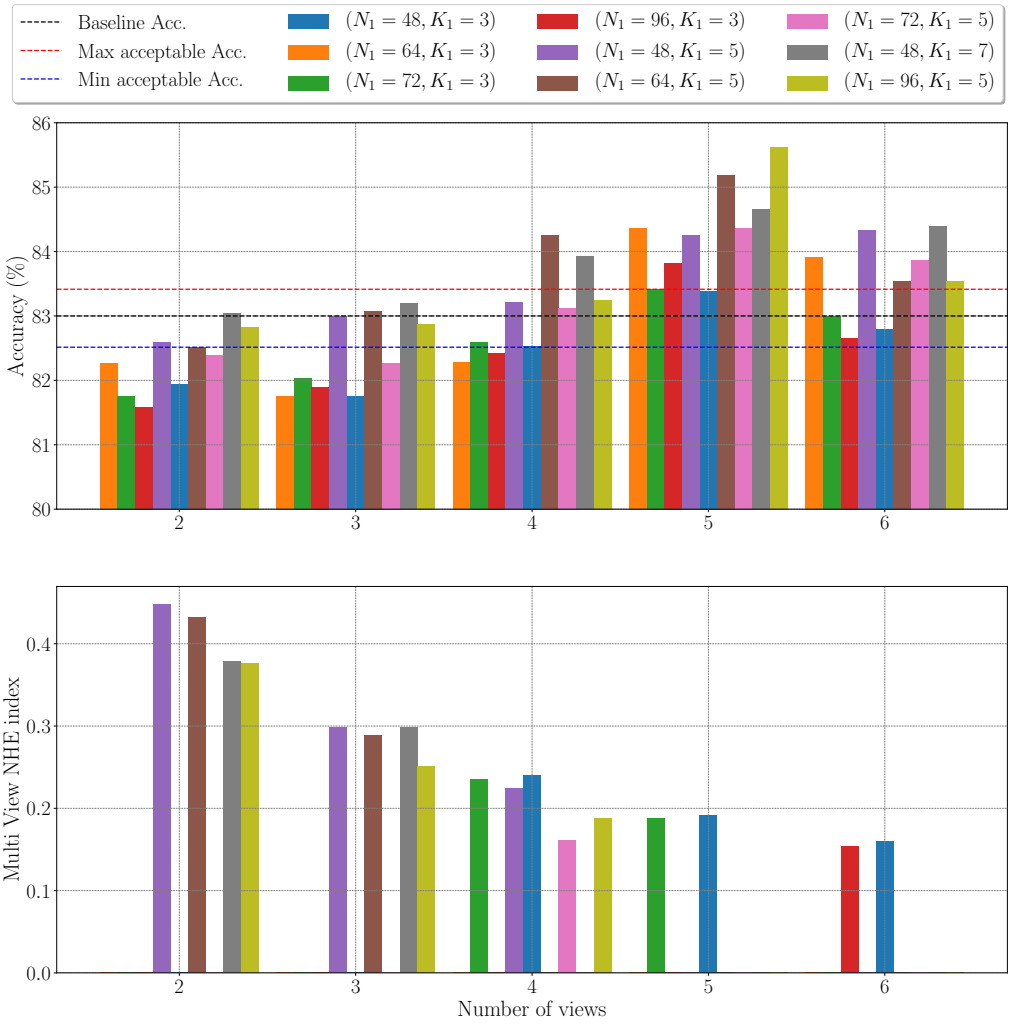


FIGURE 4.17 – La figure du haut reporte les taux de classification pour $v \in [2;6]$ et leur index de performance multi-vues. Ici, seuls les réseaux multi-vues dont les performances Acc_{mv} sont proches à +/- 0.5% du réseau initial sont considérés.

primitive et *fps* est le nombre d'images par seconde que le couple {capteur-fpga} est capable de traîter. Le tableau 4.4 reporte les deux meilleurs configurations Cfg1 et Cfg2 trouvés à partir des résultats la section précédente. La première est destinée à un système de 4 caméras basées sur des FPGA Cyclone III 120KLEs et la deuxième à un système de 2 caméras Cyclone10GX220. Les deux architectures sont capables de fonctionner à 53 MHz, fréquence de fonctionnement du capteur avec des images en 640x480 à 30 images par seconde. Dans le cas où le nombre de primitives de $\hat{F}^1$ est fixé à $N_1 = 48$, la bande passante du système est de $48 \times 27 \times 27 = 34$ KBits à comparer aux 8.2 Mbits de l'image en sortie de capteur. En considérant un accélérateur à PE couplé à une caméra standard, l'architecture présentée réduit le débit de données d'un facteur 6x si la sortie du capteur est réglée en QVGA (320x240).

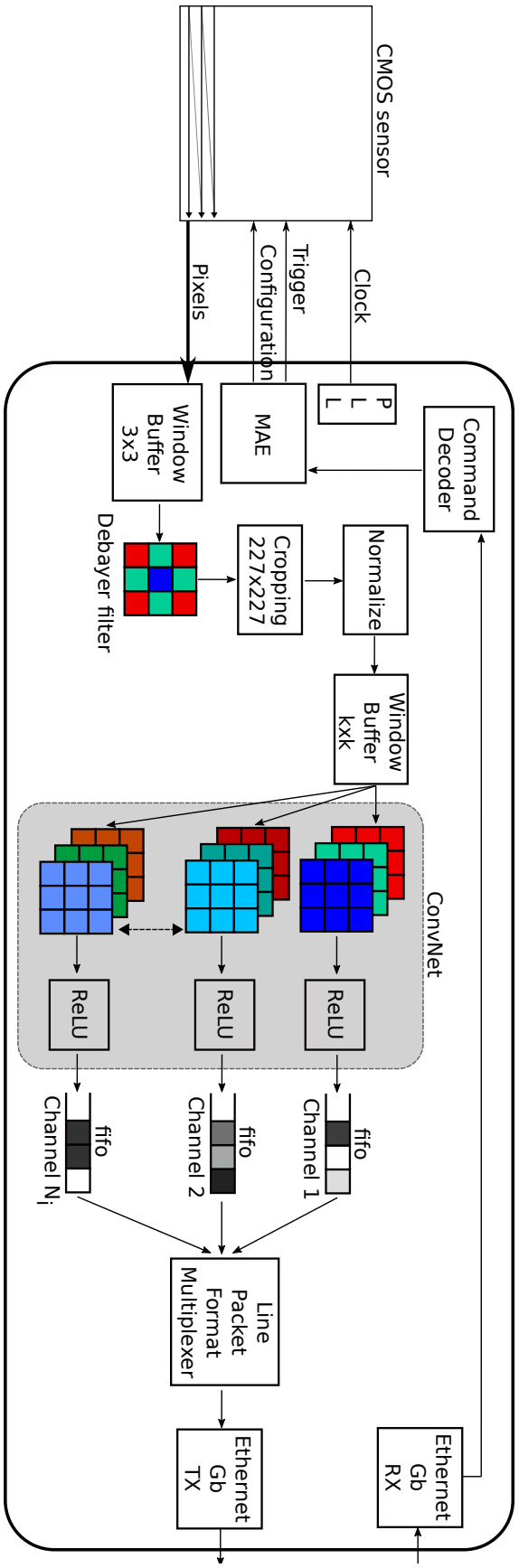


FIGURE 4.18 – Schéma synoptique du réseau de convolutions implémenté dans les têtes de caméras.

TABLE 4.4 – Résultats de synthèse des parties *front-end* sur des FPGA typés embarqué : Le design comprend entre autres, la machine à états configurant le capteur, le pré-traitement bayer→RGB, le pipeline de la première couche du réseau et une pile de communication UDP/IP vers un lien Ethernet Gigabit.

(N_1, K_1)	Conv 1	Full design(% Total)	Bandwidth MB/s (30 img/s)
(48,3)	70806 LEs + 28740 Registers	59% (Cyclone III)	1.001
(48,5)	146180 LEs + 113920 Registers	109% (Cyclone III)	1.001
(48,5)	39561 ALMs + 112681 Registers	69% (Cyclone 10GX)	1.001

4.2.6 Temps de calcul du schéma d'accélération mixte

$$T_{new}(SIMD) = T_{total}(SIMD) - \sum_{i=0}^{i=L} T_{SIMD}(L_i) + \sum_{i=0}^{i=L} T_{DHM}(L_i) \simeq T_{total}(SIMD) - \sum_{i=0}^{i=L} T_{SIMD}(L_i) \tag{4.19}$$

Cette section démontre le potentiel d'un système d'inférence multi-vues mixte incorporant le DHM sur les têtes de caméra et un modèle de calcul conventionnel sur CPU, GPU ou FPGA symbolisant le noeud central de calcul. Ce coeur est composé soit d'un SoC ARM Cortex-A57 soit d'un GPU embarqué JETSON Nano soit de l'accélérateur PipeCNN [Wan17] décrit en OpenCL. Comme montré par le tableau 4.5, l'accélération openCL sur FPGA n'est pas capable de traiter plus de 3 images par seconde mais l'ensemble multi-vues optimisé proposé multiplie par 4 le débit de données. La première couche $(N_1, k_1) = (96, 11)$ du réseau dense initial requiert près de 69 ms de temps de calculs tandis qu'une des versions optimisée à 4 caméras $(N_1, k_1) = (48, 3)$ n'occupe que 3 ms du temps d'inférence total avec *PipeCNN* ou 8.4ms pour la version $(N_1, k_1) = (48, 5)$ à deux caméras. Il est important de noter que cette optimisation n'est possible que dans le cas où un ensemble convexe de caméras visualise un même objet. Les données en entrée du réseau multi-vues suivent la même trajectoire directe que dans le cas mono-vue. Par conséquent, une image en sortie de capteur requiert 205M opérations MAC dans le cas multi-vues où $(N_1, k_1) = (48, 3)$ et 660M opérations MAC pour le réseau initial.

4.2.7 Conclusion

Ce chapitre a démontré l'intérêt d'une inférence mixte entre le modèle de calcul DHM exclusif aux FPGA et le modèle avec ressources partagées pouvant s'exécuter sur tout

TABLE 4.5 – Temps d'inférence reporté sur SoC ARM Cortex A57, GPU embarqué, et FPGA (PipeCNN).

Device	ARM Cortex-A57 1.43 GHz	Jetson Nano MaxN 0.92 GHz	Cyclone V SoC PipeCNN 150 MHz
AlexNet CNN	221.6 ms	31.5 ms	367.5 ms
(48,3, $\hat{H}$) CNN(all layers)	119.5 ms (-46%)	13.7 ms (-56%)	91.3 ms (-75%)
Cfg1 CNN(w/o 1st layer)	106.2 ms (-52%)	13.2 ms	88.1 ms (-76%)
(48,5, $\hat{H}$) CNN(all layers)	120.1 ms (-46%)	13.2 ms	96.0 ms (-73%)
Cfg2 CNN(w/o 1st layer)	104.7 ms (-52%)	12.6 ms	87.6 ms (-76%)

type de plateforme. Cependant, le modèle flots de données matériels reste extrêmement coûteux et requiert des optimisations comme celles appliquées dans le chapitre précédent. Le multi-vues intervient sur cet aspect, c'est-à-dire sur la capacité du réseau à supporter un fort taux de compression. L'architecture de caméra multi-vues proposée est scindée en deux parties où la première supporte un schéma d'inférence directement sur le flot de données et la deuxième poursuit le reste de l'inférence grâce à une accélération conventionnelle. Le gain de temps d'inférence est notable sur systèmes embarqués comme le montre les résultats du tableau 4.5 où le réseau AlexNet est embarqué sur un système entièrement basé sur des FPGA.

Chapitre 5

Proposition d'une plateforme matérielle multi-vues

Résumé : *Ce dernier chapitre concerne la partie technique de la thèse. La construction d'une caméra custom n'est pas triviale et nécessite de répondre à des besoins spécifiques. L'architecture de la caméra multi-vues découle des résultats trouvés en amont de ce chapitre mais répond aussi à une volonté d'en faire un outil pérenne de recherche et développement. La première section montre comment une caméra à base de FPGA pert de répondre au problème de synchronisation de prises de vues. La partie suivante propose d'établir le cahier des charges et une partie du développement de l'ensemble. Le contexte sanitaire et professionnel ont limité le temps de production du système multi-vues avec la société Italienne partenaire.*

5.1 Exemples de caméras multi-vues sur FPGA

Comme la dernière partie de ce travail de thèse repose sur les étapes de la conception d'une caméra multi-vues embarquée, il est intéressant de remarquer que le nombre de réalisations restent limitées et cantonnées à leur but applicatif premier. *Afshari et Popovic* [AJB+13, PASL13] ont conjointement construit un dôme de réticules, nommé caméra panoptique, et un algorithme de reconstruction d'image selon une vue semi-sphérique. Cet algorithme assemble les images de 20 capteurs au format CIF (352×288) et génère une image panoramique de 1024×288. Le moteur de rendu embarqué est orienté flot de données avec un calcul de la position de chaque pixel de chaque caméra sur la demi-sphère. Les paramètres de la géométrie sont contenus dans desLUT et les unités DSP

(A) (B) (C)

FIGURE 5.1 – Caméras multi-vues intelligentes (FPGA) issues des travaux de *Popovic et al.*

du FPGA Virtex 5 transforment chacune des images selon un plan de référence. L'intensité des images est normalisée selon une caméra de référence puis assemblées par *alpha-blending*. Les mêmes auteurs proposeront un architecture similaire [PCL+17] mais appliquée à plus grande échelle. Les 49 caméras du systèmes sont connectées à une carte contenant 8 FPGA Virtex 5 dont 7 de ces circuits sont dédiés à 7 caméras chacun. Cette quantité de FPGA n'est pas surprenante du point de vue de la technologie puisque un unique FPGA ne propose pas de suffisamment d'entrées/sorties pour gérer autant d'imageurs. De même, le nombre de DSP embarqués est insuffisant pour gérer simultanément les 49 transformations géométriques en temps réel. C'est pourquoi, le calcul est distribué sur plusieurs FPGA et que le système global est minutieusement étudié pour gérer la synchronisation et les mouvements de données vers le circuit reprogrammable final. Plus tard, ce même algorithme est réutilisé dans une application spécifique de détection de drones [DEN+20]. La caméra est composée de 4 FPGA Virtex 7, chacun pilotant 3 caméras de 5 méga-pixels pour un total de 12 caméras. Le système est capable de générer un panorama de très haute définition (320 Méga-pixels) à 3 images par seconde ou 15 images par secondes en sélectionnant une partie du panorama à 21.6 Méga-pixels. La bande passante du système atteint 7.7 GB/s dans le cas du panorama complet. Les caméras intelligentes du système contribuent au calcul en prétraitant la projection des images sur une portion de la demi-sphère et embarquent également un algorithme de soustraction d'images pour détecter les objets mobiles. Les images sont récupérées par un FPGA concentrateur puis envoyés à une station de travail disposant d'un GPU pour la classification d'objets.

D'autres réalisations de systèmes multi-vues embarqués, toujours à base de circuits

reprogrammables, mais moins denses ont été étudiés comme la caméra stéréo bas coût de *Ahlberg et al.* [ALE+11] ou de *Monroy et al.* [MHK+15]. Ces solutions maisons prennent avantage des FPGA pour créer une architecture centralisée capable de gérer le flot de données des capteurs. Cet état de l'art des systèmes multi-vues basés sur FPGA démontre que ce type de circuit est idéal pour traiter la partie d'un algorithme avec le modèle flots de données ou pour piloter une grande quantité de capteurs d'images de manière synchronisée.

5.2 Synchronisation de prise de vue

La synchronisation d'images est un problème fréquemment rencontré sur des systèmes multi-visions et différemment appréhendé selon le type de capteur et/ou d'architecture de caméra. Les capteurs Rolling-shutter sont très utilisés à cause de leur coût de production relativement bas comparé aux capteurs Global-shutter. La différence principale vient de la façon dont l'image est capturée puis extraite du capteur... Le système multi-vues proposé est réalisé autour de capteurs global-shutter pour cette raison. Cette particularité est indispensable en calibration stéréo en ligne où les éléments de la scène doivent être figés dans l'espace pour obtenir l'estimation la plus précise de la matrice fondamentale [SP10]. En *free-viewpoint* vidéo, il est également indispensable de garantir une synchronisation parfaite pour générer des rendus 3D réalistes. En général, les systèmes sur étagère ne sont pas intrinsèquement conçus pour répondre à ce type de problématique et divers mécanismes sont nécessaires pour atténuer la déviation temporel produit par un ensemble de caméras. Une caméra multi-vues embarquée basée sur des circuits reprogrammables permet d'éliminer ce problème avec un contrôle fin des signaux de commandes sur les capteurs d'images.

5.2.1 Propositions de l'état de l'art

Le protocole IEEE 1588 (v1 et v2) PTP est une surcouche du protocole ethernet qui propose de synchroniser plusieurs appareils à travers un réseau ethernet. La précision d'horloge garantie est de l'ordre de la milli-seconde pour la première version et sous la micro seonde pour la deuxième. Ce protocole se situe au dessus de la couche physique du modèle TCP/IP gérant les liaisons par cuivre, optique ou radio. Les auteurs de [SP10]

parviennent à synchroniser plusieurs caméras à-posteriori de la prise de vue, connaissant la géométrie de la scène. Leur méthode consiste à générer puis analyser les silhouettes des personnes se trouvant dans le champ des caméras. Une séquence est synchronisée si les tangentes aux silouhettes vérifient la contrainte épipolaire. Dans [BAIH09], les auteurs synchronisent des capteurs rolling-shutter à l'aide d'un stroboscope.*Sousa et al.* [SWSD15] règlent la tension du synthétiseur de fréquence VCO des capteurs d'image pour minimiser l'écart de synchronisation entre plusieurs caméras USB. Enfin, la caméra de *Willburn et al.* [Wil04, WLH] utilisent le signal de déclenchement (*trigger*) des dizaines de capteurs d'image *rolling shutter* gérés par plusieurs FPGAs. Avec cette technique, les auteurs sont capables de générer des séquences de ralenti à plus de 1000 images par seconde à partir d'une centaine de caméras. Cette dernière solution est une des plus simple à mettre en oeuvre, particulièrement quand des FPGA sont couplés aux capteurs. La caméra multi-vues présentée dans cette thèse est en quelque sorte une version plus compacte et plus mobile. Elle prend notamment avantage des dernières avancées technologiques tels que les liens sériels haute vitesse, les processeurs embarqués et une interface mémoire DDR4 dont la bande passante permet en théorie de soutenir 15 GO/s de données. Le fpga central peut théoriquement mémoriser jusqu'à

$$N_{cam} = \frac{BW_{ddr}}{v \times BW_{cam}} \tag{5.1}$$

$$= \frac{DataBW_{ddr}}{v \times BW_{cam}} \tag{5.2}$$

5.2.2 Synchronisation sur FPGA

Les FPGA offre la possibilité de contrôler très précisément les timings des signaux internes et externes pour un coût matériel négligeable. La solution retenue emploie des capteurs *global-shutter OnSemi* dotés d'une commande de déclenchement externe (*trigger*). Un FPGA doté de plusieurs ports de communication ou un ordinateur conventionnel peuvent envoyer un signal de déclenchement sur toutes les caméras par le biais d'un signal commun pour assurer la simultanéité de la commande. Le protocole Ethernet permet de cibler une machine en particulier (*unicast*) ou toutes les machines présentes sur

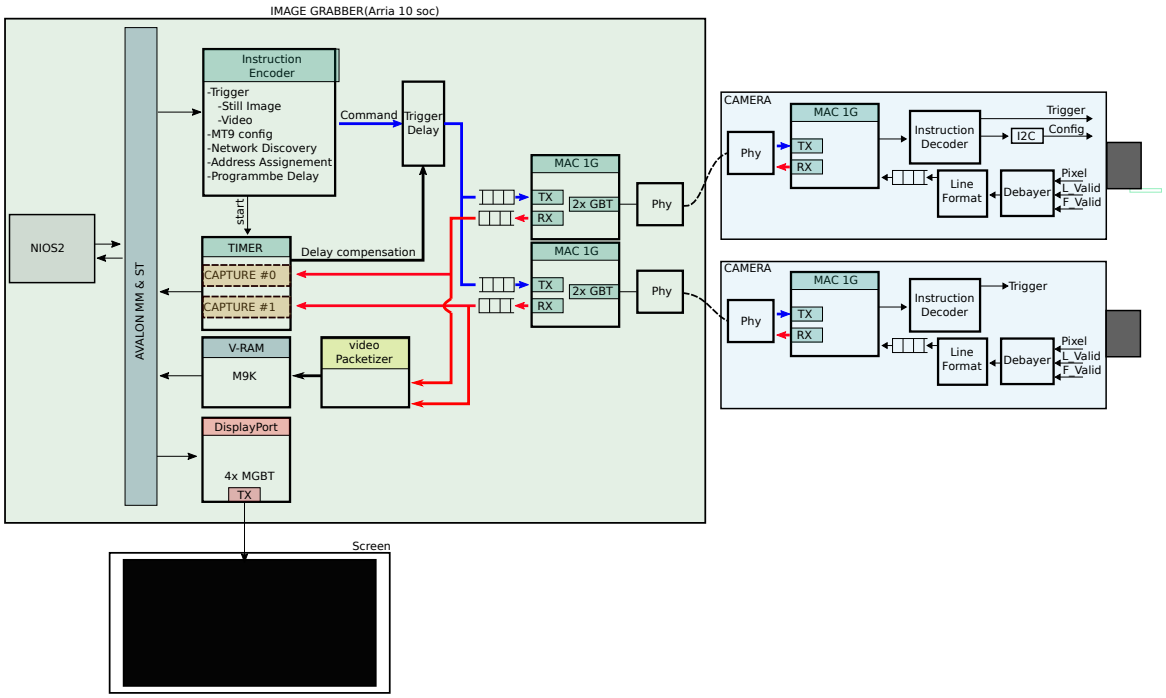


FIGURE 5.2 – Un FPGA central pilote deux têtes de caméra jumelles, elles aussi basées sur des FPGA.

un réseau (*broadcast*. La caméra proposée utilise cet avantage avec un coeur central envoyant le même signal de déclenchement à toutes les caméras satellites au même instant. Lorsque le FPGA central envoie une commande de déclenchement, les N modules MAC envoient une trame spécifique à leur caméra associée. Le temps est définie par le chemin parcouru par une trame c'est-à-dire les latences combinées des modules MACs ($T(MAC_{central})$, $T(PHY_{camera})$) , Phy ($T(PHY_{central})$ et $T(PHY_{camera})$) et de la machine à états $T(MAE)$ qui décode la commande reçue.

$$T_{trigger} = T(MAC_{central}) + T(PHY_{central})\,^{1} + T(PHY_{camera}) + T(MAC_{camera}) + T(MAE) \tag{5.3}$$

Le terme $L_{acquisition}$ est lui aussi une combinaison des temps d'intégration, de conversion et d'envoi de la première ligne de pixels de l'image prise par une tête de caméra. Le coeur de caméra reçoit la première ligne de pixels de chacune des têtes de caméra à l'instant $t_3(i)$ où i est l'indice de la i-ème caméra. Un pointeur matériel enregistre les temps d'arrivée de chaque image dans une mémoire interne du FPGA central. L'architecture de base est présentée dans la figure 5.2. Le graphe **??** reporte les résultats de synchronisation de la

1. Ici des Marvell 88E1111

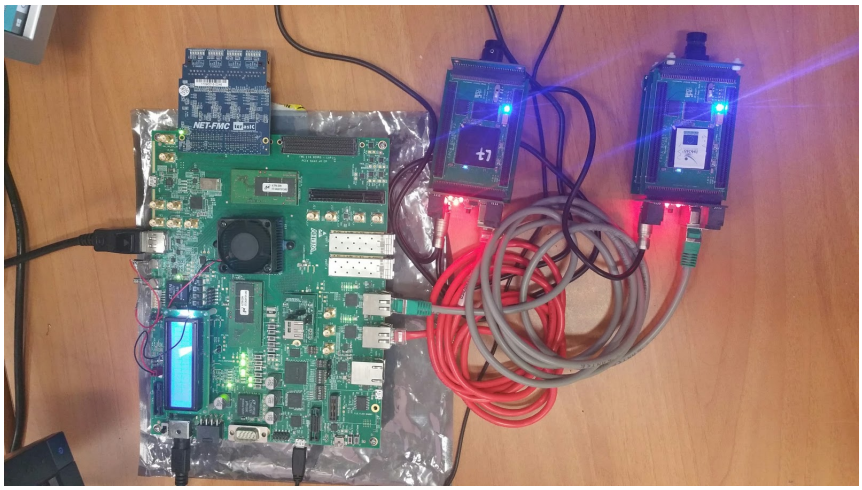


FIGURE 5.3 – Système d'acquisition synchrone de prises de vues entre 2 Dreamcam pilotées par un FPGA Arria10.

solution avec 2 caméras identiques (DreamCam) et un coeur central composé d'une carte d'évaluation ARRIA10 soc Intel FPGA. La différence de temps mesurée entre l'arrivée de la première ligne de chaque caméra est négligeable par rapport à la fréquence de prise de vue maximale (60Hz) et correspond à un Ici, les 2 capteurs sont réglés pour délivrer des images de 512×512 pixels. Le module central envoie une commande de déclenchement simultanée dès que le premier pixel de la première ligne de la dernière caméra est reçue afin de garantir la synchronisation. Le protocole embarqué de chaque côté de la caméra de test est illustré dans la figure 5.4 :

Une deuxième solution consiste à implémenter le protocole UDP/IP au dessus d'Ethernet pour s'affranchir du noeud central et travailler avec des switchs classiques. Le protocole, décrit par la figure 5.8, suit un modèle flot de donnée où chaque couche du modèle TCP-IP est traversé une seule et unique fois :

Cette méthode consomme plus de ressources sur le FPGA d'une caméra puisqu'elle doit embarquer un moteur d'encodage/décodage de trames IPv4. Pour être totalement reconnu sur un réseau IP basique, le moteur matériel doit aussi être en mesure de répondre aux requêtes ICMP et par voie de conséquence, aux requêtes ARP. La première est destinée à répondre aux échos des machines présentes sur un réseau tandis que la deuxième permet d'identifier une machine par son adresse IP. Cet ajout représente 41% de ressources logiques additionnelles sur la matrice du FPGA comparé (pas d'IP 6484 LEs) (10859 LES, 561,840 / 3,981,312 (14 %)). L'entête UDP/IP elle ne représente que 28 octets supplémentaires sur une trame de données.

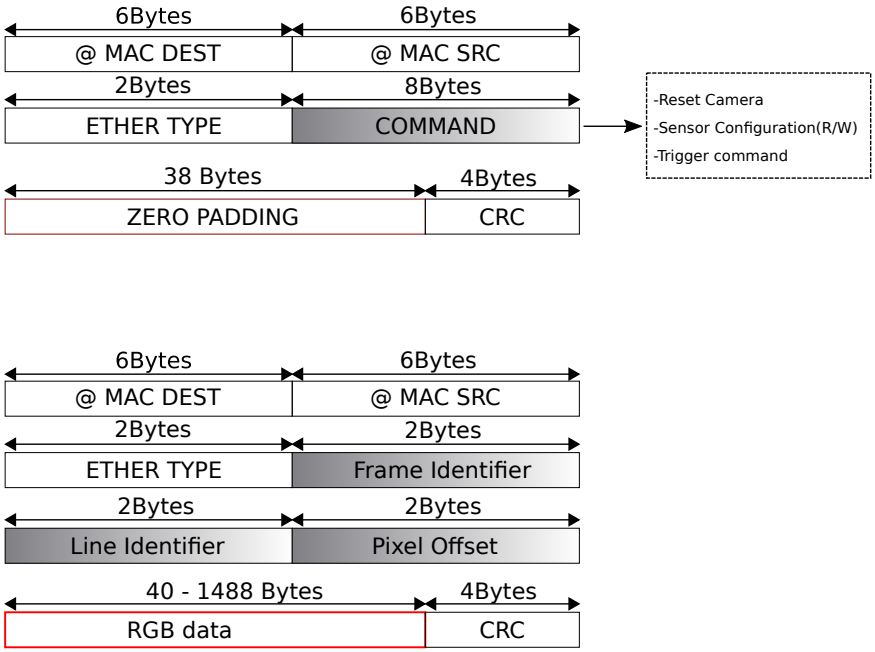


FIGURE 5.4 – Architecture des trames de communications entre les FPGA du système.

FIGURE 5.5 – Variations du temps de réponse (ping) entre le FPGA central et les têtes de caméra.

FIGURE 5.6 – Variations du temps de réponse pour la capture d'images

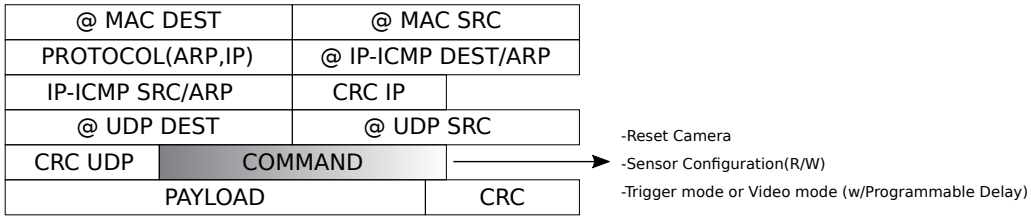


FIGURE 5.7 – Entête du protocole UDP-IP avec gestion des requêtes de découverte réseau (ARP) et de réponse au ping (ICMP)

	LEs	Memory (bits)
RAW	6480	137154
UDP/IP	10859	561,840

TABLE 5.1 – Nombre d'éléments logiques et de mémoires avec la surcouche UDP/IP et sans (RAW)

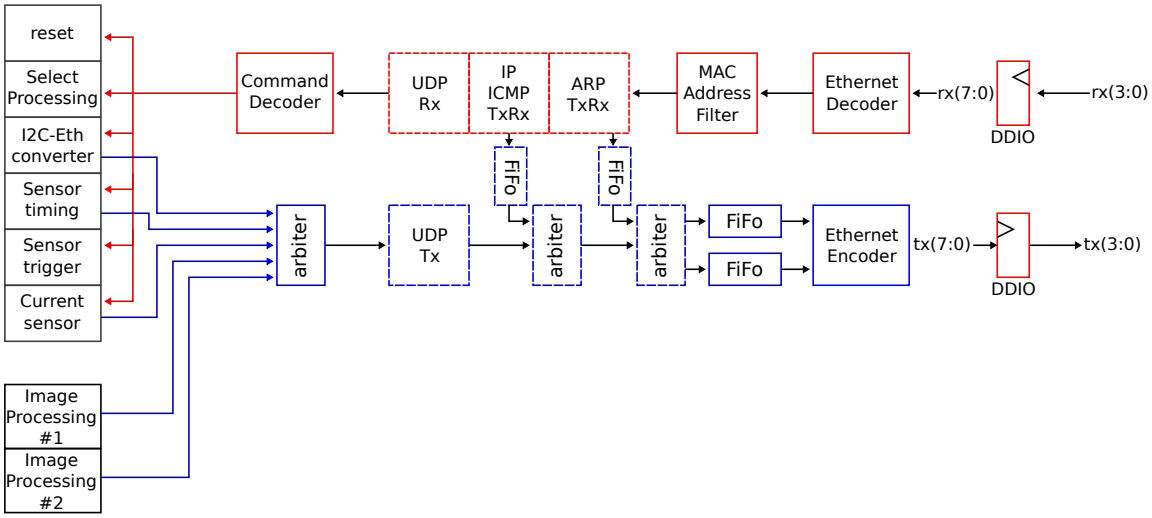


FIGURE 5.8 – Chemins de données en réception et émission présents dans une tête de caméra. Les blocs optionnels sont représentés par des tirets.

Le circuit final présent dans une caméra est illustré dans la figure 5.8

5.3 Limites : bande passante mémoire externe et nombre de serdes

La bande passante de l'interface mémoire et le nombre d'entrées/sorties déterminent la limite du nombre de têtes de caméras pouvant être connectées simultanément au FPGA central. Plus précisément, un capteur d'image couleur RGB de taille $(U \times V)$ codées sur b bits consomme une bande passante BW_{RGB} pour une fréquence de f images par secondes comme décrit par l'équation 5.4.

$$BW^{b}_{RGB} = 3 \times U \times V \times f \times b \tag{5.4}$$

Le nombre de caméras N^{cam} est théoriquement limité par la bande passante de la mémoire externe du centralisateur.

$$N^{cam}_{theorique} = \frac{BW_{DDR}}{max(BW_{link}, BW^{b}_{RGB})} \tag{5.5}$$

Dans le cas présent, la limite se situe au niveau du lien BW_{link} capé à 1000 Mbits/s soit 125MB/s. Une solution consiste à dédier un DMA pour chacune des 9 caméras, ceux-ci étant arbitrés automatiquement par le logiciel de conception de système sur puce Qsys. Un DMA permet de supporter N caméras (19.2Go/s) simultanément en mémorisant ligne par ligne les images issues des têtes de caméras. Le goulet d'étranglement se situe donc au niveau des modules DDR donné pour 15 Go/s sur le module ARRIA 10. Le nombre de serdes est aussi un facteur déterminant dans la conception d'une caméra multi-vues. Les serdes sont des cicruits numériques dédiés situés à la périphérie d'un FPGA et destinés à soutenir différentes formes de protocoles de communication. Par exemple, le protocole PCI-Express est basé sur des lignes sérialisation/désérialisation agrégées, chacune supportant plusieurs gigabits de données par seconde. Le protocole SGMII est une variante des protocoles standards GMII et RGMII pour la communication bas niveau d'ethernet. Il utilise des entrées/sorties différentielles fonctionnant à 1.25Gb/s pour communiquer avec un phy externe via une surcouche de codage 8b/10b. Ce type d'entrées/sorties est massivement présent sur les FPGA de milieu et haut-de-gamme avec un maximum de 96 paires pour un Stratix 10. Ces 96 paires permettraient théoriquement de supporter un total de 48 caméras.

5.4 Développement

Le système final, *Hydra*, est une caméra multi-têtes composée de 9 capteurs d'images. Plusieurs astuces techniques permettent d'améliorer sensiblement l'embarquabilité de la caméra comme l'injection POE+ ratifié dans le document IEEE 802.3at-2009. Cette spécification permet de fournir l'énergie aux systèmes distants dans une limite de 100m et pour libérer le système des contraintes de géométrie fréquemment rencontrées dans les systèmes de ce type. L'idée reste toujours de fournir un socle de base cohérent avec la notion de système embarqué pour de futures recherches. Comme l'idée véhiculée dans cette thèse consiste à pré-traiter une partie d'un CNN sur la partie *front-end*, c'est-à-dire les têtes de caméras, chaque module possède son propre FPGA embarqué. L'ensemble du système se veut modulaire dans le sens où les têtes de caméras peuvent être pilotées par un switch traditionnel et accessibles depuis un réseau de bureau ou alors pilotées par

un FPGA central orienté multi-média plus facilement transportable dans un système de navigation autonome.

5.4.1 Têtes de caméra

La possibilité de calculer en ligne une certaine partie d'un algorithme représente un avantage majeur dans de nombreuses applications de vision, en particulier tout ce qui concerne la vision multi-vues. Ce principe de décentralisation a été introduit dans le chapitre précédent où un CNN est divisé en 2 sections distinctes. La partie *front-end* est supporté par un modèle flot de données et décharge un processeur central du nombre d'opérations initialement supporté par ce dernier. Les têtes de caméra du système sont basées sur des FPGA Cyclone10GX capables de communiquer avec un coeur de caméra, décrit plus loin dans la section 5.4.2. Un FPGA gère sa propre pile de communication et jusqu'à deux capteurs d'images, l'un connecté avec un bus parallèle et l'autre avec le protocole MIPI. Dans un soucis de compacité, ces têtes de caméras sont divisées en 3 parties. La première embarque le module de puissance et de communication. Un circuit LT4275 convertit la puissance reçue à travers la connectique Ethernet (-54V/320 mA), une tension de 5V et 4A afin d'alimenter l'ensemble du système. Ce circuit permet de travailler soit en POE+ classe 0 pour un maximum de 12 Watts soit en classe 4 pour un maximum de 25 Watts théoriques bien que le système final est dessiné pour en recevoir 20 par têtes de caméra. Un connecteur d'alimentation externe de 5V et 4A est embarqué en parallèle pour pouvoir utiliser la caméra dans un contexte différent, par exemple un switch non POE ou directement connecté à une station de travail standard. La deuxième partie intègre la partie *processing* avec le FPGA pilotant l'ultime carte contenant un capteur d'image *global shutter* ON-SEMI AR0135. Le FPGA reste entièrement reprogrammable à distance via un module de mémoire FLASH pouvant contenir plusieurs *bitstream* à la fois afin de changer à la volée le type de *processing*. Le produit final est présenté dans les figures 5.10 et possède des dimensions contenues de 80x80mm. L'entraxe minimum entre deux capteurs est donc de 80mm en vertical et d'environ 110 mm en diagonale.

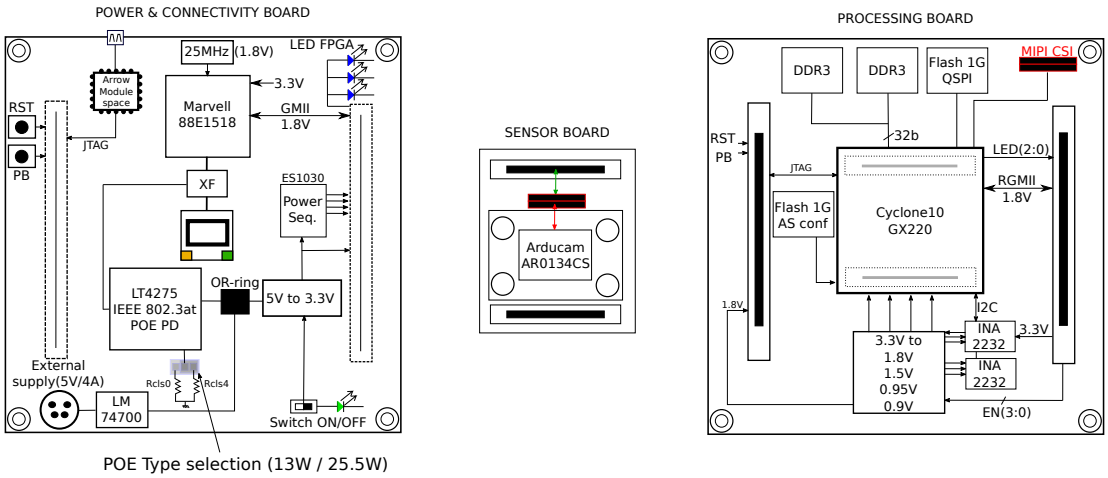


FIGURE 5.9 – Têtes de caméra Cyclone10 avec capteur global shutter

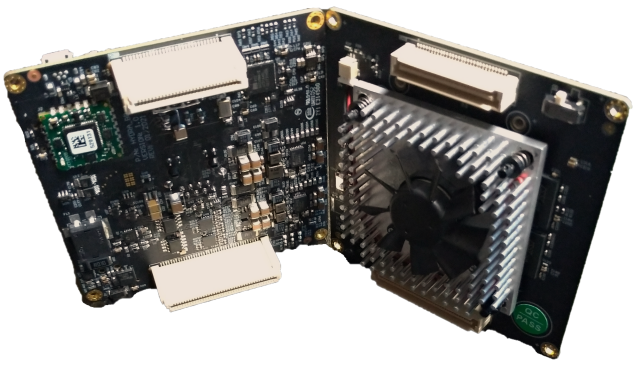


FIGURE 5.10 – Réalisation des têtes de caméra

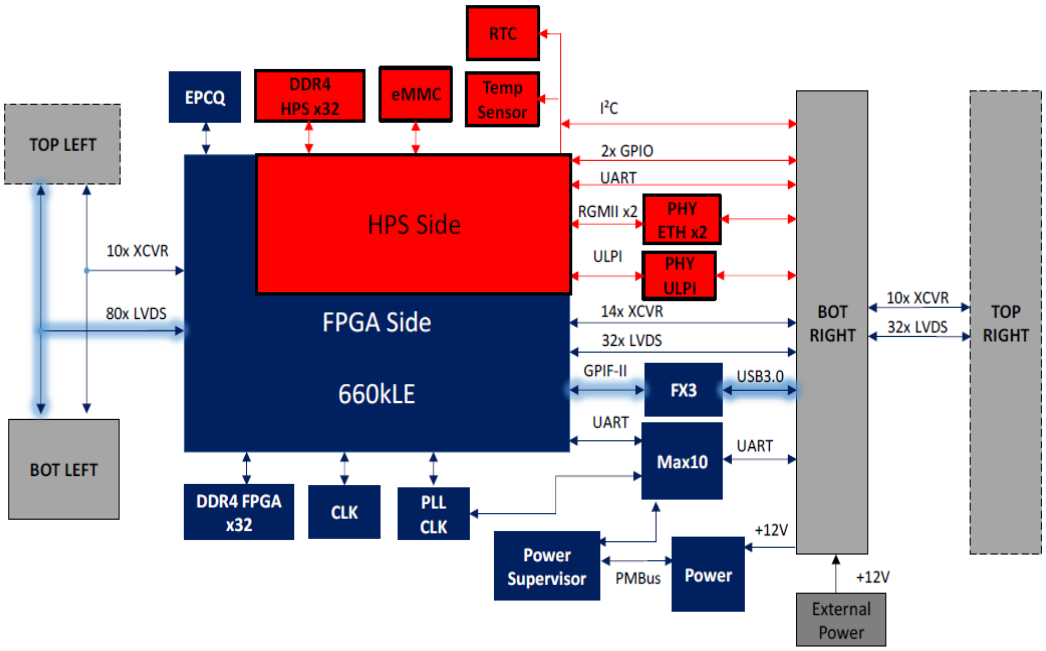


FIGURE 5.11 – Architecture du système sur module Arria10

5.4.2 Centraliseur

La partie centrale de caméra reprend l'architecture interne développée dans le cadre des expérimentations de synchronisation. Elle sert notamment à gérer l'envoi des commandes, la réception des données des 9 têtes de caméras et peut servir en tant que *back-end* pour procéder au reste de l'inférence d'un réseau de convolution. Un module du commerce **ReflexCES** doté d'un FPGA ARRIA10 de 660 KLEs est choisi pour alléger la charge de développement d'une nouvelle caméra multi-étages. Ce choix se justifie pleinement car cette plateforme possède déjà la majorité des entrées/sorties nécessaires à la caméra imaginée mais possède aussi son propre circuit d'alimentation comme le montre la figure 5.11. Le FPGA Intel embarquée expose 24 paires différentielles très haute vitesse, 42 Mbits de mémoireRAM interne, de 2 processeurs embarqués ARM Cortex-A9 et de 16 GB de mémoire externe. Les 9 caméras sont connectées par l'intermédiaire de 18 paires différentielles et alimentées par une seconde carte recevant l'alimentation générale donnée pour -54V et 300 Watts. Une partie de cette alimentation est transmise aux caméras par 3 circuits d'injection POE LTC4266 dotés de 4 ports chacun, où seulement 3 sont utilisés pour fournir 20 Watts par caméra comme montré dans la figure 5.12. Une partie de la puissance sert à alimenter le coeur de la caméra et son FPGA embarqué. Les liens serdes

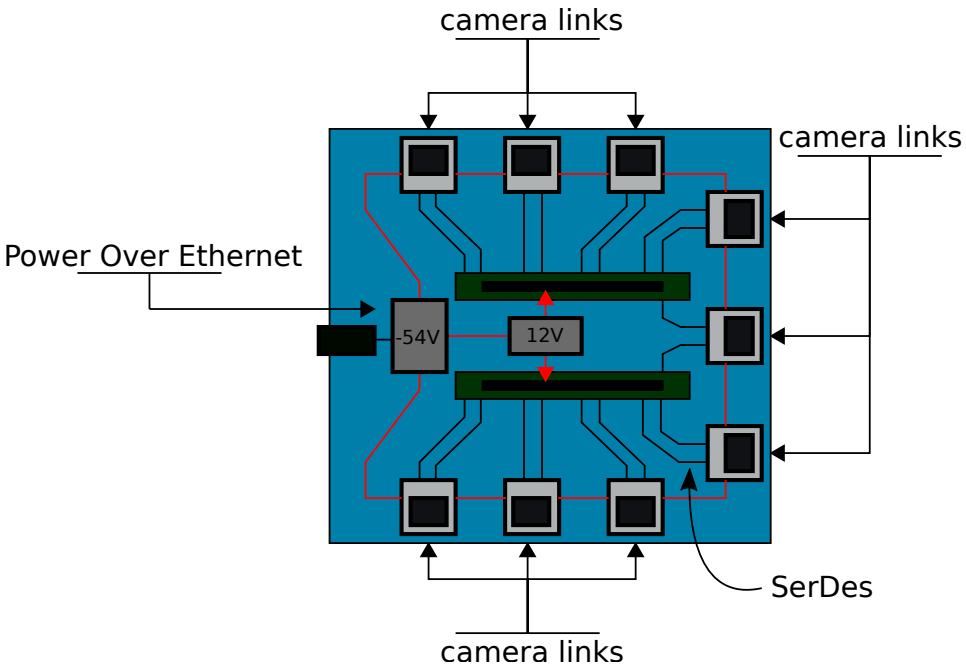


FIGURE 5.12 – Partie puissance gérée intégré au système central

	ALM	Memory (bits)	PLL	Serdes RX	Serdes TX	Total I/O
ARRIA 10 SX066	36,295 / 251,680 (14 %)	12,849,442 / 43,642,880 (29 %)	27 / 64 (42 %)	10 / 24 (42 %)	14 / 24 (58 %)	342 / 604 (57 %)

TABLE 5.2 – Ressources matérielles du système de base du FPGA central.

haute vitesse restants sont utilisés pour communiquer à 10 Gb/s avec un serveur externe par le biais d'une connectique SFP+ toujours basé sur le protocol UDP/IP. Ce lien optionnel a été pensé pour servir de pont entre les têtes de caméras et un calculateur externe. Les 9 caméras peuvent ainsi transmettre des images de 1280x960 à 30 images par seconde sans mémorisation de la part du FPGA. Deux modules d'affichage displayport et HDMI (Analog Device ADV7513) occupent les entrées/sorties restantes du système sur module pour visualiser en temps réel les données issues des capteurs sur des écrans embarqués. Au total, la plateforme de base dotée de tous les éléments évoqués ci-dessus

L'ensemble du système central se résume donc à trois cartes de dimensions 96mmx85mm. L'une est dédiée au dialogue avec l'extérieur et vient s'attacher directement en dessous de la carte contenant le FPGA. La denière carte est dédiée à la gestion des têtes de caméra. Le concept de ces deux cartes est illustré dans la figure 5.13. Cette partie de la caméra est en cours de réalisation chez le fournisseur responsable de la fabrication des têtes de caméras. Au final, cette topologie reste similaire à celles employées dans la littérature scientifique mais possèdent des atouts supplémentaires représentés par la modularité du

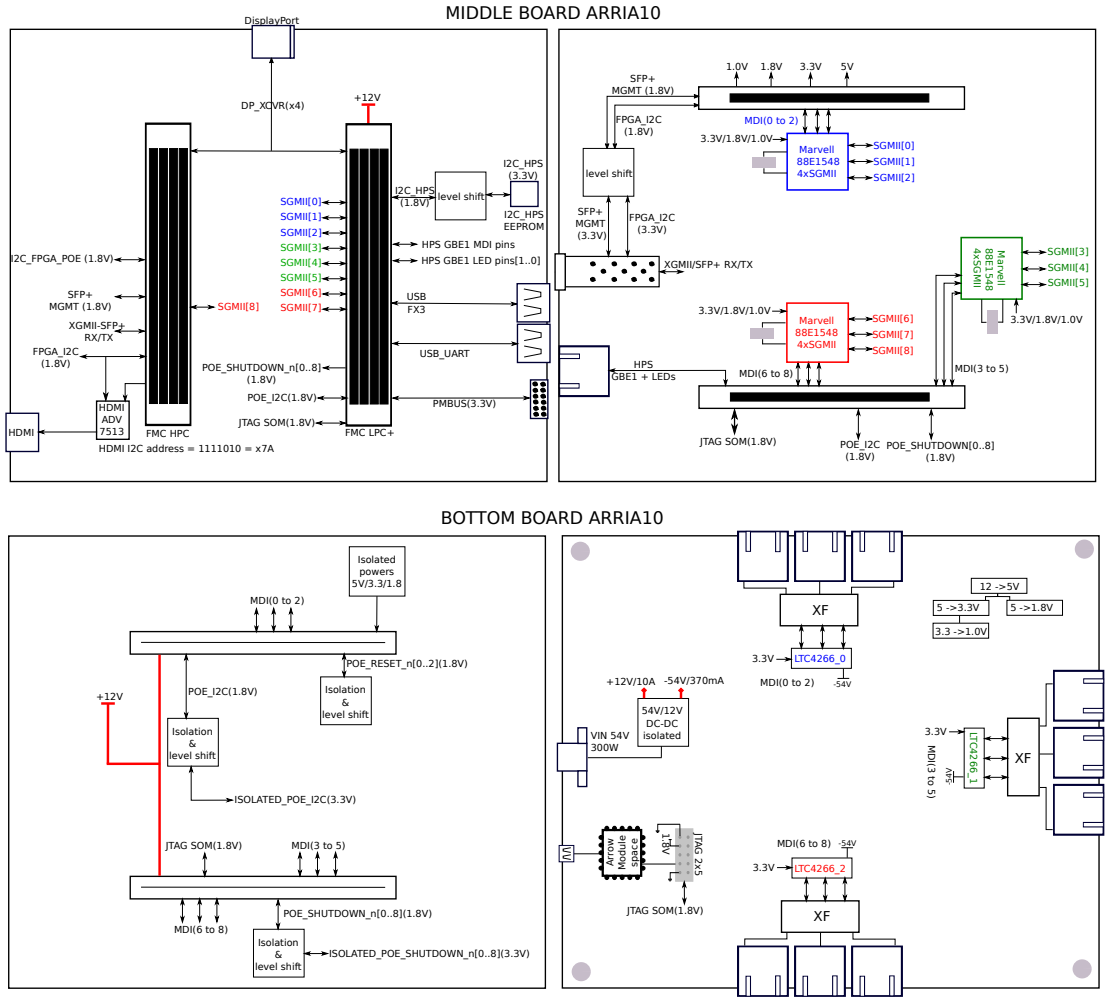


FIGURE 5.13 – Ensemble des cartes de support pour le système sur module Arria10

système. Le prototype de caméra multi-vues présenté dans cette dernière section est la résultante des idées émises dans le chapitre précédent. Plus précisément, un FPGA central pilote et connecte toutes les têtes de caméras. En dehors du spectre de l'intelligence artificielle, cette architecture possède l'avantage de synchroniser chaque prise de vue avec une déviation temporelle de l'ordre de quelques dizaines de nano-secondes. Le principe de centralisation peut être poussé à l'extrême en permettant au noeud de communication de fournir la puissance aux caméras connexes via les liens de transmission pour réduire l'encombrement en centralisant la source d'énergie. En additionnant l'ensemble de ces optimisations, l'architecture proposée fournit un socle de caméra multi-vues embarquée, différente des systèmes actuellement disponibles car pensée en premier lieu pour accélérer l'inférence d'IA mais également pour fournir une instrumentation suffisamment générique pour de futurs projets de recherche. Les tableaux 5.3 et 5.4 proposent un récapitulatif technique des caméras embarquées de la littérature pour mettre en perspective les capacités de la caméra développée.

Etat de l'art des caméras multi-vues embarquées ou utilisant des FPGA	FPGA	GPU/CPU	Cameras	bande-passante capteurs (8bpp)	Couleur	Connectique capteurs
Gidel	Arria10	-	jusqu'à 100 (annoncé) via des modules de connexions supplémentaires	N.C.	oui	MiPi, Fibre Optique, Camera Link
e-con systems	-	Nvidia Jetson Tx2	jusqu'à 4	up to 4K, 30fps	oui (UYUV)	MiPi vers Coaxial
FaceBook360 surround	-	-	13	1384x1036, 30 fps	oui	Usb3.1
High performance imaging with large camera arrays [Wil06]	>100 Spartan 2	4 CPU	>100	320x240	oui	FireWire
Pantoptic Camera [PASL13]	Virtex 5	-	20	352x288	non	LVDS
"Design and implementation of real-time multi-sensor vision systems" [PCL+17]	7 + 1 Virtex 5	-	49	352x288	oui(RGB)	LVDS, Bus PLB
"Real-time omni-directional imaging plateform for drone detection and tracking" [DEN+20]	4+1 Virtex 7	-	16	5120x3840, 15fps	non	Fibre Optique
"Gimme" [ALE+11]	Spartan 3A DSP	Intel Atom	2	640x480, 5.5 fps	oui	sur carte
"Hexacam" [MHK+15]	Spartan 6	-	6	960x540, 9fps	oui	MiPi-CSi
Hydra	9x Cyclone10, 1x Arria10 SoC	2xCortex A9	up to 9 (up to 2x9 with embedded MiPi port)	1280x720 bits uncompressed 45 fps	oui	1GbE

TABLE 5.3 – Récapitulatif technique des caméras intelligentes de l'état de l'art (1/2)

Etat de l'art des caméras multi-vues embarquées ou utilisant des FPGA	Mode d'alimentation capteurs	Synchronisation	Affichage Embarqué	Connectivité externe	portée des caméras	Géométrie
Gidel	non	oui	-	PCI-Express, 4x10GbE SFP+	>100m (optical)	6 DoF
e-con systems	coaxial	oui	hdmi, MiPi	1GbE, usb3.0	30 cm	6 DoF
FaceBook360 surround	USB3 ou 12V	N.C.	-	PCI-Express 8x	-	Convexe
High performance imaging with large camera arrays [Wil06]	Dédié	oui	non	-	-	Matrice
Pantoptic Camera [PASL13]	Dédié	N.C.	-	USB 2.0	-	Convexe
"Design and implementation of real-time multi-sensor vision systems" [PCL+17]	Dédié (sur carte)	oui	-	USB2.0, 1GbE	30 cm	Convexe
"Real-time omni-directional imaging plateform for drone detection and tracking" [DEN+20]	Dédié (sur carte)	oui	-	PCI-Express 2.0 (40Gb/s)	>100m	Convexe
"Gimme" [ALE+11]	Dédié (sur carte)	oui	-	100Mb Ethernet	-	Stéréo
"Hexacam" [MHK+15]	MiPi	oui	-	1GbE	-	Convexe
Hydra	POE type 2 -54V/20W (autonomes) ou 5V/4A	oui	Hdmi, Displayport	USB3.1, 10GbE SFP+, 1GbE via SoC	jusqu'à 100m	6 DoF

TABLE 5.4 – Récapitulatif technique des caméras intelligentes de l'état de l'art (2/2)

Chapitre 6

Conclusions et perspectives

Les systèmes multi-vues possèdent de nombreux atouts si les défis techniques qu'ils induisent sont relevés. Les FPGA possèdent suffisamment d'atouts pour piloter des dizaines de caméras de manière simultanée par un seul et unique circuit contrairement aux autres systèmes bien plus coûteux en terme de matériel. Les FPGA sont d'autant plus intéressants puisque certains algorithmes liés à la vision sont transposables dans un modèle flots de données. Ce modèle *feed-forward* est commun à la plupart des architectures de réseaux de convolutions. Nous avons vu dans les chapitres **??** et 4 que le nombre de paramètres d'un CNN pouvaient être dramatiquement réduits en profitant de plusieurs points de vues d'un objet. Le nombre de paramètres et/ou leur distribution à travers un réseau a un impact significatif sur le modèle d'accélération choisi. Par exemple, un accélérateur à PE est plus sensible à l'organisation des poids de convolutions alors qu'un modèle DHM est agnostique à leur agencement. Dans ce dernier cas, l'empreinte matérielle du modèle d'inférence est directement corrélé à la quantité de poids de convolution non-nuls en appelant d'autant d'opérateurs MAC. La quantification des poids et des primitives apporte également un gain notable en terme d'espace de stockage mémoire et d'énergie lors des transferts des paramètres vers l'accélérateur. La taille d'un multiplieur à opérande constante est directement lié au nombre de bits des paramètres et des primitives. et par conséquent, permet de réduire l'empreinte matérielle d'un accélérateur orienté flots de données pour augmenter le facteur d'intégration du CNN dans un FPGA. Un contexte multi-vues peut engager toutes ces méthodes pour réduire la taille d'un réseau de convolutions. Cette thèse a démontré qu'un réseau MVCNN possède plusieurs points optimaux pour lesquels le nombre de paramètres d'un modèle est inférieur au réseau de base. Diverses approches méritent d'être explorées pour la suite de ce travail et

sont listées ci-dessous :

Réseau hétérogène multi-vues :

L'axe de recherche proposé est d'utiliser des architectures de réseaux différentes dans le système multi-caméras. Par exemple, un système à deux caméras pourrait utiliser une architecture de type MobileNet sur la première caméra et une architecture de type SqueezeNet sur la deuxième. Les paramètres seraient alternativement mis à jour pendant l'étape d'entraînement.

Apprentissage par renforcement sur la géométrie :

L'idée est d'invoquer un algorithme d'apprentissage par renforcement RL afin d'optimiser le taux de compression α et la géométrie $R_\theta(x, y, z)$. La trajectoire des caméras serait décidé "en ligne" par l'agent RL en fonction des sorties du programme d'entraînement, c'est à dire le taux de compression et la précision du réseau.

Distillation par séparation de classes :

De manière générale, un réseau conçu pour différencier des centaines de classes sera sur-paramétré sur des problématiques moins complexes. Cette idée est formalisée et démontrée par *Hinton et al.* [HVD15] par un algorithme de distillation de connaissance. Le *dataset* ImageNet [DDS+09, RDS+14] est subdivisé en de multiples sous-catégories $P_1, P_2, ..., P_n$ puis réparties sur plusieurs réseaux de convolutions $NET_1, NET_2, ..., NET_n$ moins paramétrés. Cet ensemble de réseaux regroupe la totalité des connaissances du réseau dense. L'idée serait de démontrer que l'architecture de notre caméra est capable d'appliquer ce principe mais d'une manière sensiblement différente, en utilisant une méthode de compression. Comme appliqué par *Hinton et al.*, le *dataset* serait divisé en V parties correspondant au nombre de têtes de caméra lesquelles sont initialisées avec un réseau dense NET_d. Ces têtes de caméras sont arrangées en une matrice pour simuler une seule et même caméra. D'après les conclusions obtenues dans ce manuscrit, chaque réseau est potentiellement surparamétré puisque les sous-ensembles du *dataset* sont moins difficiles à résoudre. En utilisant le même principe de compression et de regroupement d'activations, un algorithme entraînerait séquentiellement chacune des têtes de caméras sur sa propre partie du *dataset* en appliquant un taux de compression graduel. L'objectif final reste similaire à celui étudié dans cette thèse, à savoir minimiser le

nombre de paramètres total du réseau pour embarquer le modèle flot de données. Puisque les caméras ont des connaissances complémentaires, l'idée est de trouver une équilibre dans la répartition des classes sur les V caméras afin d'obtenir un taux de compression maximal sur l'ensemble de la caméra multi-vues.

Réseau spécifiquement conçu pour exploiter les temps morts dus au sous-échantillonage :

L'annexe 1 a montré que les temps morts du modèle flot de données d'une couche IVR peuvent se traduire par une baisse de ressources matérielles. Cette idée peut servir de base pour créer un réseau optimal exploitant le sous-échantillonage intra-convolutions.

Annexe A

Optimisation annexe du DHM : convolutions avec stride

La tactique la plus optimale pour inférer le graphe d'un CNN consiste à implémenter toutes convolutions par des unités de multiplication-accumulation dédiées, mais reste hors de portée des FPGA embarqués. Les architectures de CNN les plus récentes ont abandonné la méthode de sous-échantillonnage par *max-pooling* au profit d'un sous-échantillonage intra-convolutions ($s \geq 2$). Les réseaux actuels embarquent plus de couches que les précédents et par conséquent, le sous-échantillonnage peut intervenir à l'intérieur même de l'étape de convolution. Par exemple, le réseau AlexNet comporte 5 couches de convolutions. La première couche L_1 compresse la dimension $(U_0,\ V_0)$ des données d'entrée d'un facteur 8 par l'intermédiaire d'un stride $s_1 = 4$ et d'un stride $s_{maxpool1} = 2$ à l'intérieur du *max-pooling*. Le réseau MobileNetv2 comporte 17 couches de convolutions et le même facteur de compression est atteint au bout de la 7-ème couche de convolution. Les couches avec un stride (s>1) présentent une opportunité de réduire l'empreinte matérielle en échangeant la logique utilisée par les opérations de multiplications-accumulation par des unités de convolutions sérialisées et de la mémoire embarquée (FIFO, ROM). *Abdelouahab et al.* [ABP18] ont démontré qu'une convolution sérialisée A.1 occupait plus d'espace sur la matrice d'un FPGA que sa contrepartie implémentée selon le modèle DHM. En effet, l'empreinte mémoire de l'espace de stockage des paramètres de convolutions, l'empreinte matérielle des opérateurs MAC 8 bits génériques et celle de la logique de séquençage nécessite plus de place que celle appelée par des opérateurs MAC spécialisés par leur poids de convolutions. Hors, dans le cas d'une accélération au plus près du capteur, deux paramètres peuvent être exploités pour réduire l'empreinte matérielle

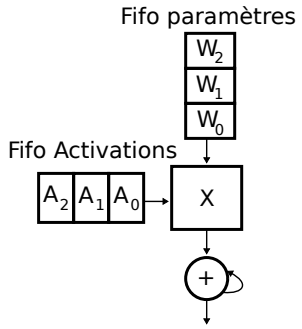


FIGURE A.1 – Figure représentant un opérateur MAC sérialisé. Une FIFO contient les paramètres de convolutions et une deuxième FIFO contient les valeurs des activations. Dans ce cas, le multiplieur n'est plus spécialisé par la taille des opérandes et la logique interne aux FIFO appelle une des ressources matérielles supplémentaires.

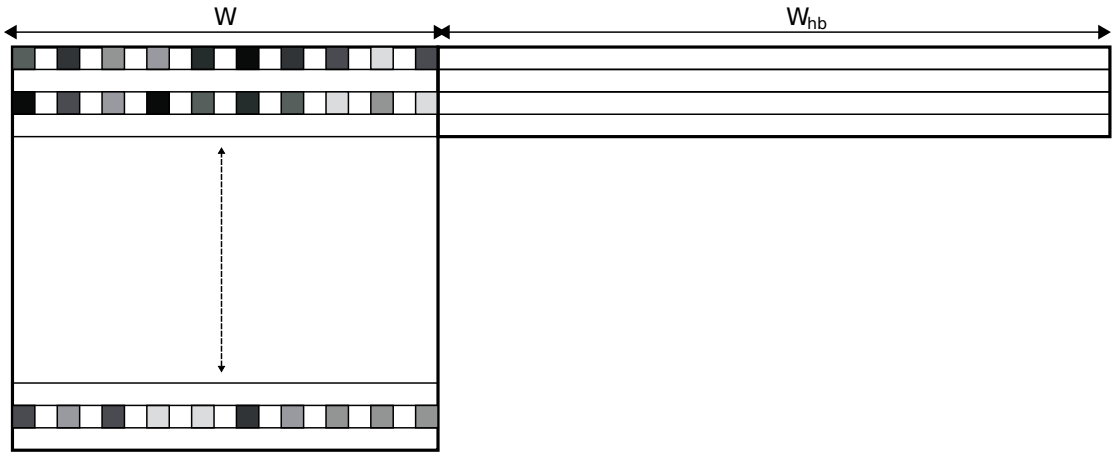


FIGURE A.2 – La figure illustre les temps d'inactivité, représentés par les zones blanches, dans le cas où $s = 2$.

à savoir :

- le paramètre de stride s
- la zone morte horizontale U_{hb} après chaque ligne d'un capteur d'image CMOS

Le premier laisse un accélérateur DHM inactif un pixel sur deux puis pendant une ligne complète alors que le deuxième le laisse inactif pendant $2 \times U_{hb}$ coups d'horloge comme montré dans la figure A.2. Pour un capteur CMOS *OnSemi*, cette zone dure typiquement 370 coups d'horloge. L'idée consiste à remplir ces temps morts par du travail effectif sur des unités MAC génériques en nombre suffisant afin que le temps de calcul couvre le temps total U_{tot} disponible. Ce budget temporel est décrit par l'équation A.1

$$U_{tot}(i) = s_i(\times U_i + U_{Hblank}) \tag{A.1}$$

où U_{tot} est le nombre de cycles d'horloge disponibles pour effectuer les convolutions. $\mathcal{M}_{fifo}$ FIFO sont nécessaires pour mémoriser les activations de la couche *depthwise* précédente. Cette quantité, en octets, est définie dans l'équation A.2 :

$$\mathcal{M}_{fifo}(i) = C_{int}(i) \times \frac{U_i}{s_i} \tag{A.2}$$

Une machine à états récupère les données d'activation séquentiellement depuis les mémoires $\mathcal{M}_{fifo}$ et active les opérateurs MAC correspondants. Les opérateurs d'un même canal de convolution sont activés avec un décalage correspondant à un coup d'horloge et délivrent une donnée d'une ligne d'une activation après C_{int} coups d'horloge. Par conséquent la mémoire contenant les poids de convolutions $M_{weights}$ peut être partagée moyennant un registre à décalage inséré entre les opérateurs [1]. Le nombre de mémoire RAM $M_{weights}$ est donné par l'équation A.3.

$$\mathcal{M}_{weights}(i) = C_{int}(i) \times C_{out}(i) \tag{A.3}$$

L'idée finale consiste à réutiliser des unités mac élémentaires pendant toute la durée U_{tot}. Le nombre optimal d'unités pour un canal de convolutions doit satisfaire la contrainte exprimée dans l'équation A.4 :

$$PE(i) > \lceil \frac{U_i \times C_{int}}{U_{tot}(i) - \frac{U_i}{2}} \rceil \tag{A.4}$$

Le retrait de $\frac{U_i}{2}$ cycles d'horloge est nécessaire pour commencer la séquence de calcul sans discontinuité entre les unités de calculs. Au final, l'accélérateur introduit une latence de $C_{int} + \frac{U_i}{2}$ coups d'horloge pour délivrer la première donnée d'une activation. L'équation A.5 illustre l'idée avec la couche IVR d'indice 2 du réseau MobileNetv2.

$$\begin{cases} C_{int}(2) = 48 \\ U_2 = 56 \\ U_{tot} - \frac{U_2}{2} = 824 \end{cases} \implies PE(2) = 4 \tag{A.5}$$

Dans cet exemple, chaque unité PE gère 4 canaux de convolutions chacune en provenant des FIFO où chacune effectue la totalité des calculs en $U_{tot} - \frac{U_2}{2}$ coups d'horloge.

1. Illustré par une série de bascule D dans la figure A.3

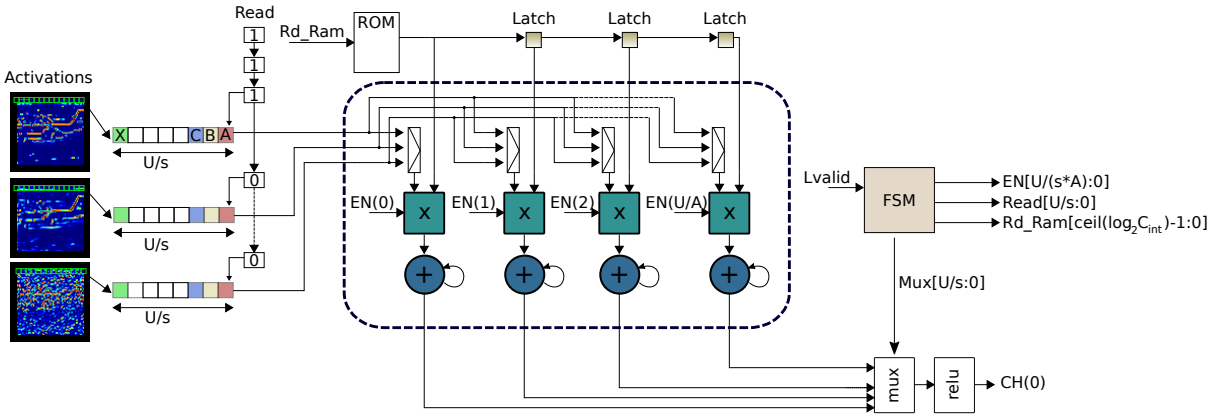


FIGURE A.3 – Illustration d'une unité mixte de convolution optimisée. La machine d'états principale commande la lecture des FIFO, le multiplexage pour livrer les données aux entrées des PE et le multiplexage pour commander l'ordre des sorties. En contrepartie, l'accélérateur utilise plus de mémoires et introduit une latence supplémentaire pour délivrer les résultats de convolutions.

Mobilenetv2($\rho = 0.5$), T_{hb} = 370 T_{clk}(MT9M031)	DHM (AGP = 50%)	PE/Channel	Flow-PE		
	Elements logiques		Elements logiques	RAM	Registres
L2(U = 56, C_{depth} = 48, C_{linear} = 8)	13490	4	6851 (-52%)	27648	4148
L4(U = 28, C_{depth}= 96, C_{linear} = 16)	28088	4	10121 (-64%)	36864	6644
L7(U = 14, C_{depth} = 96, C_{linear} = 32)	54360	2	16887(-68.9%)	36864	6931
L14(U = 7, C_{depth} = 288, C_{linear}= 80)	399247	4	44421 (-88.8%)	202752	18932

TABLE A.1

A.0.1 Résultats et analyse

Les résultats affichés dans le tableau A.1 démontre que les observations préliminaires faîtes par *Abdelouahab et al.* sont justes avec 2.2x, 1.96x ressources supplémentaires pour les sous modules couches du réseau non optimisées. Les optimisations amenées dans un contexte de sous-échantillonnage et une analyse plus poussée du fonctionnement du système permettent de réduire efficacement le nombre de ressources de la partie *linear bottleneck* d'une couche. Aux couches 4, 7 et 14 de MobileNetv2, ces nouveaux modules appellent respectivement 2x, 2.8x, 3.2x et 5.2x moins de ressources que leur équivalent DHM compressée. En contrepartie, cette nouvelle architecture invoque de la mémoire embarquée pour contenir les activations et les poids de convolutions.

Bibliographie

[AAY17] Reza Abbasi-Asl and Bin Yu. Structural Compression of Convolutional Neural Networks Based on Greedy Filter Pruning. pages 1–15, 2017.

[ABC+16] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, and Google Brain. TensorFlow : A System for Large-Scale Machine Learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation - OSDI '16*, pages 265–284, 2016.

[Abd19] Kamel Abdelouahab. Reconfigurable hardware acceleration of CNNs on FPGA-based smart cameras. 2019.

[ABP18] Kamel Abdelouahab, François Berry, and Maxime Pelcat. The Challenge of Multi-Operand Adders in CNNs on FPGAs : How not to solve it! pages 1–4, 2018.

[AJB+13] Hossein Afshari, Laurent Jacques, Luigi Bagnato, Alexandre Schmid, Pierre Vandergheynst, and Yusuf Leblebici. The PANOPTIC camera : A plenoptic sensor with real-time omnidirectional capability. *Journal of Signal Processing Systems*, 70(3) :305–328, 2013.

[ALE+11] Carl Ahlberg, Jörgen Lidholm, Fredrik Ekstrand, Giacomo Spampinato, Mikael Ekström, and Lars Asplund. GIMME - A General Image Multiview Manipulation Engine. *Proceedings - 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011*, pages 129–134, 2011.

[AOC+17] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C Ling, and Gordon R Chiu. An OpenCL(TM) Deep Learning Accelerator on Arria

10. In ACM, editor, *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*, pages 55–64, Monterey, California, USA, 2017. ACM.

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. page 37, 1968.

[APS+17a] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, and F. Berry. Tactics to Directly Map CNN Graphs on Embedded FPGAs. *IEEE Embedded Systems Letters*, 2017.

[APS+17b] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, Cedric Bourrasset, Jean-Charles Quinton, and François Berry. Hardware Automated Datafow Deployment of CNNs. Technical report, Universite Clermont Auvergne, 2017.

[ASS+18] Eman Ahmed, Alexandre Saint, Abd El Rahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamila Aouada, and Bjorn Ottersten. A survey on Deep Learning Advances on Different 3D Data Representations. 1(1) :1–35, 2018.

[BAIH09] Derek Bradley, Bradley Atcheson, Ivo Ihrke, and Wolfgang Heidrich. Synchronization and rolling shutter compensation for consumer video camera arrays. *2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, pages 1–8, 2009.

[BBD+19] Michael Broxton, Jay Busch, Jason Dourgarian, Matthew DuVall, Daniel Erickson, Dan Evangelakos, John Flynn, Ryan Overbeck, Matt Whalen, and Paul Debevec. A low cost multi-camera array for panoramic light field video capture. *SIGGRAPH Asia 2019 Posters, SA 2019*, pages 2018–2019, 2019.

[BBH03] Myron Z. Brown, Darius Burschka, and Gregory D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8) :993–1008, 2003.

[BHHS18] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *Advances in Neural Information Processing Systems*, 2018-Decem :5145–5153, 2018.

[BJM+06] Ignacio Bravo, Pedro Jiménez, Manuel Mazo, José Luis Lázaro, and Alfredo Gardel. Implementation in FPGAS of Jacobi method to solve the eigenvalue and eigenvector problem. *Proceedings - 2006 International Conference on Field Programmable Logic and Applications, FPL*, pages 729–732, 2006.

[BJTM16] Vassileios Balntas, Edward Johns, Lilian Tang, and Krystian Mikolajczyk. PN-Net : Conjoined Triple Deep Network for Learning Local Image Descriptors. 2016.

[BL07] Matthew Brown and David G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1) :59–73, 2007.

[BM98] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. *Proceedings of the Annual ACM Conference on Computational Learning Theory*, pages 92–100, 1998.

[BMK13] Tali Basha, Yael Moses, and Nahum Kiryati. Multi-view scene flow estimation : A view centered variational approach. *International Journal of Computer Vision*, 101(1) :6–21, 2013.

[BNHS18] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. 2018.

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF : Speeded Up Robust Features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.

[CDB15] Matthieu Courbariaux, Jean Pierre David, and Yoshua Bengio. Training deep neural networks with low precision multiplications. In *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, number Section 5, pages 1–10, 2015.

[CGSC16] Christopher B. Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal Correspondence Network. (Nips) :1–9, 2016.

[CPC16] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An Analysis of Deep Neural Network Models for Practical Applications. *arXiv e-print*, 2016.

[CX14] Jason Cong and Bingjun Xiao. Minimizing computation in convolutional neural networks. In *Proceedings of the International Conference on Artificial Neural Networks - ICANN '14*, pages 281–290. Springer, 2014.

[D.99] Lowe D. Object Recognition from Local Scale-Invariant Features. *ICCV*, 482 :35–40, 1999.

[DBN+20] Shail Dave, Riyadh Baghdadi, Tony Nowatzki, Sasikanth Avancha, Aviral Shrivastava, and Baoxin Li. Hardware Acceleration of Sparse and Irregular Tensor Computations of ML Models : A Survey and Insights. *arXiv*, pages 1–58, 2020.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '09*, pages 248–255. IEEE, 2009.

[DEN+20] Bilal Demir, Selman Ergunay, Gokcen Nurlu, Vladan Popovic, Beat Ott, Peter Wellig, Jean Philippe Thiran, and Yusuf Leblebici. Real-time high-resolution omnidirectional imaging platform for drone detection and tracking. *Journal of Real-Time Image Processing*, 17(5) :1625–1635, 2020.

[DFI+15] Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet : Learning optical flow with convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter :2758–2766, 2015.

[DRP+19] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. D2-Net : A Trainable CNN for Joint Detection and Description of Local Features. 2019.

[DT05] Navneet Dalal and Bill Triggs. Histogram of oriented gradients for human detection in video. *CVPR*, pages 172–176, 2005.

[ESH15] Ali Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. *WWW 2015 - Proceedings of the 24th International Conference on World Wide Web*, pages 278–288, 2015.

[FLG14] Simon Fuhrmann, Fabian Langguth, and Michael Goesele. MVE-A Multi-View Reconstruction Environment. *Eurographics Workshop on Graphics and Cultural Heritage (2014)*, pages 11–18, 2014.

[FOP+18] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. A Configurable Cloud-Scale DNN Processor for Real-Time AI. *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14, 2018.

[FZZ+18] Yifan Feng, Zizhao Zhang, Xibin Zhao, Rongrong Ji, and Yue Gao. GVCNN : Group-View Convolutional Neural Networks for 3D Shape Recognition. *Cvpr*, pages 264–272, 2018.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, 9 :249–256, 2010.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '14*, pages 580–587, 2014.

[Gir15] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision - ICCV '15*, pages 1440–1448, 2015.

[GLX+17] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. FP-DNN : An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates. In *Proceedings of the IEEE Annual International Symposium on Field-Programmable Custom Computing Machines - FCCM '17*, pages 152–159. IEEE, 2017.

[GPY+20] Adam Golinski, Reza Pourreza, Yang Yang, Guillaume Sautiere, and Taco S Cohen. Feedback Recurrent Autoencoder for Video Compression. pages 1–29, 2020.

[GYC16] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. *Advances in Neural Information Processing Systems*, pages 1387–1395, 2016.

[GZY+17] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. A Survey of FPGA Based Neural Network Accelerator. *arXiv e-print*, 2017.

[Har97] Richard I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6) :580–593, 1997.

[HGDG20] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2) :386–397, 2020.

[Hir07] Heiko Hirschm. SGM : Stereo Processing by Semi-Global Matching. *Tpami*, pages 1–14, 2007.

[HLB18] Loc N. Huynh, Youngki Lee, and Rajesh Krishna Balan. D-pruner : Filter-based pruning method for deep convolutional neural network. *EMDL 2018 - Proceedings of the 2018 International Workshop on Embedded and Mobile Deep Learning*, pages 7–12, 2018.

[HLJS15] Xufeng Han, Thomas Leung, Yangqing Jia, and Rahul Sukthankar. MatchNet : Unifying Feature and Metric Learning for Patch-Based Matching. *CVPR*, 2015.

[HLL11] Yong Seok Heo, Kyong Mu Lee, and Sang Uk Lee. Robust Stereo matching using adaptive normalized cross-correlation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(4) :807–822, 2011.

[HLM+16] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE : Efficient Inference Engine on Compressed Deep Neural Network. *Proceedings of the International Symposium on Computer Architecture - ISCA '16*, 16 :243–254, 2016.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua :2261–2269, 2017.

[HMD15] Song Han, Huizi Mao, and William J. Dally. Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. pages 1–14, 2015.

[HMD16] Song Han, Huizi Mao, and William J Dally. Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *Proceedings of the International Conference on Learning Representations - ICLR'16*, pages 1–13, 2016.

[HPN+16] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, Bryan Catanzaro, and William J. Dally. DSD : Dense-Sparse-Dense Training for Deep Neural Networks. 2016.

[HPTD15] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems - NIPS'15*, pages 1135–1143, 2015.

[HPTT16] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network Trimming : A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. 2016.

[HSST04] David R. Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis : An overview with application to learning methods. *Neural Computation*, 16(12) :2639–2664, 2004.

[HU16] Vishakh Hegde and Sheema Usmani. Parallel and Distributed Deep Learning. *Tech Report*, 2016.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *Advances in Neural Information Processing Systems - NIPS'15*, 2015.

[HZ04] Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[HZC+17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv e-print*, 2017.

[HZRS16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '16*, pages 770–778, 2016.

[HZRS16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision - ECCV'16*, pages 630–645. Springer, 2016.

[HZS06] Guang Bin Huang, Qin Yu Zhu, and Chee Kheong Siew. Extreme learning machine : Theory and applications. *Neurocomputing*, 70(1-3) :489–501, 2006.

[HZS17] Yihui He, Xiangyu Zhang, and Jian Sun. Channel Pruning for Accelerating Very Deep Neural Networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob :1398–1406, 2017.

[HZWK16] Xushen Han, Dajiang Zhou, Shihao Wang, and Shinji Kimura. CNN-MERP : An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks. *Proceedings of the 34th IEEE International Conference on Computer Design, ICCD 2016*, pages 320–327, 2016.

[IMA+16] Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, Song Han, William J Dally, and Kurt Keutzer. SqueezNet : AlexNet accuracy with 50x fewer parameters and 0.5MB Model Size. *arXiv e-print*, arXiv :1602 :1–5, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei, editors, *Proceedings of the International Conference on Machine Learning - ICML '15*, volume 37, pages 448–456, Lille, France, 2015.

[JLD16a] Edward Johns, Stefan Leutenegger, and Andrew J Davison. Pairwise Decomposition of Image Sequences for Active Multi-View Recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition - CVPR'16*, pages 3813–3822, 2016.

[JLD16b] Edward Johns, Stefan Leutenegger, and Andrew J. Davison. Pairwise Decomposition of Image Sequences for Active Multi-View Recognition. pages 3813–3822, 2016.

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe : Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the ACM International Conference on Multimedia - MM'14*, 2014.

[JZL+] J., A. Everingham M. Zisserman, VanGool L., Williams C. K. I., and Winn. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.

[Kan19] Hyeong-Ju Kang. Accelerator-Aware Pruning for Convolutional Neural Networks. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2019.

[KB14] Diederik P Kingma and Jimmy Ba. Adam : A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations - ICLR'15*, 2014.

[KGL+] Jin Hee Kim, Brett Grady, Ruolong Lian, John Brothers, and Jason H Anderson. FPGA-Based CNN Inference Accelerator Synthesized from Multi-Threaded C Software.

[KH92] Anders Krogh and John A. Hertz. A Simple Weight Decay Can Improve Generalization. *nips*, 1992.

[KMN16] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. RotationNet : Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints. 2016.

[KMT+17] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D : Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob :1530–1538, 2017.

[Kri09] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, Technical Report, University of Toronto, 2009.

[Kri12] Alex Krizhevsky. ImageNet Classification with Deep Convolutional Neural Networks. *American Journal of PharmacoGenomics*, 4(4) :253–262, 2 2012.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 1 ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, 2012.

[KVB18] Alexandros Kouris, Stylianos I. Venieris, and Christos Savvas Bouganis. Cascade C NN : Pushing the performance limits of quantisation in convolutional neural networks. *Proceedings - 2018 International Conference on Field-Programmable Logic and Applications, FPL 2018*, pages 155–162, 2018.

[KYHZ16] Liyanaarachchi Lekamalage Chamara Kasun, Yan Yang, Guang Bin Huang, and Zhengyou Zhang. Dimension Reduction with Extreme Learning Machine. *IEEE Transactions on Image Processing*, 25(8) :3906–3918, 2016.

[LBD+90] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems - NIPS'90*, pages 396–404, 1990.

[LC10] Yann Lecun and Cortes Corrina. MNIST handwritten digit database, 2010.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. *arXiv preprint*, arXiv :1312, 2013.

[LFG+18] Zhengfa Liang, Yiliu Feng, Yulan Guo, Hengzhu Liu, Wei Chen, Linbo Qiao, Li Zhou, and Jianfeng Zhang. Learning for Disparity Estimation Through Feature Constancy. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2811–2820, 2018.

[LG15] Andrew Lavin and Scott Gray. Fast Algorithms for Convolutional Neural Networks. *arXiv e-print*, arXiv : 150, 2015.

[LHB04] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2, 2004.

[LKD+17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient. *Iclr 2017*, (2016) :1–13, 2017.

[LLS+17] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient CNN through Network Slimming. *Iccv*, pages 2736–2744, 2017.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO : Common Objects in Context. In *Proceedings of the European Conference on Computer Vision - ECCV'14*. Springer, 2014.

[LQL+18] Hao Liu, Fangchao Qu, Yingjian Liu, Wei Zhao, and Yitong Chen. A drone detection with aircraft classification based on a camera array. *IOP Conference Series : Materials Science and Engineering*, 322(5), 2018.

[LRM15] Tsung Yu Lin, Aruni Roychowdhury, and Subhransu Maji. Bilinear CNN models for fine-grained visual recognition. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter :1449–1457, 2015.

[LSC17] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations. *arXiv e-print*, 2017.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '15*, pages 3431–3440, 2015.

[LW17] Jian-Hao Luo and Jianxin Wu. An Entropy-based Pruning Method for CNN Compression. 2017.

[LXH+19] Liqiang Lu, Jiaming Xie, Ruirui Huang, Jiansong Zhang, Wei Lin, and Yun Liang. An efficient hardware accelerator for sparse convolutional neural networks on FPGAS. *Proceedings - 27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*, pages 17–25, 2019.

[LXPX18] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. Frequency-domain dynamic pruning for convolutional neural networks. *Advances in*

Neural Information Processing Systems, 2018-Decem(NeurIPS) :1043–1053, 2018.

[LYZ19] Yingming Li, Ming Yang, and Zhongfei Zhang. A Survey of Multi-View Representation Learning. *IEEE Transactions on Knowledge and Data Engineering*, 31(10) :1863–1883, 2019.

[LZZ+18] Jian Hao Luo, Hao Zhang, Hong Yu Zhou, Chen Wei Xie, Jianxin Wu, and Weiyao Lin. ThiNet : Pruning CNN Filters for a Thinner Net. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(8) :1, 2018.

[MCVS17] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*, pages 45–54, 2017.

[MHK+15] Abraham Monrroy, Manato Hirabayashi, Shinpei Kato, Masato Edahiro, Takefumi Miyoshi, and Satoshi Funada. HexaCam : An FPGA-based Multi-view Camera System. 2015.

[MHP+17] Huizi Mao, Song Han, Jeff Poo, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv*, (Nips) :1–10, 2017.

[Mit18] Sparsh Mittal. *A survey of FPGA-based accelerators for convolutional neural networks*, volume 32. Springer London, 2018.

[MNCM17] Asit Mishra, Eriko Nurvitadhi, Jeffrey J. Cook, and Debbie Marr. WRPN : Wide reduced-precision networks. *arXiv*, pages 1–11, 2017.

[MS15] Daniel Maturana and Sebastian Scherer. VoxNet : A 3D Convolutional Neural Network for Real-Time Object Recognition. pages 922–928, 2015.

[MTS+19] Iaroslav Melekhov, Aleksei Tiulpin, Torsten Sattler, Marc Pollefeys, Esa Rahtu, and Juho Kannala. DGC-Net : Dense geometric correspondence network. *Proceedings - 2019 IEEE Winter Conference on Applications of Computer Vision, WACV 2019*, pages 1034–1042, 2019.

[MZ18] Suyog Gupta Michael Zhu. TO PRUNE, OR NOT TO PRUNE : EXPLORING THE EFFICACY OF PRUNING FOR MODEL COMPRESSION. *ICLR 2018*, 51(22-24) :4478–4488, 2018.

[MZKD19] Bradley McDanel, Sai Qian Zhang, H. T. Kung, and Xin Dong. Full-stack optimization for accelerating CNNs using powers-of-two weights with FPGA validation. *Proceedings of the International Conference on Supercomputing*, pages 449–460, 2019.

[MZZS18] Ningning Ma, Xiangyu Zhang, Hai Tao Zheng, and Jian Sun. Shufflenet V2 : Practical guidelines for efficient cnn architecture design. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11218 LNCS :122–138, 2018.

[NSB+17] Eriko Nurvitadhi, Suchit Subhaschandra, Guy Boudoukh, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason OngGeeHock, Yeong Tat Liew, Krishnan Srivatsan, and Duncan Moss. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks ? In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*, pages 5–14, 2017.

[PASL13] Vladan Popovic, Hossein Afshari, Alexandre Schmid, and Yusuf Leblebici. Real-time implementation of Gaussian image blending in a spherical light field camera. *Proceedings of the IEEE International Conference on Industrial Technology*, pages 1173–1178, 2013.

[PCL+17] Vladan Popovic, Ömer Cogal, Yusuf Leblebici, Kerem Seyid, and Abdulkadir Akin. Design and implementation of real-time multi-sensor vision systems. *Design and Implementation of Real-Time Multi-Sensor Vision Systems*, pages 1–257, 2017.

[PGC17] Adam Paszke, Sam Gross, and Soumith Chintala. Automatic differentiation in pytorch. *nips*, 2017.

[Pho75] Bui Tuong Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, 18(6) :311–317, 1975.

[PPK03] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang. Super-resolution image reconstruction : A technical overview. *IEEE Signal Processing Magazine*, 20(3) :21–36, 2003.

[PPTM19] Matteo Poggi, Davide Pallotti, Fabio Tosi, and Stefano Mattoccia. Guided Stereo Matching. 2019.

[Qi] Charles R Qi. PointNet : Deep Learning on Point Sets for 3D Classification and Segmentation.

[QWY+16] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*, pages 26–35, New York, NY, USA, 2016. ACM.

[RAS17] Ignacio Rocco, Relja Arandjelovic, and Josef Sivic. Convolutional neural network architecture for geometric matching. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 39–48, 2017.

[RCA+18] Ignacio Rocco, Mircea Cimpoi, Relja Arandjelović, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Neighbourhood Consensus Networks. (NeurIPS), 2018.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once : Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '16*, 2016.

[RDS+14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3) :211–252, 2014.

[RF16] Joseph Redmon and Ali Farhadi. YOLO9000 : Better, Faster, Stronger. *arXiv e-print*, 2016.

[RHW86] D.E Rumelhart, G.E Hinton, and R.J Williams. Learning Internal Representations By Error Propagation (original), 1986.

[Roa11] Kahuta Road. Eigenvalues Calculation. 7(10) :5939–5946, 2011.

[RORF16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-net : Imagenet classification using binary convolutional

neural networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9908 LNCS :525–542, 2016.

[RSP19] Siamak Ravanbakhsh, Jeff Schneider, and Barnabás Póczos. Deep learning with sets and point clouds. *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, pages 1–12, 2019.

[SDC+19] Benoit Steiner, Zachary Devito, Soumith Chintala, Sam Gross, Adam Paszke, Francisco Massa, Adam Lerer, Gregory Chanan, Zeming Lin, Edward Yang, Alban Desmaison, Alykhan Tejani, Andreas Kopf, James Bradbury, Luca Antiga, Martin Raison, Natalia Gimelshein, Sasank Chilamkurthy, Trevor Killeen, Lu Fang, and Junjie Bai. PyTorch : An Imperative Style, High-Performance Deep Learning Library. *NeuroIPS*, (NeurIPS), 2019.

[SFM17] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing CNN accelerator efficiency through resource partitioning. *Proceedings - International Symposium on Computer Architecture*, Part F1286(c) :535–547, 2017.

[Sha] Lin Shao. Neural Network for 3D object classification. pages 1–6.

[SHZ+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2 : Inverted Residuals and Linear Bottlenecks. 2018.

[SMBM15] Baoguang Shi, Student Member, Song Bai, and Student Member. DeepPano : Deep Panoramic Representation for 3-D Shape Recognition. 22(12) :2339–2343, 2015.

[SMC+] Vivienne Sze, Senior Member, Yu-hsin Chen, Student Member, and Tienju Yang. Efficient Processing of Deep Neural Networks : A Tutorial and Survey. pages 1–32.

[SMKLm15a] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learnedmiller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. *Ieee Iccv*, pages 945–953, 2015.

[SMKLM15b] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *Proceedings of the IEEE International Conference on Computer Vision - ICCV '15*, pages 945–953. IEEE, 2015.

[SP10] Sudipta N. Sinha and Marc Pollefeys. Camera network calibration and synchronization from silhouettes in archived video. *International Journal of Computer Vision*, 87(3) :266–283, 2010.

[SP11] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1665–1672, 2011.

[SSZ01] D. Scharstein, R. Szeliski, and R. Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Proceedings - IEEE Workshop on Stereo and Multi-Baseline Vision, SMBV 2001*, (1) :131–140, 2001.

[Sun13] Shiliang Sun. A survey of multi-view machine learning. pages 2031–2038, 2013.

[SWSD15] Ricardo M. Sousa, Martin Wäny, Pedro Santos, and Morgado Dias. Multi-camera synchronization core implemented on USB3 based FPGA platform. *Image Sensors and Imaging Systems 2015*, 9403 :940306, 2015.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint*, arXiv :1409 :1–14, 2014.

[SZAB17] Nima Sedaghat, Mohammadreza Zolfaghari, Ehsan Amiri, and Thomas Brox. Orientation-boosted Voxel nets for 3D object recognition. *British Machine Vision Conference 2017, BMVC 2017*, pages 1–18, 2017.

[TCP+18] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet : Platform-Aware Neural Architecture Search for Mobile. 2018.

[Tec18] Virginia Tech. ATTENTION-BASED GUIDED STRUCTURED SPARSITY. pages 1–4, 2018.

[TL19] Mingxing Tan and Quoc V. Le. EfficientNet : Rethinking model scaling for convolutional neural networks. *36th International Conference on Machine Learning, ICML 2019*, 2019-June :10691–10700, 2019.

[VKB18a] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. Toolflows for Mapping Convolutional Neural Networks on FPGAs. *ACM Computing Surveys*, 51(3) :1–39, 2018.

[VKB18b] Stylianos I Venieris, Alexandros Kouris, and Christos-savvas Bouganis. Toolflows for Mapping Convolutional Neural Networks on FPGAs : A Survey and Future Directions. 0(0), 2018.

[Wan17] Dong Wang. PipeCNN : An OpenCL-based FPGA Accelerator for Convolutinal Neural Networks. In *Proceedings of the International Conference on Field-Programmable Technology - FPT '17*, 2017.

[Wil04] Bennett Wilburn. High Performance Imaging Using Arrays of Inexpensive Cameras. (December) :128, 2004.

[Wil06] Bennett Wilburn. High-performance imaging with large camera arrays. *SIGGRAPH 2006 - ACM SIGGRAPH 2006 Courses*, 1(212) :45–113, 2006.

[WLH] Bennett Wilburn, Marc Levoy, and Mark Horowitz. High-Speed Videography Using a Dense Camera Array 2 . Previous Work 3 . High-Speed Videography Using An Array of Cameras.

[WPS19] Chu Wang, Marcello Pelillo, and Kaleem Siddiqi. Dominant Set Clustering and Pooling for Multi-View 3D Object Recognition. pages 1–12, 2019.

[WS] Zhirong Wu and Shuran Song. 3D ShapeNets : A Deep Representation for Volumetric Shapes.

[WWW+16] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in Neural Information Processing Systems*, pages 2082–2090, 2016.

[WZL15] Songtao Wu, Shenghua Zhong, and Yan Liu. ResNet. *CVPR*, 2015.

[XTX13] Chang Xu, Dacheng Tao, and Chao Xu. A Survey on Multi-view Learning. pages 1–59, 2013.

[XXS+15] Zhige Xie, Kai Xu, Wen Shan, Ligang Liu, Yueshan Xiong, and Hui Huang. Projective Feature Learning for 3D Shapes with Multi-View Depth Images. In *Computer Graphics Forum*, 2015.

[Yan97] Yann Le Cun. OBD(Optimal Brain Damage). 9(1) :50, 1997.

[YGW+19] Xiaoyu Yu, Jianlin Gao, Yuwei Wang, Jie Miao, Ephrem Wu, Heng Zhang, Yu Meng, Bo Zhang, Biao Min, and Dewei Chen. A data-center FPGA acceleration platform for convolutional neural networks. *Proceedings - 29th International Conference on Field-Programmable Logic and Applications, FPL 2019*, pages 151–158, 2019.

[YLC+18] Ruichi Yu, Ang Li, Chun Fu Chen, Jui Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching Yung Lin, and Larry S. Davis. NISP : Pruning Networks Using Neuron Importance Score Propagation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[YLL+18] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. MVSNet : Depth inference for unstructured multi-view stereo. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11212 LNCS :785–801, 2018.

[YLLW18] Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers. (2017) :1–11, 2018.

[YTZW18] Zhi Xin Yang, Lulu Tang, Kun Zhang, and Pak Kin Wong. Multi-View CNN Feature Aggregation with ELM Auto-Encoder for 3D Shape Recognition. *Cognitive Computation*, 10(6) :908–921, 2018.

[YWT12] Jun Yu, Meng Wang, and Dacheng Tao. Semisupervised multiview distance metric learning for cartoon synthesis. *IEEE Transactions on Image Processing*, 21(11) :4636–4648, 2012.

[YZT+19] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing Hao Xue, and Qingmin Liao. Deep Learning for Single Image Super-Resolution : A Brief Review. *IEEE Transactions on Multimedia*, 21(12) :3106–3121, 2019.

[ZC05] Cha Zhang and Tsuhan Chen. Multi-view imaging : Capturing and rendering interactive environments. *Computer Vision for Interactive and Intelligent Environments 2005*, 2005 :51–67, 2005.

[ZDG+18] Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. Cambricon-S : Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2018-Octob :15–28, 2018.

[ZG17] Michael Zhu and Suyog Gupta. To prune, or not to prune : exploring the efficacy of pruning for model compression. 2017.

[ZH19] Chaoyang Zhu and Kejie Huang. An efficient hardware accelerator for sparse convolutional neural networks on FPGAS. *Proceedings - 27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*, pages 17–25, 2019.

[ZJZ+19] Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. Neural Network Distiller : A Python Package For DNN Compression Research. 2019.

[ŽL16] Jure Žbontar and Yann Lecun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17 :1–32, 2016.

[ZLS+15a] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*, FPGA, pages 161–170, 2015.

[ZLS+15b] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. *Proc. 2015 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays - FPGA '15*, pages 161–170, 2015.

[ZXXS17] Jing Zhao, Xijiong Xie, Xin Xu, and Shiliang Sun. Multi-view learning overview : Recent progress and new challenges. *Information Fusion*, 38 :43–54, 2017.

[ZZLS18] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet : An Extremely Efficient Convolutional Neural Network for Mobile Devices. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.