Codages optimisés pour la conception d'accélérateurs matériels de réseaux de neurones profonds

Clément Metz

université PARIS-SACLAY

Codages optimisés pour la conception d'accélérateurs matériels de réseaux de neurones profonds

Optimized coding techniques for the design of deep neural network hardware accelerators

Thèse de doctorat de l'université Paris-Saclay

École doctorale n◦ 575, electrical, optical, bio : physics and engineering (EOBE)

Spécialité de doctorat : Sciences de l'information et de la communication
Graduate School : Informatique et sciences du numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche **Institut LIST** (Université Paris-Saclay, CEA), sous la direction d'**Antoine DUPRET**, Docteur (HDR), et le co-encadrement d'**Olivier BICHLER**, Docteur

Thèse soutenue à Paris-Saclay, le 18 décembre 2023, par

Clément METZ

Composition du jury

Membres du jury avec voix délibérative

Sylvie LE HEGARAT-MASCLE Professeur, Université Paris-Saclay	Présidente
Eva DOKLADALOVA Professeur, ESIEE - Université Paris-Est	Rapporteur & Examinatrice
Frédéric PETROT Professeur, ENSIMAG - Grenoble INP - Université de Grenoble	Rapporteur & Examinateur
Maxime PELCAT Docteur (HDR), INSA Rennes - Université Clermont Auvergne	Examinateur

THESE DE DOCTORAT
NNT : 2023UPAST190

université PARIS-SACLAY

ÉCOLE DOCTORALE
Physique et ingénierie : Electrons, Photons, Sciences du vivant (EOBE)

Titre : Codages optimisés pour la conception d'accélérateurs matériels de réseaux de neurones profonds

Mots clés : Intelligence artificielle, réseaux neuronaux, vision par ordinateur, codage, quantification

Résumé : Par leurs domaines d'application très divers (santé, énergie, défense, finance, navigation autonome...), les réseaux de neurones constituent une composante importante des outils d'apprentissage automatique. Les performances des réseaux de neurones sont grandement influencées par la complexité de leur architecture en nombre de couches, de neurones et de connexions. Mais l'entraînement et l'inférence de réseaux de plus en plus grands implique une sollicitation croissante de ressources matérielles et des temps de calcul plus longs. À l'inverse, leur portabilité se retrouve bridée sur des systèmes embarqués aux faibles capacités mémoire et/ou calculatoire.

L'objectif de cette thèse est d'étudier et de concevoir des méthodes permettant de réduire l'empreinte matérielle des réseaux de neurones tout en préservant au mieux leurs performances. Nous nous restreignons aux réseaux de convolution dédiés à la vision par ordinateur en étudiant les possibilités offertes par la quantification. La quantification vise à réduire l'empreinte matérielle des réseaux en mémoire, en bande passante et en opérateurs de calculs, par la réduction du nombre de bits des paramètres et des activations.

Les contributions de cette thèse consistent en une nouvelle méthode de quantification post-entraînement reposant sur l'exploitation des corrélations spatiales des paramètres du réseau, une approche facilitant l'apprentissage des réseaux très fortement quantifiés, ainsi qu'une méthode visant à combiner la quantification en précision mixte et le codage entropique sans perte.

Cette thèse se limite essentiellement aux aspects algorithmiques, mais les orientations de recherche ont été fortement influencées par la contrainte de faisabilité matérielle des propositions apportées.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Physique et ingénierie : Electrons, Photons, Sciences du vivant (EOBE)

Title : Optimized coding techniques for the design of deep neural network hardware accelerators
Keywords : Artifical intelligence, neural networks, computer vision, coding, quantization

Abstract : Neural networks are an important component of machine learning tools because of their wide range of applications (health, energy, defence, finance, autonomous navigation, etc.). The performance of neural networks is greatly influenced by the complexity of their architecture in terms of the number of layers, neurons and connections. But the training and inference of ever-larger networks translates to greater demands on hardware resources and longer computing times. Conversely, their portability is limited on embedded systems with low memory and/or computing capacity.

The aim of this thesis is to study and design methods for reducing the hardware footprint of neural networks while preserving their performance as much as possible. We restrict ourselves to convolution networks dedicated to computer vision by studying the possibilities offered by quantization. Quantization aims to reduce the hardware footprint, in terms of memory, bandwidth and computation operators, by reducing the number of bits in the network parameters and activations.

The contributions of this thesis consist of a new post-training quantization method based on the exploitation of spatial correlations of network parameters, an approach facilitating the learning of very highly quantized networks, and a method aiming to combine mixed precision quantization and lossless entropy coding.

The contents of this thesis are essentially limited to algorithmic aspects, but the research orientations were strongly influenced by the requirement for hardware feasibility of our solutions.

Table des matières

1 - Introduction

L'intelligence artificielle (IA) au sens large est présente depuis des décennies dans notre environnement et dans nos activités sous diverses formes. Elle est notamment au coeur de récentes avancées techniques aux impacts multiples sur la vie quotidienne et professionnelle de chacun. Des applications spectaculaires l'ont mise sur le devant de la scène : on pense aux Large Language Models (LLM) tels que ChatGPT [Liu et al., 2023], dont l'irruption a bousculé le champ d'utilisation de l'IA, en la faisant passer de l'arrière-plan à l'usage direct et conscient par l'utilisateur non spécialiste.

Mais l'IA est une technologie coûteuse. Les ressources mémoire (stockage comme bande passante) et calculatoires nécessaires pour utiliser, et plus encore pour entraîner un modèle de langage tel que GPT4 [OpenAI, 2023] sont prodigieuses : plusieurs mois d'entraînement sur des centaines – sinon des milliers – de processeurs graphiques (Graphics Process Units, GPUs) de dernière génération à plus de 10.000\$ pièce, pour environ 2×10^{11} paramètres (~ 1 To). Mais ce n'est pas tout, car il faut ensuite pouvoir accéder suffisamment rapidement à ces paramètres au moment de l'inférence, ce qui suppose des contraintes importantes en bande passante sur le matériel. Le débit d'un disque SSD standard s'établit entre 0.5 et 5 Go/s, soit au moins 200 secondes pour accéder à l'ensemble des paramètres de GPT4. Une barre de RAM avoisine quant à elle les 50 Go/s, mais ses capacités de stockage sont plus restreintes. Anticiper les problématiques d'usage et de déploiement de ces outils ainsi que d'en réduire le coût énergétique s'impose donc dans un but de sobriété énergétique comme de faisabilité technique.

Les capacités calculatoires disponibles et les performances atteignables des modèles d'IA ont toujours été fortement liées. Aucune des récentes avancées en langage, en reconnaissance vocale ou en vision n'aurait pu aboutir sans un usage intensif du calcul parallèle. Mais le tassement de la loi de Moore, qui se heurte aux limites de la physique, tend à faire penser qu'il pourrait un jour devenir difficile de continuer à augmenter les capacités du matériel. Un changement de paradigme se profile donc, et l'IA ne pourra probablement plus se contenter de la course au gigantisme (tant des modèles que des supercalculateurs) dans laquelle sont engagés les géants du secteur.

Les problématiques de capacité du matériel sont d'ores et déjà bien connues dans le contexte embarqué. La conduite autonome ne peut par exemple pas s'accommoder de temps de latence qui excèdent le temps de réaction hu-

main, ce qui impose un traitement embarqué des données des capteurs du véhicule de sorte à éliminer les délais de transmission concomitants d'un traitement cloud. L'ordinateur de bord doit donc être capable d'analyser lui-même ces données en temps réel, et de le faire avec précision. Pour d'autres applications, les données ont un caractère sensible : c'est le cas de la reconnaissance faciale. Le respect de la vie privée de l'utilisateur peut imposer, selon la législation [UE, 2016], un traitement des données personnelles localisé sur le terminal et non sur un serveur.

1.1 . Deep learning embarqué

1.1.1 . Enjeux de compression et d'accélération

La demande croissante des applications en temps réel et basse consommation pose de nouveaux défis pour le déploiement matériel de réseaux neuronaux. Dans le matériel embarqué, les ressources telles que la mémoire, la puissance de calcul et la bande passante sont souvent plus significativement limitées que dans les fermes de serveurs. La compression et l'accélération des réseaux de neurones sont donc devenus des thèmes de recherche centraux, cruciaux pour surmonter ces défis et rendre le deep learning accessible à un plus grand nombre de supports et d'applications.

Les avancées en matière d'architectures neuronales et de possibilités d'entraînement (disponibilité de données en très grandes quantités, calcul haute performance) permettent l'utilisation de modèles de plus en plus grands et complexes (Figure 1.2), démontrant leurs capacités inédites.

De fait, augmenter la taille d'un modèle en ajoutant des couches, des neurones ou des connexions est généralement un moyen simple et efficace d'en améliorer les performances. Les modèles de vision les plus récents tels qu'EfficientNet [Tan and Le, 2020], ConvNext [Liu et al., 2022b] ou ViT [Dosovitskiy et al., 2021], sollicitent des centaines de gigaoctets de mémoire à l'entraînement et possèdent plusieurs dizaines de millions de paramètres. Déployer de tels réseaux sur des plateformes embarquées peut s'avérer impossible en pratique, à moins de faire appel à des techniques de compression et d'accélération pour réduire leur consommation.

En effet, les contraintes inhérentes aux matériels à ressources limitées posent un défi technique supplémentaire pour le déploiement des réseaux. Les choix de matériels spécialisés doivent s'adapter aux contraintes de consommation énergétique, de coût et d'espace disponible. Comparés aux GPUs, généralement utilisés pour l'entraînement et très flexibles dans leurs usages, les FPGAs (field-programmable gate array) et les ASICs (Application-Specific

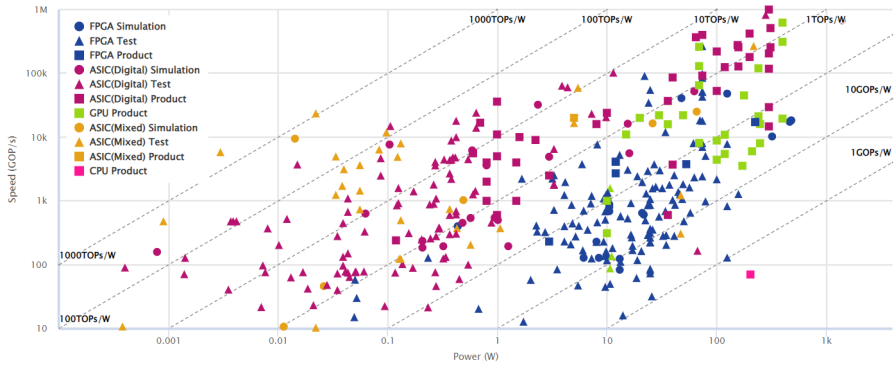

Figure 1.1 – Comparaison d'une collection d'accélérateurs de réseaux de neurones [Guo et al.]

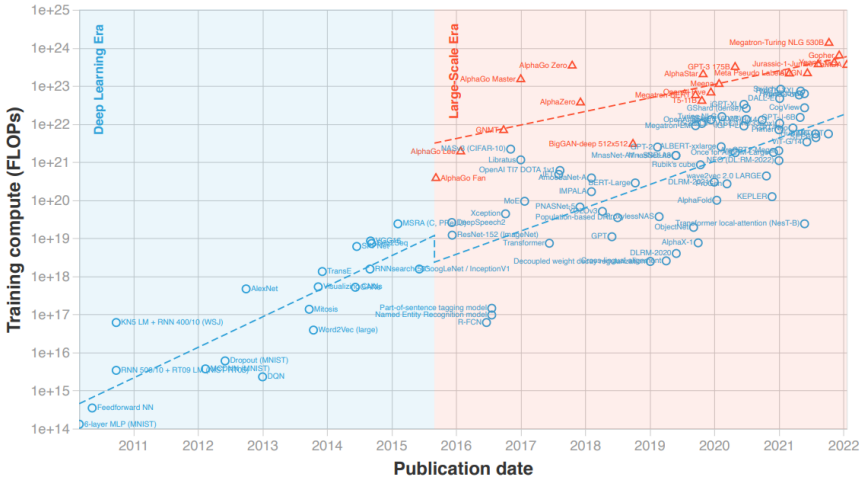

Figure 1.2 – Évolution de la complexité d'entraînement de certains réseaux de l'état de l'art en fonction du temps [Sevilla et al., 2022]

Integrated Circuit) offrent des qualités appréciables pour l'accélération et l'efficacité énergétique de l'inférence (cf Figure 1.1) mais nécessitent des efforts de design et de programmation plus poussés, ce qui peut faire flamber le temps de développement et le coût du système.

1.1.2 . Importance du choix de la cible matérielle

Bien souvent, c'est le choix d'une cible matérielle qui dicte un choix d'une méthode de compression, et non l'inverse. Dans le domaine de la compression des réseaux de neurones, il n'existe pas de benchmark unique d'évaluation. Il existe au contraire une variété d'objectifs et de contraintes, qui évoluent en fonction de la cible matérielle et des performances visées : il est évident qu'un réseau utilisé sur le cloud pour la reconnaissance faciale sur Facebook [Taigman et al., 2014] ne peut avoir les mêmes spécifications ni le

même budget de ressources qu'un réseau utilisé pour identifier des particules à haute énergie au LHC [Alimena et al., 2020].

Exemple du TPU

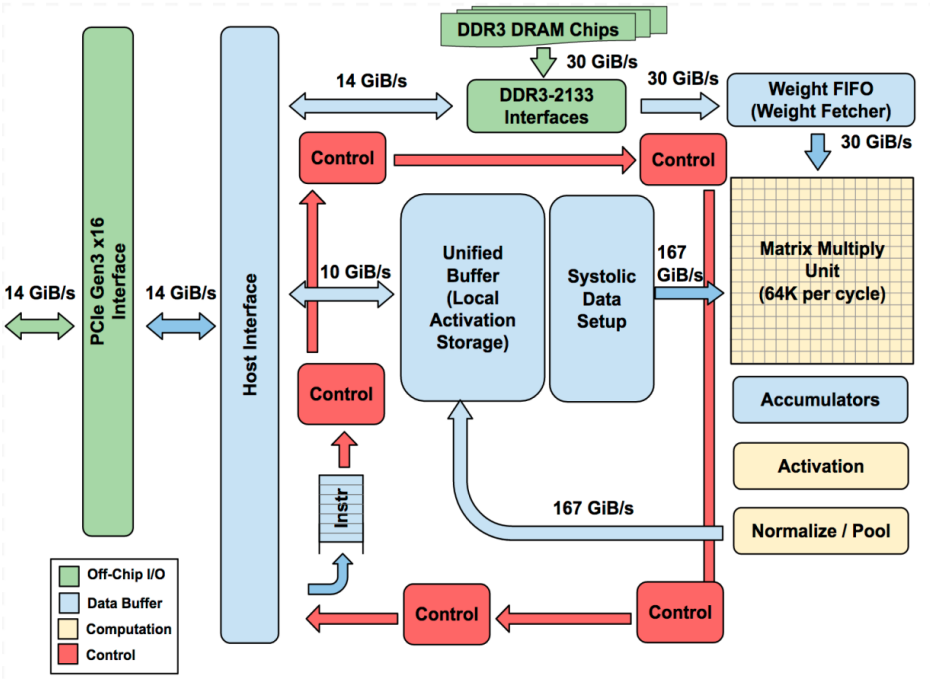

Figure 1.3 – Diagramme du TPU [Jouppi et al., 2017]

Le TPU (Tensor Processing Unit) [Jouppi et al., 2017] est un bon exemple de cible matérielle pour l'inférence. Utilisé sur les serveurs cloud de Google, il s'agit d'un ASIC dédié à l'accélération de réseaux de neurones (voir Figure 1.3). Deux facteurs limitent la performance du TPU :

- La puissance de calcul disponible (notamment pour la Matrix Multiply Unit), qui détermine le nombre maximal d'opérations pouvant être effectuées par unité de temps.
- La vitesse du transfert de données (notamment en provenance de la mémoire DRAM), qui détermine la bande passante du TPU.

Parmi ces deux facteurs, celui qui borne en pratique les performances diffère en fonction du type d'application (voir Figure 1.4).

Figure 1.4 – Performance du TPU par application [Jouppi et al., 2017]

Sur cette figure, on place en abscisse le nombre d'opérations effectuées par octet de paramètres du réseau, et en ordonnée le nombre de téraopérations par seconde. Les points de fonctionnement atteignables du TPU sont délimités par une courbe plafond (en bleu), et on place ceux de différentes applications sur le graphe (des LSTM, des MLP et des CNN). On note deux zones de contraintes :

- La partie diagonale de la courbe, où la quantité de calculs par seconde peut augmenter avec le nombre d'opérations : le système n'est pas limité par les calculs, il est donc limité par la bande passante.
- La partie horizontale de la courbe, où la quantité de calculs par seconde atteint un plafond : le système est limité par les calculs, et pas par la bande passante.

Ainsi, la nature de la limitation matérielle (bande passante des canaux de communications, ressources mémoire) dépend du type de matériel utilisé et de l'application. Rien ne sert par exemple de réduire la complexité calculatoire si c'est la bande passante qui est limitante. Il faut donc prendre en compte tous ces facteurs au moment du choix d'une technique de compression.

1.2 . Métriques, plateformes, benchmarks

1.2.1 . Métriques

Plusieurs métriques coexistent pour évaluer le coût d'utilisation d'un réseau. En fonction des travaux, on privilégiera la taille mémoire totale à l'inférence, la taille mémoire une fois le fois le réseau compressé (pour transmettre ses paramètres sur un canal de communication), son nombre de multiplieurs-accumulateurs (Multiply-Accumulate, MACs), ou encore le nombre d'opérations à virgule flottante (FLoating-Point Operations per Second, FLOPS) nécessaires à son inférence. Ces métriques permettent d'estimer la complexité d'une architecture, se traduisant par une latence ou un coût énergétique plus ou moins importants.

1.2.2 . Plateformes

L'entraînement s'effectue généralement en amont du déploiement, même si certains travaux [Lin et al., 2022] démontrent la faisabilité d'un entraînement directement sur matériel. Pour entraîner les réseaux, on utilise principalement deux plateformes de calcul : PyTorch (Meta) [Paszke et al., 2019] et Tensorflow (Google) [Abadi et al., 2016]. On passe ensuite par des systèmes d'export tels que TensorRT pour effectuer la transition vers le matériel. Il existe également des plateformes directement orientées déploiement : N2D2 (CEA), hls4ml [Fahim et al., 2021] (CERN), ou encore OpenVINO [Gorbachev et al., 2019] (Intel), qui offrent leurs propres fonctionnalités d'export.

1.2.3 . Benchmarks

Dans le domaine de la vision par ordinateur, il existe des benchmarks reconnus pour l'évaluation sur certaines tâches, consistant en des datasets d'images. On présente succinctement les principaux benchmarks pour la reconnaissance d'images, tâche la mieux représentée dans l'état de l'art de la compression.

MNIST [Lecun et al., 1998] est une base d'images de chiffres écrits à la main et numérisés en niveaux de gris, avec une résolution de 28 par 28 pixels. Créé en 1994, c'est le dataset le plus célèbre en vision par ordinateur. Victime de sa simplicité, MNIST est de moins en moins utilisé en pratique : l'état de l'art de son taux de reconnaissance avoisine les 99.9%. De nos jours, on lui préfère des datasets avec plus de classes et des images de plus haute résolution.

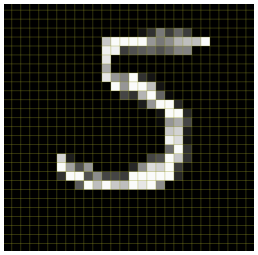

Figure 1.5 – Exemple d'image issue de MNIST.

CIFAR [Krizhevsky, 2009] est une base d'images de la vie courante, en couleurs, de résolution 32 par 32. Il vient en deux versions : CIFAR10 et CIFAR100, comprenant respectivement 10 et 100 classes.

Figure 1.6 – Exemple d'image issue de CIFAR100.

Enfin, ImageNet [Russakovsky et al., 2015] est une base de plus d'un million d'images de résolution plus élevée que les datasets précédents, qui n'en contiennent quant à eux que quelques dizaines de milliers. Il est constitué d'images tierces prélevées sur internet et réparties en 1000 classes annotées. De 2010 à 2017, il a été au coeur du challenge ILSVRC (ImageNet Large Scale Visual Recognition Challenge), qui a marqué le début de l'ère des réseaux de convolution pour la reconnaissance d'images [Krizhevsky et al., 2017].

Figure 1.7 – Exemple d'image issue d'ImageNet.

La performance d'un réseau sur la tâche de reconnaissance d'images s'évalue en fonction de la précision Top-k (Top-k Accuracy) sur un échantillon d'images test non aperçues lors de l'entraînement. Le réseau renvoie une distribution de probabilité d'appartenance aux classes du dataset, et si l'image fait partie des k premières, elle est considérée comme bien reconnue. On calcule ensuite le ratio entre le nombre d'images de test bien reconnues et le nombre total d'images, ce qui donne la précision Top-k. On utilise très souvent la précision Top-1, plus rarement la Top-5.

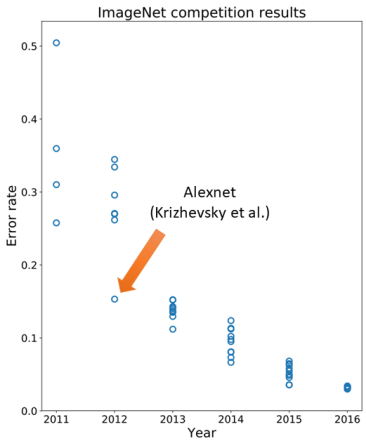

Figure 1.8 – Performance des 10 meilleures participations à ILSVRC par année [Wikipedia, 2018] (Error rate $= 1-$ Top-5 Accuracy)

1.3 . Entraînement

1.3.1 . Hyperparamètres et généralisation

L'entraînement d'un réseau de neurones repose généralement sur l'algorithme de rétropropagation du gradient (Algorithme 1) [Rumelhart et al., 1986] par rapport à une fonction de coût (loss) qui calcule la distance entre le résultat attendu et le résultat renvoyé par le modèle. La mise à jour des paramètres du réseau est conditionnée par un optimiseur, qui peut être le SGD (Stochastic Gradient Descent) [Ruder, 2017], Adam [Kingma and Ba, 2017], ou encore RMSprop [Hinton, 2012]. Chacun de ces optimiseurs possède des "hyperparamètres" qui sont des paramètres qui conditionnent l'évolution de leur état et de l'entraînement.

L'hyperparamètre le plus connu est le taux d'apprentissage (learning rate) noté α qui conditionne l'amplitude de mise à jour des paramètres. La sélection de α est souvent effectuée par une recherche de type grid search (on compare l'évolution de la loss de validation sur quelques époques et on choisit le α qui la fait diminuer le plus vite). On peut également citer le momentum, noté μ, qui ajoute de l'inertie à la mise à jour des paramètres : si un paramètre a été réduit à la mise à jour précédente, il aura tendance à être réduit également par la suite.

Le choix des hyperparamètres est un processus long qui demande beaucoup de travail d'expérimentation, car la plupart du temps, la seule manière de savoir si un choix convient est d'entraîner le réseau. L'évaluation correcte des performances d'un modèle dépend donc grandement de la rigueur de l'expérimentateur et de l'ajustement correct des hyperparamètres.

La capacité de généralisation du réseau entraîné désigne sa capacité à utiliser son apprentissage sur de nouvelles données non rencontrées lors de l'entraînement. Si cette capacité est trop faible, on est alors en présence d'un phénomène de surapprentissage (overfitting). On peut influencer la capacité de généralisation grâce aux data augmentations (ensemble de transformations que l'on applique aux données du dataset d'entraînement pour les rendre plus générales) [Perez and Wang, 2017], ou à la régularisation. L'objectif de la régularisation est d'empêcher une spécialisation trop importante du réseau. En exemples de techniques de régularisation, on peut lister le Dropout [Srivastava et al., 2014] (suppression temporaire et aléatoire de connexions à l'apprentissage), le weight decay [Krogh and Hertz, 1991] (pénalité en norme L2 sur l'amplitude des poids), ou encore le label smoothing [Müller et al., 2019] (altération légère des labels pour lisser le paysage de la loss).

1.3.2 . Limites d'apprentissage

Les modèles de réseaux de neurones existants sont très inférieurs au cerveau humain desquels ils s'inspirent, à la fois du point de vue de la plasticité (apprentissage online), de la diversité des usages, comme de l'efficacité énergétique : en effet, le cerveau humain est capable d'apprendre toutes de sortes de tâches chaque jour et fonctionne à une puissance de 20W, tandis qu'un A100 de Nvidia nécessite 250W de puissance sur deux jours pour entraîner un simple Resnet à reconnaître des images. Un tel écart s'explique de plusieurs manières :

- Le cerveau comporte bien plus de neurones ($\sim 10^{11}$) [Herculano-Houzel, 2009] qu'un ANN ($\sim 10^4$ pour un Resnet), et bien plus de connexions ($\sim 10^{14}$ contre $\sim 10^7$ poids pour un Resnet), lui permettant d'apprendre une représentation du monde beaucoup plus exhaustive et fine.
- Le cerveau humain bénéficie du phénomène d'apprentissage multimodal : il apprend suivant une variété de stimuli, et pas seulement en regardant des images ou des textes. Il bénéficie également d'une capacité à transposer ses connaissances d'un domaine à un autre, alors qu'un ANN est spécialisé sur une tâche bien particulière et n'a qu'une vision très restreinte du monde.
- Le modèle du neurone utilisé pour les ANN n'est aucunement fidèle au neurone biologique. Les réseaux à spikes (SNN) s'en rapprochent et sont plus efficaces énergétiquement que les ANN [Wu et al., 2022], mais sont souvent moins performants sur des tâches complexes du fait des difficultés relatives à leur entraînement [Plagwitz et al., 2023].
- Les réseaux les plus grands et les plus performants comme les LLM requièrent des temps d'apprentissage très longs (plusieurs mois sur des

dizaines ou des centaines de GPUs) et de gigantesques masses de données d'entraînement [OpenAI, 2023], tandis qu'un humain est capable d'apprendre à partir d'un nombre très restreint d'échantillons.

La qualité d'un réseau est donc déterminée par deux éléments : la topologie du réseau (type de modèle, nombre de neurones, de couches, nature des connexions...), et l'efficacité de son apprentissage. Sur ce second point, il existe une multitude de manières d'améliorer à la marge la performance finale du réseau en affinant les hyperparamètres, voire en les faisant évoluer au cours de l'entraînement, par exemple pour le learning rate : Loshchilov and Hutter [2017].

1.4 . Réseaux de convolution

Les réseaux de convolution (Convolutional Neural Networks), sont une architecture spécifique de réseaux de neurones profonds particulièrement adaptée au traitement d'images et de données structurées. Les CNN sont très utilisés dans l'embarqué du fait de la sparsité de leur topologie. Dans ce qui suit, on présente succinctement certaines architectures classiques pour la reconnaissance d'images sur lesquelles on a été amené à expérimenter tout au long de cette thèse.

1.4.1 . Principes généraux

Les CNN sont composés de plusieurs types de couches : les couches de convolution, les couches non-linéaires dites "d'activation", les couches de pooling et les couches linéaires "fully connected".

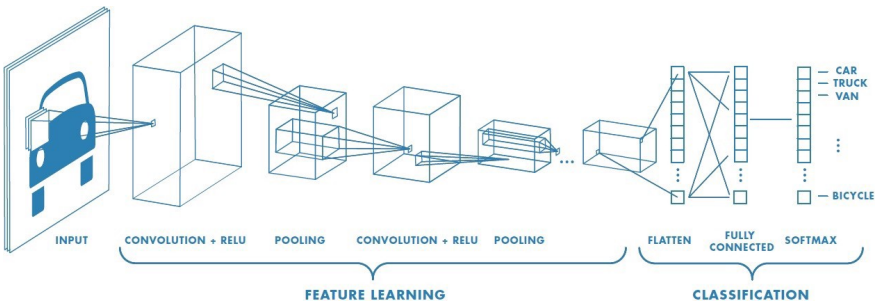

Figure 1.9 – Schéma fonctionnel d'un réseau convolutif [MATLAB, 2021]

Le coeur opérationnel des CNN est leur extracteur de caractéristiques, composé d'un empilement de couches de convolution. Les noyaux de convolution sont appliqués de manière glissante sur l'ensemble des entrées de la

couche, à commencer par l'image elle-même qui constitue l'entrée de la première couche. Cette opération permet de détecter des caractéristiques locales telles que les bords, les textures ou les motifs spécifiques présents dans l'image. À mesure que le flux d'activations avance dans le réseau, les caractéristiques extraites dans chacun des canaux représentent des éléments de plus en plus sophistiqués et globaux de l'image d'entrée, permettant ainsi la classification ou la régression à la sortie.

Une non-linéarité est généralement appliquée à la sortie des couches de convolution. On fait souvent le choix de la ReLU (rectified linear unit) (Figure 1.10), à la fois pour sa simplicité d'implémentation et pour sa non distorsion des valeurs positives, qui permet une atténuation des problèmes d'explosion du gradient comme de gradient évanescent [Bengio et al., 1994].

La couche de pooling agrège les résultats sorties des couches précédentes pour en réduire la cardinalité. Plusieurs opérations linéaires ou non existent (max-pooling, average pooling....) et permettent de réduire la résolution des caractéristiques à l'intérieur du réseau. On en emploie souvent une juste avant la couche fully connected servant de classifieur, de sorte à transformer chaque caractéristique en un nombre unique (logit).

Les couches fully connected, qui sont similaires à celles d'un perceptron, sont utilisées en fin de réseau pour la classification finale ou la prédiction des résultats. Elles prennent en entrée les caractéristiques extraites par les couches de convolution et les transforment en une sortie appropriée pour la tâche donnée (classification ou régression).

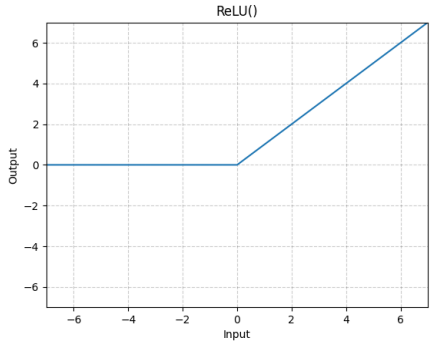

Figure 1.10 – Fonction d'activation Rectified Linear Unit (ReLU)

L'efficacité de cette architecture, notamment en reconnaissance d'images, a révolutionné la vision par ordinateur. Elle a été popularisée par Y. LeCun

dans les années 90 grâce à une application à la lecture automatique de codes postaux [LeCun et al., 1989]. Il est à noter que l'hégémonie des CNN pour la vision est aujourd'hui disputée par l'arrivée du transformer [Vaswani et al., 2023], issu du domaine du langage (NLP). Mais le transformer est un modèle coûteux du point de vue matériel, ce qui le rend moins pertinent pour une mise en œuvre dans des systèmes embarqués basse consommation. Pour cette raison, on a exclu les transformers de notre champ d'étude.

1.4.2 . Quelques architectures classiques

Resnet

Le réseau de neurones à connexions résiduelles (Resnet) a été introduit par [He et al., 2015, 2016]. Il a été conçu pour résoudre le problème de gradient évanescent, qui engendre une dégradation de la performance des réseaux avec l'augmentation de leur profondeur.

L'équation de propagation d'un bloc Resnet peut être formulée comme suit (Equation 1.1) :

$$\mathbf{x_{n+1}} = \mathbf{x_n} + \mathcal{F}(\mathbf{x_n}, \mathcal{W}) \tag{1.1}$$

où $\mathbf{x_n}$ est l'entrée du bloc résiduel, $\mathbf{x_{n+1}}$ sa sortie, et $\mathcal{F}(\mathbf{x_n}, \mathcal{W})$ représente la fonction non linéaire définie par les paramètres $\mathcal{W}$ du bloc. La sortie du bloc est obtenue en ajoutant l'entrée à la sortie de la fonction non linéaire : c'est la connexion résiduelle, qui a pour but de faciliter la propagation de l'information à travers le réseau (voir Figure 1.11).

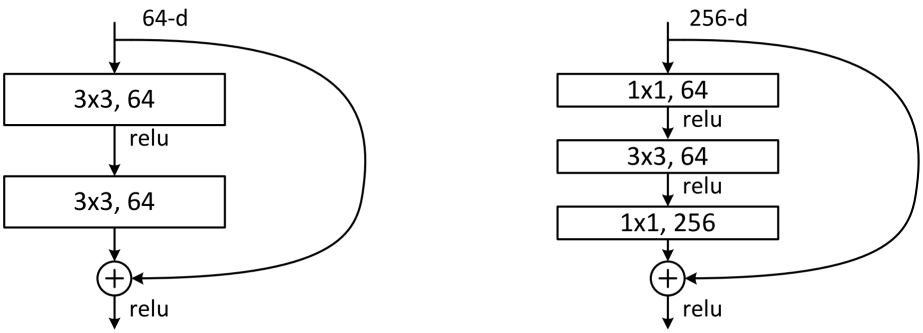

Figure 1.11 – Les deux types de blocs d'un Resnet [He et al., 2015]

L'architecture Resnet a connu un grand succès sur les tâches de vision, en particulier sur la classification d'images, la détection d'objets et la segmentation sémantique. Elle a permis de construire des réseaux beaucoup plus profonds (atteignant plus de 100 couches) et performants avec une durée d'entraînement soutenable. Le concept de connexion résiduelle est réutilisé dans la plupart des architectures plus récentes.

MobileNet

MobileNet est une famille d'architectures de réseaux neuronaux conçue spécifiquement pour les applications de vision sur des appareils mobiles et embarqués. Son objectif est de réduire le coût computationnel des réseaux neuronaux. Elle a été introduite par Howard et al. [2017] et a ensuite été améliorée avec les versions MobileNetV2 [Sandler et al., 2018] et MobileNetV3 [Howard et al., 2019].

Le bloc principal de MobileNet effectue une opération de convolution séparable en profondeur (depthwise separable convolution, Figure 1.12), qui se décompose en deux étapes :

- Convolution en profondeur (depthwise convolution) : Cette étape applique une convolution individuelle pour chaque canal d'entrée, utilisant un noyau de taille 3×3. Cela permet de réduire le nombre de paramètres et les opérations de calcul.
- Convolution point à point (pointwise convolution) : Après la convolution en profondeur, une convolution classique de kernel 1×1 est appliquée pour combiner les informations spatiales des différents canaux.

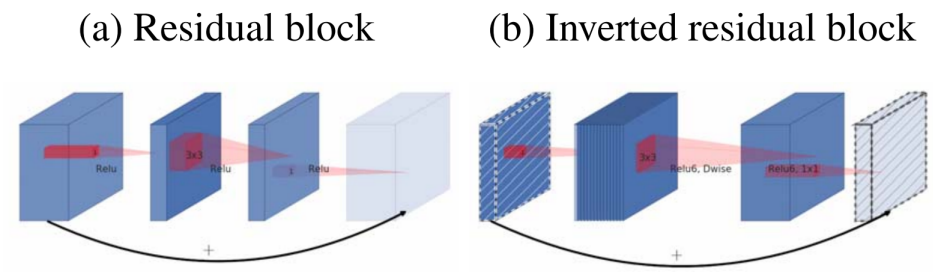

Figure 1.12 – Différence entre un bloc résiduel (de type Resnet), et une convolution séparable en profondeur (bloc résiduel inversé, de type MobileNet) [Sandler et al., 2018].

L'avantage de MobileNet est sa sparsité topologique : au lieu de calculer une convolution 3×3 par couple canal d'entrée/canal de sortie, on se contente de calculer une convolution par canal d'entrée et de recombiner les canaux entre eux par des convolutions 1×1 beaucoup moins coûteuses. En effet, pour m canaux d'entrée, n canaux de sortie et un kernel de taille $k \times k$, une convolution classique nécessite k^2mn poids, tandis qu'une convolution séparable en profondeur n'en utilise que $k^2m + mn$.

EfficientNet

EfficientNet [Tan and Le, 2020] est une famille d'architectures de réseaux neuronaux conçue pour obtenir un bon équilibre entre la précision et l'efficacité computationnelle. EfficientNet se base sur le même type de blocs que MobileNetv2, et y ajoute des blocs squeeze-and-excite [Hu et al., 2017]. La structure d'EfficientNet repose sur une règle de mise à l'échelle de la largeur, de la profondeur et de la résolution déterminée expérimentalement. Plutôt que d'optimiser séparément chaque dimension, EfficientNet propose une méthode systématique pour les ajuster de manière équilibrée ("Compound scaling", Figure 1.13), en utilisant un paramètre ϕ de mise à l'échelle global, appelé "coefficient de mise à l'échelle" :

$$\text{depth} : d = \alpha^{\phi} \tag{1.2}$$
$$\text{width} : w = \beta^{\phi} \tag{1.3}$$
$$\text{resolution} : r = \gamma^{\phi} \tag{1.4}$$
$$\text{s.t. } \alpha.\beta^2.\gamma^2 \approx 2 \tag{1.5}$$

où α, β, γ sont supérieurs ou égaux à 1 et déterminés par une recherche heuristique.

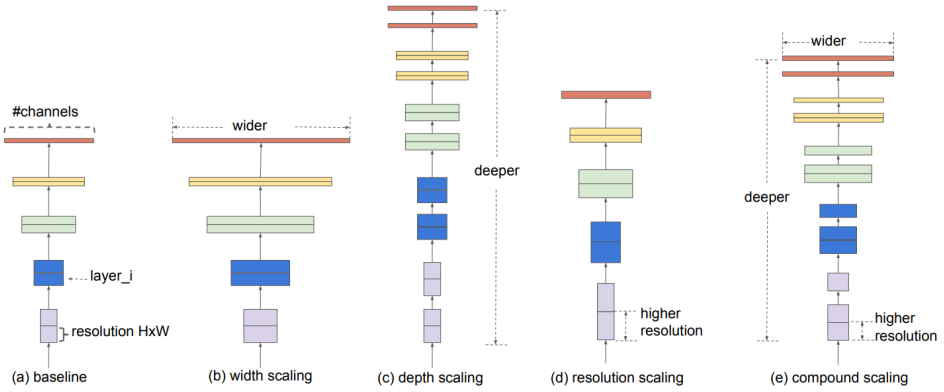

Figure 1.13 – Compound scaling et autres types de scaling [Tan and Le, 2020].

1.4.3 . Neural Architecture Search

La recherche d'architecture est un moyen à part entière d'optimiser les performances des réseaux moyennant des contraintes matérielles. Cette branche de recherche porte le nom de Neural Architecture Search (NAS). Mais la NAS n'est pas l'unique moyen de parvenir à cet objectif. Dans de nombreuses applications, on préférera utiliser une architecture déterminée que l'on compressera a posteriori pour la ramener dans les limites du budget matériel de l'application visée.

1.5 . Objectifs de la thèse

L'applicabilité pratique de la méthode de compression choisie joue un rôle prépondérant : la meilleure méthode en théorie (qui conserve le mieux la précision pour un taux de compression donné selon un certain critère, par exemple) n'est pas forcément la méthode à privilégier en pratique. De fait, les contraintes de rapidité de déploiement, la complexité d'un réentraînement du réseau, le disponibilité des données d'entraînement, ou encore les compétences nécessaires à l'implémentation matérielle sont autant de critères de choix importants pour l'industrie.

Ainsi, trouver le meilleur levier pour minimiser l'empreinte des réseaux de neurones est un problème intrinsèquement mal posé, dont la solution optimale varie selon l'objectif recherché et les moyens à disposition. Dans un cas, il peut être pertinent de concevoir entièrement une architecture pensée spécifiquement pour la tâche, tandis que dans un autre, on préférera élaguer une architecture existante, ou encore recourir au calcul approché pour accélérer l'inférence.

En ce qui concerne cette thèse, on a choisi de concentrer nos recherches sur la quantification. Parmi les techniques existantes, la quantification se distingue en effet par l'efficacité et la simplicité de son implémentation matérielle, au sens où elle permet d'agir directement sur la complexité du calcul par une réduction de la précision des opérateurs utilisés. Bien que l'implémentation matérielle constitue l'objectif ultime, seules des mises en œuvre algorithmiques de techniques de compression ont été réalisées. Néanmoins, la faisabilité matérielle des propositions apportées a constamment fait l'objet d'une attention particulière.

La problématique de cette thèse peut ainsi être résumée par l'interrogation suivante : "Comment quantifier un réseau de manière à atteindre un objectif de compression donné, tout en dégradant le moins possible ses capacités à résoudre le problème pour lequel il a été conçu ?".

1.6 . Organisation de la thèse

On commencera par dresser dans la partie 2 un panorama non exhaustif des méthodes existantes de quantification des réseaux de neurones en essayant d'établir une classification des différentes approches en fonction des hypothèses expérimentales et des types de quantifieurs utilisés. Dans les parties suivantes, on présentera trois solutions explorées au cours de cette thèse. La partie 3 étudie l'hypothèse de la quantification post-training (don-

nées de réentraînement disponibles en quantité très limitée), et propose un quantifieur exploitant les corrélations entre paramètres voisins d'une même couche pour réduire l'erreur de quantification. Dans la partie 4 est introduite une modification de l'algorithme d'apprentissage afin de prendre en compte les contraintes de convergence liées à la quantification en très basse précision. Enfin, la partie 5 s'intéresse à la cooptimisation de la quantification avec celle d'un autre type de compression : le codage de source. On conclura en donnant une perspective du domaine de la compression des réseaux de neurones.

2 - État de l'art

Dans cette partie, on fait état des principales approches existantes pour compresser les réseaux de neurones. On supposera que l'on dispose d'une architecture fixe dont on cherche à optimiser les performances étant donné un certain budget (mémoire ou calculatoire). On ne prétend pas ici offrir une revue exhaustive de l'ensemble des méthodes de compression existantes. Pour cela, on renvoie le lecteur aux méta-analyses suivantes : Blalock et al. [2020], Krishnamoorthi [2018]. On réalisera un inventaire plus détaillé des méthodes de quantification.

2.1 . Paradigmes de compression

Sans que cela soit spécifique aux réseaux de neurones, on peut diviser les techniques de compression en deux courants principaux : la compression sans pertes et la compression avec pertes. Une technique de compression sans pertes se présente sous la forme d'une fonction bijective dont l'inverse est la fonction de décompression. Une technique de compression avec pertes autorise quant à elle la perte d'information.

2.1.1 . Compression sans pertes

Dans le cadre des réseaux neuronaux, la compression sans pertes présente l'avantage majeur de garantir l'absence de dégradations sur les performances du réseau. En pratique, la compression sans pertes est fréquemment utilisée en support d'une ou plusieurs techniques de compression avec pertes.

Codage entropique

Le principe de fonctionnement du codage entropique repose sur la théorie de l'information et la notion d'entropie informationnelle [Shannon, 1948].

L'entropie $H(X)$ est une mesure de l'incertitude ou du désordre dans un ensemble de données. Elle est définie mathématiquement comme suit :

$$H(X) = -\sum_{x \in \Omega(X)} p(X = x) \times log_2(p(X = x)) \tag{2.1}$$

où X est la variable aléatoire modélisant la source d'information, $\Omega(X)$ son univers (l'ensemble des symboles possibles), et $p(x)$ est la probabilité d'occurrence d'un symbole x. Une distribution très "prévisible" a une entropie plus faible qu'une distribution qui l'est moins. En cela, la notion d'entropie de

Shannon correspond à l'entropie thermodynamique, qui est une mesure de désordre.

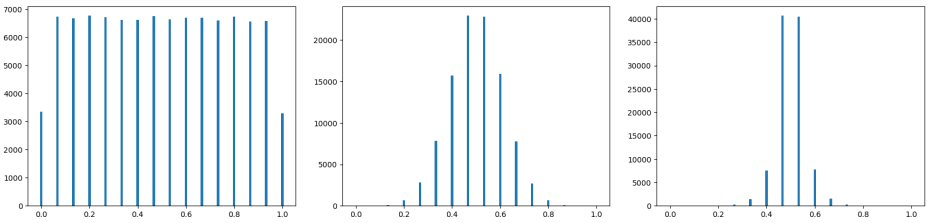

Figure 2.1 – De gauche à droite, trois exemples de tirages de distributions de moins en moins entropiques. Abscisse : valeur du symbole. Ordonnée : nombre de tirages.

D'après le théorème du codage de source [Shannon, 1948], un flux de données peut être compressé sans pertes jusqu'à une certaine borne dépendante de la distribution de ses symboles :

Théorème du codage de source

N variables aléatoires indépendantes et identiquement distribuées d'entropie $H(X)$ peuvent être compressées en plus de $NH(X)$ bits avec une probabilité négligeable de perte d'information. Réciproquement, elles ne peuvent pas être compressées en moins de $NH(X)$ bits sans que la perte d'information soit un événement presque sûr.

Le codage entropique assigne des codes de longueur variable aux symboles en utilisant leurs probabilités d'occurrence. Les symboles fréquents sont représentés par des codes courts, tandis que les symboles moins fréquents sont représentés par des codes plus longs, permettant ainsi de compresser les données. Le codage entropique est le plus utile lorsque la distribution à encoder est peu entropique. Le codage de Huffman [Huffman, 1952] et le codage arithmétique [Langdon, 1984] sont deux exemples d'algorithmes de codage entropique largement utilisés. Le codage de Huffman construit un arbre binaire de forme dictée par les probabilités des symboles. Le codage arithmétique représente quant à lui l'ensemble des données par un nombre décimal compris entre 0 et 1, en utilisant des intervalles de tailles proportionnelles aux probabilités des symboles.

DeepCABAC

DeepCABAC [Wiedemann et al., 2020] résulte de l'adaptation de CABAC (Context-based Adaptive Binary Arithmetic Coder) à la compression de ré-

seaux de neurones. Initialement conçu pour la compression vidéo au standard H264/AVC, ce codeur comporte plusieurs étapes et fonctionne par rétroaction :

- Quantification : le codeur choisit l'hyperparamètre de pas de quantification en prenant en compte la perte en performance du réseau et la longueur du code obtenu à la fin des étapes suivantes.
- Binarisation : les poids quantifiés sont binarisés suivant un arbre binaire de décision similaire à celui de CABAC.
- Modélisation du contexte : les probabilités des symboles sont estimées en fonction des symboles voisins, et non d'une distribution fixe.
- Codage arithmétique : la séquence des symboles est encodée grâce à l'estimation de leurs probabilités.

DeepCABAC n'est pas utilisable à l'inférence, le coût de son décodage étant trop important. Mais dans un scénario d'apprentissage fédéré (federated learning), qui exploite des canaux de bande passante limitée, il est intéressant de pouvoir transmettre les données d'entraînement, les paramètres du réseau ainsi que leurs gradients en limitant l'usage des ressources.

Transformées type Fourier

Les décompositions fréquentielles sont très utilisées dans le domaine de la compression d'images, notamment dans le standard JPEG pour la transformée en cosinus discrète (DCT) (cf. Equation 2.2).

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \quad \text{for} \quad k = 0, \ldots, N-1 \tag{2.2}$$

où le signal temporel (x_i) est transformé en signal fréquentiel (X_k).

La DCT décompose l'image en une combinaison de différentes fréquences. La plupart des images ont une certaine redondance ou corrélation spatiale : les pixels voisins ont tendance à avoir des valeurs similaires. La DCT exploite cette propriété en représentant l'image comme une somme de signaux sinusoïdaux de différentes fréquences et amplitudes. En pratique, l'image est d'abord divisée en blocs de pixels. Ensuite, pour chaque bloc, la DCT génère une matrice de coefficients. Ces coefficients représentent l'amplitude des différentes fréquences présentes dans le bloc. Lorsque les représentations fréquentielles des signaux sont clairsemées (sparse), la DCT conserve l'information et peut être assimilée à une compression sans pertes, aux erreurs d'arrondi près. Sinon, on peut forcer la sparsité en tronquant les coefficients d'amplitude plus faible qui ont une contribution négligeable à l'aspect de l'image.

Gueguen et al. [2018] exploite une DCT ainsi que d'autres éléments du codec jpeg pour réduire la complexité de l'inférence d'un Resnet-50, en effectuant l'inférence sur une image compressée.

2.1.2 . Compression avec pertes

La compression avec pertes des réseaux de neurones offre des avantages significatifs en termes de simplification de l'inférence, tels que la possibilité de réduire la complexité des calculs (en réduisant leur précision ou bien leur quantité), ou la réduction du nombre d'accès mémoire (en réduisant le nombre de paramètres ou en augmentant leur sparsité). Contrairement aux approches sans pertes, elle peut également entraîner une dégradation de la prédiction, en particulier à des niveaux de compression agressifs. Cependant, simplifier un modèle de la sorte peut aussi l'aider à mieux généraliser, et ainsi, de manière contre-intuitive, à améliorer ses performances [Esser et al., 2020, Jin et al., 2019].

Weight sharing

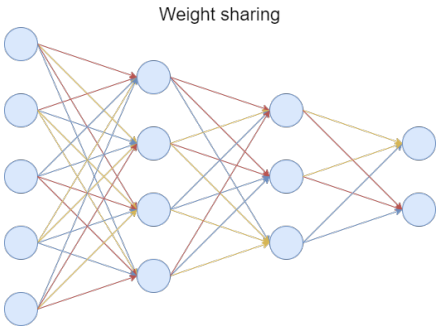

Figure 2.2 – Weight sharing : les connexions de même couleur partagent un même poids

Le partage des poids (weight sharing, Figure 2.2) permet de réduire le nombre de paramètres d'un réseau. Il consiste à réutiliser les mêmes poids à plusieurs endroits. L'idée de partage des poids est à la base de la conception des réseaux de convolution : les mêmes poids sont réutilisés pour le calcul sur l'ensemble d'une caractéristique d'entrée. Le weight sharing a pour effet de réduire le nombre de paramètres à entraîner ou à garder en mémoire, au prix d'une réduction de la flexibilité du modèle.

Le weight sharing fait partie des techniques utilisées dans Han et al. [2016]. Ce papier fondamental dans le domaine de la compression des réseaux de neurones commence par effectuer un pruning des poids de faible amplitude

du réseau, dont les poids restants sont ensuite réentrainés. L'étape de weight sharing consiste à calculer un clustering des poids, les associant aux centroïdes les plus proches selon l'algorithme des k-moyennes. Ces centroïdes sont les poids partagés. Pour finir, le réseau est réentrainé une dernière fois afin d'ajuster la position de ces centroïdes.

Pruning

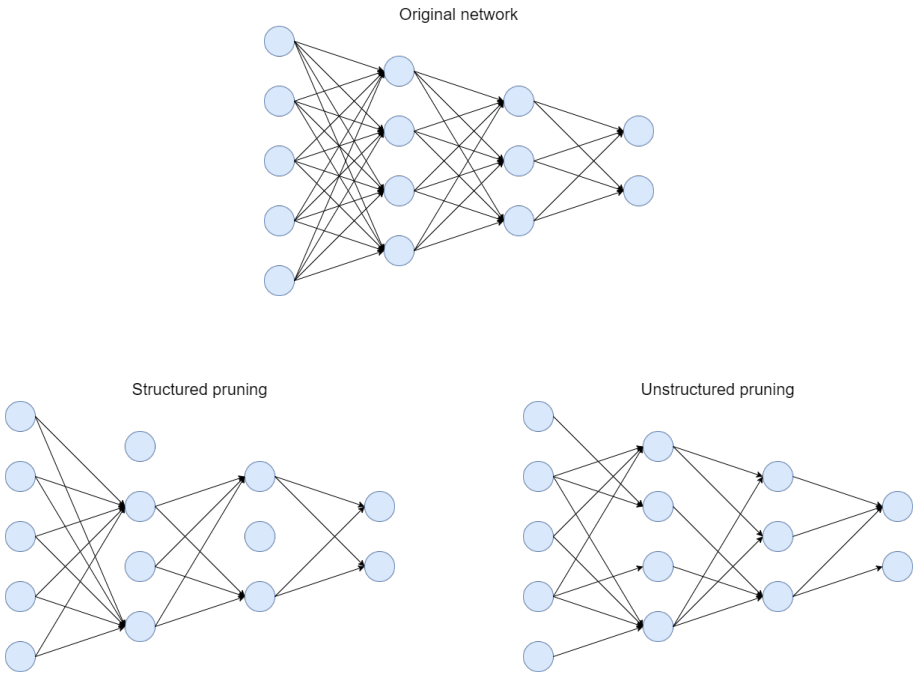

Figure 2.3 – Pruning structuré et pruning non structuré

Le pruning est une technique utilisée pour réduire la taille d'un réseau en éliminant les connexions les moins "importantes" entre les neurones. Il vise à améliorer l'efficacité et les performances du réseau en réduisant le nombre de paramètres à apprendre et en accélérant les calculs lors de l'inférence. Le pruning peut être appliqué à différentes échelles (Figure 2.3), que ce soit au niveau des poids individuels (pruning non structuré) ou des neurones entiers (pruning structuré).

L'idée principale du pruning est basée sur le fait que, lors de l'apprentissage, toutes les connexions ne contribuent pas de manière équivalente à la performance globale du modèle. Certains poids peuvent être moins importants ou même redondants : en les supprimant, on peut obtenir un modèle plus compact.

Le pruning, en particulier lorsqu'il est agressif, peut entraîner une perte de performance du réseau. Pour atténuer cet effet, on peut utiliser des techniques de réentrainement pour réajuster les poids restants du réseau afin de prendre en compte le changement de topologie. Cette étape de réentrainement consiste à reprendre l'apprentissage sur le modèle pruné. Il est possible d'appliquer le pruning de manière itérative, en effectuant plusieurs cycles de pruning et de réentrainement pour obtenir le modèle optimal.

L'une des méthodes de pruning les plus courantes consiste à évaluer l'importance des poids du réseau par des critères tels que leur magnitude [Han et al., 2016] ou leur sensibilité aux perturbations [Molchanov et al., 2019]. Par exemple, les poids ayant une valeur proche de zéro ou qui ne varient pas significativement lors de l'apprentissage peuvent être considérés comme moins importants. Une fois que l'importance des poids a été évaluée, on peut les éliminer en les mettant à zéro ou en les supprimant complètement du modèle.

On peut aussi faire de l'apprentissage sporadique (sparse training) : on maintient un certain nombre de connexions à 0, quitte à changer ces connexions itérativement, aléatoirement ou par l'apprentissage d'un masque [Guo et al., 2016]. Enfin, une approche très populaire [Frankle and Carbin, 2019] repose sur l'hypothèse du ticket gagnant : pour tout réseau dense, il existe un sous-réseau (winning ticket) dont la précision peut atteindre celle du réseau dense en un nombre similaire d'itérations.

Low-rank approximation

L'approximation à rang inférieur (low-rank approximation) est une technique de compression d'opérateur matriciel. Elle revient à trouver des matrices U et V de taille réduite et telles que :

$$W \approx UV^T \tag{2.3}$$

Pour réduire la taille des matrices de poids, on recourt fréquemment à la décomposition aux valeurs singulières (Singular Value Decomposition, SVD) [Khodak et al., 2021].

Comme pour le pruning, l'approximation à rang inférieur présente des pertes en performance de plus en plus importantes à mesure que le rang de compression décroît. Pour atténuer cet effet, on peut apprendre les rangs des matrices de la factorisation durant l'entraînement [Idelbayev and Carreira-Perpiñán, 2020].

Quantification

La quantification des réseaux de neurones consiste à réduire la précision des paramètres et des activations. Cette méthode repose sur le fait que la représentation de nombres en virgule flottante utilisée par défaut dans la plupart des applications peut être surdimensionnée pour l'usage recherché, et qu'une précision inférieure peut suffire sans compromettre de manière significative les performances du modèle. La quantification a été le domaine d'intérêt principal de cette thèse, et on développera son état de l'art plus en détails au paragraphe suivant.

La quantification permet de réduire la taille du réseau sans modifier son architecture. Elle permet aussi d'accélérer l'inférence en réduisant non seulement les temps d'accès mémoire, mais aussi la complexité des opérateurs. En effet, il devient possible d'utiliser des additionneurs et des multiplieurs en précision réduite, plus rapides et plus économes que les opérateurs 32 bits.

On distingue deux courants principaux pour la quantification [Krishnamoorthi, 2018] : les méthodes post-training (PTQ) et les méthodes quantization-aware training (QAT). La PTQ consiste à prendre un réseau préentrainé et à le quantifier sans réentrainement. Les méthodes QAT nécessitent quant à elles un pipeline d'entrainement complet, sont généralement plus complexes, plus longues à mettre en oeuvre, et demandent plus de ressources et d'expertise. Dans le cas général, la QAT est indiquée lorsque l'on cherche à minimiser la dégradation des performances du réseau, car elle donne les meilleurs résultats [Zhou et al., 2018]. En revanche, lorsque l'objectif est un déploiement rapide et peu coûteux, les avantages de la PTQ deviennent intéressants [Choukroun et al., 2019].

2.1.3 . Bilan des paradigmes de compression

On classe les différentes approches de compression présentées dans cette section dans la Table 2.1 en fonction de leurs objectifs (lossy ou lossless, compression des poids, compression des activations, accélération de l'inférence). Parmi les approches avec pertes, le pruning et la quantification se distinguent : elles permettent de compresser à la fois les poids et les activations, réduisant de fait le besoin de stockage mémoire, la complexité des transferts mémoire et celle du calcul. Il est à noter que la classification ci-dessous est simplifiée et n'a pas valeur absolue. In fine, les capacités de compression et d'accélération des différentes techniques présentées dépendent de leur implémentation matérielle. Ainsi, on ne comptera une technique valide pour l'accélération que si elle est pensée à l'origine pour l'accélération.

Technique	Lossless	Poids	Activations	Accélération
Codage entropique	✓	✓	✗	✗
Weight sharing	✗	✓	✗	✓
Pruning	✗	✓	✗/✓[1]	✗/✓[1]
Low-rank factorization	✗	✓	✗	✓
Fourier-like	✗	✗	✓	✓
Quantification	✗	✓	✓	✓

Table 2.1 – Tableau des approches de compression.

2.2 . Paradigmes de quantification

Dans cette section, on présente les principales approches existantes pour la quantification des réseaux de neurones.

2.2.1 . Quantification post-entraînement

La quantification post-training (PTQ) regroupe des méthodes généralement économes, dont l'objectif est un déploiement rapide et à peu de frais d'un réseau quantifié avec une recherche de performance modérée.

Banner et al.

Banner et al. [2018] regroupe trois solutions qui ne nécessitent aucun réentraînement ou finetuning.

- La quantification des poids $W \rightarrow \widehat{W}$ induit une distorsion de leur distribution, modifiant leur biais (Equation 2.4) et leur variance (Equation 2.5). La compensation de cette distorsion (Equation 2.6) entraîne une amélioration des performances.

$$\mu = \mathbb{E}(W) - \mathbb{E}(\widehat{W}) \tag{2.4}$$

$$\xi = \frac{||W - \mathbb{E}(W)||_2}{||\widehat{W} - \mathbb{E}(\widehat{W})||_2} \tag{2.5}$$

$$\widehat{W} \leftarrow \xi(\widehat{W} + \mu) \tag{2.6}$$

- Différents canaux d'une même couche peuvent impacter la performance du réseau dans des proportions différentes. À partir de cette observation, Banner et al. [2018] adapte la précision de quantification en allouant une précision différente pour chaque canal (2.7), de sorte à minimiser l'erreur quadratique moyenne (Mean-Squared Error, MSE) et à

1. ✗ si le pruning est non structuré, ✓ s'il l'est.

ce que la précision moyenne de la couche reste inchangée. Ci-dessous, M_i est la précision allouée au canal i, α_i la valeur maximum constatée sur le canal i en valeur absolue, et B le budget total en bits.

$$M_i = \lfloor log_2(\frac{\alpha_i^{\frac{2}{3}}}{\sum_j \alpha_j^{\frac{2}{3}}}.B) \rceil \tag{2.7}$$

- Les activations sont quantifiées via la méthode ACIQ : l'approximation de leur distribution à une gaussienne ou une laplacienne permet d'aboutir à un paramètre de clipping optimal au sens de la MSE.

Adaround

Nagel et al. [2020] est une méthode de quantification des poids. Adaround démontre la sous-optimalité de l'arrondi au plus proche : l'arrondi le plus proche ne permet pas d'obtenir le réseau quantifié le plus performant. Il détaille une piste pour prendre en compte la performance du réseau lors de l'arrondi, en se ramenant au problème plus simple de l'optimisation d'une loss locale (couche par couche), ensuite relaxé en un problème d'arrondi. En d'autres termes, Adaround apprend la meilleure manière d'arrondir les poids du réseau afin de reproduire le plus fidèlement possible les sorties intermédiaires des couches de la baseline non quantifiée sur un batch de données d'entraînement. Ci-dessous, on écrit l'équation d'optimisation d'Adaround (Equation 2.8). Le premier terme est le problème de reconstruction, où f_a est la fonction d'activation, $\mathbf{W}$ les poids, $\mathbf{x}$ les activations, $\widetilde{\mathbf{W}}$ les poids pseudo quantifiés avec le bruit d'arrondi $\mathbf{V}$ (que l'on fait converger vers 0 ou 1 au fur et à mesure de l'entraînement). $\lambda f_{reg}(\mathbf{V})$ est le terme d'optimisation lagrangienne permettant de passer progressivement d'un paramètre en précision flottante à un paramètre quantifié, où λ est un hyperparamètre et f_{reg} une fonction de régularisation.

$$\arg\min_{\mathbf{V}} ||f_a(\mathbf{W}\mathbf{x}) - f_a(\widetilde{\mathbf{W}}\mathbf{x})||_F + \lambda f_{reg}(\mathbf{V}) \tag{2.8}$$

Brecq

La contribution majeure de Li et al. [2021] est l'intuition de la reconstruction par blocs. Là où Nagel et al. [2020] cherche à minimiser le bruit de quantification couche par couche, Brecq identifie des dépendances entre les couches et en déduit qu'une reconstruction plus globale est une meilleure approche. Mais si réentraîner tout le réseau est en théorie plus avantageux, les contraintes du post-training et en particulier la limitation à un petit nombre d'échantillons causent facilement de l'overfitting lorsque de grandes quantités de paramètres

sont entraînés simultanément. Pour remédier à ce dilemme, Brecq propose une voie médiane qui reconstruit les sorties intermédiaires des blocs résiduels, et parvient à s'approcher des niveaux de performance des techniques QAT sur la reconnaissance d'images.

Data-Free Quantization (DFQ)

Data-free quantization Nagel et al. [2019] permet la quantification post-training des poids et des activations sans recourir à aucune donnée, ni de calibration ni de finetuning. DFQ met en évidence l'existence de disparités d'amplitude importantes entre les différents canaux d'une couche de MobileNet-v2, ainsi que d'un biais sur l'activation de sortie induit par la quantification. Pour remédier à cela, DFQ calcule un paramètre d'égalisation entre les canaux de couches successives de sorte à maximiser la précision de quantification :

$$s_i = \frac{1}{r_i^{(2)}} \sqrt{r_i^{(1)} r_i^{(2)}} \tag{2.9}$$

où $r_i(1)$ est l'amplitude de quantification du canal de la première couche, et $r_i^{(2)}$ celle de la seconde. Il suffit ensuite d'ajuster l'amplitude de quantification des canaux en les multipliant ou en les divisant par ce facteur pour aboutir à un réseau dont la sortie est identique, mais dont les canaux sont égalisés.

Ensuite, pour corriger le biais constaté sur l'activation de sortie, il suffit de retrancher ce biais à la sortie. En faisant l'hypothèse d'une distribution gaussienne de l'entrée de la couche, on peut calculer une estimation de ce biais sans utiliser de données de calibration :

$$\mathbb{E}[y] = \mathbb{E}[\tilde{y}] - \mathbb{E}(\epsilon x) \tag{2.10}$$

où y est la sortie de la couche non quantifiée, $\tilde{y}$ est la sortie de la couche quantifiée, ϵ est le biais induit sur les poids par la quantification et x est l'entrée de la couche, qui peut être estimée de deux manières, soit via la couche de batch normalization précédente, soit en comparant l'activation avant et après la quantification.

2.2.2 . Quantification à l'apprentissage

La quantification à l'apprentissage (Quantization-Aware Training, QAT) regroupe des approches plus performantes que la PTQ mais aussi plus coûteuses. On préférera notamment la QAT pour des quantifications très agressives et pour lesquelles la performance est critique.

STE

Entraîner un réseau quantifié est une tâche non triviale. En effet, la fonction de quantification $q : W_{n,i} \rightarrow \widehat{W_{n,i}}$ (n indice de la couche, i indice du poids) admet 0 pour dérivée presque partout. Un remplacement naïf ne permet donc pas d'apprendre les paramètres du réseau, faute de rétropropager un gradient non nul dans l'algorithme de rétropropagation du gradient [Rumelhart et al., 1986] généralement utilisé pour entraîner les réseaux de neurones :

Algorithm 1 Rétropropagation du gradient

1: **function** Backprop(Réseau N, Entrée x, Label y, Fonction perte L, Taux d'apprentissage λ)
2: Calculer l'erreur $L(N(x), y)$
3: Pour la dernière couche, mettre à jour les paramètres par descente de gradient : $W_{n,i} \leftarrow W_{n,i} - \lambda \frac{\partial L}{\partial y_n} \frac{\partial y_n}{\partial W_{n,i}}$.
4: Rétropropager l'erreur à la couche précédente par dérivation composée : $W_{n-1,i} \leftarrow W_{n-1,i} - \lambda \frac{\partial L}{\partial y_n} \frac{\partial y_n}{\partial y_{n-1}} \frac{\partial y_{n-1}}{\partial W_{n-1,i}}$. ▷ Le mode de mise à jour présenté ici est naïf. En pratique, il dépend de l'optimiseur choisi.
5: Itérer le procédé jusqu'à la première couche.
6: **end function**

Dans le cas d'un réseau quantifié, il faut remplacer $\frac{\partial y_n}{\partial W_{n,i}}$ par $\frac{\partial y_n}{\partial \widehat{W_{n,i}}} \frac{\partial \widehat{W_{n,i}}}{\partial W_{n,i}}$ dans l'algorithme de rétropropagation. Mais il existe un moyen [Bengio et al., 2013] de choisir un gradient de remplacement non nul, appelé Straight-Through Estimator (STE), permettant le passage de l'information. Par exemple, si $\widehat{W_{n,i}} = \lfloor W_{n,i} \rceil$, le STE s'écrit comme suit :

$$\frac{\partial \widehat{W_{n,i}}}{\partial W_{n,i}} = 1 \tag{2.11}$$

Quantifieurs uniformes

Les quantifieurs uniformes sont le mode de quantification le plus intuitif. Ils sont caractérisés par l'existence d'un pas de quantification constant : les niveaux de quantification sont identiques pour tous les paramètres d'une même couche et régulièrement espacés. Leur fonction de quantification repose généralement sur le calcul de l'arrondi.

Learned Step size Quantization (LSQ)

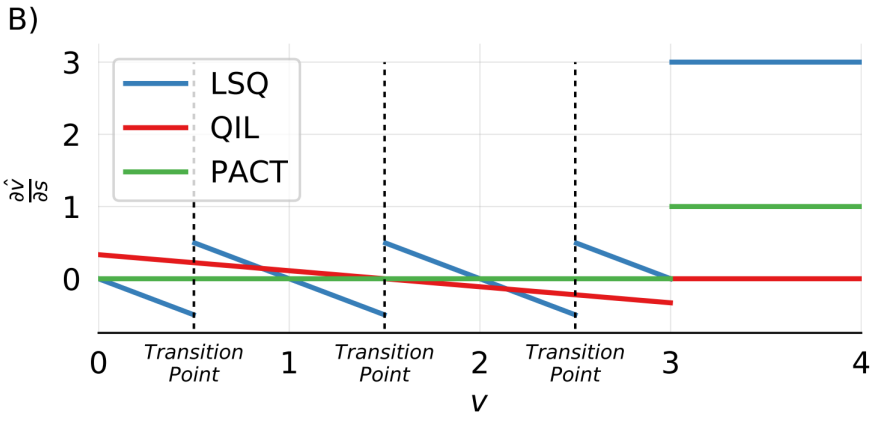

Figure 2.4 – Gradient pour la mise à jour du pas de quantification par LSQ [Esser et al., 2020]

Learned step size quantization [Esser et al., 2020] est une méthode de quantification uniforme qui apprend un pas de quantification s pour chaque couche. En effet, étant donné que les couches du réseau ont a priori des fonctions et des propriétés différentes, il peut être nécessaire d'introduire des pas de quantification distincts. En théorie, rien n'empêche de fixer ce pas de quantification en fonction des poids ou des activations, de sorte à minimiser l'erreur de quantification comme dans Banner et al. [2018], mais en pratique, une telle minimisation ne coïncide pas forcément avec l'optimalité des performances du réseau. Pour apprendre s, LSQ introduit une nouvelle manière de calculer le gradient des poids quantifiés par rapport à s, de sorte à prendre en compte la distance qui sépare chaque poids de son niveau de quantification (Figure 2.4).

En posant v l'objet à quantifier, $\widehat{v}$ sa valeur quantifiée, Q_N le nombre de niveaux de quantification négatifs et Q_P le nombre de niveaux de quantification positifs :

$$\bar{v} = \lfloor clip(\frac{v}{s}, -Q_N, Q_P) \rceil \tag{2.12}$$

$$\widehat{v} = \bar{v} \times s \tag{2.13}$$

où $clip(x, a, b) = max(a, min(x, b))$. En dérivant comme un produit, on obtient :

$$\frac{\partial \widehat{v}}{\partial s} = \begin{cases} -\frac{v}{s} + \lfloor \frac{v}{s} \rceil & \text{si } -Q_N < \frac{v}{s} < Q_P \\ -Q_N & \text{si } \frac{v}{s} \leq -Q_N \\ Q_P & \text{si } \frac{v}{s} \geq Q_P \end{cases} \tag{2.14}$$

On obtient un gradient de norme plus élevée lorsque v s'éloigne de son niveau de quantification, ce qui se comprend intuitivement par l'idée qu'une petite modification de v a plus d'importance lorsque v est proche d'une transition (car la valeur quantifiée $\widehat{v}$ risque de changer), que lorsque v est proche de son point de quantification.

Scale-Adjusted Training (SAT)

Scale-adjusted training [Jin et al., 2019] propose une analyse du processus de rétropropagation du gradient lors de l'entraînement, constate deux facteurs d'inefficacité des méthodes de quantification existantes et en dérive deux règles à mettre en place pour les corriger :

- La loss cross-entropique présente des régions de saturation : lorsque les logits sortent d'un certain intervalle, l'optimisation perd en efficacité. Il convient donc de restreindre la variance des logits, ce qui peut être fait en mettant à l'échelle les poids quantifiés $\widehat{W}$ de la dernière couche du classifieur selon l'équation 2.15, où n est le nombre de classes et $\mathbb{V}$ la variance.
- Le réseau apprend mieux lorsque les gradients sont du même ordre de grandeur à chaque endroit du réseau. Pour homogénéiser les gradients, chaque couche linéaire doit être suivie d'une couche de batch normalization, ou bien faire l'objet d'une mise à l'échelle.

$$\widehat{W} \leftarrow \frac{1}{n\mathbb{V}(\widehat{W})}\widehat{W} \tag{2.15}$$

Non-Uniform to Uniform Quantization (N2UQ)

Non-Uniform to Uniform Quantization [Liu et al., 2022a] part du constat que les techniques de quantification non-uniformes offrent plus de souplesse que les techniques uniformes, dans le sens où les niveaux de quantification peuvent être concentrés aux endroits importants, ou bien s'étirer pour modéliser la queue d'une distribution, ce qui est impossible avec une approche uniforme standard. En revanche, il est possible de modifier la position des seuils de transition entre points de quantification sans rompre le caractère uniforme, en modifiant simplement la règle d'arrondi (Figure 2.5). N2UQ introduit des paramètres de positions pour modéliser les seuils de transition et les apprend en adaptant le STE. Par ailleurs, la distribution des poids quantifiés peut être très inégale et très concentrée sur les niveaux de quantification les plus bas. Liu et al. [2022a] propose un mécanisme de régularisation entro-

pique pour encourager une meilleure dispersion et maximiser la dispersion des poids.

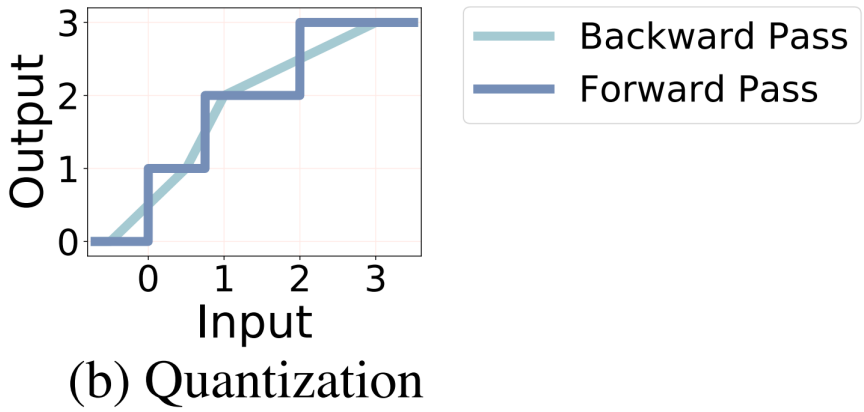

Figure 2.5 – Fonction de quantification et G-STE [Liu et al., 2022a]

Quantifieurs non-uniformes

Par opposition aux quantifieurs uniformes, les quantifieurs non-uniformes ne disposent pas d'un pas de quantification constant. Leurs contructions sont variées, dictées par la volonté de quantifier de manière plus précise certains paramètres. Ils peuvent avoir une structure prédéterminée, comme dans Li et al. [2020], ou bien reposer sur le clustering [Han et al., 2016].

La souplesse d'un schéma de quantification non-uniforme lui permet de capturer plus efficacement l'information en adaptant mieux la distribution de ses niveaux de quantification à celle des paramètres du réseau, typiquement en forme de gaussienne. Cependant, les quantifieurs non uniformes, en particulier non structurés, sont difficiles à mettre en œuvre de manière efficace sur du matériel. C'est pourquoi la quantification uniforme est souvent privilégiée en pratique, en raison de sa simplicité d'implémentation, permettant de recourir à des opérations en nombres entiers.

Deep compression

Han et al. [2016] présente une technique de quantification reposant sur le clustering (Figure 2.6). Les centroïdes sont initialisés par l'algorithme des k-moyennes, dont l'objectif est de minimiser la somme des distances au carré entre l'ensemble des poids à quantifier et le centroïde le plus proche pour chaque poids.

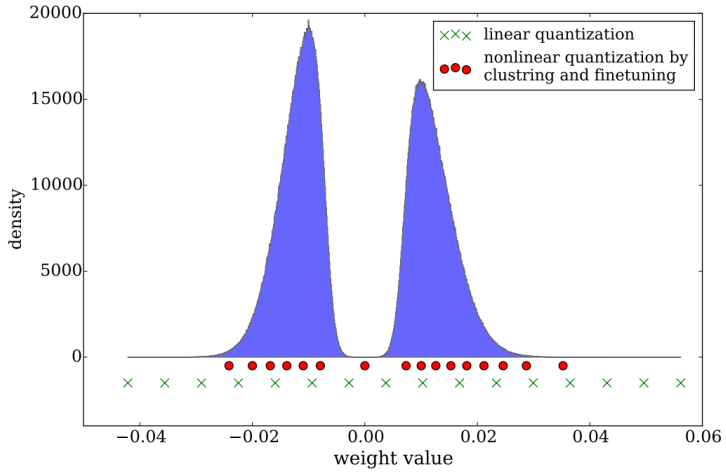

Figure 2.6 – Fonction de quantification de Deep Compression [Han et al., 2016]

L'intérêt d'un tel schéma est de concentrer les niveaux de quantification aux points critiques de la distribution, là où les paramètres à quantifier sont les plus nombreux. Les centroïdes C_k (en rouge sur la Figure 2.6) sont ensuite affinés par rétropropagation du gradient de la loss L suivant l'équation 2.16, établie par dérivation composée.

$$\frac{\partial L}{\partial C_k} = \sum_i \frac{\partial L}{\partial W_i} \frac{\partial W_i}{\partial C_k} = \sum_i \frac{\partial L}{\partial W_i} \mathbb{1}(I_i = k) \tag{2.16}$$

où I_i correspond au numéro du centroïde le plus proche de W_i et $\mathbb{1}$ est la fonction indicatrice.

Additive Power-of-Two quantization (APoT)

Li et al. [2020] s'appuie sur l'idée de la quantification en puissances de 2 dont l'objectif est de permettre une représentation précise des nombreux paramètres de valeurs proches de 0 tout en facilitant l'opération de multiplication, nécessaire pour calculer la convolution), la ramenant à de simples décalages sur les bits (équation 2.17) :

$$2^x r = \begin{cases} r & \text{si } x = 0 \\ r \ll x & \text{si } x > 0 \\ r \gg x & \text{si } x < 0 \end{cases} \tag{2.17}$$

Cependant, la quantification en puissances de 2 possède certains défauts, notamment celui d'être trop précise au voisinage de 0. En effet, pour une quantification sur 4 bits, les deux seuils non nuls les plus petits sont 2^{-15} et 2^{-14}. Une quantification aussi précise est inutile pour la performance du réseau. Par ailleurs, les deux seuils les plus grands sont 0.5 et 1, pouvant causer de lourdes erreurs de quantification sur les poids les plus élevés. Li et al.

[2020] propose une voie médiane, consistant à placer les niveaux de quantification selon le schéma suivant (pour une précision de $2n$ bits et des poids non signés) :

$$Q(\gamma, n) = \gamma \times \{\sum_{i=0}^{n-1} p_i\} \text{ avec } p_i \in \{0, \frac{1}{2^i}, \frac{1}{2^{(i+n)}}, \frac{1}{2^{(i+2n)}}\} \tag{2.18}$$

Où γ est un facteur d'échelle appris. On peut obtenir d'autres schémas en faisant varier le nombre de termes de la somme et à l'intérieur de chaque "bin" p_i. La Figure 2.7 est un exemple de schéma de quantification utilisant 4 bits obtenu par Li et al. [2020].

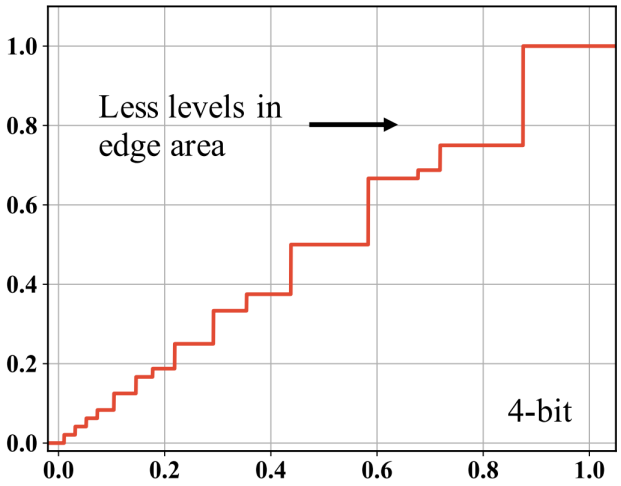

Figure 2.7 – Fonction de quantification d'APoT [Li et al., 2020]

FP8 quantization

L'apparition du matériel dédié pour le calcul 8-bit à virgule flottante a récemment renforcé l'intérêt pour les techniques de quantification utilisant ce format. Kuzmin et al. [2022] constate que le choix de la taille de l'exposant est déterminant pour les applications PTQ, un grand exposant permettant de mieux représenter les distributions comprenant beaucoup d'outliers, tandis qu'une telle représentation perd en importance pour le QAT, qui s'accommode aussi bien d'un schéma uniforme.

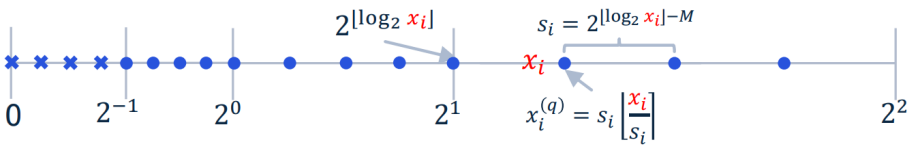

Figure 4: Graphic depiction of the FP8 quantizer.

Figure 2.8 – Niveaux de quantification de Kuzmin et al. [2022]

Quantifieurs par blocs

La quantification par blocs est une autre approche consistant à regrouper des paramètres pour les quantifier ensemble. Partant du même constat d'hétérogénéité de la distribution des valeurs à quantifier que la quantification non uniforme, les quantifieurs par blocs généralisent cette dernière en dimension supérieure. L'approche la plus fréquente repose sur la quantification-produit [Jégou et al., 2011], décomposant l'espace de recherche en produits d'espaces plus petits.

Les quantifieurs par blocs permettent d'atteindre les meilleurs taux de compression mémoire à performance égale, mais leur usage est limité en pratique par leur complexité d'implémentation qui ne permet généralement pas l'inférence en précision réduite, sauf combinaison avec une méthode de quantification uniforme [Fan et al., 2020]. En général, ils font usage de tables pour stocker les points de quantification, ce qui a le double inconvénient d'ajouter un overhead mémoire pour stocker la table et de nécessiter des adresses de plus en plus longues à mesure la table s'agrandit.

And the bit goes down

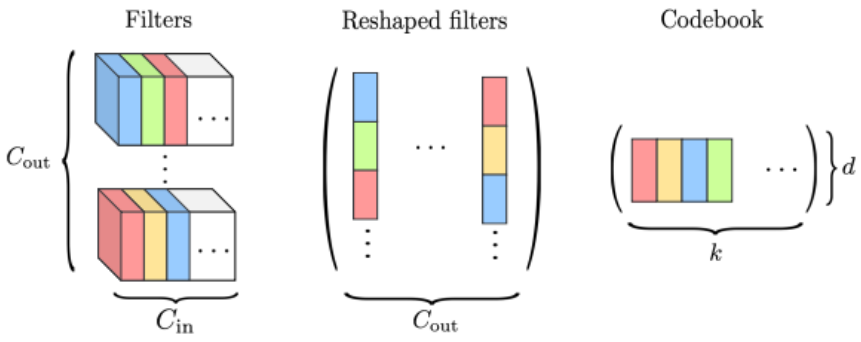

Figure 2.9 – Schéma de fonctionnement de Stock et al. [2020]

Stock et al. [2020] initialise les points de quantification par un clustering pondéré, dans l'objectif de reconstruire le plus fidèlement possible la sortie des couches du réseau. Les points de quantification sont stockés dans une table et entraînés par distillation selon une rétropropagation de gradient similaire à [Han et al., 2016].

2.2.3 . Quantification en précision mixte

Enfin, on identifie une dernière classe de méthodes de quantification qui étendent le problème le quantification à la recherche de la précision : les méthodes à précision mixte (mixed precision). Toutes les méthodes présentées jusqu'ici disposent d'une précision de quantification fixée au départ de l'entraînement. Cette précision peut varier entre les couches : en effet, il est commun de laisser la première couche de convolution et la dernière couche du classifieur en 8-bit, car la diminution de leur précision a tendance à dégrader plus fortement les performances du réseau. La précision mixte généralise cette pratique en permettent d'ajuster individuellement la précision de chacune des couches du réseau.

La précision mixte présente l'avantage d'une meilleure précision atteignable à taille du réseau ou coût calculatoire égaux, du moins en théorie, car l'espace de recherche englobe celui de la précision fixe. En revanche, elle va de pair avec une augmentation de la complexité matérielle : le fait de faire cohabiter plusieurs précisions de calcul au moment de l'inférence oblige à prendre en charge les différents opérateurs adaptés à ces différentes précisions. Elle peut s'appliquer aux poids comme aux activations. La recherche de la précision optimale peut se restreindre à une précision par bloc du réseau, à une précision par couche, ou bien s'étendre à une précision par canal.

Hardware-friendly Mixed-precision Quantization block (HMQ)

Habi et al. [2020] propose un moyen d'optimiser conjointement la précision de quantification et le seuil de saturation du quantifieur grâce à un "HMQ Block". Afin de contourner le caractère non différentiable du paramètre de précision, celui-ci est optimisé de manière stochastique. L'équation 2.19 représente une distribution aléatoire (où les $g_{t,b}$ sont des tirages d'une distribution de Gumbel) dont les paramètres $\widehat{\pi}_{t,b}$ sont appris. Les $P_{T,B}(T=t,B=b|g_{t,b})$ servent à paramétrer le quantifieur (en nombre de seuils b, seuil maximum de quantification t et pas de quantification Δ) selon les équations 2.20 et 2.21.

$$P_{T,B}(T=t,B=b|g_{t,b}) = \frac{exp(\frac{log(\widehat{\pi}_{t,b})+g_{t,b}}{\tau})}{\sum_{t'\in T}\sum_{b'\in B} exp(\frac{log(\widehat{\pi}_{t',b'})+g_{t',b'}}{\tau})} \tag{2.19}$$

$$\widehat{\Delta} = \sum_t \sum_b \Delta_{t,b} P_{T,B}(T = t, B = b | g_{t,b}) \tag{2.20}$$

$$\widehat{t} = \sum_t t P_T(T = t) \tag{2.21}$$

À l'inférence, la paire (t, b) est fixée à $\arg\max_{t,b}(\{\widehat{\pi_{t,b}}\})$.

Hessian-Aware Quantization of neural networks with mixed precision (HAWQ)

HAWQv2 [Dong et al., 2019a] cherche à mesurer l'importance relative des couches du réseau à l'aide d'un proxy. Il commence par démontrer que la valeur moyenne de la trace de la hessienne H_i des poids de la couche est un meilleur proxy que la plus grande valeur propre utilisée dans HAWQ [Dong et al., 2019b]. Dong et al. [2019b] calcule la trace de manière efficace, et en tire un ordre d'importance entre les couches, ce qui permet de réduire considérablement l'espace de recherche des précisions. Pour finir, il assigne à chacune des couches une précision en bits de sorte à minimiser la métrique Ω (équation 2.22), représentant la sensibilité totale du réseau au bruit de quantification en fonction d'une certaine assignation.

$$\Omega = \sum_{i=1}^{L} \overline{Tr(H_i)} ||\widehat{W_i} - W_i||_2^2 \tag{2.22}$$

Fracbits

Fracbits [Yang and Jin, 2020] traite les précisions comme des paramètres à part entière et intègre leur apprentissage à la phase de rétropropagation du gradient. Les précisions sont considérées comme des nombres flottants, et le calcul de la fonction de quantification s'effectue par pondération, selon l'équation 2.23.

$$q_{\lambda_i}(W_i) = (1 - (\lambda_i - \lfloor \lambda_i \rfloor)) q_{\lfloor \lambda_i \rfloor}(W_i) + (\lambda_i - \lfloor \lambda_i \rfloor) q_{\lceil \lambda_i \rceil}(W_i) \tag{2.23}$$

$\frac{\partial \lfloor \lambda_i \rfloor}{\partial \lambda_i}$ étant nul presque partout, le gradient de $q_{\lambda_i}(W_i)$ par rapport à λ_i est non nul, et les λ_i peuvent par conséquent être appris. Passé un certain stade de l'entraînement, les λ_i sont arrondis et fixés le temps d'une phase de finetuning.

2.3 . Conclusion de l'état de l'art

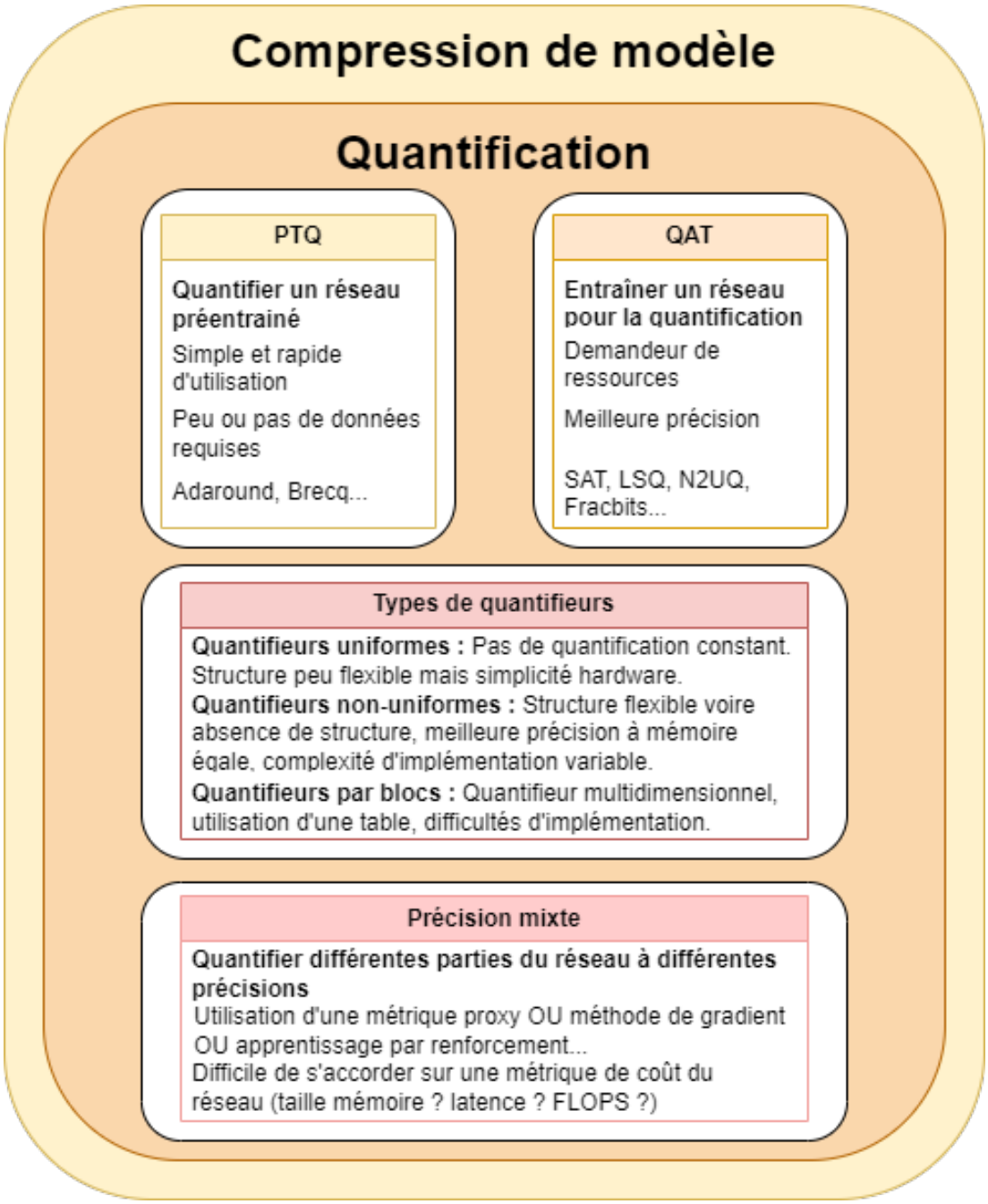

Figure 2.10 – Catégories de méthodes identifiées dans l'état de l'art de la quantification

On résume sur la Figure 2.10 les grandes catégories de méthodes identifiées dans cette partie. Outre la distinction entre post-training et quantization-

aware training, les méthodes de distinguent par la nature du quantifieur utilisé et par la recherche ou non d'une répartition optimale des précisions au sein des différentes parties du réseau.

Method	PTQ	QAT	Mixed P.	Accel.	Top-1 Acc.
[Banner et al., 2018]	✓	✗	✗/✓	✓	+
Brecq [Li et al., 2021]	✓	✗	✗/✓	✓	++
LSQ [Esser et al., 2020]	✗	✓	✗	✓	+++
APoT [Li et al., 2020]	✗	✓	✗	✓	+++
VectorQ [Stock et al., 2020]	✗	✓	✗	✗	++++
Fracbits [Yang and Jin, 2020]	✗	✓	✓	✓	++++

Les méthodes de quantification uniformes sont les plus utilisées en pratique, en conséquence de leur simplicité d'implémentation qui facilite leur déploiement sur une cible embarquée. Cependant, les méthodes non-uniformes et par blocs démontrent des capacités théoriques de compression de modèle bien supérieures du fait de leur meilleure adéquation à la distribution des paramètres du réseau et de leur plus grande capacité de représentation. Ce constat amène le travail de recherche présenté dans la partie suivante, dont l'objectif est le développement d'une méthode de quantification PTQ par blocs compatible avec des calculs en précision réduite, et donc avec une accélération matérielle.

Il est également à noter que les précisions de quantification agressives, notamment la binarisation, ont tendance à causer des dégradations importantes de la performance du réseau. Ce constat simple oriente un second travail de recherche, focalisé sur le QAT, qui vise à atténuer ces dégradations en proposant une recette générique d'adaptation de l'algorithme d'entraînement d'une méthode quelconque de quantification uniforme.

Enfin, un troisième axe de recherche exploré au cours de la thèse a été l'optimisation conjointe de la quantification en précision mixte et de la compression entropique, via une minimisation à l'apprentissage de l'entropie des paramètres du réseau. En d'autres termes, on a cherché l'équilibre optimal entre agressivité de la quantification et codage non destructif du point de vue mémoire et calculatoire.

3 - Lattice Quantization

L'état de l'art de la quantification fait la part belle aux quantifieurs uniformes [Jin et al., 2019, Banner et al., 2018, Choukroun et al., 2019], qui permettent de simplifier drastiquement l'inférence grâce au calcul en nombres entiers. Cependant, des types de quantifieurs plus flexibles [Han et al., 2016, Li et al., 2020] offrent un meilleur rapport précision/mémoire, grâce à une structure s'adaptant mieux à la distribution des paramètres du réseau. Lattice Quantization [Metz et al., 2023] vise à faire la synthèse entre simplicité d'implémentation d'un quantifieur uniforme et performance d'un quantifieur par blocs à travers un nouveau type de quantifieur.

Contributions

- On présente un nouveau quantifieur, basé sur la structure algébrique des réseaux euclidiens, pour permettre une quantification plus flexible et plus précise des poids que les techniques reposant sur la quantification uniforme.
- On étudie l'impact de la quantification des activations en conjonction avec ce quantifieur, pour permettre l'accélération de l'inférence.
- On teste notre quantifieur dans deux scénarios différents, en fonction de la disponibilité des données. On le compare avec l'état de l'art de la PTQ.
- On fournit une analyse du fonctionnement du quantifieur et de son surcoût mémoire.

3.1 . Observations

La structure du quantifieur doit répondre à deux exigences :

- S'adapter à la distribution des poids du réseau.
- Permettre l'accélération matérielle de l'inférence.

Ces deux points sont notamment satisfaits par Kuzmin et al. [2022] et Li et al. [2020], mais ces méthodes relèvent d'une quantification scalaire, qui néglige les relations entre paramètres, desquelles les quantifieurs par blocs peuvent tirer parti.

L'entraînement d'un réseau de convolution donne généralement lieu à une distribution en cloche de ses paramètres (voir Figure 3.1 pour l'exemple du Resnet-50). Cette donnée justifie l'intérêt de "densifier" les niveaux de quantification au voisinage de 0, pour minimiser l'erreur de quantification en ajou-

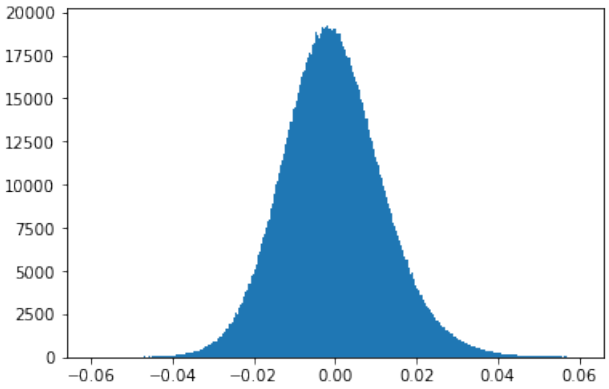

Figure 3.1 – Histogramme des poids de la couche de convolution 3×3 du bloc 4.0 d'un Resnet-50 (ImageNet). Abscisse : valeur du bin de poids. Ordonnée : nombre de poids du bin.

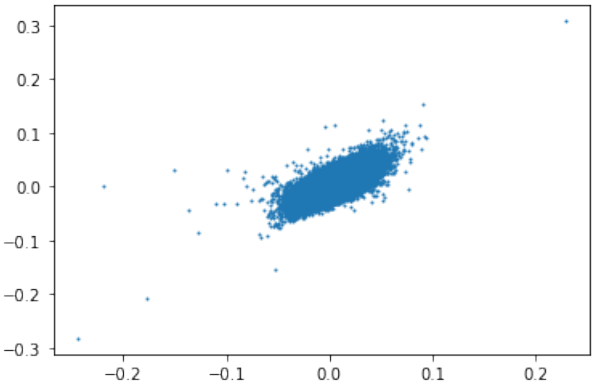

Figure 3.2 – Distribution jointe des poids 1 et 2 de chaque kernel de la couche de convolution 3×3 du bloc 4.0 d'un Resnet-50 (ImageNet). Abscisse : valeur poids 1. Ordonnée : valeur poids 2.

tant de la précision là où les paramètres sont les plus nombreux, mais ne nous donne aucune information sur la répartition multidimensionnelle de ces paramètres. On s'intéresse donc au tenseur des poids d'une couche de convolution : $\mathbf{W_{cout,cin,k^2}}$, où c_{in} est le nombre des canaux d'entrée, c_{out} celui des canaux de sortie et k^2 la taille de chaque kernel de convolution. Dans la suite, on supposera $k = 3$. Pour chaque $1 \leq i \leq c_{out}, 1 \leq j \leq c_{in}$, on ajoute le point $(W_{i,j,1}, W_{i,j,2})$ sur la Figure 3.2. La forme particulière du nuage de points obtenu peut être observée sur de nombreuses autres architectures de réseaux convolutionnels.

On note la présence de corrélations entre les paramètres d'un même filtre de convolution. Sur la Figure 3.3, pour chaque $1 \leq i \leq c_{out}, 1 \leq j \leq c_{in}$, on

trace en ligne m et colonne n le point $(W_{i,j,m}, W_{i,j,n})$, de sorte à obtenir la matrice de corrélation des poids en fonction de leur position dans le filtre (le poids 1 est situé en haut à gauche, le poids 2 en haut au milieu...). On note $\rho_{X,Y}$ le coefficient de corrélation de Pearson de deux variables aléatoires X et Y :

$$\rho_{X,Y} = \frac{\mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))]}{\sigma_X \sigma_Y} \tag{3.1}$$

où $\sigma(X)$ est l'écart-type de la variable aléatoire X et $\mathbb{E}$ l'espérance mathématique.

Ce coefficient mesure la force des corrélations linéaires de deux distributions entre -1 et 1, avec -1 = anticorrélation parfaite, 1 = corrélation parfaite, 0 = indépendance. On considère les $W_{.,.,m}$ comme des variables aléatoires, desquelles les $W_{i,j,m}$ sont des tirages, et on calcule leurs coefficients de corrélation deux à deux. Sur la diagonale, on trace l'histogramme des $W_{i,j,m}$.

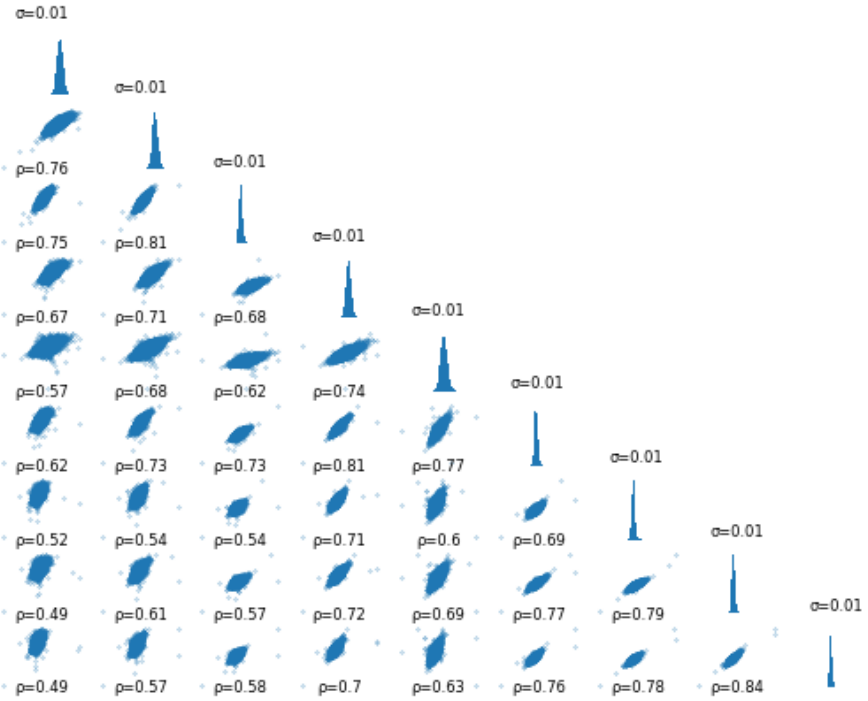

Figure 3.3 – Matrice de corrélation de la couche de convolution 3×3 du bloc 4.0 d'un Resnet-50 (ImageNet)

Illustrée dans le cas d'un Resnet-50, ces mesures confirment expérimentalement l'existence de corrélations linéaires entre les poids d'un même filtre. Autrement dit, plus certains poids d'un filtre sont grands, et plus il y a de

chances que les autres poids du même filtre le soient également. Le même type de distribution peut être constaté au sein des couches 3×3 d'autres topologies telles que VGG [Simonyan and Zisserman, 2015] ou Densenet [Huang et al., 2018]. Il serait donc judicieux que notre quantifieur puisse épouser ces corrélations, comme représenté sur la Figure 3.4 (droite).

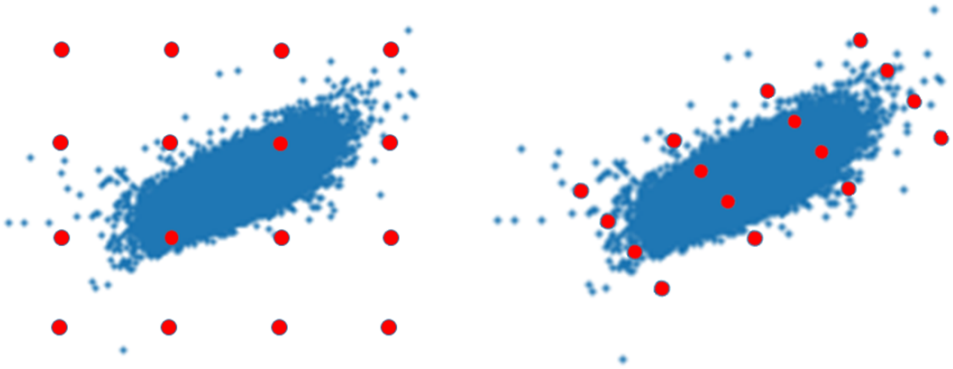

Figure 3.4 – Gauche : structure d'un quantifieur scalaire uniforme. Droite : structure souhaitée pour le quantifieur de LatticeQ. En rouge les points de quantification.

On constate par la même occasion que la structure d'un quantifieur scalaire uniforme (comme ceux de Banner et al. [2018], Nagel et al. [2020], Li et al. [2021]) semble très inadaptée à la prise en compte de ces corrélations, du fait de l'éloignement de certains de points de quantification de la distribution des poids.

3.2 . Méthodes

Dans cette partie, on présente les notions et les méthodes qui ont servi à l'implémentation de Lattice Quantization (LatticeQ).

3.2.1 . Réseaux euclidiens

On commence par définir formellement la notion de réseau euclidien sur laquelle se base la structure de notre quantifieur. Les réseaux euclidiens présentent trois intérêts majeurs pour la quantification :

1. L'existence de vecteurs générateurs dispensant de l'utilisation d'une table exhaustive pour stocker les points de quantification.
2. L'existence de réseaux présentant des caractéristiques algébriques particulières de densité.
3. Une zone de quantification plus flexible, et potentiellement plus adaptée à des données corrélées linéairement.

On justifie ici les deux premiers points. On mettra en évidence le troisième au paragraphe 3.5.

Définition 1

Un réseau euclidien Λ de $\mathbb{R}^n$ est un sous-groupe discret de $\mathbb{R}^n$ pour l'addition tel que $\mathbb{R}^n$ est engendré par Λ.

Exemple

$\mathbb{Z}^n$ est un réseau euclidien. Il s'agit du réseau cubique. La répartition des niveaux d'une quantification uniforme (Figure 3.5 gauche) correspond à un réseau cubique tronqué, ici représenté en dimension 2.

Propriété 1

Soit Λ un réseau de $\mathbb{R}^n$, il existe une famille $\mathcal{B}$ de n vecteurs du réseau telle que tout élément de Λ s'exprime de manière unique comme combinaison linéaire à coefficients entiers des éléments de $\mathcal{B}$. Une telle famille porte le nom de base.

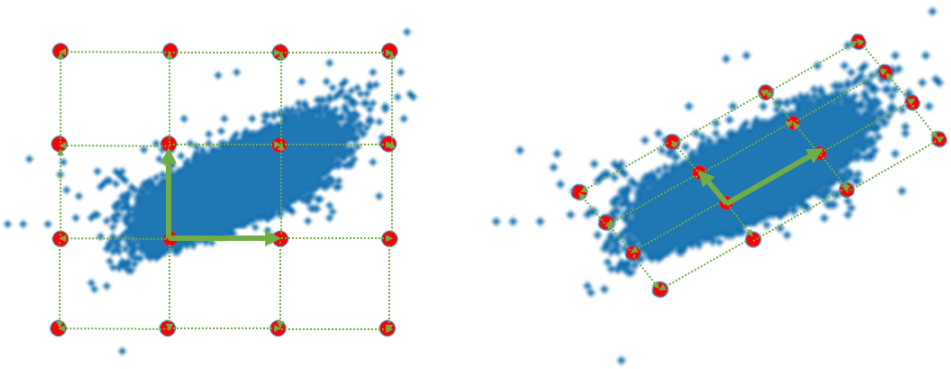

Figure 3.5 – Bases de quantification (flèches vertes trait plein) pour la quantification uniforme (gauche) et pour la quantification en réseau euclidien (droite)

Idée de la preuve

Soit $\mathcal{B}$ une famille de n vecteurs libres de Λ, de normes les plus petites possible. La minimalité des normes garantit qu'aucun vecteur du réseau n'est inclus dans le parallélogramme délimité par ces vecteurs (à part ses sommets qui sont les combinaisons entières "en 0-1" des vecteurs choisis). On considère un vecteur λ du réseau. Sur le modèle d'une division euclidienne, on retranche à λ les parties entières de ses coordonnées suivant les vecteurs de

$\mathcal{B}$. On trouve que le résultat est inclus dans le parallélogramme, ce qui prouve qu'il est nul, et donc que λ s'écrit comme combinaison entière des vecteurs de $\mathcal{B}$.

Remarque

La connaissance d'une base d'un réseau Λ est donc suffisante pour connaître l'ensemble des points du réseau. Dans la perspective d'utiliser un réseau euclidien pour quantifier une distribution de poids, cette observation permet de s'affranchir de la nécessité de conserver en mémoire une représentation exhaustive de l'ensemble des points de quantification.

Définition 2

Soit $\mathcal{B} = (b_1, ..., b_n)$ une base d'un réseau euclidien Λ. La maille de Λ associée à $\mathcal{B}$ est le polytope convexe $\{\sum_{i=1}^{n} \mu_i b_i,\ 0 \leq \mu_i \leq 1\}$.

Définition 3

Le covolume d'un réseau Λ est un invariant de la base du réseau. Il est défini par la quantité $det(b_1, ..., b_n)$. Il s'agit du volume d'une maille.

Remarque

En dimension 1, la notion de covolume correspond à la longueur de l'unique vecteur de base, en dimension 2, à l'aire d'une maille, et en dimension 3, au volume d'une maille. Pour comparer les "densités" de deux réseaux distincts, il est nécessaire de normaliser leurs covolumes.

Définition 4

On appelle rayon de couverture d'un réseau Λ la quantité :

$$\mu_\Lambda = inf\{R \in \mathbb{R}_+, \bigcup_{\lambda \in \Lambda} B(\lambda, R) = \mathbb{R}^n\} \tag{3.2}$$

En d'autres termes, il s'agit du plus petit réel R tel qu'un recouvrement par des boules de rayon R centrées en chaque vecteur λ du réseau recouvre intégralement l'espace (voir Figure 3.6).

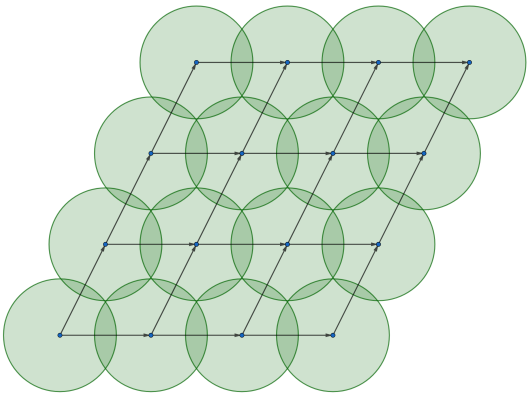

Figure 3.6 – Recouvrement du plan par des disques de rayon $\geq \mu_\Lambda$, centrés en chaque $\lambda \in \Lambda$.

Remarque

Soit un réseau de rayon de couverture μ en dimension n, on peut alors garantir que tout point de l'espace se situe à une distance d'au plus μ d'un vecteur de ce réseau. μ constitue donc une borne supérieure pour l'erreur de quantification d'un vecteur sur ce réseau.

Propriété 2

En dimension 2 et au-delà, il existe des réseaux euclidiens plus denses, au sens du rayon de couverture, que le réseau cubique.

Exemple

En dimension 2, le réseau hexagonal normalisé possède un rayon de couverture $\mu = \sqrt{\frac{2}{3\sqrt{3}}} \approx 0.62$, tandis que le réseau cubique normalisé possède un rayon de couverture $\mu = \frac{\sqrt{2}}{2} \approx 0.71$.

Remarque importante

Il faut noter que le schéma de quantification uniforme utilisé classiquement n'est rien d'autre qu'une quantification en réseau cubique. La quantification en réseau euclidien n'est donc qu'une généralisation de l'approche scalaire.

3.2.2 . Quantification des poids

Jusqu'ici, on a utilisé le terme "quantification" sans détailler formellement à quoi il faisait référence. Dans le cas d'une quantification scalaire, quantifier

revient à trouver le niveau de quantification le plus proche, ce qui peut être fait très simplement via un arrondi ou une table. Dans le cas de la quantification vectorielle, on fait également appel à une table qui répertorie tous les points de quantification. La recherche naïve du point de quantification le plus proche est alors d'une complexité linéaire en la taille de la table. Au cours de l'entraînement, c'est le coût de la recherche qui limite la taille de cette table, tandis qu'à l'inférence, c'est le coût mémoire du stockage de la table qui devient limitant. Dans notre cas, la notion de base d'un réseau euclidien nous exonère de la nécessité de stocker toute la table des points de quantification, puisque la donnée de la base seule suffit.

Un réseau euclidien est de cardinal infini par définition. Afin d'utiliser cette structure comme quantifieur, il faut d'abord se ramener à une structure de cardinal fini et donc la tronquer. On procède comme suit : soit Λ un réseau euclidien de base $\mathcal{B} = (\mathbf{b_i})_{1 \leq i \leq n} \in \mathbb{R}^n$ et b une certaine précision donnée en bits. On note Q l'ensemble des points de quantification. On pose :

$$Q = \{q \in \mathbb{R}^n, q = \sum_{i=1}^{n} \delta_i \mathbf{b_i},\ \forall i \in \{1, ..., n\},\ -2^{b-1} \leq \delta_i \leq 2^{b-1} - 1\} \tag{3.3}$$

Dans le cadre de notre approche, quantifier, autrement dit trouver le point de notre réseau euclidien de quantification le plus proche d'un vecteur quelconque, devient un problème classique de la théorie des réseaux euclidiens. Ce problème porte le nom de Closest Vector Problem (CVP) :

CVP : "Soit $x \in \mathbb{R}^n$, Λ un réseau de $\mathbb{R}^n$ et d une distance dans $\mathbb{R}^n$, trouver $\lambda \in \Lambda$ tel que $d(x, \lambda) = min\{d(x, l),\ l \in \Lambda\}$."

Algorithm 2 Nearest plane algorithm

function Babai(Basis $\mathcal{B} = (b_i)_{1 \leq i \leq n}$, Vector t, Bitwidth b)
  $\mathcal{B}^* \leftarrow GramSchmidt(\mathcal{B}) = (b_i^*)_{1 \leq i \leq n}$
  $r \leftarrow t$
  for $j \in \{n, ..., 1\}$ **do**
    $u_j \leftarrow clip(\frac{<r, b_j^*>}{<b_j^*, b_j^*>}, -2^{b-1}, 2^{b-1} - 1)$
    $r \leftarrow r - \lfloor u_j \rceil b_j$
  end for
  return $x = \sum_{j=1}^{n} \lfloor u_j \rceil b_j = t - r$
end function

Le CVP est NP-difficile [Micciancio, 1998], mais il existe des algorithmes d'approximation pour le résoudre en temps polynomial. L'algorithme du plan

le plus proche (Nearest Plane Algorithm, Algorithme 2) Babai [1986] est un de ces algorithmes d'approximation. On choisit de l'utiliser du fait de sa simplicité d'implémentation et de sa fiabilité en basse dimension. La notation $< x, y >$ désigne le produit scalaire et $\lfloor x \rceil$ l'arrondi.

Algorithm 3 Gram-Schmidt algorithm

function GramSchmidt(Basis $\mathcal{B}$)
  $b_1^* \leftarrow b_1$
  for $j \in \{2, ..., n\}$ **do**
    $b_j^* \leftarrow b_j - \sum_{i=1}^{j-1} < b_j, b_i^* > b_i^*$
  end for
  return $\mathcal{B}^* = (b_j^*)_{1 \leq j \leq n}$
end function

Enfin, sur le même principe que Stock et al. [2020], on se sert de la quantication produit (product quantization) [Jégou et al., 2011] pour séparer les blocs de paramètres à quantifier. La dimension de la base de quantification doit être égale à la taille des blocs, de sorte à ce qu'elle engendre tout l'espace. Expérimentalement, on trouve que la dimension optimale est de 3 (voir Table 3.2).

Quantification par couche / par canal

Il est possible de quantifier les poids du réseau par canal, auquel cas on choisira pour chaque canal un pas de quantification différent pour une méthode uniforme et une base de quantification différente pour LatticeQ. La quantification par canal est particulièrement avantageuse dans le cas d'une hypothèse de PTQ stricte, sans réentraînement, afin d'adapter le quantifieur aux différences d'échelle entre les poids des différents canaux. En revanche, elle se paie en termes de complexité mémoire et d'implémentation matérielle [Nagel et al., 2021].

3.2.3 . Déquantification des poids

Chaque bloc quantifié est représenté en mémoire par ses coordonnées dans la base de quantification (entières et comprises entre -2^{b-1} et $2^{b-1} - 1$, par définition). Soit notre base de quantification $\mathcal{B}$:

$$\mathcal{B} = (\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}) = ((b_{1,1}, b_{1,2}, b_{1,3}), (b_{2,1}, b_{2,2}, b_{2,3}), (b_{3,1}, b_{3,2}, b_{3,3})) \tag{3.4}$$

avec chaque $b_{i,j}$ réel. Soit un filtre de convolution 3×3 quantifié F^q :

$$F^q = \begin{pmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{pmatrix} \tag{3.5}$$

F^q comporte 3 blocs quantifiés : (f_1, f_2, f_3), (f_4, f_5, f_6) et (f_7, f_8, f_9). La convolution $\hat{F}$ représentée par F^q est donc la concaténation de 3 blocs 1×3 :

$$\hat{F} = Dequant(F^q) = \begin{pmatrix} (f_1\mathbf{b_1} + f_2\mathbf{b_2} + f_3\mathbf{b_3}) \\ (f_4\mathbf{b_1} + f_5\mathbf{b_2} + f_6\mathbf{b_3}) \\ (f_7\mathbf{b_1} + f_8\mathbf{b_2} + f_9\mathbf{b_3}) \end{pmatrix} \tag{3.6}$$

Si l'on choisit $\mathcal{B}$ comme l'homothétie d'une base orthonormale ($\mathcal{B} = \lambda \times I_n$), la structure du quantifieur est alors exactement celle de la quantification uniforme classique (on appellera cette version simplifiée "Cubic LatticeQ" dans la suite). En pratique, on peut utiliser $Dequant$ pour calculer la convolution à l'inférence, mais on propose dans la suite une implémentation optimisée qui supporte les calculs en précision réduite.

3.2.4 . Inférence optimisée

Réduire le coût du stockage mémoire des poids réduit également le coût des transferts mémoire et est donc favorable à l'accélération de l'inférence. Cependant, pour exploiter tout le potentiel de la quantification, on souhaite pouvoir utiliser des opérateurs low-bit. On veut montrer qu'il existe une manière d'implémenter l'inférence de manière efficace avec LatticeQ.

Soit $(C_i^{in})_{1 \leq i \leq in}$ l'ensemble des features d'entrée (quantifiées) et $(C_j^{out})_{1 \leq j \leq out}$ l'ensemble des features de sortie. Soit W^q le tenseur des poids quantifiés. Soit $\mathcal{B}^{ext}$ la base de quantification étendue en dimension 9 :

$$\begin{aligned} \mathcal{B}^{ext} = ((\mathbf{b_1}, \mathbf{0}, \mathbf{0}), (\mathbf{b_2}, \mathbf{0}, \mathbf{0}), (\mathbf{b_3}, \mathbf{0}, \mathbf{0}), (\mathbf{0}, \mathbf{b_1}, \mathbf{0}), (\mathbf{0}, \mathbf{b_2}, \mathbf{0}), \\ (\mathbf{0}, \mathbf{b_3}, \mathbf{0}), (\mathbf{0}, \mathbf{0}, \mathbf{b_1}), (\mathbf{0}, \mathbf{0}, \mathbf{b_2}), (\mathbf{0}, \mathbf{0}, \mathbf{b_3})) \end{aligned}$$

où $(\mathbf{0}, \mathbf{b_i}, \mathbf{0})$ représente le vecteur $(0, 0, 0, b_{i,1}, , b_{i,2}, , b_{i,3}, 0, 0, 0)$. On a alors :

$$\hat{F} = \sum_{i=1}^{9} f_i \times B_i^{ext} \tag{3.7}$$

Calculer l'inférence revient à calculer C_j^{out} pour tout j. Dans une couche non quantifiée, C_j^{out} est calculé comme suit :

$$C_j^{out} = \sum_{i=1}^{in} Conv(W_{i,j}; C_i^{in}) \tag{3.8}$$

Avec LatticeQ, on change l'ordre des opérations :

$$C_j^{out} = \sum_{k=1}^{9} \sum_{i=1}^{in} W_{i,j,k}^q Conv(B_k^{ext}; C_i^{in}) \tag{3.9}$$

Comme $Conv(B_k^{ext}; C_i^{in})$ ne dépend pas du canal de sortie, on peut commencer par calculer $Conv(B_k^{ext}; C_i^{in})$ pour tous i, k ($9 \times in$ convolutions). La base $\mathcal{B}$ étant elle-même quantifiée uniformément via un quantifieur min/max, ces convolutions peuvent donc être calculées en low-bit. Il suffit ensuite de combiner ces résultats intermédiaires en multipliant par les entiers $W^q_{i,j,k}$ pour tout j afin d'obtenir la feature de sortie selon l'équation 3.9. Avec ce réordonnancement, la complexité de l'inférence sur cette couche devient $9.in$ convolutions low-bit et $9.in.out$ multiplications et additions scalaires.

3.2.5 . Choix de la base de quantification

On dispose d'un algorithme pour quantifier sur un réseau euclidien et d'un moyen optimisé de calculer l'inférence. Il ne reste plus qu'à sélectionner un "bon" réseau euclidien pour la quantification. Banner et al. [2018], Choukroun et al. [2019] fixent comme objectif pour leurs quantifieurs la réduction de la moyenne des carrés de l'erreur (MSE) entre le tenseur des poids quantifiés et le tenseur non quantifié. Expérimentalement, on trouve que minimiser la moyenne des cubes (Mean-Cube Error, MCE) donne des résultats légèrement supérieurs (Table 3.1). On explique ce constat par le fait que la MCE accorde plus d'importance aux poids éloignés de tout point de quantification, et donc par conséquents éloignés de 0, qui pèsent particulièrement sur la qualité de l'inférence. Comme algorithme de recherche de la base de quantification, on adopte une simple recherche aléatoire (Algorithme 4). Cette recherche est exécutée plusieurs fois de suite afin de maximiser les chances de découvrir une base performante.

Algorithm 4 FindBasis

function FindBasis(Weight tensor W, Basis dimension dim, Bit-width $bits$, Temperatures T)
  $\mathcal{B} \leftarrow I_{dim}$
  for $0 \leq s \leq |T|$ **do**
    $\mathcal{B}' \leftarrow \mathcal{B}' + Sample(\mathcal{G}(0, \sigma = T(s), |\mathcal{B}|))$
    if $MCE(W, W^q_{\Lambda_{\mathcal{B}'}}) < MCE(W, W^q_{\Lambda_{\mathcal{B}}})$ **then**
      $\mathcal{B} \leftarrow \mathcal{B}'$
    end if
  end for
  return $\mathcal{B}$
end function

Table 3.1 – Comparaison des loss MSE et MCE pour Resnet-18 avec poids quantifiés 4-bit.

	Top-1 accuracy	
Network	MSELoss	MCELoss
Resnet-18	67.85	**67.94**

Table 3.2 – Impact du choix de la dimension de la base pour les couches 3×3 sur la performance finale avec poids quantifiés 4-bit.

	Top-1 accuracy	
Network	$dim = 3$	$dim = 9$
Resnet-18	**67.94**	61.90

La transformation aléatoire que l'on utilise est une perturbation par addition d'un bruit gaussien multivarié d'écart-type $T(s)$, où T est une suite de paliers de température pouvant être ajustée en tant qu'hyperparamètre. Comme précisé au paragraphe précédent, il est possible d'effectuer une inférence low-bit avec LatticeQ à condition de quantifier la base. Pour ce faire, on peut ajouter un quantifieur min/max pour $\mathcal{B}'$ entre les lignes 4 et 5 de l'algorithme 4.

3.2.6 . Stratégies post-entraînement

L'état de l'art de la PTQ ne s'accorde pas sur les hypothèses expérimentales. On répertorie les différentes hypothèses existantes dans la classification suivante, inspirée par celle de Nagel et al. [2019] :

- **Niveau 1a.** Aucune donnée disponible et aucun finetuning des paramètres du réseau. "Aussi simple qu'un appel à une API". Les seules entrées nécessaires sont la définition du modèle et ses paramètres [Nagel et al., 2019].
- **Niveau 1b.** Aucun finetuning des paramètres du réseau. Une faible quantité de données non annotées peuvent être utilisées pour calibrer les activations. La quantification par canaux est largement privilégiée [Banner et al., 2018, Choukroun et al., 2019, Zhao et al., 2019].
- **Niveau 2.** Des échantillons non annotés sont nécessaires pour procéder à un finetuning des poids et des activations. La quantification par couches devient le standard à partir de ce niveau [Li et al., 2021, Choukroun et al., 2019, Nagel et al., 2020, Wang et al., 2020a].
- **Niveau 3.** QAT. Nécessite l'intégralité du des données d'entraînement, un pipeline complet d'optimisation et une recherche d'hyperparamètres pour atteindre une bonne performance [Jin et al., 2019, Esser et al., 2020, Liu et al., 2022a].

Naturellement, les méthodes de niveau plus élevé affichent de meilleures

précisions, mais sont plus complexes à mettre en pratique car elles requièrent soit des données supplémentaires, soit un entraînement plus long et plus coûteux. Pour mettre LatticeQ à l'épreuve, on la teste sous les deux ensembles d'hypothèses les plus fréquents en PTQ : 1b et 2.

3.2.7 . Quantification des activations

Quantifier les activations n'est pas l'objet de LatticeQ, mais on note qu'elle peut être combinée avec n'importe quelle autre méthode pour cela de sorte à accélérer l'inférence. On propose néanmoins une manière simple de quantifier les activations à seule fin de comparaison pour le niveau 1b.

À l'instar de Banner et al. [2018], on choisit de quantifier les activations en utilisant un batch de test non annoté. Comme pour les poids, on peut soit quantifier les activations par canal, soit les quantifier par couche. La quantification des activations par couche en moins de 8 bits cause une dégradation importante de la précision du réseau. Pour une quantification par canal, il est en revanche possible d'atteindre une précision de 4 bits pratiquement sans pertes en utilisant une méthode similaire à McKinstry et al. [2019].

Soit C une feature à quantifier et $q \in]0, 1[$ une probabilité. Soit α_C le quantile de probabilité q de C, interprété comme une distribution :

$$p_{c \in C}(c \leq \alpha_C) = q \tag{3.10}$$

On estime α_C en effectuent une seule inférence. On utilise ensuite α_C comme seuil de saturation pour la quantification des activations du canal C. On choisit différents quantiles en fonction de l'architecture du réseau et de la précision souhaitée pour les activations (voir Table 3.3 pour les quantiles choisis pour Resnet).

Table 3.3 – Quantiles choisis pour Resnet

Bitwidth	2	3	4	6	8
Quantile	0.992	0.9991	0.9997	0.99999	1

3.3 . Expériences

On évalue LatticeQ sur la tâche de classification d'images du dataset ImageNet Russakovsky et al. [2015]. Toutes les expériences sont réalisées à l'aide de la plateforme PyTorch Paszke et al. [2019]. Les modèles préentrainés pro-

viennent de la librairie torchvision.models. À partir de résultats expérimentaux, on fixe les dimensions des bases de quantification à 3 pour les couches de convolution 3×3, et 2 pour les couches 1×1. On quantifie la première et la dernière couche en 8 bits (selon les pratiques de l'état de l'art) avec CubicLatticeQ, n'ayant pas trouvé d'avantage à utiliser LatticeQ en dimension > 1 sur ces couches. Pour chaque expérience, on donne la précision en bits et la précision top-1.

3.3.1 . Niveau 1b : LatticeQ sans échantillons de finetuning

On suppose dans cette partie qu'aucune donnée n'est disponible pour la quantification des poids et que seul un batch de test est disponible pour la calibration des activations (si elles sont quantifiées). Les résultats de LatticeQ sont ici comparés à ceux de deux méthodes de l'état de l'art du niveau 1b [Banner et al., 2018, Choukroun et al., 2019]. ZS LatticeQ (Zero-shot LatticeQ) désigne notre méthode restreinte aux poids, de sorte qu'elle ne nécessite strictement aucune donnée, ni de finetuning, ni de calibration.

On utilise à la fois la quantification par canal et la correction de biais proposée par Banner et al. [2018], tout à fait applicable en parallèle de LatticeQ. La correction de biais est calculée non seulement sur les poids quantifiés, mais également au moment de la recherche de la base par l'algorithme 4 pour des précisions inférieures ou égales au 4-bit, de sorte à ce que la base de quantification prenne en compte la correction de biais. La ligne 5 de l'algorithme est alors remplacée par :

if $MCE(W, \beta(W^q_{\Lambda_{\mathcal{B}'}})) < MCE(W, \beta(W^q_{\Lambda_{\mathcal{B}}}))$ **then**
  $\mathcal{B} \leftarrow \mathcal{B}'$
end if

où β est la fonction de correction de biais.

Les résultats de la table 3.4 montrent les améliorations substantielles apportées par LatticeQ sur les approches existantes. Notre méthode surpasse Banner et al. [2018] avec les mêmes hypothèses expérimentales, ainsi que la méthode OMSE+opt Choukroun et al. [2019] sur tous les modèles testés. LatticeQ parvient à limiter à moins de 1% la perte de précision par rapport au modèle préentraîné sur Resnet avec des poids 4-bit, et à 3% avec des poids 3-bit, là où les performances de Banner et al. [2018] s'effondrent. Les résultats de la table 3.5 montrent que LatticeQ est compatible avec la quantification des activations. Malgré la simplicité de notre méthode de quantification des activations, les performances de LatticeQ demeurent supérieures à celles des méthodes existantes sur la quasi-totalité les modèles testés. Comme attendu au regard de l'état de l'art, les modèles plus compacts tels que Densenet et MobileNet sont moins résilients à la quantification que VGG et Resnet.

Table 3.4 – LatticeQ avec correction de biais sur ImageNet. On utilise la quantification par canal. Les résultats de Choukroun et al. [2019] sont issus du papier, ceux de Banner et al. [2018] sont reproduits à l'aide du code source. La notation "WkAn" désigne la quantification k-bit des poids et n-bit des activations (32-bit équivaut à la précision flottante FP32).

Network	Method	Top-1 accuracy W4A32	W3A32	W2A32
Resnet-18 (69.8)	ZS LatticeQ (Ours)	**69.0**	**66.7**	**41.7**
	Banner et al.	67.5	43.2	1.2
	OMSE+opt	67.1		
Resnet-50 (76.0)	ZS LatticeQ (Ours)	**75.6**	**73.6**	**47.3**
	Banner et al.	74.8	67.4	0.4
	OMSE+opt	74.7		
VGG16-bn (73.4)	ZS LatticeQ (Ours)	**72.9**	**70.9**	**40.7**
	Banner et al.	71.6	65.9	0.1
Densenet-121 (74.4)	ZS LatticeQ (Ours)	**73.3**	**68.9**	**10.4**
	Banner et al.	69.8	54.2	0.5
	OMSE+opt	71.7		
MobileNet-v2 (71.9)	ZS LatticeQ (Ours)	**68.2**	**48.9**	**0.3**
	Banner et al.	62.8	13.9	0.1

3.3.2 . Niveau 2 : LatticeQ avec échantillons de finetuning

Les travaux les plus récents du domaine de la PTQ exploitent souvent les données d'un batch de calibration afin de finetuner les poids quantifiés [Nagel et al., 2020, Li et al., 2021, Wang et al., 2020a]. Cette pratique permet d'atténuer les différences en performance entre PTQ et QAT sur une variété d'architectures et de tâches. On a souhaité démontrer la compatibilité de LatticeQ avec cette pratique de finetuning. En utilisant le résultat de la reconstruction par blocs de Li et al. [2021], on implémente un mécanisme simple de finetuning. Soit $\mathcal{L}$ une fonction de perte (loss). À chaque nouveau bloc B_i, on calcule x_i l'entrée de ce bloc dans le réseau préentraîné et x_i^q l'entrée de ce bloc dans le réseau quantifié. Pour chaque bloc, le problème d'optimisation est donné par l'équation 3.11 :

$$\arg\min_{B_i^q} \mathcal{L}(B_i(x_i), B_i^q(x_i^q)) \tag{3.11}$$

Table 3.5 – LatticeQ avec correction de biais sur ImageNet, et quantification des activations. On utilise la quantification par canal pour les poids, par canal pour les activations 4-bit, par couche pour les activations 8-bit.

		Top-1 accuracy				
Network	Method	W4A32	W4A8	W4A4	W3A8	W2A8
Resnet-18 (69.8)	LatticeQ (Ours)	**68.99**	**69.04**	**67.05**	**66.35**	**39.20**
	Banner et al.		67.4	64.3	43.4	1.3
	OMSE+opt	67.1				
Resnet-50 (76.0)	LatticeQ (Ours)	**75.63**	**75.57**	**71.87**	**73.37**	**47.15**
	Banner et al.		74.8	70.3	67.5	0.4
	OMSE+opt	74.7				
VGG16-bn (73.4)	LatticeQ (Ours)	**72.89**	**72.89**	**71.08**	**70.90**	**40.67**
	Banner et al.		71.6	70.4	65.9	0.1
	OMSE+opt	72.3				
Densenet-121 (74.4)	LatticeQ (Ours)	**73.30**	70.16	**66.97**	**64.75**	**9.86**
	Banner et al.		70.2	63.0	53.8	0.4
	OMSE+opt	71.7				
MobileNet-v2 (71.9)	LatticeQ (Ours)	**68.23**	**66.89**	**50.90**	**45.45**	**0.34**
	Banner et al.		61.1	36.3	10.2	0.1

Table 3.6 – LatticeQ with data samples

		Top-1 accuracy	
Network	Method	W4A32	W3A32
Resnet-18 (69.8)	LatticeQ (Ours)	**69.3**	**68.6**
	Adaround	68.6	
	Bitsplit	69.1	66.8
Preresnet-18 (71.0)	LatticeQ (Ours)	**70.5**	**69.3**
	Brecq	70.3	69.0

On commence par initialiser les bases de quantification pour chaque couche à l'aide de l'algorithme 4. On sélectionne un échantillon de 512 images de ca-

libration, et on les sépare en 4 batchs de 128. En utilisant cet échantillon, on effectue 2000 époques d'entraînement pour chaque bloc, et on apprend ses poids et ses bases de quantification à l'aide de l'optimiseur Adam [Kingma and Ba, 2017]. Comme loss $\mathcal{L}$, on utilise la MSE sur la sortie des blocs intermédiaires du réseau. Les gradients sont rétropropagés à travers l'arrondi de l'algorithme 2 à l'aide du STE (straight-through estimator). Le taux d'apprentissage (learning rate) est fixé à 10^{-6}. On compare ici LatticeQ avec des méthodes de l'état de l'art du niveau 2. On a reproduit les expériences de Li et al. [2021] via le code source avec une quantification par couche, standard sous ces hypothèses expérimentales.

La table 3.6 montre que LatticeQ a également du potentiel sous l'hypothèse du finetuning. Notre méthode maintient la perte en précision aux alentours de 1% sur un Resnet18 quantifié en 3 bits et améliore légèrement les résultats de Brecq sur Preresnet.

3.4 . Étude d'ablation

Table 3.7 – LatticeQ par canal sans correction de biais

	Top-1 accuracy		
Network	FP32	W4A8	W4A4
Resnet-18	69.8	67.2	66.0
Resnet-50	76.0	74.6	70.6
VGG-16bn	73.4	72.4	70.4
Densenet-121	74.4	70.5	66.5
MobileNet-v2	71.9	58.0	31.7

On présente les résultats de LatticeQ en retirant la correction de biais dans la table 3.7. Il est à noter que notre quantifieur par canal demeure proche de la précision des réseaux préentraînés Resnet et VGG malgré cette ablation. La correction de biais améliore la précision de notre méthode, qui reste supérieure aux méthodes existantes en son absence. On fournit les résultats de quantification par couche dans l'hypothèse zero-shot (table 3.8), pour comparaison avec les résultats par canal (table 3.7). La quantification par canal demeure nécessaire pour atteindre une précision proche du réseau préentraîné sur la plupart des architectures en l'absence de finetuning. Malgré tout, il est à noter qu'avec LatticeQ, quantifier par couche en 8-bit (poids et activations) est presque sans impact sur la performance des Resnets.

Table 3.8 – LatticeQ par couche

	Top-1 accuracy		
Network	FP32	W8A8	W4A8
Resnet-18	69.8	69.5	59.5
Resnet-50	76.0	75.9	70.2
VGG-16bn	73.4	73.3	68.5
Densenet	74.4	71.3	59.4
MobileNet-v2	71.9	70.6	12.9

3.5 . Analyse

3.5.1 . Erreur de quantification et structure du quantifieur

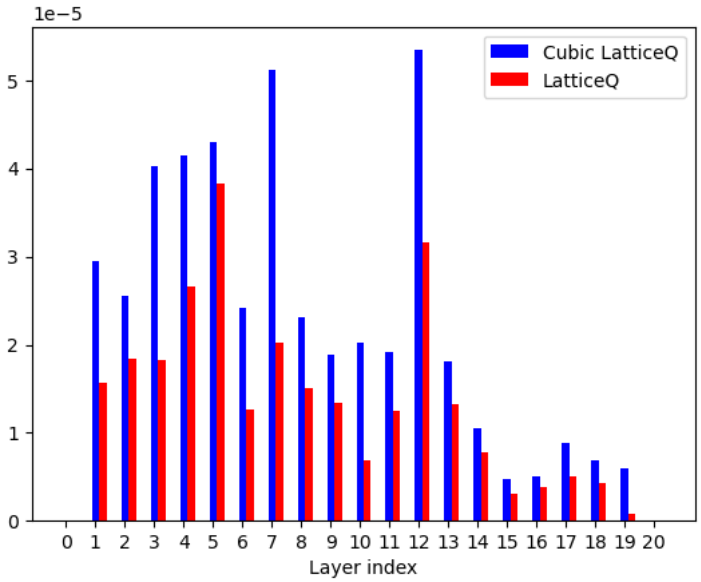

Figure 3.7 – Erreur de quantification par couche d'un Resnet18, comparaison entre LatticeQ et CubicLatticeQ (uniforme). Ordonnée : valeur de la MCE.

On démontre ici les avantages en termes d'erreur de quantification (Figure 3.7) et de performance fonctionnelle (Table 3.9) liés à l'utilisation d'un réseau euclidien déformable plutôt qu'un réseau cubique (= quantification uniforme) pour quantifier. Nos résultats confirment l'hypothèse initiale que les corrélations entre paramètres d'un réseau de neurones peuvent être exploitées dans un but de compression et d'accélération.

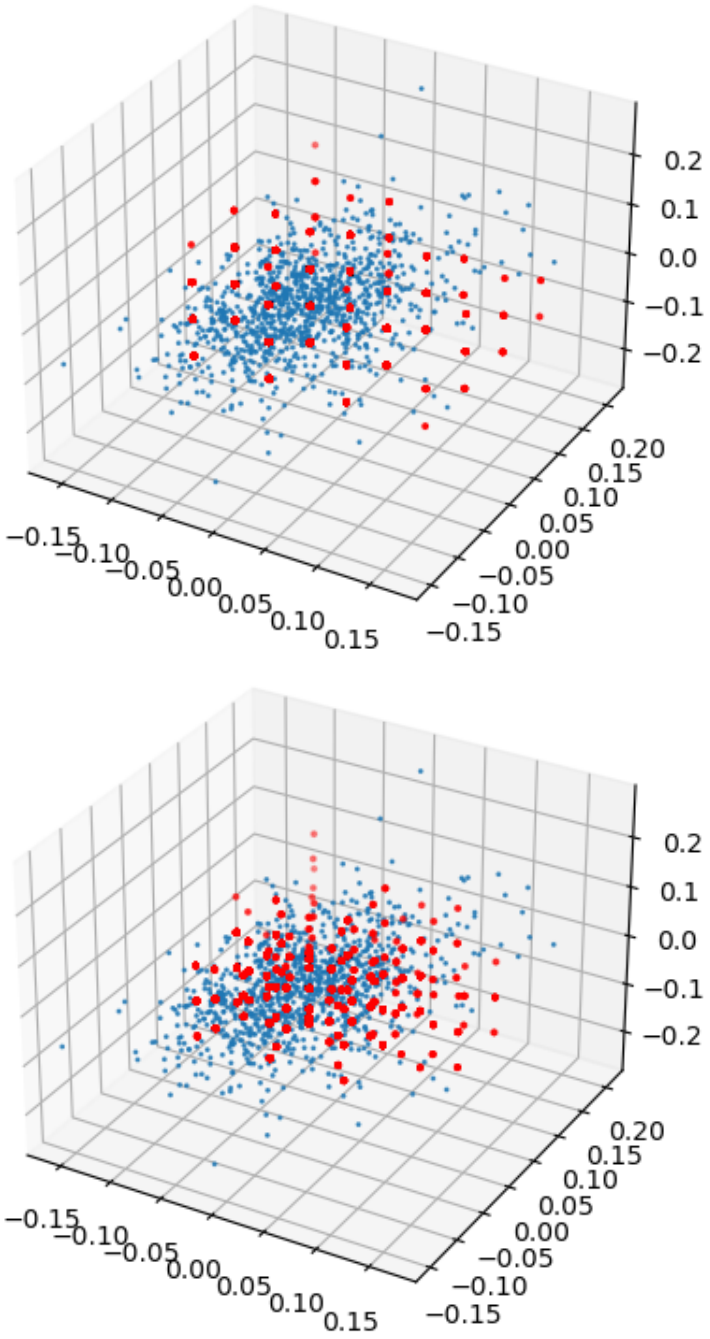

Figure 3.8 – Haut : Points de quantification de CubicLatticeQ (rouge) et blocs de poids (bleu), Bas : Points de quantification de LatticeQ (rouge) et blocs de poids (bleu). *On représente ici les points de quantification 2-bit pour une meilleure lisibilité.*. Axe i : valeurs poids i.

Sur la Figure 3.8, chaque bloc de poids est représenté par un point bleu, et chaque point de quantification par un point rouge. On constate que LatticeQ permet d'augmenter la concentration des points de quantification aux endroits critiques de la distribution. La structure de notre réseau de quantification s'adapte donc bel et bien à la distribution des filtres.

Table 3.9 – Comparison between baseline per-channel LatticeQ and baseline per-channel Cubic LatticeQ (scalar quantization).

Network	Method	FP32	W4A8
Resnet-18	LatticeQ	69.8	**67.2**
	Cubic LatticeQ	69.8	57.6

3.5.2 . Surcoût mémoire

Table 3.10 – Surcoût mémoire de LatticeQ avec prise en compte des bases de quantification. Le pourcentage de compression est donné en comparaison avec le modèle préentraîné FP32.

Type	Network	W4 mem.	Comp. %	Bits/weight
Per layer	Resnet-18	6.100 MB	86.94%	4.00
	Resnet-50	13.78 MB	86.49%	4.00
	Densenet-121	4.465 MB	85.86%	4.00
	MobileNet-v2	2.376 MB	82.88%	4.00
Per channel	Resnet-18	6.158 MB	86.82%	4.02
	Resnet-50	14.01 MB	86.26%	4.04
	Densenet-121	4.555 MB	85.57%	4.04
	MobileNet-v2	2.547 MB	81.65%	4.17

Enfin, on calcule le coût mémoire de LatticeQ par couche et par canal pour une quantification 4-bit (W4 mem.). Chaque fois qu'une base est utilisée pour quantifier un canal ou une couche, on stocke $32 + n^2.b$ bits, où n est la dimension de la base et b le nombre de bits utilisés pour coder les éléments de la base. 32 bits sont dus au facteur de scaling. Dans la Table 3.10, on répertorie l'impact en surcoût mémoire de LatticeQ sur certaines architectures.

On donne les ratios de compression pour les modèles entiers première et dernière couche comprises (Comp. %) et son équivalent exprimé en nombre de bits par poids (Bits/weight). On constate que le surcoût dû au stockage des bases est négligeable pour la quantification par couche, et représente moins d'1% du coût total pour la quantification par canal, sauf pour MobileNet (à cause des convolutions en profondeur). Par conséquent, à performance égale, la pénalité du surcoût mémoire dû aux bases peut être négligée devant les bénéfices octroyés par LatticeQ en termes de capacité de compression.

3.6 . Conclusion et perspectives

Dans cette partie, on a présenté LatticeQ, méthode de quantification post-training pour les poids des réseaux convolutionnels. LatticeQ exploite les réseaux euclidiens pour aboutir à un quantifieur flexible, capable de s'adapter à la distribution des poids. Ce travail a donné lieu à la publication d'un article à la conférence DATE 2023, ainsi qu'à une présentation au workshop "Machine Learning for Systems" de la conférence NeurIPS 2022. LatticeQ présente un intérêt particulier pour le déploiement de modèles préentraînés en précision flottante sur du matériel contraint en mémoire. Elle ne requiert aucun entraînement et aucune donnée, ce qui la rend facile d'utilisation. On a montré que LatticeQ surpasse l'état de l'art de la PTQ en basse précision sur plusieurs architectures de réseaux convolutionnels bien connues, avec moins d'1% de surcoût mémoire. Enfin, on rappelle que LatticeQ n'est qu'une généralisation de la quantification uniforme. N'importe quelle méthode de quantification reposant sur un quantifieur uniforme possède donc son équivalent "Lattice" : dans ce papier, on a notamment étudié les équivalents "Lattice" de Banner et al. [2018] et Li et al. [2021].

Du côté des perspectives, il est plausible que l'intérêt du quantifieur en réseau euclidien s'étende au-delà de l'hypothèse PTQ, étant donné que cette méthode démontre de bonnes performances y compris lorsque des données de finetuning sont disponibles. Enfin, LatticeQ pourrait inspirer la conception d'autres quantifieurs de structures différentes de celle des quantifieurs uniformes classiques.

Les résultats de la PTQ restent malgré tout limités en comparaison de la QAT, du fait des bénéfices du réentraînement, qui permet de compenser jusqu'à un certain point la perte en performance due à la quantification. Dans l'optique de quantifier plus fortement le réseau, on s'intéresse dans la partie suivante à la QAT, et en particulier à la quantification à des résolutions agressives, où les pertes en performance deviennent inévitables. On cherchera à comprendre la source de ces pertes et à les atténuer.

4 - Quantification agressive à l'apprentissage

Après avoir consacré la première partie de cette thèse à une méthode PTQ, on a souhaité explorer le Quantization-Aware Training (QAT) et en particulier les résolutions de quantification très agressives, inaccessibles en post-training. Courbariaux et al. [2015], Rastegari et al. [2016] sont les deux exemples fondateurs de la littérature sur la quantification binaire. Ces deux méthodes, ainsi que toutes les méthodes QAT non pensées expressément pour la quantification binaire mais pouvant être utilisées en précision binaire, présentent d'importantes dégradations dont on a cherché à comprendre l'origine. Le but de cette partie est de proposer une amélioration du pipeline classique de la QAT permettant d'atténuer les contraintes que les très basses résolutions font peser sur l'entraînement.

Contributions

- On étudie l'impact de la quantification agressive sur la qualité de l'apprentissage lors de l'entraînement QAT.
- On en déduit une modification du pipeline d'entraînement simple et applicable à tout quantifieur uniforme.
- On évalue notre méthode sur deux réseaux de l'état de l'art et deux méthodes de quantification uniforme (SAT et N2UQ).

4.1 . Principes de la QAT

Comme on l'a noté en introduction, les méthodes quantization-aware training (QAT) reposent sur l'estimateur "straight-through" (STE) pour l'entraînement du réseau. Cet estimateur permet de contourner la fonction de quantification de sorte à rétropropager un gradient non nul. Il existe plusieurs manières d'utiliser le STE pour faire la mise à jour des poids, qui sont répertoriées dans Li et al. [2017], et que l'on résume ci-après.

Arrondi déterministe

L'équation de mise à jour des paramètres des méthodes de type arrondi déterministe s'écrit sous la forme suivante :

$$w_{t+1} = Q_d(w_t - \alpha_t \nabla f(w_t)) \tag{4.1}$$

où Q_d est la fonction d'arrondi déterministe, α_t le taux d'apprentissage ($\in \mathbb{R}_+$), f la fonction de loss. Le problème de l'arrondi déterministe réside dans la

faible exploration de l'espace des solutions que celui-ci induit. En effet, au cours de l'apprentissage, un même paramètre w_t peut ne jamais changer de valeur. Afin de remédier à ce défaut, les deux algorithmes suivants sont privilégiés.

Arrondi stochastique

L'équation de mise à jour des paramètres des méthodes de type arrondi stochastique s'écrit sous la forme suivante :

$$w_{t+1} = Q_s(w_t - \alpha_t \nabla f(w_t)) \tag{4.2}$$

où Q_s est l'arrondi stochastique (arrondi aléatoire en fonction de la proximité des points de quantification). L'avantage de l'arrondi stochastique (et de l'arrondi déterministe), est que w_t peut être mis à jour sans avoir à conserver une copie des poids en précision flottante. Cela rend l'arrondi stochastique particulièrement intéressant pour l'apprentissage embarqué, lorsque les ressources disponibles pour l'entraînement sont limitées.

Binary Connect

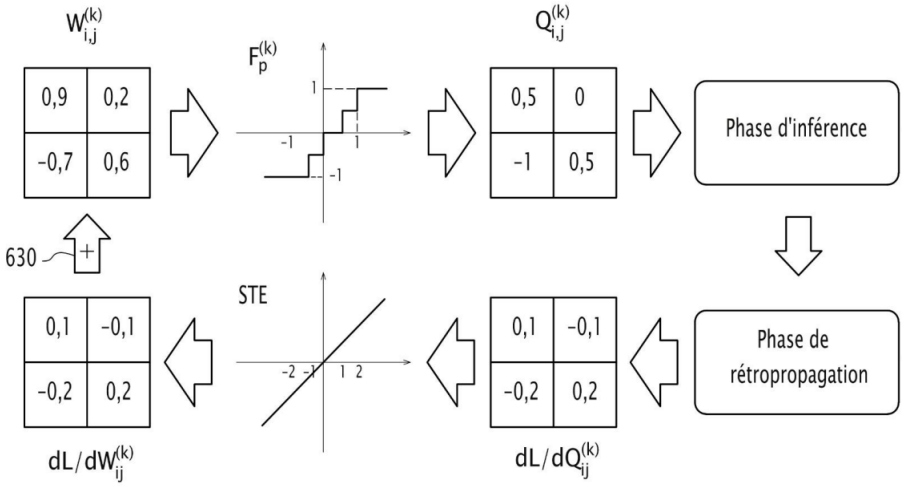

Figure 4.1 – Schéma fonctionnel de l'inférence, de la rétropropagation, et de la mise à jour des paramètres par Binary Connect

L'équation de mise à jour des paramètres des méthodes de type binary connect s'écrit sous la forme suivante :

$$w_{t+1} = w_t - \alpha_t \nabla f(Q(w_t)) \tag{4.3}$$

où Q est la fonction de quantification. La Figure 4.1 schématise la mise à jour des paramètres dans un pipeline d'entraînement de type BinaryConnect.

4.2 . Limites d'apprentissage spécifiques à la quantification

Comme mentionné en introduction (cf. 1.3.2), il existe une multitude de techniques pour améliorer à la marge la performance d'un réseau de neurones par la régularisation, en adaptant l'optimiseur [Loshchilov and Hutter, 2017] ou la loss [Müller et al., 2019]. Mais aucune d'entre elles n'est spécifique à la quantification.

Lorsque ses paramètres sont quantifiés, le réseau est soumis à une contrainte de résolution de ses paramètres qui réduit la quantité totale d'information disponible, bornant ainsi ses performances atteignables. En effet, le STE [Bengio et al., 2013] permet certes d'entraîner un réseau quantifié, mais il ne garantit pas une "bonne" convergence [Li et al., 2017] : les algorithmes d'arrondi stochastique (Equation 4.2) et de binary connect (Equation 4.3) rencontrent tous deux un "plancher de précision", qui les empêche en théorie d'atteindre les performances de l'entraînement non quantifié. Un tel constat est en cohérence avec l'intuition, car l'espace des solutions du problème d'entraînement QAT est strictement inclus dans l'espace des solutions du problème d'entraînement général.

Mais le réseau est également soumis à une contrainte spécifique à l'entraînement QAT lorsque les activations sont quantifiées, que l'on cherchera à mettre en évidence dans le paragraphe suivant. Notre méthode visera à alléger cette contrainte pour permettre au réseau d'apprendre plus aisément en basse précision de quantification.

4.3 . Impact de la quantification des activations sur l'apprentissage

L'objectif de ce paragraphe est de montrer comment la quantification des activations impacte la convergence des paramètres du réseau. Nous cherchons en particulier à évaluer l'impact de la quantification des activations sur la précision des gradients. En reprenant l'équation 4.3, on interprète l'erreur de quantification comme un bruit ϵ_t sur les activations x en entrée de la couche. Naturellement, le bruit de quantification augmente lorsque les niveaux de quantification se raréfient, autrement dit, lorsque la quantification devient plus agressive. On écrit l'équation 4.4 de mise à jour des paramètres :

$$w_{t+1} = w_t - \alpha_t \nabla f_{\hat{w}_t}(x + \epsilon_t) \tag{4.4}$$

où w_t sont les poids à l'étape t de l'entraînement, α_t le taux d'apprentissage, ϵ_t le bruit de quantification sur les activations et x les activations en entrée de la couche pendant la passe forward. Il est commun de considérer que $\nabla f_{\hat{w}_t}$ possède un certain degré de régularité, notamment qu'elle est L-lipschitzienne ("deux entrées proches d'une couche donnent sensiblement la même sortie"). Cela permet d'écrire :

$$||\nabla f_{\hat{w}_t}(x + \epsilon_t) - \nabla f_{\hat{w}_t}(x)|| \leq L||\epsilon_t|| \tag{4.5}$$

On obtient l'encadrement :

$$w_t - \alpha_t(\nabla f_{\hat{w}_t}(x) + L||\epsilon_t||) \leq w_{t+1} \leq w_t - \alpha_t(\nabla f_{\hat{w}_t}(x) - L||\epsilon_t||) \tag{4.6}$$

Autrement dit, les perturbations dues à la quantification sur la descente de gradient sont bornées par $\alpha_t L||\epsilon_t||$, le produit du taux d'apprentissage α_t, de la mesure de régularité du gradient de la loss L, et de l'amplitude du bruit de quantification $||\epsilon_t||$. Ces perturbations sont nécessaires et permettent l'apprentissage QAT, mais si leurs amplitudes augmentent dans de trop grandes proportions par rapport à $\nabla f_{\hat{w}_t}(x)$, la qualité de la descente de gradient risque d'être affectée du fait de l'instabilité induite. En conséquence, l'entraînement peut être ralenti, voire ne pas démarrer.

4.4 . Méthode proposée

En vertu de l'encadrement 4.6, le bruit de quantification à chaque rétropropagation est proportionnel au taux d'apprentissage α_t. Or, ce taux d'apprentissage décroît au fur et à mesure de l'entraînement, de sorte à explorer l'espace des paramètres de plus en plus finement. Par conséquent, lors de cette phrase d'exploration plus fine durant laquelle les paramètres du réseau se stabilisent et "varient moins vite", le bruit de quantification perd en amplitude relative par rapport aux paramètres. On fait l'hypothèse qu'il est possible de tirer profit de cette stabilité pour introduire de plus en plus de perturbations liées à la quantification à mesure que l'entraînement avance, et que cela limiterait les dommages sur la convergence. Par la suite, on appellera cette hypothèse "hypothèse de stabilisation progressive".

Afin de limiter l'impact du bruit de quantification sur l'apprentissage, on propose donc une nouvelle approche qui consiste à débuter l'entraînement en utilisant une quantification moins agressive des activations que la précision visée, puis à faire décroître la précision au fur et à mesure des époques.

À la fin de l'entraînement, la précision des activations est fixée à l'objectif, comme pour un apprentissage QAT classique. Notre approche s'apparente à celle de Zhuang et al. [2017], mais on y apporte une meilleure progressivité de sorte à rendre le processus utile pour les très basses précisions.

4.4.1 . Précision étendue

On commence par présenter une extension du domaine des précisions de quantification, appelée précision étendue. En temps normal, on quantifie les paramètres et les activations sur un nombre de bits b, qui correspond à 2^b points de quantification. Cela permet d'établir une relation bijective entre le domaine quantifié (en nombre de bits) et la valeur effective du point de quantification. Mais rien n'oblige en pratique à établir cette bijection. Durant l'entraînement, il est parfaitement envisageable de restreindre le nombre de points de quantification à une valeur quelconque qui ne soit pas nécessairement une puissance de 2.

Ci-dessous, l'équation 4.7 donne l'ensemble des points de quantification d'une méthode uniforme sur le domaine $[0, 1]$ avec précision étendue à n points de quantification :

$$Q = \{\frac{k}{n-1}, 0 \leq k \leq n-1\} \tag{4.7}$$

Un exemple très simple de quantifieur à n points de quantification sur [0,1] est donné par l'équation 4.8 :

$$\text{Quantize}_n(x) = \frac{\lfloor (n-1) \times x \rceil}{n-1} \tag{4.8}$$

La précision étendue permet donc une plus grande progressivité, donnant par exemple accès à la précision ternaire. Dans l'optique de faire baisser cette précision au cours de l'apprentissage, et en considérant notre hypothèse de stabilisation progressive, disposer d'un plus grand nombre de paliers de précision intermédiaires devrait améliorer la performance finale du réseau.

4.4.2 . Mode de décroissance

On fait le choix d'une décroissance exponentielle du nombre de points de quantification, car ce type de décroissance permet à l'entraînement de s'attarder plus longtemps sur des précisions proches de la précision finale qu'une décroissance linéaire. À chaque époque, on calcule le nombre de points comme suit (Equation 4.9) :

$$n_k = \lfloor max(C^k.2^{b_0}, 2^{b_f}) \rceil \tag{4.9}$$

où n est le nombre de points de quantification à l'époque k, b_f est la précision finale recherchée, b_0 est la précision initiale (typiquement $b_0 = b_f + 2$), et

C est une constante de décroissance inférieure ou égale à 1 (typiquement $C = 0.97$).

4.4.3 . Progressivité étagée

Une optimisation supplémentaire de l'approche consiste à étager la décroissance de la précision en fonction de la profondeur des couches du réseau. En effet, le bruit de quantification des couches les plus profondes se répercute sur les couches les moins profondes au cours de la rétropropagation. Il est donc pertinent de réduire dans un premier temps la précision des couches les moins profondes, de sorte à préserver au mieux la qualité des gradients issus des couches plus profondes. On résume les changements apportés au processus d'entraînement dans l'algorithme 5.

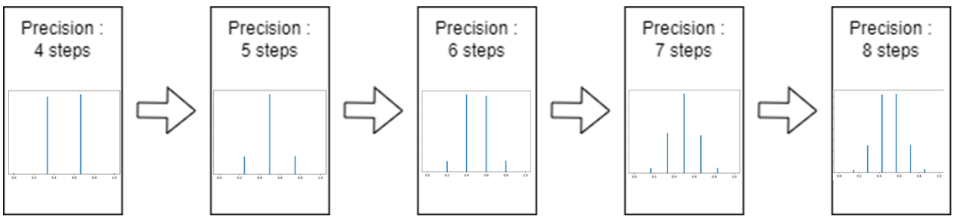

Figure 4.2 – Exemple de répartition étagée des précisions parmi les blocs d'un réseau

Algorithm 5 Progressive quantization

function Train(Network $\mathcal{N}$, Decay constant C, Large enough number of epochs ep, Sequence of Network Blocks $(\mathcal{B})$, Initial precision b_0, Objective precision b_f, Stage progressiveness s)
  for epoch $\in \{1, ..., ep\}$ **do**
    for block $\in (\mathcal{B})$ **do**
      block.precision $\leftarrow \lfloor max(C^{max(0, epoch - s \times \text{block.index})}.2^{b_0}, 2^{b_f}) \rceil$
    end for
    Run network training epoch
  end for
end function

4.4.4 . Application à SAT

SAT [Jin et al., 2019] (voir 2.2.2) est une méthode de quantification uniforme. En premier lieu, on a souhaité mettre notre approche à l'épreuve sur cette méthode très simple et performante. Le quantifieur des activations de SAT est inspiré de Choi et al. [2018] (Equation 4.10). L'adaptation de SAT à notre approche est triviale et ne nécessite que de redéfinir la fonction de quantification (Quantize) selon l'équation 4.8.

$$x^q = \alpha \times \text{Quantize}(\frac{\text{clamp}_\alpha(x)}{\alpha}) \tag{4.10}$$

Table 4.1 – Impact de la quantification progressive des activations avec SAT sur des niveaux de quantification agressifs pour ImageNet.

		Top-1 accuracy		
Network	Method	W3A3	W2A2	W1A1
Resnet-18 (69.8)	Progressive SAT		65.5	**57.6**
	SAT		**65.6**	57.1
Mobilenet-v2 (71.9)	Progressive SAT	**62.2**	**47.1**	
	SAT	53.9	–	

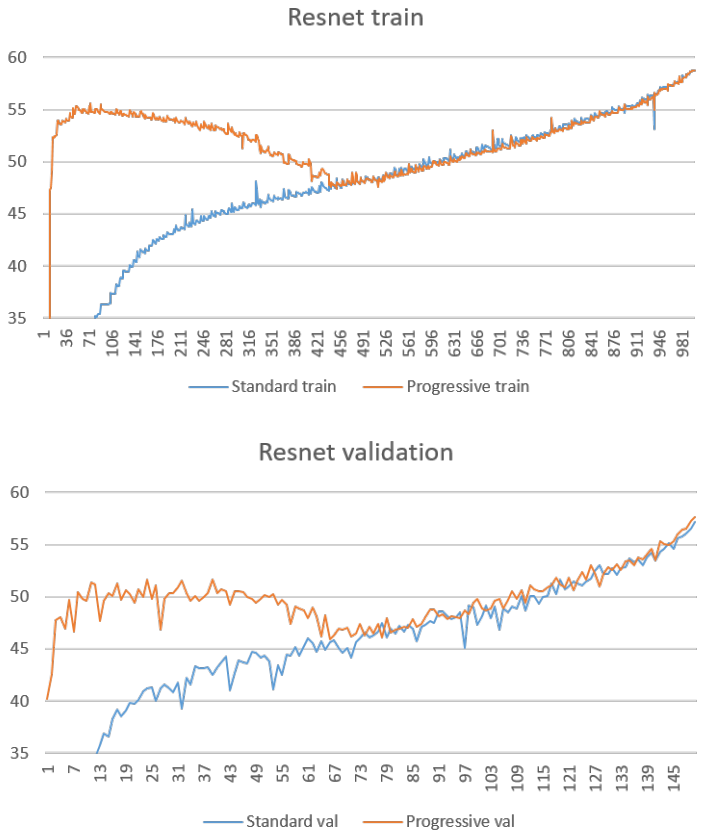

Figure 4.3 – Resnet18 (W1A1), précision Top-1 d'entraînement et de validation, progressive en rouge et standard en bleu.

La Table 4.1 fait l'inventaire des expériences menées avec la quantification progressive (Progressive SAT) sur Resnet-18 et Mobilenet-v2, entraînés sur ImageNet. On compare notre approche avec l'entraînement SAT standard sans modification du pipeline d'apprentissage. On constate un léger progrès en binaire sur Resnet-18, et un progrès très substantiel sur Mobilenet-v2 avec plus de 8% gagnés en 3-bit. On note également que l'entraînement 2-bit de Mobilenet-v2 diverge avec l'optimiseur SGD (Stochastic Gradient Descent), tandis qu'il converge avec notre approche.

Sur la Figure 4.3 (haut), on représente la précision d'entraînement (Top-1 training accuracy) de notre Resnet18 binaire sur ImageNet avec et sans progressivité de la précision. On constate sans surprise un bien meilleur démarrage de l'entraînement progressif (au cours des premières époques, les activations sont quantifiées en 3-bit). Les deux courbes finissent par se rejoindre lorsque la précision de l'entraînement progressif atteint sa valeur finale, mais le réseau entraîné progressivement bénéficie d'une meilleure capacité de généralisation, ce qui lui permet d'atteindre une précision de validation légèrement supérieure (voir Figure 4.3, bas).

4.4.5 . Application à N2UQ

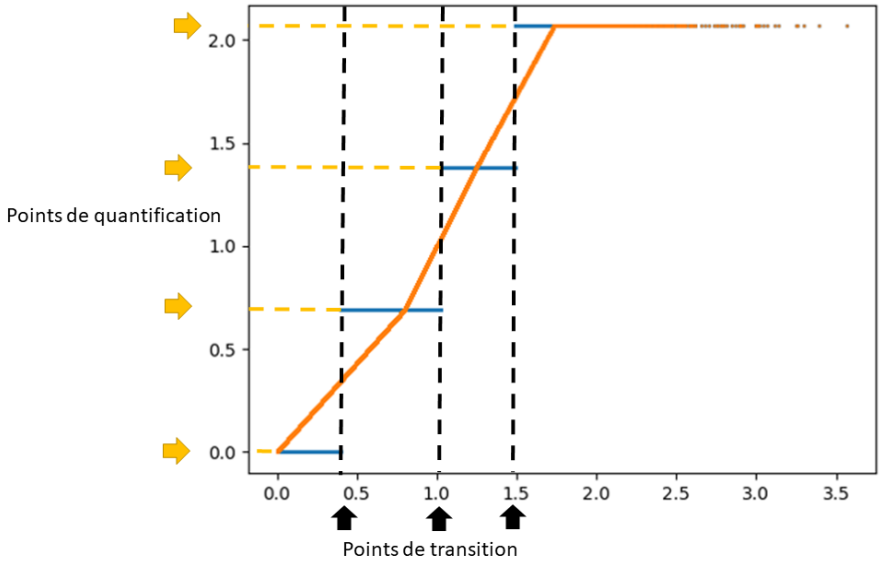

Figure 4.4 – Fonction de quantification N2UQ (bleu) et G-STE (orange). On pointe en ordonnée les points de quantification (régulièrement espacés), et en abscisse les points dits "de transition". Enfin, la largeur horizontale des segments de pente du G-STE (branches) sont les α_i appris.

La quantification des activations est le facteur limitant principal de la performance des réseaux en basse précision. N2UQ [Liu et al., 2022a](voir 2.2.2) présente une méthode originale, avec une fonction de quantification des activations plus souple et plus générale que SAT, tout en satisfaisant la contrainte d'uniformité (niveaux de quantification régulièrement espacés) importante pour la simplicité d'implémentation. On a cherché à savoir si notre approche pouvait présenter un intérêt sur ce type particulier de fonctions d'activation "non-uniform to uniform", aux résultats prometteurs. L'objet de ce paragraphe est d'adapter cette fonction de quantification pour la rendre compatible avec notre approche.

La fonction de quantification des activations de N2UQ est constante par morceaux (voir Figure 4.4), et caractérisée par des tailles de paliers distinctes. Ces tailles de paliers sont déterminées par des paramètres $(\alpha_i)_{1\leq i\leq n}$ apprenables. On commence par présenter la fonction utilisée pour les activations de N2UQ. L'algorithme 6 est utilisé pour calculer cette fonction.

On présente dans l'algorithme 7 (voir Annexe 4.6) une adaptation de l'algorithme 6 pour permettre d'ajuster le nombre de niveaux de quantification au fur et à mesure de l'apprentissage tout en maintenant le même estimateur Generalized-STE pour la rétropropagation (voir Liu et al. [2022a] pour la définition du G-STE). L'intérêt de l'algorithme 7 est de pouvoir conserver les mêmes paramètres α_i tout au long de l'apprentissage tout en adaptant le forward à un nombre de points de quantification N plus élevé que la précision finale n. Cet algorithme calcule le point de transition suivant en parcourant les branches de la fonction par morceaux du G-STE. On représente sur la Figure 4.5 une fonction N2UQ ainsi adaptée. Le nombre de branches reste le même, seule la précision du forward change.

Algorithm 6 N2UQ activation function

function N2UQActivationFunction(Input x, Number of steps n, Lower domain limit start, Quantization step s)
    threshold $\leftarrow$ start $+ \frac{\alpha_1}{2}$
    if $x <$ threshold **then**
        return 0
    end if
    for $1 \leq i \leq n-1$ **do**
        threshold $\leftarrow$ threshold $+ \frac{\alpha_i}{2} + \frac{\alpha_i+1}{2}$
        if $x <$ threshold **then**
            return $i \times s$
        end if
    end for
    return $n \times s$
end function

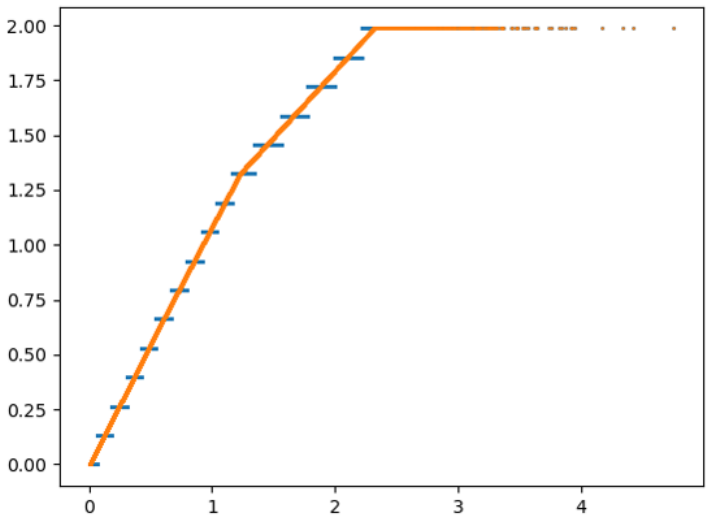

Figure 4.5 – Fonction de quantification N2UQ (bleu) adaptée à notre approche d'apprentissage progressif, et G-STE (orange).

Dans la Table 4.2, on reporte les résultats de quelques entraînements sur Resnet-18 et Mobilenet-v2 à basses précisions sur ImageNet. La comparaison des résultats entre notre approche et l'approche standard confirme le bien fondé de la quantification progressive pour les activations N2UQ. On remarque que cette approche est d'autant plus utile que la quantification est agressive : si le gain est marginal sur Resnet-18 en 2-bit, il est beaucoup plus important en binaire et sur Mobilenet-v2, architecture réputée moins robuste à la quantification.

Table 4.2 – Impact de la quantification progressive des activations avec N2UQ sur des niveaux de quantification agressifs pour ImageNet.

		Top-1 accuracy	
Network	Method	W2A2	W1A1
Resnet-18 (69.8)	Progressive N2UQ	**67.5**	**58.4**
	N2UQ	67.3	56.6
Mobilenet-v2 (71.9)	Progressive N2UQ	**55.1**	
	N2UQ	53.4	

4.5 . Conclusion et perspectives

Dans cette seconde partie, on a présenté un moyen d'améliorer l'entraînement QAT des réseaux de neurones fortement quantifiés par l'introduction de trois techniques : la quantification en précision étendue, la progressivité de l'apprentissage quantifié et l'étagement de cette progressivité à l'intérieur du réseau. Ce travail a donné lieu à un dépôt de brevet. L'entraînement des réseaux quantifiés en très basse précision demeure un défi pour l'état de l'art actuel, les dégradations dues à la perte d'information rendant sa convergence difficile. On a néanmoins contribué à rapprocher légèrement cet objectif en adaptant le pipeline d'entraînement aux conditions particulières qu'implique la quantification agressive, aboutissant à une amélioration sensible de l'ordre de 1 à 2% de la méthode N2UQ [Liu et al., 2022a] en 2-bit et en binaire sur les architectures Resnet et Mobilenet-v2, et jusqu'à 8% en 3-bit avec la méthode SAT sur Mobilenet-v2.

Dans cette partie, on a démontré l'impact du bruit de quantification des activations sur l'apprentissage. Les perspectives de ce travail incluent par conséquent la recherche de nouveaux moyens d'atténuer les contraintes que fait peser ce bruit de quantification sur la vitesse de convergence.

Mais la diminution de la résolution de quantification n'est pas l'unique moyen de compresser un réseau (voir 2.1). Combiner différentes approches (quantification et pruning par exemple) peut mener à des modèles performants et facilement embarquables. La partie suivante explore une de ces approches hybrides entre quantification, sparsification et codage de source pour atteindre des taux de compression compétitifs, tout en étant compatible avec l'inférence embarquée.

4.6 . Annexe : adaptation de l'algorithme 6

Algorithm 7 Adapted N2UQ activation function

function AdaptedN2UQActivationFunction(x, Final number of steps n, Current number of steps $N > n$, start, s)
    threshold $\leftarrow$ start $+ \frac{\alpha_1}{2} \times \frac{n}{N}$
    if $x <$ threshold **then**
        return 0
    end if
    current_following_qpoint $\leftarrow$ start
    $j \leftarrow 0$
    for $i \leftarrow 1$ to $n - 1$ **do**
        current_following_qpoint $\leftarrow$ current_following_qpoint $+ \alpha_i$
        while threshold $+ \alpha_i \times \frac{n}{N} \leq$ current_following_qpoint **do** ▷ Calcul du prochain point de transition tant qu'il reste sur la même branche
            $j \leftarrow j + 1$
            threshold $\leftarrow$ threshold $+ \alpha_i \times \frac{n}{N}$
            if $x <$ threshold **then**
                return $j \times \frac{s.n}{N}$
            end if
        end while
        if threshold $+ \alpha_i \times \frac{n}{N} >$ current_following_qpoint **then** ▷ Même calcul quand le point de transition change de branche
            $j \leftarrow j + 1$
            threshold $\leftarrow$ current_following_qpoint $+ \frac{\alpha_{i+1}}{s} \times (\frac{s.n}{N} -$ (current_following_qpoint $-$ threshold) $\times \frac{s}{\alpha_i})$
            if $x <$ threshold **then**
                return $j \times \frac{s.n}{N}$
            end if
        end if
    end for
    return $N \times \frac{s.n}{N}$
end function

5 - Compression ANS-neuronale

La dernière partie de cette thèse explore la passerelle entre la quantification des réseaux de neurones (en particulier la quantification en précision mixte) et le codage entropique. La taille mémoire des paramètres d'un réseau de neurones est un critère déterminant pour son embarquabilité, étant donné qu'elle détermine en partie la taille de la puce nécessaire à leur stockage, mais également l'énergie nécessaire au transfert des données de la mémoire aux unités de calcul. La taille mémoire des paramètres d'un réseau peut certes être réduite par la quantification, mais il est possible d'aller encore plus loin sans pertes en utilisant le codage de source (voir 2.1.1).

Tartaglione et al. [2021] est un exemple de méthode utilisant une régularisation basée sur l'entropie de sorte à rendre un réseau de neurones plus facilement compressible, mais ne s'intéresse pas au codage lui-même, se focalisant plutôt sur la réduction d'entropie pendant l'entraînement. Oktay et al. [2020] est un autre exemple d'utilisation du codage entropique pour la compression, mais son approche de décomposition fréquentielle par transformée de Fourier discrète n'est pas compatible avec l'acccélération du fait de sa complexité algorithmique ($\mathcal{O}(n \log_2(n))$, avec n la longueur du vecteur). À l'opposé, l'ambition de cette partie est d'ouvrir la voie au codage à l'inférence, c'est-à-dire de rendre possible le stockage "on edge" des paramètres encodés pour ne les décoder qu'au moment de les utiliser pour le calcul.

Contributions

- On met en évidence l'intérêt d'un quantifieur avec un point o pour la compression.
- On étudie un codage entropique particulier, le codage ANS (Asymetric Numeral Systems), et on démontre ses avantages dans le contexte embarqué.
- On adapte une méthode de quantification en précision mixte existante [Yang and Jin, 2020] à un objectif entropique, et on met en oeuvre la quantification avec point o et le codage ANS pour évaluer la compressibilité des réseaux ainsi obtenus.

5.1 . Adaptation de Fracbits

On choisit une méthode de quantification en précision mixte du fait de la supériorité de cette approche par rapport à la précision fixe en termes de coût de stockage mémoire. En effet, un réseau quantifié en précision mixte peut atteindre un taux de compression plus important qu'en précision fixe à dégradation de performance égale [Yang and Jin, 2020, Habi et al., 2020, Yang et al., 2021, Zhang et al., 2021], même si un tel apport n'est pas sans conséquences sur l'implémentation matérielle.

5.1.1 . Détail de la méthode

Fracbits (voir 2.2.3) permet d'intégrer la sélection des précisions de quantification des couches ou des canaux au processus d'apprentissage. On veut explorer la compatibilité de cette quantification en précision mixte avec une forme d'optimisation entropique de la taille du réseau. En effet, réduire la précision des poids d'une couche du réseau a un impact significatif sur leur entropie, et on souhaiterait que ce soit l'entropie (et non la taille du réseau) qui dicte l'évolution de ces précisions.

Le principe de l'apprentissage de Fracbits est le même quel que soit son mode de fonctionnement (poids uniquement ou bien poids et activation). Pour apprendre les précisions de quantification en même temps que les paramètres du réseau, on a recours à l'optimisation lagrangienne, c'est-à-dire que l'on ajoute un terme de contrainte à la loss (Equation 5.1) :

$$\mathcal{L}^{tot} = \mathcal{L}^{cls} + \kappa.\mathcal{L}^{constraint} \tag{5.1}$$

où $\mathcal{L}^{cls}$ est la loss de classification (typiquement l'entropie croisée), $\mathcal{L}^{constraint}$ est le terme de loss supplémentaire qui modélise la contrainte, et κ est un hyperparamètre qui garantit l'équilibre entre les deux.

Fracbits a deux modes de fonctionnement, qui correspondent chacun à une contrainte particulière. Le premier est le mode "poids uniquement", où la contrainte est la taille du réseau (Equation 5.2) :

$$\mathcal{L}^{size} = \sum_{w} \lambda_w \tag{5.2}$$

où λ_w est le paramètre de précision courant en bits (flottant) du poids w.

Le second mode est le mode "poids et activations", où la contrainte est l'intensité calculatoire (Equation 5.3) :

$$\mathcal{L}^{compute} = \sum_{wa} \lambda_w \lambda_a \tag{5.3}$$

où λ_a (flottant) est le paramètre de précision courant en bits de l'activation a. Autrement dit, pour chaque canal d'entrée et chaque filtre, on somme les

produits des précisions courantes des poids et des activations.

À l'issue d'un certain nombre paramétrable d'époques (fixé par défaut à 80% du total de l'entraînement), on procède à l'arrondi des paramètres de précision λ_w et λ_a. Afin que l'arrondi ne fasse pas dépasser l'objectif de compression, on effectue une recherche dichotomique d'un terme d'ajustement "offset". Les précisions finales sont fixées aux valeurs $\lambda_w - \text{offset}$ et $\lambda_a - \text{offset}$.

5.1.2 . Améliorations

Dans ce paragraphe, on présente deux améliorations apportées à Fracbits, qui se traduisent chacune par un gain en performance de la méthode.

Clipping des poids

Le quantifieur utilisé par Fracbits est celui de SAT, inspiré de DoReFa [Zhou et al., 2018] pour les poids (Equation 5.4) et de PACT [Choi et al., 2018] pour les activations (Equation 4.10) :

$$w_{i,j}^q = \text{Quantize}(\frac{1}{2} \times (\frac{\tanh(w_{i,j})}{\max_{r,s}|\tanh(w_{r,s})|} + 1)) \tag{5.4}$$

La première amélioration porte sur le mode de clipping de l'équation 5.4. L'objectif d'un tel clipping est de limiter la dispersion des paramètres grâce au comportement asymptotique de la fonction tangente hyperbolique. Dans le contexte de la quantification, il peut sembler intéressant à première vue de limiter cette dispersion, de sorte à réduire la dynamique des paramètres et l'erreur de quantification. Or, nous cherchons à diminuer l'entropie des paramètres dans le but de mieux compresser le réseau grâce au codage entropique. On a donc supprimé ce clipping pour étudier son impact sur la qualité de l'entraînement. En pratique, on trouve que le clipping en tangente hyperbolique n'apporte rien, et est même néfaste pour la performance du quantifieur (voir Table 5.1). On choisit donc de le remplacer par un clipping naïf (Equation 5.5). Ce résultat n'est pas l'objet principal de cette partie, mais chercher à l'expliquer pourrait en soi faire l'objet de nouveaux travaux.

$$w_{i,j}^q = \text{Quantize}(\frac{1}{2} \times (\frac{w_{i,j}}{\max_{r,s}|w_{r,s}|} + 1)) \tag{5.5}$$

Table 5.1 – Étude du clipping DoReFa.

Network	Method	**Top-1 accuracy** W2A32 [1]
Mobilenet-v2 (71.9)	Fracbits Naive clip	**70.3**
	Fracbits Tanh clip	69.3

Progressivité de l'hyperparamètre κ

Dans l'équation 5.1, l'hyperparamètre κ sert à garantir l'équilibre entre la loss cross-entropique, optimisant les performances en classification, et le terme de contrainte visant à compresser le réseau. Dans Fracbits, κ est laissé fixe à une valeur élevée tout au long de l'entraînement, ce qui cause une convergence très rapide des paramètres de précision. Inspiré par les résultats de la partie 4 qui démontrent le bénéfice de la quantification progressive, on choisit d'ajuster ce paramètre progressivement tout au long des 20 premières époques à l'aide d'une croissance linéaire jusqu'à une valeur seuil, afin de réduire petit à petit la précision du réseau. Le résultat expérimental de la Table 5.2 est une nouvelle preuve du bien-fondé de l'approche progressive.

Table 5.2 – Étude de la progressivité de κ.

Network	Method	**Top-1 accuracy** W2A32
Mobilenet-v2 (71.9)	Fracbits Naive clip progressive κ	**71.0**
	Fracbits Naive clip constant κ	70.3

5.1.3 . Objectif entropique

En plus des deux modes de fonctionnement décrits dans le paragraphe 5.1.1, on en propose un troisième dont l'objectif est la réduction de l'entropie des poids du réseau (Equation 5.6). Le but de ce mode de fonctionnement est de permettre à l'expérimentateur de définir une certaine taille de réseau souhaitée post-codage entropique, contrairement au mode de fonctionnement

1. Précision mixte par couche, première et dernière couche en 8-bit, reste des couches en 2-bit par poids en moyenne.

"poids uniquement" (Equation 5.2), qui ne définit qu'un objectif pré-codage entropique.

$$\mathcal{L}^{entropy} = \frac{|\sum_W |W|((1-\lambda_W+\lfloor\lambda_W\rfloor)H_{W(\lfloor\lambda_W\rfloor)} + (\lambda_W - \lfloor\lambda_W\rfloor)H_{W(\lceil\lambda_W\rceil)}) - \hat{H}|}{\hat{H}} \tag{5.6}$$

où les W sont les groupes de paramètres du réseau (couches ou canaux), λ_W les paramètres de précision de ces groupes, $H_{W(k)}$ l'entropie des poids quantifiés à la précision k, et $\hat{H}$ est l'objectif d'entropie totale à atteindre.

On peut ainsi calculer le gradient de $\mathcal{L}^{entropy}$ selon les λ :

$$\frac{\partial \mathcal{L}^{entropy}}{\partial \lambda_W} = \frac{|W|}{\hat{H}}(H_{W(\lceil\lambda_W\rceil)} - H_{W(\lfloor\lambda_W\rfloor)}) \tag{5.7}$$

5.2 . Optimisation entropique

Dans cette partie, on étudie l'impact d'une diminution de l'entropie des poids du réseau sur sa performance en fonction de sa précision de quantification. On met également en évidence l'intérêt d'un point zéro pour la compressibilité des poids quantifiés par l'utilisation d'un nombre impair de points de quantification.

5.2.1 . Entropie, précision et performance du réseau

On cherche à appliquer un codage entropique aux poids d'un réseau quantifié. On dispose ainsi de deux leviers de compression : le premier est la précision de quantification, et le second est le codage entropique lui-même. Pour augmenter le taux de compression, on peut donc agir sur ces deux leviers, c'est-à-dire opter pour une quantification plus agressive ou bien pousser le réseau à être moins entropique. Sachant que l'on cherche à minimiser la taille d'un réseau compressé, on voudrait savoir laquelle de ces deux stratégies est la meilleure au sens de la préservation de la performance du réseau sur sa tâche.

On expérimente sur SAT dans un premier temps, afin de pouvoir fixer une précision de quantification commune à tout le réseau, sauf pour la première et la dernière couche qui demeurent en 8-bit (selon la pratique commune de l'état de l'art). L'expression de l'objectif entropique (Equation 5.6) devient simplement :

$$\mathcal{L}^{entropy} = \frac{|\sum_W |W| \times H_W - \hat{H}|}{\hat{H}} \tag{5.8}$$

Estimateur différentiable pour l'entropie binaire d'une loi normale

L'entropie discrète H_W (Equation 2.1) n'est pas dérivable selon les paramètres de poids W. Pour calculer le gradient des paramètres du réseau selon notre objectif entropique, on doit donc passer par un proxy différentiable.

L'entropie différentielle étend la définition d'entropie de Shannon aux lois de probabilité continues. L'entropie binaire différentielle d'une variable aléatoire réelle X de densité f s'écrit :

$$H(X) = -\int_{t\in\Omega(X)} f(t)\log_2(f(t))dt \tag{5.9}$$

où $\Omega(X)$ est le support de X. Et si X suit une loi normale :

$$H(X) = \log_2(\sigma(X)\sqrt{2\pi e}) \tag{5.10}$$

Ou sous sa forme variance :

$$H(X) = \frac{1}{2}\log_2(V(X)2\pi e) \tag{5.11}$$

L'estimateur $\bar{\sigma}(X) = \sqrt{\bar{x^2} - \bar{x}^2}$ de l'écart-type est différentiable par rapport aux tirages x. On en déduit un estimateur différentiable de l'entropie binaire différentielle pour la loi normale : $\bar{H}(X) = log_2(\bar{\sigma}(X)\sqrt{2\pi e})$. À supposer que les poids du réseau suivent une loi normale (ce qui est une bonne approximation en pratique), on a donc là une approximation différentiable de leur entropie. Cependant, dans le cadre de la QAT, les poids sont quantifiés. Dans le paragraphe suivant, on étudie l'impact de la fonction arrondi $\lfloor X \rceil$ sur l'entropie d'une variable aléatoire continue à support fini.

Impact de l'arrondi sur l'entropie d'une loi continue à support fini

Soit $f(t)$ la densité d'une variable aléatoire X symétrique d'espérance nulle à support fini $[-N, N]$, avec $N \in \mathbb{N}$. Son entropie binaire s'écrit :

$$H(X) = -\int_{-N}^{N} f(t)\log_2(f(t))dt \tag{5.12}$$

Par la relation de Chasles, on peut écrire :

$$H(X) = -\sum_{i=-N}^{N}\int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t)\log_2(f(t))dt \tag{5.13}$$

L'entropie de $\lfloor X \rceil$ s'écrit quant à elle :

$$H(\lfloor X \rceil) = -\sum_{i=-N}^{N} p(\lfloor X \rceil = i)\log_2(p(\lfloor X \rceil = i)) \tag{5.14}$$

Ce qui revient à :

$$H(\lfloor X \rceil) = -\sum_{i=-N}^{N} p(i-\frac{1}{2} < X \leq i+\frac{1}{2}) \log_2(p(i-\frac{1}{2} < X \leq i+\frac{1}{2})) \quad (5.15)$$

$$H(\lfloor X \rceil) = -\sum_{i=-N}^{N} \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t)dt \times \log_2(\int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(u)du) \quad (5.16)$$

On remarque que $\lfloor X \rceil$ a la même entropie qu'une variable aléatoire continue $\tilde{X}$ de densité $g(t) = \int_{\lfloor t \rceil - \frac{1}{2}}^{\lfloor t \rceil + \frac{1}{2}} f(u)du$ [1] . On écrira indifféremment par la suite $H(\lfloor X \rceil)$ ou $H(\tilde{X})$.

On calcule l'écart à l'aide de 5.13 et de 5.16 :

$$H(\tilde{X}) - H(X) = \sum_{i=-N}^{N} \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t) \log_2(\frac{f(t)}{\int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(u)du})dt \quad (5.17)$$

$$H(\tilde{X}) - H(X) = \int_{-N}^{N} f(t) \log_2(\frac{f(t)}{\int_{\lfloor t \rceil - \frac{1}{2}}^{\lfloor t \rceil + \frac{1}{2}} f(u)du})dt \quad (5.18)$$

On identifie une divergence de Kullback-Leibler.

$$H(\tilde{X}) - H(X) = D_{\mathsf{KL}}(\tilde{X}||X) \quad (5.19)$$

On cherche à borner la différence $H(\tilde{X}) - H(X)$. Par le théorème des valeurs intermédiaires, on sait que f prend au moins une fois la valeur $\int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(u)du$ sur l'intervalle $[i-\frac{1}{2}, i+\frac{1}{2}]$ (sans quoi f est toujours strictement supérieure/inférieure à sa moyenne sur cet intervalle, ce qui est absurde). Soit c_i tel que $f(c_i) = \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(u)du$, et soit $L(t) = \log_2(\frac{f(t)}{\int_{\lfloor t \rceil - \frac{1}{2}}^{\lfloor t \rceil + \frac{1}{2}} f(u)du})$. On applique l'inégalité des accroissements finis à $L(t)$, pour $t \in [i-\frac{1}{2}, i+\frac{1}{2}]$:

$$|L(t) - L(c_i)| \leq |t - c_i| \times sup_{[i-\frac{1}{2}, i+\frac{1}{2}]}|L'| \quad (5.20)$$

Par définition de c_i, $L(c_i) = 0$.

$$|L(t)| \leq |t - c_i| \times sup_{[i-\frac{1}{2}, i+\frac{1}{2}]}|L'| \quad (5.21)$$

Et en injectant 5.21 dans 5.17 :

$$|H(\tilde{X}) - H(X)| \leq \sum_{i=-N}^{N} sup_{[i-\frac{1}{2}, i+\frac{1}{2}]}|L'| \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t)|t - c_i|dt \quad (5.22)$$

1. g est une fonction en escalier qui prend en tout point t la valeur moyenne de f sur l'intervalle $[\lfloor t \rceil - \frac{1}{2}, \lfloor t \rceil + \frac{1}{2}]$

$$|H(\tilde{X}) - H(X)| \leq \frac{1}{ln(2)} \sum_{i=-N}^{N} sup_{[i-\frac{1}{2},i+\frac{1}{2}]} |\frac{f'}{f}| \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t)|t - c_i|dt \quad (5.23)$$

Dans l'intégrale, t et c_i appartiennent tous deux à l'intervalle $[i-\frac{1}{2}, i+\frac{1}{2}]$, donc $|t - c_i| \leq 1$. On obtient :

$$|H(\tilde{X}) - H(X)| \leq \frac{1}{ln(2)} \sum_{i=-N}^{N} sup_{[i-\frac{1}{2},i+\frac{1}{2}]} |\frac{f'}{f}| \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t)dt \quad (5.24)$$

Application au cas gaussien

On suppose désormais que X suit une loi normale centrée de variance σ^2 et coupée sur $[-N, N]$ ($X \sim \text{clamp}(\mathcal{N}(0, \sigma^2), -N, N)$), et on suppose N suffisamment grand pour que la probabilité de la queue de la distribution soit négligeable. Soit $t'_i = \text{argmax}_{t \in [i-\frac{1}{2},i+\frac{1}{2}]} |\frac{f'(t)}{f(t)}|$. En reprenant 5.24 :

$$|H(\tilde{X}) - H(X)| \leq \frac{1}{ln(2)} \sum_{i=-N}^{N} |\frac{f'(t'_i)}{f(t'_i)}| \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t)dt \quad (5.25)$$

En remplaçant f par la densité de la loi normale :

$$|H(\tilde{X}) - H(X)| \leq \frac{1}{ln(2)} \sum_{i=-N}^{N} |\frac{t'_i}{\sigma^2}| \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} f(t)dt \quad (5.26)$$

t'_i appartient à l'intervalle $[i - \frac{1}{2}, i + \frac{1}{2}]$, donc $t - 1 \leq t'_i \leq t + 1$. Par la relation de Chasles :

$$|H(\tilde{X}) - H(X)| \leq \frac{1}{ln(2)\sigma^2} (\int_{-N}^{0} (-t + 1)f(t)dt + \int_{0}^{N} (t + 1)f(t)dt) \quad (5.27)$$

$$|H(\tilde{X}) - H(X)| \leq \frac{1}{ln(2)\sigma^2} (\int_{-N}^{0} -tf(t)dt + \int_{0}^{N} tf(t)dt + \int_{-N}^{N} f(t)dt) \quad (5.28)$$

La somme des deux premières intégrales est l'espérance d'une loi demi-normale (coupée sur $[0, N]$), c'est-à-dire de la valeur absolue d'une loi normale. Elle vaut $\sigma\sqrt{\frac{2}{\pi}}$. On fait l'approximation que l'effet du coupage sur $[0, N]$ sur l'espérance est négligeable. La troisième intégrale vaut 1 par propriété d'une densité. On conclut :

$$|H(\lfloor X \rceil) - H(X)| \leq \frac{1}{ln(2)} (\frac{1}{\sigma}\sqrt{\frac{2}{\pi}} + \frac{1}{\sigma^2}) \quad (5.29)$$

Ainsi, d'après 5.29, si σ est suffisamment grand, $H(X)$ est une bonne approximation de $H(\lfloor X \rceil)$, et donc l'entropie différentielle 5.11 est un bon proxy pour l'entropie discrète de la distribution quantifiée $\lfloor X \rceil$. L'objectif entropique peut donc être dérivé par rapport aux poids en injectant 5.11 dans 5.8 :

$$\frac{\partial \mathcal{L}^{entropy}}{\partial W} = \frac{\frac{\partial V(W)}{\partial W}}{2ln(2)V(W)\hat{H}} \tag{5.30}$$

$$\frac{\partial \mathcal{L}^{entropy}}{\partial W} = \frac{\frac{\partial (E(W^2)-E(W)^2)}{\partial W}}{2ln(2)V(W)\hat{H}} \tag{5.31}$$

$$\frac{\partial \mathcal{L}^{entropy}}{\partial W} = \frac{W - E(W)}{n.ln(2)V(W)\hat{H}} \tag{5.32}$$

où n est le nombre de paramètres de la couche. On note que si les poids sont centrés, le gradient 5.32 est une forme de weight decay [Krogh and Hertz, 1991], dépendant de la couche et de la dispersion des poids. Ce gradient donne une valeur analytique au paramètre du weight decay.

Résultats expérimentaux

On commence par vérifier la qualité de notre approximation. On entraîne un Resnet-18 (16-32-64-128, ~ 400k paramètres) sur cifar10 avec l'optimiseur SGD, et on le quantifie en 4-bit avec la méthode SAT. On calcule la valeur d'entropie réelle et la valeur donnée par notre approximation à l'issue de l'entraînement en fonction de divers objectifs de compression, donnés en Mo ($H \times |W|$). Le résultat est reporté dans la Table 5.3.

Table 5.3 – Évaluation de la qualité du proxy sur une variété de précisions de quantification et d'objectifs entropiques

Network	Quantization	Metric	Baseline	**Size (MB)**		
Resnet-18	W4A32	$H \times \|W\|$	0.228	0.120	0.099	0.062
		Proxy	0.229	0.12	0.10	0.06
	W3A32	$H \times \|W\|$	0.132	0.077	0.069	0.063
		Proxy	0.132	0.07	0.06	0.05
	W2A32	$H \times \|W\|$	0.088	0.078	0.067	0.059
		Proxy	0.088	0.08	0.06	0.05

En 4-bit, les valeurs d'entropie calculées par notre proxy sont très proches des valeurs réelles d'entropie des poids du réseau (avec une erreur relative de l'ordre de 1 à 3%). Aux précisions plus basses et lorsque l'objectif entropique est plus agressif, la qualité du proxy tend à se dégrader, ce qui coïncide avec le constat 5.29 : lorsque l'écart-type de la distribution est faible, le proxy est peu fidèle à l'entropie discrète des poids quantifiés.

Sur la Figure 5.1, on représente la distribution des poids 4-bit d'une couche du réseau entraîné sans objectif entropique, que l'on compare avec celle de la même couche du réseau entraîné avec un objectif entropique. On note l'impact de $\mathcal{L}^{entropy}$, qui augmente la concentration des poids autour des valeurs médianes. Là où les poids sont dispersés sur 15 valeurs sans loss entropique, ils sont concentrés sur 6 valeurs seulement avec l'ajout de l'objectif entropique. Ceci se traduit par une diminution de la borne entropique de compression sur la couche concernée : de 48.59 Ko à 28.46 Ko.

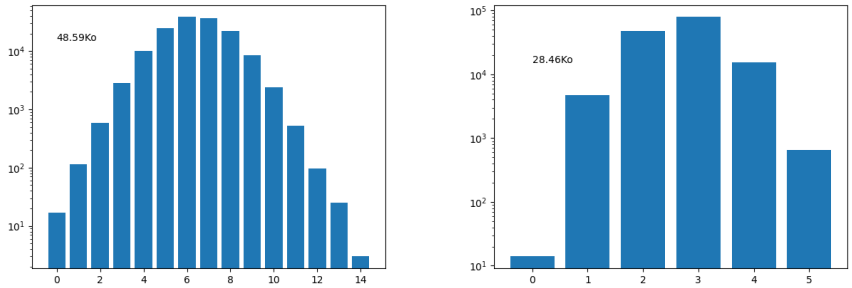

Figure 5.1 – Distribution des poids d'une couche d'un Resnet-18 4-bit entraîné sur cifar10, sans objectif entropique (gauche), et avec objectif entropique (droite). Abscisse : niveau de quantification. Ordonnée : nombre de poids.

Précision de quantification et objectif entropique

Table 5.4 – Performance du réseau en fonction de la précision de quantification et de l'entropie des poids

Network	Quantization		**Size (MB) - Top-1**		
Resnet-18	W4A32	$H \times \|W\|$	0.229	0.100	0.085
		Top-1 Acc	90.63	89.72	86.98
	W3A32	$H \times \|W\|$	0.132	0.077	0.69
		Top-1 Acc	90.71	90.19	89.25
	W2A32	$H \times \|W\|$	0.088	0.078	0.067
		Top-1 Acc	89.63	89.74	89.38

On mène plusieurs expériences pour comparer les impacts de notre objectif entropique et d'une quantification plus agressive sur la performance du

réseau. Dans la Table 5.4, on inventorie les performances obtenues pour plusieurs objectifs de compression et plusieurs précisions de quantification, et on trace le résultat sur la Figure 5.2.

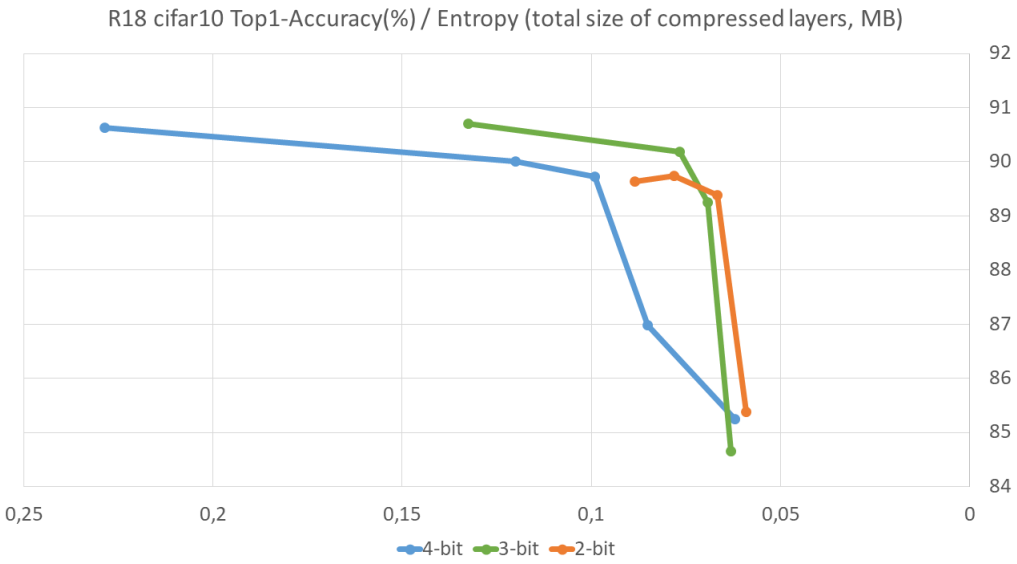

Figure 5.2 – Performance d'un Resnet-18 sur cifar10 en fonction de l'entropie des poids du réseau et de leur précision. Abscisse : taille mémoire totale des couches de convolution compressées. Ordonnée : Top-1 Accuracy.

En étudiant les résultats, on remarque d'abord qu'une précision de quantification élevée n'offre pas nécessairement de meilleurs résultats de performance (ce que confirment Jin et al. [2019], Esser et al. [2020]). Par ailleurs, en regardant l'impact du critère entropique sur la performance à divers degrés de précision de quantification, on constate qu'une précision de quantification plus faible permet d'atteindre des entropies plus basses avec des pertes de performance moins importantes. En plus de réduire la taille mémoire intrinsèque du réseau, la précision de quantification a donc un impact sur son potentiel de compressibilité.

5.2.2 . Quantification avec point zéro

Dans ce paragraphe, on s'interroge sur l'opportunité d'utiliser un quantifieur en précision étendue (voir 4.4.1). Plus spécifiquement, on souhaiterait un nombre impair de seuils afin d'avoir un point de quantification en 0. En effet, du fait de la distribution naturelle des poids dans le réseau entraîné, ce point de quantification particulier capterait une très grande partie des poids, et donc contribuerait potentiellement à minimiser l'entropie des poids quantifiés. Afin de valider cette intuition, on reproduit l'expérience de la Table 5.4 avec des poids ternaires (c'est-à-dire de type {-1,0,1}) et avec des poids sur 5 seuils. On reporte les résultats dans la Table 5.5 et sur la Figure 5.3.

Adaptation de l'estimateur de l'entropie au cas impair

En expérimentant, on remarque que la qualité de notre estimateur différentiable de l'entropie se dégrade fortement en ternaire, ce qui s'explique par le fait que l'écart-type de la distribution ternaire est très faible : l'erreur entre $H(X)$ et $H(\lfloor X \rceil)$ est trop importante (cf. Equation 5.29). On l'adapte pour cette précision en particulier : en supposant simplement que la distribution est ternaire et symétrique, on a :

$$H(X) = -\sum_{x=-1}^{1} p(X=x)\log_2(p(X=x)) \tag{5.33}$$

Par symétrie :

$$H(X) = -p(X=0)\log_2(p(X=0)) - 2p(X=1)\log_2(p(X=1)) \tag{5.34}$$

Or :

$$V(X) = E(X^2) - E(X)^2 = 2p(X=1) \tag{5.35}$$

Donc :

$$H(X) = -(1-V(X))\log_2(1-V(X)) - V(X)\log_2(\frac{V(X)}{2}) \tag{5.36}$$

Notre estimateur de l'entropie adapté à la précision ternaire devient donc :

$$H(x) = -(1-V(x))\log_2(1-V(x)) - V(x)\log_2(\frac{V(x)}{2}) \tag{5.37}$$

où $V(x)$ est la variance empirique.

Résultats expérimentaux

Table 5.5 – Performance du réseau en fonction de la précision de quantification et de l'entropie des poids

Network	Quantization		**MB/Top-1 Acc**		
Resnet-18	5 thresholds	$H \times \|W\|$	0.07	0.051	0.043
		Top-1 Acc	90.21	87.8	82.78
	ternary	$H \times \|W\|$	0.008		
		Top-1 Acc	81.75		

Les résultats sur 5 seuils valident notre intuition, on parvient à compresser davantage le réseau avec moins de pertes grâce au point zéro. Sur l'expérience de quantification ternaire, la dégradation de performance est certes

importante, mais il faut noter que le taux de compression atteint est lui aussi très important (environ 10 fois plus élevé que la baseline 2-bit sur les couches compressées, du fait de la sparsité très importante de la distribution des poids).

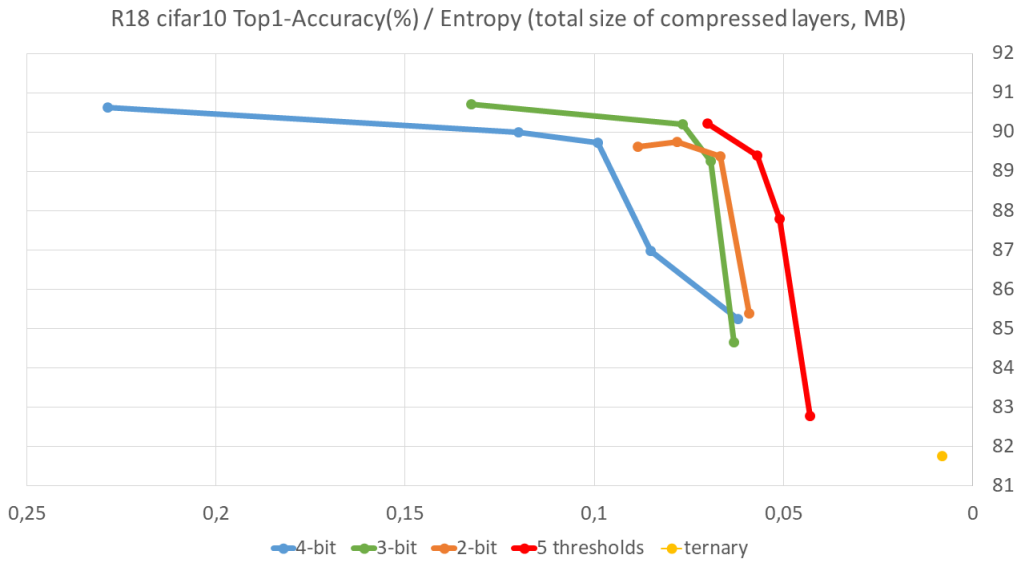

Figure 5.3 – Performance d'un Resnet-18 sur cifar10 en fonction de l'entropie des poids du réseau et de leur précision, avec précisions étendues.

Le défaut d'un nombre de seuils quelconque est la nécessité de stocker les poids sur un nombre de bits supérieur au logarithme en base 2 du nombre de seuils, et donc de perdre de l'information. Mais on rappelle que l'on veut encoder les poids à l'inférence. Cette considération matérielle n'est donc pas à prendre en compte pour le stockage mémoire, mais seulement pour l'implémentation des opérateurs situés en aval du décodeur.

5.2.3 . Bilan

On a montré qu'il existe un lien entre précision de quantification et entropie des poids. Ainsi, pour compresser au mieux un réseau, il paraît judicieux de jouer sur les deux aspects en adoptant un quantifieur aussi agressif que possible et avec un point zéro. L'utilisation d'une méthode de quantification à précision mixte telle que Fracbits est donc tout indiquée afin d'atteindre les précisions et les entropies les plus basses possibles dans les couches les moins sensibles et les plus peuplées.

Par la suite, on s'intéresse au décodeur. Afin d'exploiter au mieux notre méthode, il est crucial de disposer d'un décodeur entropique efficace à la fois du point de vue de la compression et de l'implémentation matérielle.

5.3 . Codage ANS

La nature des contraintes matérielles liées à l'inférence diffèrent en fonction du type de matériel utilisé (CPU, GPU, TPU, ASIC, FPGA...), si bien qu'il serait vain de prétendre avoir un codeur entropique optimal pour toutes ces architectures à la fois. On peut néanmoins effectuer quelques hypothèses sur un cahier des charges commun à ces dernières pour notre décodeur.

- Être compatible avec la programmation SIMD (Single Instruction Multiple Data), et donc avec la parallélisation.
- Être compatible avec le modèle flot de données.
- Avoir une bande passante élevée pour ne pas ralentir le transfert de données.
- Être assez performant en compression pour apporter un gain mémoire substantiel qui justifie son utilisation.

La littérature de la compression de réseaux de neurones [Han et al., 2016, Wiedemann et al., 2020, Oktay et al., 2020, Tartaglione et al., 2021] présente divers exemples d'utilisation de codeurs entropiques [Huffman, 1952, Golomb, 1966, Langdon, 1984] mais il apparaît qu'aucun d'entre eux ne satisfait les trois critères susmentionnés. Par exemple, le codage de Huffman et le code de Golomb n'atteignent pas la borne entropique tandis que le codage arithmétique est réputé trop complexe au décodage (utilisant deux états pour représenter un intervalle et nécessitant une renormalisation). Le codage ANS (Asymmetric Numeral Systems) [Duda, 2009] existe quant à lui en plusieurs variantes. On s'intéresse en particulier à la variante tANS (tabled ANS), dont le décodeur peut être implémenté à l'aide d'une look-up table (LUT).

5.3.1 . Principe de fonctionnement

Le décodeur de tANS consiste en un automate fini. Soit n le nombre de symboles à encoder (dans notre cas : le nombre de points de quantification), le nombre d'états de l'automate peut être fixé à $l = 2^b n$ où b est un entier. À chaque état est affecté un symbole (plusieurs états peuvent décoder le même symbole), un nombre de bits à lire dans la chaîne à décoder ainsi qu'un entier utile pour calculer l'état suivant.

Sur la Figure 5.4 est représentée une telle machine à états pour une certaine distribution binaire {a,b}. Pour encoder un "a" à partir d'un certain état, on suit la transition partant du haut de cet état et on concatène la chaîne binaire qui étiquette la transition. Pour décoder, on parcourt l'automate dans le sens inverse en fonction des bits lus dans la chaîne. On remarque que certaines transitions ont une chaîne binaire vide : c'est tout l'avantage d'ANS par rapport au codage de Huffman, pour lequel chaque symbole est encodé par au moins 1 bit. On donne le pseudocode du décodeur dans la partie 5.6.

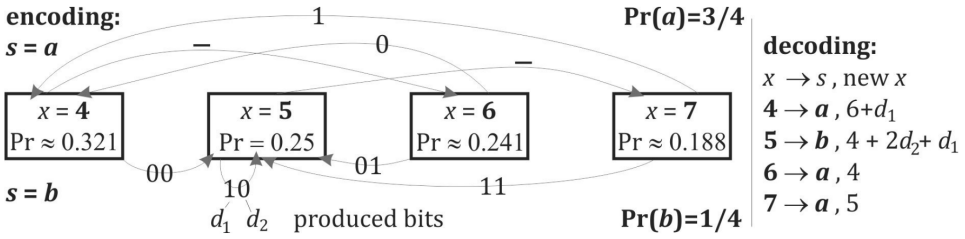

Figure 5.4 – Machine à états pour l'encodage et le décodage ANS [Duda, 2009].

La construction de cette machine à états dépend de la distribution : le nombre d'états affectés à un symbole dépend de la probabilité de ce symbole (plus il est rare et moins il a d'états associés). Duda [2009] montre que la qualité de la compression dépend en général de l'initialisation des états. Naturellement, plus il y a d'états disponibles (plus b est grand), et meilleure est la compression, l'idéal étant d'avoir $p_s = \frac{l_s}{l}$ pour chaque symbole s, où l_s est le nombre d'états associés à s et p_s la probabilité réelle du symbole s.

5.3.2 . Borne de compression

Il est possible de borner asymptotiquement l'écart ΔH entre la longueur du code et l'entropie de la distribution (Equation 5.38, Duda [2009]) :

$$\Delta H \leq \frac{1}{l^2 ln(4)} \sum_s \frac{1}{p_s} (\frac{p_s}{2\text{min}_{s'} p_{s'}} + \frac{1}{2})^2 + \mathcal{O}(l^{-3}) \tag{5.38}$$

L'écart avec la borne entropique décroît quadratiquement avec le nombre d'états. En pratique, $l = 4n$ (4 fois plus d'états que de symboles) donne généralement un écart de l'ordre de 0.01 bit par symbole [Duda, 2009].

5.3.3 . Implémentation

Il reste à adresser le surcoût mémoire dû au stockage de la table elle-même. C'est un facteur important, car la table doit être stockée en permanence au plus près de l'unité de calcul. À titre d'exemple, on suppose un décodeur à 64 états de symboles 4-bit. Dans chacun des 64 états, on doit stocker 4 bits pour le symbole à décoder, un nombre de bits à lire (lui-même encodé sur $\lceil \log_2(\log_2(64)) \rceil = 3$ bits), et une transition encodée sur $\log_2(64) = 6$ bits. Au total, toute l'information du décodeur peut être stockée sur $13 \times 64 = 832$ bits, soit 104 octets. Pour plus de simplicité, on pourrait vouloir encoder chaque élément de la table sur un octet, soit une taille mémoire totale de 192 octets.

Sur la Figure 5.5, on propose un schéma d'implémentation simple de décodeur à 64 états qui repose sur une LUT et un additionneur 6-bit. Le flux de poids compressés arrive dans une structure de type FIFO (First-In First-Out) qui interagit avec la table de décodage. L'état initial est envoyé à la table, qui

répond en émettant trois signaux. L'additionneur est utilisé pour calculer l'état suivant.

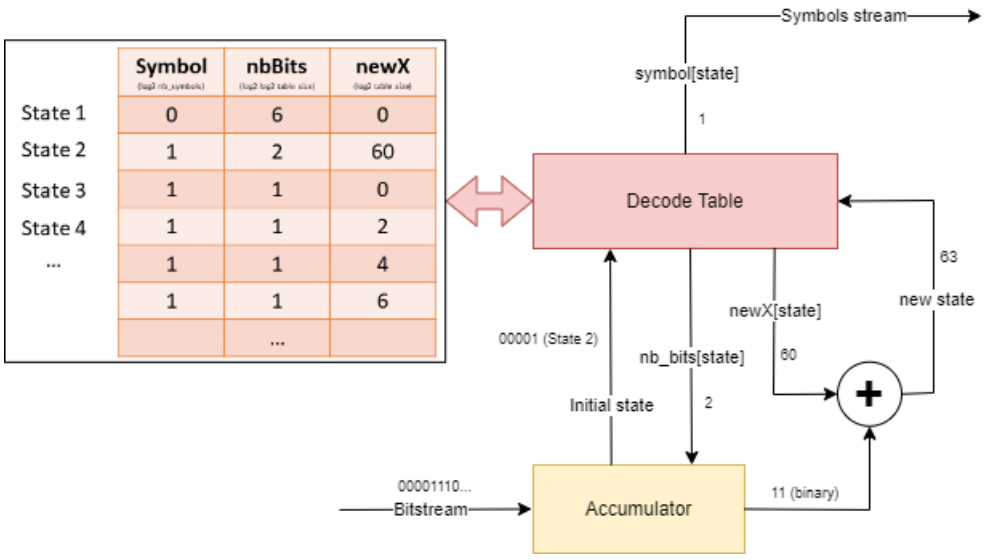

Figure 5.5 – Diagramme du décodeur ANS

Il est à noter que la présence de l'additionneur n'est pas forcément nécessaire. On peut aussi précalculer les additions et tabuler leurs résultats à l'intérieur de la table de décodage. Dans ce cas, il faut mémoriser chaque transition pour chaque état, ce qui se traduit par une augmentation de la taille de la table. Si l est le nombre d'états, il faut alors stocker de l'ordre de l^2 transitions, chacune représentée par un numéro d'état (un octet). Dans l'exemple du décodeur de la Figure 5.5, il faut stocker au maximum 64 transitions par état, pour un total de 64×64 octets, soit 4 KiB. On représente cette architecture de décodeur sans additionneur sur la Figure 5.6.

ANS est compatible avec le modèle flot de données, ce qui lui donne un avantage sur les codeurs reposant sur un contexte (comme Wiedemann et al. [2020] par exemple, qui utilise les valeurs de deux symboles précédemment décodés pour décoder le symbole courant). On peut également le rendre compatible avec la programmation parallèle (SIMD). Pour cela, il faut encoder séparément les flux de données parallèles. On peut alors réutiliser le même décodeur sur plusieurs flux simultanés. Au lieu d'avoir un état unique, le décodeur aura dans ce cas un vecteur d'états. L'impact de la séparation des flux parallèles sur la capacité de compression sera étudié dans 5.4.1.

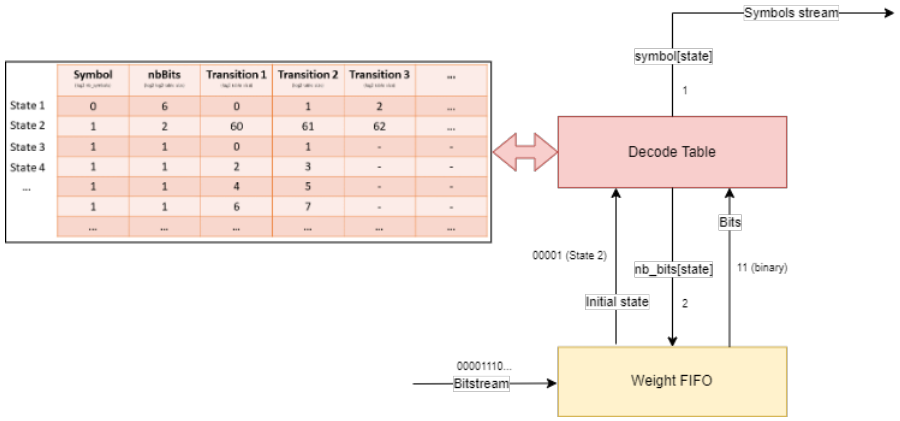

Figure 5.6 – Diagramme du décodeur ANS sans additionneur

5.4 . Expériences

On mène des expériences sur ImageNet et sur les réseaux Resnet-18 et Resnet-50 [He et al., 2015]. À l'heure de la rédaction de ce manuscrit, les résultats de cette partie n'incluent pas l'objectif entropique 5.32, l'auteur n'étant pas encore parvenu à des résultats concluants sur cette base de données. Ils incluent en revanche la quantification avec point zéro.

Table 5.6 – Performance des réseaux et tailles mémoire avec poids quantifiés et compressés, ImageNet, et performances de trois méthodes de l'état de l'art de la compression.

Network	Method	Top-1 Acc	Size	$H\|W\|$
Resnet-18 69.8% 46.8 MB	[Tartaglione et al., 2021]	68.8%	3.6 MB	
	[Oktay et al., 2020]	70.0%	2.78 MB	
	Fracbits (weights only)	68.5%	2.01 MB	1.99 MB
	Ours (weights only)	69.0%	**1.67 MB**	1.65 MB
Resnet-50 76.1% 102.5 MB	[Tartaglione et al., 2021]	71.3%	5.5 MB	
	[Oktay et al., 2020]	73.5%	5.91 MB	
	[Wiedemann et al., 2020]	73.7%	5.25 MB	
	[Wiedemann et al., 2020]	74.5%	10.4 MB	
	Ours (weights only)	75.6%	**4.52 MB**	4.39 MB

Pour chaque couche, on compresse tous les poids en une seule fois avec ANS (table de décodage de 256 états), et on reporte la taille mémoire atteinte. Sur ces réseaux, on note une supériorité nette de l'approche avec point zéro associée à la quantification en précision mixte sur les trois méthodes de l'état de l'art reportées, qui utilisent toutes la compression entropique. Il faut par ailleurs souligner que Oktay et al. [2020] doit entraîner un décodeur en plus des paramètres du réseau, que Tartaglione et al. [2021] utilise la méthode de compression LZMA [Ziv and Lempel, 1977], et que Wiedemann et al. [2020] utilise un codage de Golomb doublé d'un codage arithmétique avec contexte. Les décodeurs de ces codes sont beaucoup plus complexes qu'ANS et inutilisables en pratique à l'inférence (ce qui n'est pas l'objectif de ces trois références).

Afin de contrôler que le point zéro a bel et bien un rôle dans la réduction d'entropie du réseau et que cette dernière n'est pas uniquement due à Fracbits, on propose une expérience témoin sur Resnet-18 avec un objectif de 2 Mo sans point zéro de quantification (Fracbits). Le taux de reconnaissance atteint de cette manière est de 68.5%, soit moins élevé qu'avec notre méthode, malgré un objectif de compression moins ambitieux. Ce résultat démontre l'intérêt de la quantification avec point zéro pour la réduction mémoire.

Dans le paragraphe suivant, on tente d'évaluer la faisabilité d'ANS du point de vue matériel en simulant deux contraintes : la parallélisation du décodage et la réduction du nombre d'états de la table de décodage.

5.4.1 . Parallélisation du décodage et taille de la table

Comme on l'a mentionné à la fin du paragraphe 5.3.3, on souhaite pouvoir paralléliser le décodage des poids dans le but de se conformer au paradigme de programmation SIMD. On reprend le Resnet-50 de la Table 5.6, et dans chaque couche, on regroupe les poids par canal d'entrée pour les compresser séparément. On reporte les coûts mémoire obtenus dans la Table 5.7. On compare également les taux de compression obtenus en fonction de la taille de la table de décodage. Celle-ci peut être fixée par l'utilisateur, et on peut borner l'écart entre le coût mémoire obtenu et la borne entropique grâce à 5.38.

Table 5.7 – Impact de la parallélisation et de la taille de la table de décodage sur la taille du réseau compressé.

		Compressed size	
Network	Table size (l)	Single stream	Parallel streams
Resnet-50	256	4.52 MB	4.54 MB
	128	4.68 MB	4.70 MB
	64	5.02 MB	5.04 MB

On constate que la parallélisation n'augmente que très légèrement le coût mémoire (moins de 1% de surcoût). 256 états sont nécessaires pour s'approcher à 3% en écart relatif de la borne entropique (4.39 Mo), mais il est possible de descendre jusqu'à 64 états avec une pénalité de l'ordre de 15% sur le coût mémoire.

5.4.2 . Analyse

Dans ce paragraphe, on fait état des précisions de quantification pour le Resnet-50 étudié ci-dessus, ainsi que de la répartition de la complexité mémoire au sein de ses différentes couches.

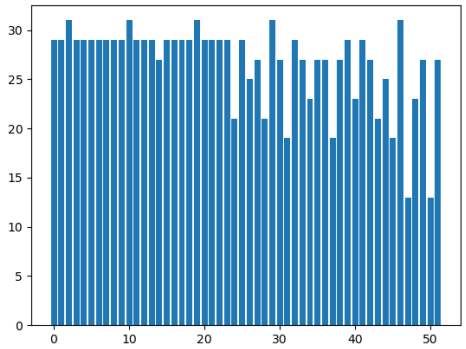

Figure 5.7 – Répartition des précisions de quantification parmi les couches du Resnet-50, en nombre de seuils. Abscisse : numéro de la couche. Ordonnée : nombre de seuils.

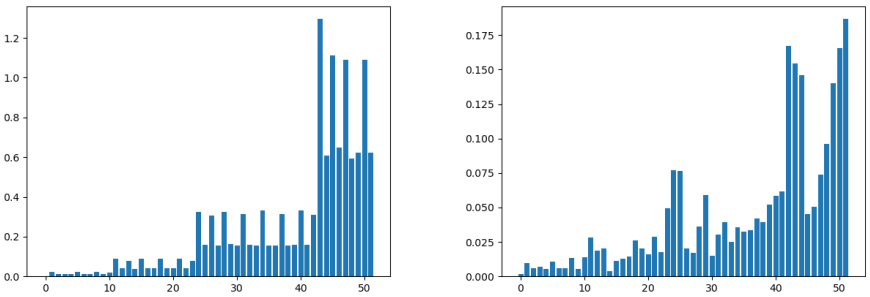

Figure 5.8 – Répartition du coût mémoire (en Mo, ordonnée) parmi les couches du Resnet-50 (numérotées en abscisses). Réseau non compressé (gauche), et compressé par ANS (droite). Les échelles des axes des ordonnées sont différentes.

Sur la Figure 5.7, on indique le nombre de seuils de quantification pour chaque couche de convolution (à l'exception de la première, restée en 8-bit). Étant donné que l'on quantifie avec un point zéro, tous ces nombres de seuils sont impairs. Le réseau est initialisé avec des précisions de 31 seuils (< 5-bit), et la première moitié du réseau demeure à peu près à cette précision à l'issue de l'entraînement. Vers la fin du réseau, les couches les plus peuplées ont en revanche vu leur précision diminuer, jusqu'à 13 seuils pour deux d'entre elles.

Sur la Figure 5.8, on regarde la taille mémoire par couche, non compressée sur la figure de gauche, compressée avec ANS sur la figure de droite. Certaines couches sont compressées jusqu'à 20 fois, et la bande passante crête passe de 1.3 Mo à 185 Ko pour la couche la plus consommatrice, soit une réduction de 86%. Le coût mémoire total nécessaire au stockage des paramètres des couches convolutives est d'environ 2.5 Mo, (4.5 au total, dont 2.0 pour le classifieur linéaire quantifié en 8-bit). Le taux de compression global des couches convolutives vaut donc $\frac{2.5\times 8.10^6}{102.5\times 8.10^6 - 32\times 2048\times 1000} = 0.0265$, soit 0.85 bit par poids. Le réseau atteint un taux de compression plus important que s'il était entièrement binarisé, mais la structure de calcul reste plus complexe du fait des opérations entières signées nécessaires au calcul des convolutions.

5.5 . Conclusion et perspectives

Dans cette troisième et dernière partie, on s'est intéressé à la combinaison de la quantification et du codage entropique pour la compression des poids du réseau à l'inférence. On a mis en évidence l'existence d'une interaction entre la précision de quantification et la compressibilité grâce à une approximation différentiable de l'entropie, rendant possible l'optimisation de l'entropie des poids par rétropropagation du gradient d'une loss. On a également montré l'intérêt d'un point zéro de quantification dans l'optique de réduire le coût mémoire d'un réseau quantifié. Pour finir, on a étudié les caractéristiques d'un codage de source particulier, le codage ANS, encore inutilisé (à notre connaissance) dans le domaine de la compression de réseaux de neurones, et qui présente des avantages pour l'implémentation *on-edge*.

Les résultats de cette étude montrent qu'un coût mémoire de moins de 1-bit en moyenne par poids (sur les couches de convolution) peut être atteint pratiquement sans pertes sur des Resnet et des tâches de reconnaissance d'images aussi complexes qu'ImageNet. Le stockage mémoire et la bande passante des réseaux peuvent donc être largement réduits moyennant l'intégration d'un décodeur de source efficace à l'inférence, tel qu'ANS.

La suite logique de ce travail est d'implémenter la décompression à l'inférence sur une puce (FPGA, TPU, ASIC...) et d'évaluer son gain matériel réel. Il a fait l'objet d'un preprint soumis en conférence.

5.6 . Annexe : décodeur ANS

On donne ci-dessous (Algorithme 8) le pseudocode du décodeur tANS. On note l'absence de multiplication et le recours à la look-up table "decodeTable".

Algorithm 8 decodeData

```
1: function decodeData(bitStream, StreamLength, decodeTable,
   tableLog)
2:     output ← []
3:     state, bitStream ← bitsToState(bitStream, tableLog)
4:     nb_decoded ← 0
5:     symbol ← 0
6:     while nb_decoded < StreamLength do
7:         symbol, state, bitStream ← decodeSymbol(state, bitStream,
   decodeTable)
8:         nb_decoded ← nb_decoded + 1
9:         output ← output + symbol
10:    end while
11:    Reverse output
12:    return output
13: end function
```

Algorithm 9 decodeSymbol

```
1: function decodeSymbol(state, bitStream, decodeTable)
2:     symbol ← decodeTable[state]['symbol']
3:     nbBits ← decodeTable[state]['nbBits']
4:     rest, bitStream ← bitsToState(bitStream, nbBits)
5:     state ← decodeTable[state]['newX'] + rest
6:     return symbol, state, bitStream
7: end function
```

Algorithm 10 Bits to state conversion

```
1:  function bitsToState(bitStream, nbBits)
2:      if nbBits = 0 then
3:          bits ← null
4:      else
5:          bits ← bitStream[−nbBits :]
6:      end if
7:      rest ← int(bits, 2)                          ▷ rest ← 0 if bits = null
8:      if bits = bitStream then
9:          remaining ← null
10:     else if nbBits = 0 then
11:         remaining ← bitStream
12:     else
13:         remaining ← bitStream[: −nbBits]
14:     end if
15:     return rest, remaining
16: end function
```

6 - Conclusion

Cette partie fait la synthèse des travaux de cette thèse et y apporte un recul critique. On donne également quelques perspectives pour les travaux présentés, et plus largement sur l'évolution du domaine de la quantification dans le contexte du deep learning.

6.1 . Synthèse des travaux

Au cours de cette thèse, on a cherché des manières de compresser un réseau de neurones tout en maintenant au mieux son niveau de performance sur sa tâche. En particulier, on a étudié trois hypothèses expérimentales correspondant à trois objectifs distincts :

- La quantification post-entraînement, qui consiste à quantifier un réseau après son entraînement (et éventuellement à le finetuner légèrement en utilisant un échantillon très restreint de données). On a cherché à repousser les limites de précision et de performance dans ce scénario en proposant en nouveau type de quantifieur plus adapté à la distribution réelle des paramètres du réseau, dans le but de minimiser l'erreur de quantification. LatticeQ présente une transformation avantageuse de l'espace des niveaux de quantification qui permet l'inférence en précision réduite, jusqu'à 4-bit pratiquement sans pertes, et atteint des performances compétitives avec l'état de l'art des méthodes PTQ sur la reconnaissance d'images [Banner et al., 2018, Li et al., 2021, Choukroun et al., 2019, Nagel et al., 2020].

- La quantification à l'apprentissage à très basse précision, qui s'accompagne de défis relatifs aux difficultés d'entraînement des réseaux dans ces conditions. On a analysé l'impact des très basses précisions sur la performance et cherché à l'atténuer en apportant une modification au pipeline d'entraînement. Notre méthode offre une amélioration simple et générique adaptable aux techniques de quantification uniforme grâce à la progressivité de l'apprentissage.

- La compression mémoire des réseaux à l'inférence, à la croisée de nombreux enjeux matériels (taille physique de la puce, bande passante, latence, efficacité énergétique...). Pour cela, on a exploré la combinaison de deux paradigmes de compression : la quantification en précision mixte et le codage de source. On a montré que ces deux approches pouvaient être conjuguées avantageusement à condition de disposer d'un type de codage de source compatible avec l'inférence, pour lequel on a identifié ANS comme candidat crédible. La compression ANS-neuronale

permet d'atteindre des tailles de réseau plus faibles que la binarisation sur ImageNet avec l'architecture Resnet, tout en limitant largement les déperditions : jusqu'à 0.85 bit par poids sur les couches convolutives de l'architecture Resnet-50 pour 0.5% de pertes.

Ces travaux ne vont pas sans un certain nombre de limitations. Tout d'abord, cette thèse s'est limitée aux aspects *software*. LatticeQ n'a pas encore fait l'objet d'une implémentation matérielle, non plus que la compression ANS pour confirmer sa compatibilité avec l'inférence. Ces deux méthodes reposent sur l'ajout d'un objet (base de quantification pour LatticeQ, décodeur pour ANS) de taille contenue, mais dont l'impact sur l'inférence reste à évaluer.

Comme on l'a écrit en introduction, les choix d'architectures matérielles précèdent généralement les choix logiciels. Faute d'avoir ciblé une architecture en particulier, les travaux de cette thèse se cantonnent principalement à la théorie et nécessitent un travail supplémentaire d'adaptation et d'implémentation matérielle pour pouvoir être utilisés en pratique.

6.2 . Méthodologie et conditions expérimentales

Dans ce paragraphe, on soulève deux problématiques importantes auxquelles l'auteur de cette thèse a été confronté au cours de ses recherches. Elles touchent à l'éthique scientifique dans le domaine du deep learning, à la méthodologie et aux conditions expérimentales.

6.2.1 . Quantification : variété des objectifs et méthodologie

Il n'existe pas de métrique unique de performance ou d'utilité d'une technique de compression de réseau de neurones. Au contraire, chaque technique définit son propre objectif (qu'il soit calculatoire, de taille mémoire, ou autre) et formule ses propres hypothèses expérimentales (PTQ stricte, PTQ finetuning, QAT, quantification des activations, première et dernière couche 8-bit...). Cette pluralité est intéressante du point de vue du monde réel, car chaque application industrielle possède ses propres contraintes matérielles et de performance, mais elle ne facilite pas les comparaisons entre méthodes.

Des benchmarks communs sont relativement bien admis : l'état de l'art a tendance à évaluer ses méthodes sur les mêmes réseaux et les mêmes jeux de données. Toutefois, les différences entre méthodes somme toute assez ressemblantes (il n'existe pas mille manières de concevoir un quantifieur uniforme) viennent parfois des à-côtés : utilisation de la distillation, utilisation de données de calibration pour le post-training, voire d'un recours plus intensif aux ressources de calcul. En effet, peut-on dire d'une méthode de quantification qu'elle est plus performante qu'une autre si son apport ne consiste qu'en

une augmentation du nombre d'époques d'apprentissage ? Ou si elle repose sur une somme de régularisations sans rapport avec la quantification pour atteindre une performance supérieure ? Ainsi, la notion de méthodologie de comparaison est cruciale pour cerner l'apport individuel d'une technique.

Parmi les méthodes de comparaison existantes, l'étude d'ablation est une pratique émergente. Présente dans de nombreuses publications, elle permet d'isoler et d'identifier précisément l'apport individuel de chacune des idées présentées. De l'avis de l'auteur de cette thèse, cette pratique est à généraliser, car elle est la seule qui permette une comparaison juste entre techniques aux hypothèses expérimentales proches mais distinctes. C'est également une pratique bénéfique du point de vue de l'éthique scientifique : une technique de quantification n'est pas un monolithe, elle est constituée de plusieurs couches et d'optimisations entremêlées, et il est du devoir de l'auteur de présenter le plus précisément possible l'apport de chacune d'entre elles dans un objectif de transparence. Certaines idées peuvent être séduisantes sur le papier mais n'avoir qu'un apport marginal en pratique, et camoufler ce fait en rajoutant des optimisations préexistantes sans rapport avec la méthode pour donner l'illusion de meilleurs résultats est constitutif d'une faute déontologique.

6.2.2 . Aspects expérimentaux

Le deep learning se situe à la croisée des sciences théoriques (analyse, statistiques...) et des sciences expérimentales (sciences cognitives, neurosciences...). Si la théorie guide les idées, la performance d'un réseau de neurones ne peut être évaluée que par l'expérience et est d'ailleurs bien souvent déterminée par les conditions expérimentales. De fait, l'accès à des ressources de calcul en quantité suffisante est la condition sine qua non de la démarche scientifique dans ce domaine. La qualité et la quantité des expériences effectuées, le nombre d'hypothèses testées, le parcours de l'espace des hyperparamètres constituent la partie immergée de l'iceberg de n'importe quelle publication, et sont incontournables pour obtenir des résultats satisfaisants.

L'évolution constante des standards expérimentaux vers des modèles toujours plus lourds (Alexnet, puis Resnet, et maintenant les transformers pour la vision) stimule le besoin de machines toujours plus puissantes. La plupart des expériences de cette thèse ont pu être menées grâce aux ressources du laboratoire qui l'a financée, sur des cartes graphiques Nvidia-A100 de dernière génération. Mais même un tel équipement serait insuffisant pour expérimenter dans de bonnes conditions sur des modèles de langage de type GPT4. Un tel recours aux ressources calculatoires pose question, non seulement du point

de vue de la consommation énergétique, mais aussi et surtout de l'accessibilité à la science.

Pas plus que les mathématiques ou la physique, le machine learning ne devrait être réservé à ceux qui possèdent la ressource économique et donc la puissance de calcul. La généralisation de l'open source pourrait constituer l'une des solutions à ce goulot d'étranglement, en participant à une diffusion libre et large de la connaissance scientifique et des algorithmes. Les grandes conférences du domaine – où sont publiées un grand nombre de contributions – telles que ICLR (International Conference on Learning Representations) ou NeurIPS (Neural Information Processing Systems) encouragent notamment la publication des codes sources. L'open source est également le choix effectué par Meta (Facebook), qui rend publics ses LLM "Llama" [Touvron et al., 2023]. L'accès libre à ces codes sources offre par ailleurs une meilleure vérifiabilité des résultats expérimentaux, à l'heure où le domaine de l'IA et la science en général rencontrent une véritable crise de la reproductibilité.

6.3 . Perspectives générales

Dans ce paragraphe, on s'intéresse à certains des débouchés potentiels de ces travaux et aux perspectives du domaine du deep learning embarqué.

6.3.1 . Unification des paradigmes de compression

Comme on l'a noté au chapitre 5, la combinaison de la quantification en précision mixte et du codage de source présente des résultats prometteurs. De manière générale, il semble que l'utilisation conjointe de plusieurs modes de compression (pruning et quantification, NAS et quantification…) [Wang et al., 2020b,c, Yang et al., 2019] permette d'atteindre des taux de compression mémoire plus importants que de miser un seul d'entre eux à la fois.

L'unification de ces différents modes par des plateformes et des librairies communes pourrait faciliter les interactions entre ces domaines aux objectifs similaires mais aux moyens distincts. Elle pourrait mener à une ère de "réseaux sur mesure", où un réseau adapté à une tâche et à ses contraintes particulières pourrait être designé de manière très souple selon des règles de construction reposant sur des briques élémentaires (types de blocs, quantification, sparsification, codage de source…).

6.3.2 . Quantifier des LLM ?

En introduction, on a évoqué les LLM (Large Language Models) et notamment l'exemple de ChatGPT. Le coût de l'exploitation de ces réseaux amène naturellement la question de leur compression, et les défis propres à la compression des LLM sont multiples. Premièrement, l'entraînement d'un LLM est une tâche très complexe, très longue et très coûteuse en soi, sans même parler d'y ajouter un entraînement QAT. Le second défi concerne l'accessibilité

aux données d'entraînement. L'entraînement d'un LLM requiert d'immenses quantités de données difficiles à obtenir pour qui n'est pas déjà un géant du numérique. Le troisième défi relève de la nature de l'entraînement des LLM qui a souvent lieu en plusieurs étapes : entraînement d'un modèle fondamental sur des données massives, puis finetuning et enfin apprentissage par renforcement avec feedback humain. Ces différentes étapes sont difficiles à répliquer pour un apprentissage QAT, ce qui explique que les recherches sur la quantification des LLM se soient jusqu'ici principalement focalisées sur le post-training [Xiao et al., 2023].

Les LLM s'appuient sur l'architecture du transformer [Vaswani et al., 2023]. Une première étape pour appréhender les enjeux relatifs à leur quantification pourrait donc être d'étudier de plus petits transformers, comme BERT [Devlin et al., 2019]. L'état de l'art [Wei et al., 2022, Bondarenko et al., 2021, Dettmers et al., 2022] établit que les enjeux de la quantification des transformers sont différents de ceux que l'on a rencontrés en quantifiant des réseaux de convolution. Les activations des transformers ont une dynamique élevée et des outliers structurés dont la distribution complexifie la quantification en basse précision. Bondarenko et al. [2021] observe néanmoins que la QAT permet d'améliorer grandement les performances de la quantification sur BERT. L'exploration de cette piste constitue donc un axe de recherche naturel pour la quantification des LLM.

6.3.3 . Pour finir

Déployer des réseaux de neurones plus frugaux et mieux dimensionnés par rapport à leur tâche peut bénéficier à tous : à l'utilisateur, aux applications, et à la société, à la fois dans un objectif de sobriété énergétique et d'embarquabilité de ces réseaux pour les applications *on-edge*. À son échelle, cette thèse a exploré diverses manières de parvenir à cet objectif par la quantification.

Dans un domaine très appliqué comme le deep learning embarqué, l'implémentation doit être à l'initiative et à l'aboutissement de tout travail de recherche. C'est pourquoi les travaux de cette thèse ont vocation à être utilisés au sein du projet ANR Deep Green, une plateforme française d'IA embarquée et de conception hardware destinée au marché français et européen.

Bibliographie

Martín Abadi, Paul Barham, Jianmin Chen, and al. TensorFlow : A system for large-scale machine learning, May 2016. URL http://arxiv.org/abs/1605.08695. arXiv :1605.08695 [cs].

J. Alimena, Y. Iiyama, and J. Kieseler. Fast convolutional neural networks for identifying long-lived particles in a high-granularity calorimeter. *Journal of Instrumentation*, 15(12) :P12006–P12006, December 2020. ISSN 1748-0221. doi : 10.1088/1748-0221/15/12/P12006. URL https://iopscience.iop.org/article/10.1088/1748-0221/15/12/P12006.

L Babai. NEAREST LATTICE POINT PROBLEM. page 13, 1986.

Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. 2018. URL https://api.semanticscholar.org/CorpusID:59292009.

Y. Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5 :157–66, 02 1994. doi : 10.1109/72.279181.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://api.semanticscholar.org/CorpusID:18406556.

Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? *ArXiv*, abs/2003.03033, 2020. URL https://api.semanticscholar.org/CorpusID:212628335.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization, 2021. URL https://api.semanticscholar.org/CorpusID:237940329.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, and al. Pact : Parameterized clipping activation for quantized neural networks. *ArXiv*, abs/1805.06085, 2018. URL https://api.semanticscholar.org/CorpusID:21721698.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit Quantization of Neural Networks for Efficient Inference. *arXiv :1902.06822 [cs, stat]*, March 2019. URL http://arxiv.org/abs/1902.06822. arXiv : 1902.06822.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect : Training deep neural networks with binary weights during propagations. In

Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, NIPS'15, page 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 () : 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35 :30318–30332, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding, 2019.

Zhen Dong, Zhewei Yao, Yaohui Cai, and al. Hawq-v2 : Hessian aware trace-weighted quantization of neural networks, 2019a. URL https://api.semanticscholar.org/CorpusID:207852310.

Zhen Dong, Zhewei Yao, Amir Gholami, and al. Hawq : Hessian aware quantization of neural networks with mixed-precision, 2019b. URL https://api.semanticscholar.org/CorpusID:148571720.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, and al. An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale, June 2021. URL http://arxiv.org/abs/2010.11929. arXiv :2010.11929 [cs].

Jarek Duda. Asymmetric numeral systems. *ArXiv*, abs/0902.0271, 2009. URL https://api.semanticscholar.org/CorpusID:13331542.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, and al. Learned Step Size Quantization. *arXiv :1902.08153 [cs, stat]*, May 2020. URL http://arxiv.org/abs/1902.08153. arXiv : 1902.08153.

Farah Fahim, Benjamin Hawks, Christian Herwig, and al. hls4ml : An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices, March 2021. URL http://arxiv.org/abs/2103.05579. arXiv :2103.05579 [physics].

Angela Fan, Pierre Stock, Benjamin Graham, and al. Training with quantization noise for extreme model compression, 2020. URL https://api.semanticscholar.org/CorpusID:215814169.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis : Finding sparse, trainable neural networks, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

S. Golomb. Run-length encodings (corresp.). *IEEE Transactions on Information Theory*, 12(3) :399–401, 1966. doi : 10.1109/TIT.1966.1053907.

Yury Gorbachev, Mikhail Fedorov, Iliya Slavutin, Artyom Tugarev, Marat Fatekhov, and Yaroslav Tarkan. Openvino deep learning workbench : Comprehensive analysis and tuning of neural networks inference. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 783–787, 2019. URL https://api.semanticscholar.org/CorpusID:207926507.

Lionel Gueguen, Alex Sergeev, Ben Kadlec, and al. Faster neural networks straight from jpeg. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/7af6266cc52234b5aa339b16695f7fc4-Paper.pdf.

K. Guo, W. Li, K. Zhong, and al. Neural network accelerator comparison [online]. URL https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29 : Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1379–1387, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/2823f4797102ce1a1aec05359cc16dd9-Abstract.html.

Hai Victor Habi, Roy H. Jennings, and Arnon Netzer. HMQ : Hardware Friendly Mixed Precision Quantization Block for CNNs. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, volume 12371, pages 448–463. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58573-0 978-3-030-58574-7. doi : 10.1007/978-3-030-58574-7_27. URL https://link.springer.com/10.1007/978-3-030-58574-7_27. Series Title : Lecture Notes in Computer Science.

Song Han, Huizi Mao, and William J. Dally. Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv :1510.00149 [cs]*, February 2016. URL http://arxiv.org/abs/1510.00149. arXiv : 1510.00149.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv :1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv : 1512.03385.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks, July 2016. URL http://arxiv.org/abs/1603.05027. arXiv :1603.05027 [cs].

Suzana Herculano-Houzel. The human brain in numbers : A linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3, 2009. URL https://api.semanticscholar.org/CorpusID:12920763.

Geoffrey Hinton. Neural Networks for Machine Learning, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Andrew Howard, Mark Sandler, Grace Chu, and al. Searching for MobileNetV3, November 2019. URL http://arxiv.org/abs/1905.02244. arXiv :1905.02244 [cs].

Andrew G. Howard, Menglong Zhu, Bo Chen, and al. MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv :1704.04861 [cs]*, April 2017. URL http://arxiv.org/abs/1704.04861. arXiv : 1704.04861.

Jie Hu, Li Shen, Samuel Albanie, and al. Squeeze-and-excitation networks, 2017. URL https://api.semanticscholar.org/CorpusID:260448641.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv :1608.06993 [cs]*, January 2018. URL http://arxiv.org/abs/1608.06993. arXiv : 1608.06993.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9) :1098–1101, 1952. doi : 10.1109/JRPROC.1952.273898.

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán. Low-rank compression of neural nets : Learning the rank of each layer. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8046–8056, 2020. doi : 10.1109/CVPR42600.2020.00807.

Qing Jin, Linjie Yang, and Zhenyu Liao. Towards Efficient Training for Neural Network Quantization. *arXiv :1912.10207 [cs]*, December 2019. URL http://arxiv.org/abs/1912.10207. arXiv : 1912.10207.

Norman P. Jouppi, Cliff Young, Nishant Patil, and al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, Toronto ON Canada, June 2017. ACM. ISBN 978-1-4503-4892-8. doi : 10.1145/3079856.3080246. URL https://dl.acm.org/doi/10.1145/3079856.3080246.

Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1) :117–128, 2011. doi : 10.1109/TPAMI.2010.57.

Mikhail Khodak, Neil A. Tenenholtz, Lester W. Mackey, and Nicoló Fusi. Initialization and regularization of factorized neural layers, 2021. URL https://api.semanticscholar.org/CorpusID:233481638.

Diederik P. Kingma and Jimmy Ba. Adam : A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv :1412.6980 [cs].

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference : A whitepaper. *arXiv :1806.08342 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1806.08342. arXiv : 1806.08342.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6) :84–90, May 2017. ISSN 0001-0782, 1557-7317. doi : 10.1145/3065386. URL https://dl.acm.org/doi/10.1145/3065386.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. pages 950–957, 1991. URL http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.

Andrey Kuzmin, Mart van Baalen, Yuwei Ren, and al. Fp8 quantization : The power of the exponent, 2022. URL https://api.semanticscholar.org/CorpusID:251710272.

G. G. Langdon. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2) :135–149, 1984. doi : 10.1147/rd.282.0135.

Y. LeCun, B. Boser, J. S. Denker, and al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4) :541–551, 1989. doi : 10.1162/neco.1989.1.4.541.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, November 1998. ISSN 00189219. doi : 10.1109/5.726791. URL http://ieeexplore.ieee.org/document/726791/.

Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets : A deeper understanding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5813–5823, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Yuhang Li, Xin Dong, and Wei Wang. Additive Powers-of-Two Quantization : An Efficient Non-uniform Discretization for Neural Networks. *arXiv :1909.13144*

[cs, stat], February 2020. URL http://arxiv.org/abs/1909.13144. arXiv : 1909.13144.

Yuhang Li, Ruihao Gong, Xu Tan, and al. BRECQ : Pushing the Limit of Post-Training Quantization by Block Reconstruction. *arXiv :2102.05426 [cs]*, February 2021. URL http://arxiv.org/abs/2102.05426. arXiv : 2102.05426.

Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. 2022.

Yiheng Liu, Tianle Han, Siyuan Ma, and al. Summary of ChatGPT-Related Research and Perspective Towards the Future of Large Language Models. *Meta-Radiology*, 1(2) :100017, September 2023. ISSN 29501628. doi : 10.1016/j.metrad.2023.100017. URL http://arxiv.org/abs/2304.01852. arXiv :2304.01852 [cs].

Z. Liu, K. Cheng, D. Huang, E. Xing, and Z. Shen. Nonuniform-to-uniform quantization : Towards accurate quantization via generalized straight-through estimation, jun 2022a. URL https://doi.ieeecomputersociety.org/10.1109/CVPR52688.2022.00489.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, and al. A ConvNet for the 2020s, March 2022b. URL http://arxiv.org/abs/2201.03545. arXiv :2201.03545 [cs].

Ilya Loshchilov and Frank Hutter. SGDR : Stochastic Gradient Descent with Warm Restarts, May 2017. URL http://arxiv.org/abs/1608.03983. arXiv :1608.03983 [cs, math].

MATLAB. Introduction to deep learning with matlab, 2021. URL https://fr.mathworks.com/campaigns/offers/deep-learning-with-matlab.html.

Jeffrey L. McKinstry, Steven K. Esser, Rathinakumar Appuswamy, and al. Discovering Low-Precision Networks Close to Full-Precision Networks for Efficient Embedded Inference. *arXiv :1809.04191 [cs]*, February 2019. URL http://arxiv.org/abs/1809.04191. arXiv : 1809.04191.

Clément Metz, Thibault Allenet, Johannes Thiele, and al. Lattice quantization. In *2023 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–2, 2023. doi : 10.23919/DATE56975.2023.10137188.

Daniele Micciancio. On the hardness of the shortest vector problem. 1998. URL https://api.semanticscholar.org/CorpusID:17010293.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning, 2019.

Rafael Müller, Simon Kornblith, and Geoffrey Hinton. *When Does Label Smoothing Help ?* Curran Associates Inc., Red Hook, NY, USA, 2019.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-Free Quantization Through Weight Equalization and Bias Correction. *arXiv :1906.04721 [cs, stat]*, November 2019. URL http://arxiv.org/abs/1906.04721. arXiv : 1906.04721.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, and al. Up or down ? adaptive rounding for post-training quantization. *ArXiv*, abs/2004.10568, 2020. URL https://api.semanticscholar.org/CorpusID:216056295.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, and al. A white paper on neural network quantization, 2021. URL https://api.semanticscholar.org/CorpusID:235435934.

Deniz Oktay, Johannes Ballé, Saurabh Singh, and Abhinav Shrivastava. Scalable model compression by entropy penalized reparameterization, 2020. URL https://openreview.net/forum?id=HkgxW0EYDS.

OpenAI. Gpt-4 technical report, 2023.

Adam Paszke, Sam Gross, Francisco Massa, and al. PyTorch : An Imperative Style, High-Performance Deep Learning Library, December 2019. URL http://arxiv.org/abs/1912.01703. arXiv :1912.01703 [cs, stat].

Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017. URL https://api.semanticscholar.org/CorpusID:12219403.

Patrick Plagwitz, Frank Hannig, Jürgen Teich, and Oliver Keszocze. To spike or not to spike ? a quantitative comparison of snn and cnn fpga implementations, 2023.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net : Imagenet classification using binary convolutional neural networks, 2016. URL https://doi.org/10.1007/978-3-319-46493-0_32.

Sebastian Ruder. An overview of gradient descent optimization algorithms, June 2017. URL http://arxiv.org/abs/1609.04747. arXiv :1609.04747 [cs].

David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https ://doi.org/10.1038/323533a0. *Nature*, (323) :533–536, 1986.

Olga Russakovsky, Jia Deng, Hao Su, and al. ImageNet Large Scale Visual Recognition Challenge. *arXiv :1409.0575 [cs]*, January 2015. URL http://arxiv.org/abs/1409.0575. arXiv : 1409.0575.

Mark Sandler, Andrew Howard, Menglong Zhu, and al. MobileNetV2 : Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer*

Vision and Pattern Recognition, pages 4510–4520, Salt Lake City, UT, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi : 10.1109/CVPR.2018.00474. URL https://ieeexplore.ieee.org/document/8578572/.

Jaime Sevilla, Lennart Heim, Anson Ho, and al. Compute Trends Across Three Eras of Machine Learning, March 2022. URL http://arxiv.org/abs/2202.05924. arXiv :2202.05924 [cs].

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3) :379–423, 1948. doi : 10.1002/j.1538-7305.1948.tb01338.x.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv :1409.1556 [cs]*, April 2015. URL http://arxiv.org/abs/1409.1556. arXiv : 1409.1556.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56) :1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Pierre Stock, Armand Joulin, Rémi Gribonval, and al. And the Bit Goes Down : Revisiting the Quantization of Neural Networks. In *ICLR 2020 - Eighth International Conference on Learning Representations*, pages 1–11, Addis-Abeba, Ethiopia, April 2020. URL https://hal.archives-ouvertes.fr/hal-02434572.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace : Closing the Gap to Human-Level Performance in Face Verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, Columbus, OH, USA, June 2014. IEEE. ISBN 978-1-4799-5118-5. doi : 10.1109/CVPR.2014.220. URL https://ieeexplore.ieee.org/document/6909616.

Mingxing Tan and Quoc V. Le. EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks. *arXiv :1905.11946 [cs, stat]*, September 2020. URL http://arxiv.org/abs/1905.11946. arXiv : 1905.11946.

Enzo Tartaglione, Stéphane Lathuilière, Attilio Fiandrotti, and al. Hemp : High-order entropy minimization for neural network compression. *Neurocomput.*, 461(C) :244–253, oct 2021. ISSN 0925-2312. doi : 10.1016/j.neucom.2021.07.022. URL https://doi.org/10.1016/j.neucom.2021.07.022.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, and al. Llama : Open and efficient foundation language models, 2023. URL https://api.semanticscholar.org/CorpusID:257219404.

UE. RGPD, 2016. URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02016R0679-20160504&qid=1532348683434.

Ashish Vaswani, Noam Shazeer, Niki Parmar, and al. Attention Is All You Need, August 2023. URL http://arxiv.org/abs/1706.03762. arXiv :1706.03762 [cs].

Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng. Towards accurate post-training network quantization via bit-split and stitching. 119 : 9847–9856, 13–18 Jul 2020a. URL https://proceedings.mlr.press/v119/wang20c.html.

Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, and Song Han. Apq : Joint search for network architecture, pruning and quantization policy, 2020b. URL https://api.semanticscholar.org/CorpusID:219687796.

Ying Wang, Yadong Lu, and Tijmen Blankevoort. Differentiable joint pruning and quantization for hardware efficiency, 2020c. URL https://api.semanticscholar.org/CorpusID:220665499.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression : Pushing the limit of low-bit transformer language models, 2022. URL https://api.semanticscholar.org/CorpusID:252545187.

Simon Wiedemann, Heiner Kirchhoffer, Stefan Matlage, and al. Deepcabac : A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 14(4) :700–714, 2020. doi : 10.1109/JSTSP.2020.2969554.

Wikipedia. Wikipedia - ImageNet, 2018. URL https://en.wikipedia.org/wiki/ImageNet.

Dengyu Wu, Xinping Yi, and Xiaowei Huang. A little energy goes a long way : Build an energy-efficient, accurate spiking neural network from convolutional neural network. *Frontiers in Neuroscience*, 16, May 2022. ISSN 1662-453X. doi : 10.3389/fnins.2022.759900. URL http://dx.doi.org/10.3389/fnins.2022.759900.

Guangxuan Xiao, Ji Lin, Mickael Seznec, and al. SmoothQuant : Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, and al., editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/xiao23c.html.

Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. Automatic neural network compression by sparsity-quantization joint learning : A constrained optimization-based approach, 2019. URL https://api.semanticscholar.org/CorpusID:208857831.

Huanrui Yang, Lin Duan, Yiran Chen, and Hai Li. Bsq : Exploring bit-level sparsity for mixed-precision neural network quantization, 2021. URL https://api.semanticscholar.org/CorpusID:231985703.

Linjie Yang and Qing Jin. Fracbits : Mixed precision quantization via fractional bit-widths, 2020. URL https://arxiv.org/abs/2007.02017.

Zhaoyang Zhang, Wenqi Shao, Jinwei Gu, Xiaogang Wang, and Ping Luo. Differentiable dynamic quantization with mixed precision and adaptive resolution, 2021. URL http://proceedings.mlr.press/v139/zhang21r.html.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Christopher De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. *CoRR*, abs/1901.09504, 2019. URL http://arxiv.org/abs/1901.09504.

Shuchang Zhou, Yuxin Wu, Zekun Ni, and al. DoReFa-Net : Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv :1606.06160 [cs]*, February 2018. URL http://arxiv.org/abs/1606.06160. arXiv : 1606.06160.

Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian D. Reid. Towards effective low-bitwidth convolutional neural networks, 2017. URL http://arxiv.org/abs/1711.00205.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3) :337–343, 1977. doi : 10.1109/TIT.1977.1055714.

Remerciements

Je tiens à remercier toutes les personnes et les institutions ayant contribué au bon déroulement de ma thèse. Ces travaux se sont déroulés au Laboratoire d'Intelligence Artificielle Embarquée (LIAE), au CEA, sous la supervision du Pr. Antoine Dupret (CEA - D-OPT), du Dr. Olivier Bichler (CEA - List) et du Dr. Johannes Thiele (Axelera.ai). Ils ont été financés par le Département Systèmes et Circuits Intégrés Numériques (DSCIN), et se sont tenus en partenariat avec l'Université Paris-Saclay.

En premier lieu, je veux remercier mes encadrants, qui ont su faire preuve d'écoute et de réactivité, et avec qui il a été agréable d'apprendre et d'échanger durant ces trois années. Je remercie également l'ensemble des membres du LIAE, mes collègues doctorants, ex-doctorants, stagiaires et permanents du laboratoire, à qui je dois bon nombre de discussions enrichissantes. Je remercie le Dr. Thibault Allenet, doctorant à mon arrivée au laboratoire, qui a contribué à mes premiers travaux de recherche et qui a exercé un précieux rôle de mentor à mon égard tout au long de ma thèse.

Sur un plan plus personnel, je remercie mes parents, qui m'ont supporté durant ces trois ans.

Je dédie cette thèse à ma grand-mère, Madame Jeanne Muller.