Compression des réseaux de neurones profonds à base de quantification uniforme et non-uniforme
Compression of Deep Neural Networks based on uniform and non-uniform quantization

Diana Resmerita[1,2] Rodrigo Cabral Farias[1] Benoît Dupont de Dinechin[2] Lionel Fillatre[1]
[1] {diana.resmerita, cabral, lionel.fillatre}@i3s.unice.fr
Université Côte d'Azur, I3S, France
[2] {diana.resmerita, benoit.dinechin}@kalray.eu
Kalray, France

Résumé

Les réseaux de neurones convolutifs sont utilisés pour les voitures autonomes, les caméras intelligentes et les smartphones en raison de leurs performances impressionnantes. Cependant, le déploiement de ces modèles sur des systèmes embarqués présente des limites. Les couches convolutionnelles et les couches fortement connectées ont des millions de paramètres et sont coûteuses en calcul. Il est donc impératif de compresser ces couches. De nombreux travaux ont proposé des techniques de compression telles que, par exemple, l'élagage, la quantification et le calcul matriciel optimisé. Cet article présente l'état de l'art dans la compression des réseaux de neurones profonds. Il propose également une étude numérique sur l'efficacité de la quantification uniforme et non-uniforme afin de réduire la taille des réseaux de neurones.

Mots Clef

Réseau de neurones profond, Compression, Quantification.

Abstract

Convolutional neural networks are used for autonomous cars, smart cameras and smartphones due to their impressive performance. However, there are limitations when deploying these models on embedded systems. Convolutional and Fully Connected layers have milions of parameters and are computationally expensive. These layers must be compressed. Several works have considered compression techniques such as pruning, quantization and efficient matrix multiplications for example. This paper summarizes recent developments in this field. It also proposes a numerical study on uniform and non-uniform quantization applied to deep neural networks to reduce their size.

Keywords

Deep Neural Network, Compression, Quantization.

1 Introduction

Les réseaux de neurones convolutifs (CNNs) sont reconnus comme étant la référence pour la classification, la détection et la segmentation des images. Les architectures des réseaux sont de plus en plus complexes et cela provoque une augmentation de la taille mémoire et de la complexité de calcul. De nos jours, les CNNs sont très demandés dans des applications embarquées telles que la détection d'objets pour les voitures autonomes et la vidéosurveillance. Les systèmes embarqués ont généralement des capacités de calcul plutôt faibles et sont limités en mémoire et en consommation d'énergie. Il est indispensable de réduire les coûts de communication, d'économiser de l'énergie et d'assurer un temps de réponse court, lorsque le CNN est utilisé pour classifier de nouvelles données.

Il existe un grand nombre de recherches sur le sujet et plusieurs méthodes ont été proposées dans la littérature et dans l'industrie. Des entreprises comme Google [1], Nvidia [2], Intel [3] et Facebook [4] ont obtenu une inférence plus rapide grâce à des calculs efficaces et une précision réduite. D'autres méthodes utilisées sont la quantification, l'élagage des réseaux, mais aussi la construction de nouveaux réseaux plus efficaces. Mais comment peut-on choisir la meilleure méthode à appliquer pour réduire la complexité d'un réseau de neurones sans perte en performance ? Comment savoir si un réseau a été bien compressé ? En particulier, quels seraient des critères d'évaluation appropriés ?

Cet article propose une comparaison entre des méthodes de quantification uniforme et non-uniforme appliquées aux paramètres d'un réseau de neurones (préalablement entraînés). On souhaite mettre en évidence quelle méthode apporte les meilleurs résultats et évaluer les différentes stratégies de compression grâce à plusieurs critères.

Ce document est organisé de la manière suivante. La section 2 présente les contraintes d'un système embarqué, des méthodes de compression de l'état de l'art issues de la littérature et les méthodes implémentées dans le milieu industriel. La section 3 introduit les réseaux de neurones profonds et leur compression. La section 4 présente les mé-

1. Google AI. https ://ai.google/tools/
2. The nvidia deep learning accelerator. http ://nvdla.org/
3. Intel(R) Math Kernel Library for Deep Neural Networks. https ://intel.github.io/mkl-dnn/index.html
4. Facebook AI Research. https ://code.fb.com/ai-research/

thodes de quantification utilisées. La section 5 décrit les expérimentations numériques effectuées. On utilise un petit modèle CNN, entraîné sur la base d'images MNIST dans le but d'analyser l'impact de l'erreur pour chaque couche, en fonction de plusieurs critères d'évaluation et pour comparer plus facilement les deux méthodes. Enfin, la section 6 conclut l'article.

2 Compression dans le Deep Learning

2.1 Deep Learning pour les systèmes embarqués

L'exécution efficace d'un réseau profond est une étape primordiale pour nombreuses applications envisagées dans le Deep Learning. Cependant, les algorithmes d'inférence conçus pour les réseaux profonds peuvent facilement submerger les ressources des systèmes embarqués. Si on exécute un réseau complexe, la plate-forme doit surmonter plusieurs défis, notamment la mémoire limitée, la puissance de calcul limitée et un temps d'inférence long. Par exemple, AlexNet [1] a 61M paramètres et il nécessite une taille mémoire de 249MB avec 724M opérations de multiplier-accumuler (MACs). Il existe des réseaux profonds comme VGG16 [2] où la taille mémoire et la charge de calcul sont encore plus élevées. La Table 1 contient des informations sur les réseaux complexes. Ces grands

TABLE 1 – Taille mémoire des réseaux.

Réseau	VGG16 [2]	ResNet152 [3]	GoogleNet [4]
Taille	552MB	232MB	53MB
Paramètres	6138M	60M	6,99M
MACs	15,5G	11,3G	1,58G

modèles sont très puissants mais consomment beaucoup d'énergie, car ils doivent être stockés dans la mémoire DRAM et récupérés à chaque utilisation. Le coût énergétique est dominé principalement par les accès mémoire. La Table 2 montre les coûts d'énergétiques des opérations arithmétiques et mémoire dans un processus 45nm CMOS [5]. On veut réduire la taille des réseaux pour pouvoir mettre le modèle dans la SRAM et éviter les accès mémoire trop coûteux. Mais, réduire la taille mémoire n'est pas suffisant pour un déploiement efficace sur des systèmes embarqués. Les CNNs sont principalement composés de couches de convolutions (CONV) et de couches fortement connectées (FC) qui représentent plus de 90% de la complexité de calcul et de la mémoire d'un CNN entier.

TABLE 2 – Consommation d'énergie pour des opérations 32 bits (Han et al. [5])

Opération	int ADD	float ADD	int MULT	float MULT	32KB SRAM	DRAM
Énergie [pJ]	0,1	0,9	3,1	3,7	5	640
Coût relatif	1	9	31	37	50	6400

Ceci motive l'apparition des réseaux légers avec un bon compromis latence-précision. MobileNet [6] a 4,24M paramètres et il nécessite une taille mémoire de 16,9MB avec 569M MACs. D'autres progrès ont été faits dans de nombreux domaines dans le but d'exécuter efficacement les réseaux des neurones.

2.2 État de l'art

Cette section présente les principales méthodes de compression qui ont été proposées dans la littérature pour réduire la complexité calculatoire et/ou la taille des réseaux de neurones.

Accélération des calculs. Afin d'accélérer l'exécution des réseaux CNNs, des algorithmes de multiplications matricielles efficaces ont été proposés. Une méthode courante pour traiter les couches consiste à utiliser GEMM [7], une procédure de multiplication de matrices qui fait partie de la bibliothéque BLAS [8]. La plupart des convolutions sont calculées avec l'argorithme FFT [9,10]. Cependant, l'algorithme de Winograd [11, 12] est performant pour les tailles de convolution plus petites qui sont plus usuelles. Les décompositions tensorielles telles que Tucker [13] et CP [14] sont aussi des outils pratiques pour accélérer l'inférence avec les réseaux ayant beaucoup de paramètres.

Élagage des réseaux. L'élagage, appelé *pruning* en anglais, est une technique qui sert à enlever les paramètres redondants. Les poids élagués occupent moins de mémoire et les multiplications relatives à ces poids sont omises. Comme présenté par Cheng et al. [15], il existe 5 techniques de pruning : *fine-grained*, *vector-level*, *kernel-level*, *group-level* et *filter-level*. Han *et al.* [16] propose une compression en 4 étapes avec élagage, quantification, encodage par Huffman et ré-entraînement. Il obtient un réseau AlexNet [1] compressé jusqu'à 35x sans sacrifier la performance. Anwar *et al.* [17] applique un élagage structuré sur 3 niveaux (filtre, kernel et intra-kernel) et réduit beaucoup la complexité des calculs des convolutions. Mao *et al.* [18] trouve que l'élagage vectoriel a des meilleures performances car il prend moins de place que l'élagage *fine-grained*.

Quantification classique. Plusieurs approches ont été publiées sur la quantification des réseaux CNNs qui traitent le problème de mémoire et celui de la rapidité de calculs. On sait que les réseaux de neurones contiennent beaucoup de redondances dans les poids (Denil *et al.* [19]). Les méthodes classiques utilisées dans la compression des signaux, comme la quantification scalaire et vectorielle, peuvent obtenir un ratio de compression très grand sans perte en *accuracy*. Ces méthodes utilisent des dictionnaires ou des *look-up tables*. Gong *et al.* [20] applique l'algorithme k-means et la quantification du produit aux couches FC et obtient un taux de compression de 16-24 pour une petite perte. Les valeurs quantifiées peuvent être encodées sans perte par des algorithmes comme Huffman. Han *et al.* [16] montre que l'étape d'encodage réduit le stockage du réseau de 20%-30%.

Quantification en faible précision. L'approche la plus étudiée actuellement est la quantification en faible précision. Dans de nombreux cas, les réseaux neuronaux sont très tolérants à des faibles précisions numériques. Un mo-

dèle à virgule flottante peut être quantifié en un modèle à virgule fixe avec presque aucune perte de précision. Gupta *et al.* [21] utilise une représentation à 16 bits en virgule fixe avec l'arrondi stochastique pour entraîner un réseau CNN. Courbariaux *et al.* [22] a des bons résultats après un entraînement avec 10 bits pour les activations et 12-bits pour les paramètres. Une telle approche apporte plusieurs avantages : l'empreinte mémoire est plus petite (on réduit la taille des données), le transfert est plus rapide et on a besoin de moins de mémoire vive et *cache* pour les activations. Par conséquent, la consommation énergétique est réduite. En théorie, si on passe de 32-bits à 8-bits, on est 4x plus efficace. Par contre, après chaque étape de calculs, les nouvelles données doivent aussi être compressées au format faible précision. Cette étape ajoute une complexité de calcul en plus.
Récemment, des nouveaux formats de données ont été introduits grâce à leur efficacité au niveau matériel. Le format *Bfloat16* [5] est une représentation en virgule flottante sur 32 bits, tronqué en 16 bits. Il conserve les caractéristiques d'un float32, mais ne prend en charge qu'une mantisse de 7 bits. Bfloat16 est plus rapide que float16 et la précision est suffisante pour les CNNs. Un autre type de données appelé *posit* [23] est conçu pour remplacer les floats. Les posits offrent une précision plus grande tout en étant plus simples au niveau matériel, donc plus économique en consommation énergétique.

Reconstruction des réseaux. Une autre approche correspond à la reconstruction des modèles les plus efficaces. Les réseaux binaires et ternaires ont des données à 1-2 bits et éliminent les multiplications. Plusieurs implémentations existent. Hubara *et al.* [24] construit le réseau BNN en remplaçant la convolution par XNOR et *popcount* pour avoir un réseau binaire. Rastegari *et al.* [25] utilise le modèle proposé par Hubara avec des facteurs d'échelle pour les valeurs, sauf pour la première et dernière couche qui restent en float32. Le réseau QNN est également proposé par Hubara [26]. Il utilise la même architecture que le BNN, mais avec des activations à 2 bits. Zhu *et al.* [27] propose un modèle ternaire qui adapte les facteurs d'échelle pour chaque couche. Malgré leur taux de compression élevé et la réduction de la complexité de calcul, il y a une baisse en précision importante. Par conséquent, l'utilisation d'un tel modèle ne semble pas assez fiable pour les applications industrielles.

2.3 Solutions industrielles

Le besoin de la compression dans l'apprentissage profond devient de plus en plus important. Les industriels contribuent significativement aux avancées dans le domaine. Une inférence plus rapide grâce à des calculs efficaces est obtenue par Gemmlowp [6], Nvidia TensorRT [7] et Rosetta [28]. Gemmlowp est une bibliothèque de multiplication de matrices en faible précision qui est utilisée par Tensorflow [8] et Tensorflow Lite [9]. La compression des poids et des entrées est faite en virgule fixe à 8 bits non-signé (uint8). La méthode utilisée est la quantification uniforme en assurant que la valeur en zéro est quantifiée sans erreur. Krishnamoorthi *et al.* [29] montre que, pour un réseau quantifié en 8 bits, la taille de stockage est réduite d'un facteur 4, l'inférence est 2-3x plus rapide sur CPU et 10x plus rapide sur des processeurs avec des capacités SIMD à virgule fixe. TensorRT utilise aussi une inférence en 8 bits réalisée à l'aide d'une quantification uniforme pour passer d'une représentation en virgule flottante à 32 bits (float32) à une représentation en virgule fixe à 8 bits signé (int8). Le biais est ignoré et les activations sont saturées à un seuil défini au moyen d'un ensemble de données de calibration. Rosetta prend inspiration de Jacob *et al.* [30] et quantifie les poids et les activations de float32 à uint8. L'approche est identique à celle de Gemmlowp. Cependant, la quantification ne s'applique pas à toutes les couches pour pouvoir améliorer la précision de la prédiction. Un jeu de données de calibration est utilisé pour définir un seuil de saturation des entrées en minimisant la norme L2 de l'erreur, contrairement à TensorRT qui minimise l'entropie relative. Caffe [31], un autre framework très connu, met en place un outil d'approximation automatique, Ristretto, pour compresser les données de float32 à n'importe quel format de données. Ristretto utilise 3 stratégies de quantification : point fixe dynamique (Courbariaux *et al.* [22]), minifloat et puissance de deux (Tang *et al.* [32]). Gysel *et al.* [33] montre que le point fixe dynamique convient le mieux. Cette approximation donne le meilleur résultat pour des faibles précisions, mais elle nécessite plus de surface de puce que le point fixe classique.
Cet article s'intéresse principalement aux techniques de compression pour réduire le stockage des paramètres d'un CNN. La rapidité des calculs nous préoccupe moins.

3 Réseau de neurones profonds

3.1 Architecture formelle

Un réseau de neurones pour la classification de C classes est une fonction complexe $h : \mathbb{R}^{n_0} \rightarrow \mathrm{S}_C$ composée de neurones formels organisés en couches. L'ensemble

$$\mathrm{S}_C = \{\mathbf{p} \in \mathbb{R}^C \mid p_i \geq 0, \sum_{i=1}^{C} p_i = 1\} \qquad (1)$$

désigne le C-simplexe. Autrement dit, étant donné un vecteur $\mathbf{x} = (x_1, ..., x_{n_0})$ de dimension n_0, un réseau de neurones génère une distribution discrète $\mathbf{p}(\mathbf{x}) = (p_1(\mathbf{x}), \ldots, p_C(\mathbf{x})) \in \mathrm{S}_C$ tel que $p_i = p_i(\mathbf{x})$ est la probabilité que $\mathbf{x}$ appartiennent à la classe $i \in \{1, \ldots, C\}$.

5. Google Using bfloat16 with TensorFlow models https ://cloud.google.com/tpu/docs/bfloat16
6. Gemmlowp : a small self-contained low-precision GEMM library. https://github.com/google/gemmlowp
7. 8 bit inference with TensorRT Nvidia. http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inferencewith-tensorrt.pdf
8. Tensorflow quantization library. https://www.tensorflow.org/api_docs/python/tf/quantization
9. Tensorflow Lite quantization solution. https://www.tensorflow.org/lite/performance/post_training_quantization

La structure d'un réseau est généralement divisée en 3 parties : couche d'entrée, couche cachée et couche de sortie. Une couche est un ensemble de neurones n'ayant pas de connexion entre eux. On note h^k la couche k d'un réseau. Pour un réseau avec L couches cachées, h^0 est la couche d'entrée, h^{L+1} est la couche de sortie et h^k avec $1 \leq k \leq L$ représente les couches cachées. L'entrée de chaque couche est pondérée par une matrice de poids W_k et un biais $\mathbf{b}_k$. Dans chaque couche, une fonction non-linéaire σ_k, appelée fonction d'activation, est appliquée à chaque combinaison pondérée. Les différentes couches sont formalisées de la façon suivante :

$$\mathbf{z}_0 = h^0(\mathbf{x}) = \mathbf{x}, \tag{2}$$

$$\mathbf{z}_k = h^k(\mathbf{x}) = \sigma_k(W_k h^{k-1}(\mathbf{x}) + \mathbf{b}_k), \tag{3}$$

$$\mathbf{z}_{L+1} = h^{L+1}(\mathbf{x}) = \sigma_{L+1}(W_{L+1} h^L(\mathbf{x}) + \mathbf{b}_{L+1}), \tag{4}$$

$$W_k \in \mathbb{R}^{n_k \times n_{k-1}},\ \mathbf{b}_k \in \mathbb{R}^{n_k},$$

$$W_{L+1} \in \mathbb{R}^{C \times n_L},\ \mathbf{b}_{L+1} \in \mathbb{R}^C. \tag{5}$$

Généralement, la couche de sortie utilise une fonction d'activation différente des couches cachées. Dans le cas d'une classification avec C classes, la fonction d'activation utilisée s'appelle *softmax* et elle est définie par

$$\text{softmax}(\mathbf{u}) = \frac{1}{\sum_{i=1}^{C} e^{u_i}} \left(e^{u_1}, \ldots, e^{u_C}\right),\ \forall \mathbf{u} \in \mathbb{R}^C. \tag{6}$$

La sortie du réseau est notée $\mathbf{p}(\mathbf{x}) = h_\theta(\mathbf{x})$ où la notation h_θ souligne que le réseau dépend du vecteur de paramètres

$$\theta = (\theta_1, ..., \theta_{L+1}),\ \text{où } \theta_k = (W_k, \mathbf{b}_k), \tag{7}$$

composé de $|\theta| = \sum_{k=1}^{L+1}(n_k n_{k-1} + n_k)$ paramètres réels. Les paramètres θ sont estimés dans la phase d'apprentissage sur un ensemble d'échantillons. On dispose d'une base d'apprentissage $(X, \mathbf{y})$ composée de N échantillons $\mathbf{x}^{(i)}$ et de N vecteurs d'étiquettes $\mathbf{y}^{(i)}$ tel que $y_c^{(i)} = 1$ si $\mathbf{x}^{(i)}$ est associée à la classe c (les autres composantes de $\mathbf{y}^{(i)}$ sont nulles). Dans le cas d'une classification, l'apprentissage modifie θ par minimisation d'une fonction de perte, l'entropie croisée, définie par

$$\sum_{i=1}^{N} H\left(\mathbf{y}^{(i)}, \mathbf{p}(\mathbf{x}^{(i)})\right) = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} \log_2 p_c(\mathbf{x}^{(i)}), \tag{8}$$

qui va indirectement maximiser la justesse du réseau. La justesse, aussi appelée *accuracy* (ACC), est définie par

$$ACC(h_\theta, X, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left\{s\left(h_\theta(\mathbf{x}^{(i)})\right) = s\left(\mathbf{y}^{(i)}\right)\right\}, \tag{9}$$

où $s(\mathbf{z})$ donne l'indice de la classe avec la plus forte probabilité dans le vecteur $\mathbf{z} \in \mathrm{S}_C$:

$$s(\mathbf{z}) = \underset{1 \leq i \leq C}{\arg\max}\, z_i. \tag{10}$$

La justesse est également évaluée sur des bases de test afin de vérifier l'erreur de généralisation du réseau.
En ce qui concerne les réseaux convolutifs (CNNs), on est intéressé par les couches de convolution (CONV) et les couches complètement connectée, en anglais appelée *fully connected* (FC). La couche FC, une couche classique du perceptron (Eq. 3), est utilisée à la fin d'un réseau pour lier les caractéristiques de l'image. La couche de convolution, donne les caractéristiques de l'image. Le résultat de la convolution d'une image I (ou une couche) de taille $H \times W$ à D canaux qui est traitée avec D' filtres est une image à deux dimensions (i, j) donnée par

$$(K_l * I)(i,j) = \sum_{d=1}^{D}\sum_{m=1}^{H'}\sum_{n=1}^{W'} K_l(m,n,d) I(i+n, j+m, d),$$

pour chaque noyau de convolution K_l de taille $H' \times W' \times D$. Les indices dans la convolution sont supposés vérifier les conditions aux bords. Un canal peut correspondre à un canal couleur ou bien à la sortie d'un filtre donné.

3.2 Généralités sur la compression

La compression a pour but de réduire l'espace nécessaire à la représentation d'une certaine quantité d'information. Une chaîne de compression est, généralement, formée de 3 étapes principales : transformation, quantification, codage et il peut y avoir aussi une étape de pré-traitement pour rendre la compression plus efficace.
L'intérêt d'appliquer un algorithme de compression à un réseau de neurones est de pouvoir reproduire au mieux ses qualités de classification avec un réseau moins coûteux en temps, en espace mémoire et en consommation énergétique. Plus précisément, il s'agit d'approcher la fonction h_θ par une fonction $h_{\hat{\theta}}$ où $\hat{\theta} = F(\theta)$ est une forme compressée de θ. La fonction de compression $F : \mathbb{R}^{|\theta|} \to \mathcal{A}$, où $\mathcal{A}$ est un ensemble discret fini, permet de réduire la taille de représentation du vecteur θ. Soit $\mathbf{p}(\mathbf{x}) = h_\theta(\mathbf{x})$, respectivement $\hat{\mathbf{p}}(\mathbf{x}) = h_{\hat{\theta}}(\mathbf{x})$, la sortie du réseau initial, resp. du réseau compressé. La divergence de Kullback-Leibler (*KL*), aussi connue comme l'entropie relative, mesure la dissimilarité entre les deux lois de probabilités $\mathbf{p} = \mathbf{p}(\mathbf{x})$ et $\hat{\mathbf{p}} = \hat{\mathbf{p}}(\mathbf{x})$:

$$\mathrm{KL}(\mathbf{p}, \hat{\mathbf{p}}) = \sum_{c=1}^{C} p_c \log\left(\frac{p_c}{\hat{p}_c}\right). \tag{11}$$

La section suivante présente deux méthodes de quantification scalaire qui vont servir à quantifier θ.

4 Méthodes de quantification

Il existe des quantificateurs scalaires et vectoriels. Un quantificateur scalaire traite chaque symbole d'entrée séparément, tandis que le quantificateur vectoriel traite un vecteur de données pour obtenir la sortie. Dans la suite, on va utiliser la quantification scalaire, car sa mise en œuvre est peu complexe.

Soit $w \in \theta$ un paramètre d'un réseau de neurones entraîné. On défini Q un quantificateur scalaire de taille $R = 2^b$, où b est le nombre de bits, qui amène $w \in P = [\theta_{min}, \theta_{max}] \subset \mathbb{R}$ sur un ensemble discret fini $\mathcal{C} = \{c_1, c_2, ..., c_R\} \subset \mathbb{R}$. Le quantificateur partitionne l'intervalle P en plusieurs sous-intervalles $P_i = [t_i, t_{i+1}[$, pour $1 \leq i \leq R$, tels que

$$\hat{w} = Q(w) = c_i \text{ ssi } w \in [t_i, t_{i+1}[. \tag{12}$$

Le pas de quantification $\Delta_i = t_{i+1} - t_i$ correspond à la largeur de l'intervalle P_i. La quantification introduit une erreur, appelée distorsion, définie par :

$$\varepsilon = Q(w) - w. \tag{13}$$

Une estimation objective de la qualité de la quantification est donnée par l'erreur quadratique moyenne, *Mean Squared Error* en anglais (MSE), entre θ et $\hat{\theta}$:

$$\text{MSE}(\theta, \hat{\theta}) = \frac{1}{|\theta|} \sum_{w \in \theta} (w - Q(w))^2. \tag{14}$$

Afin d'exploiter la redondance produite par la quantification, il est nécessaire de passer par une étape d'indexation des valeurs quantifiées, ce qui peut nécessiter l'utilisation d'un tableau de correspondance.

4.1 Quantification uniforme

Le quantificateur scalaire le plus connu est le quantificateur uniforme. Un quantificateur uniforme est un quantificateur régulier pour lequel les pas Δ_i sont constants et les niveaux de quantification c_i sont les points centraux des intervalles quantifiés. Pour quantifier les paramètres d'une couche k avec b bits, c'est-à-dire pour quantifier θ_k, on réalise les étapes de calculs suivantes :

$$\Delta = \frac{\theta_{max} - \theta_{min}}{2^b},\ i_w = \left\lfloor \frac{w}{\Delta} \right\rfloor, \tag{15}$$

$$\widehat{w} = Q(w) = \Delta i_w + \frac{\Delta}{2},\ \forall w \in \theta_k. \tag{16}$$

L'avantage de la quantification scalaire uniforme est qu'elle ne requiert qu'une multiplication. Si on connaît i_w, l'index de la valeur w, on peut en déduire facilement la valeur quantifiée avec (16). Une telle méthode est adaptée aux matériels informatiques qui ne peuvent pas accéder en parallèle à des tableaux de correspondance. La quantification uniforme n'est a priori optimale que pour une distribution de données uniforme, ce qui n'est pas le cas des paramètres d'un réseau de neurones. Cependant, les résultats dans la littérature montrent que les réseaux de neurones sont robustes aux erreurs d'approximation [34], ce qui est confirmée par la section 5.

4.2 Quantification non-uniforme

Un algorithme standard de quantification non-uniforme est l'algorithme de Lloyd [35], qui est aussi connu sous le nom de k-means. La méthode de Lloyd a comme but de construire un dictionnaire de quantification afin de minimiser la distorsion (14). Elle s'utilise sur un ensemble d'apprentissage $\mathcal{E} = \{w_1, \ldots, w_M\}$ composé de M points à quantifier où M est un entier plutôt grand. Le calcul est effectué de façon itérative vérifiant successivement 2 conditions d'optimalité. Si on connaît les niveaux de quantification c_i, on applique la condition du plus proche voisin pour calculer la meilleure partition P_i de $\mathcal{E}$. Au contraire, si on connaît les partitions P_i de $\mathcal{E}$, on peut calculer le meilleur niveau de quantification c_i en calculant le centroïde de P_i. Notons (m) et $(m+1)$ les itérations m et $m+1$, et $|P_i^{(m+1)}|$ le cardinal de $P_i^{(m+1)}$ à l'itération $m+1$. Le nombre de partitions R est fixé. L'algorithme est donné par

$$t_i^{(m+1)} = \frac{c_{i+1}^{(m)} + c_i^{(m)}}{2},\ \forall 1 \leq i \leq R, \tag{17}$$

$$c_i^{(m+1)} = \frac{1}{\left|P_i^{(m+1)}\right|} \sum_{w \in P_i^{(m+1)}} w,\ \forall 1 \leq i \leq R. \tag{18}$$

L'intérêt d'une telle quantification est qu'elle s'adapte à la distribution des données. Cependant, l'algorithme passe beaucoup de temps à calculer les distances entre les centroïdes et les autres points. Des variations existent pour accélérer l'algorithme : Hartigan [36], k-means++ [37].

5 Expérimentations numériques

5.1 Cadre expérimental

Pour les expérimentations numériques, on utilise un réseau CNN entraîné sur les données MNIST [38]. MNIST contient des images de chiffres (10 classes) de taille 28x28 pixels en noir et blanc. L'entraînement est fait sur 60×10^3 exemples et le test sur 10×10^3 exemples.

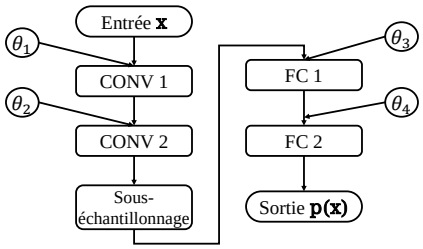

FIGURE 1 – Architecture du réseau CNN.

Pour ces évaluations, on a choisi un réseau simple qui n'a pas autant de paramètres que les réseaux présenté dans la Table 1, mais qui est représentatif des CNNs. La Figure 1 donne un schéma simple de l'architecture du réseau et la Table 3 résume les informations sur les couches. La couche de sous-échantillonage est appelée *MaxPooling* et elle n'apparaît pas dans le tableau car elle ne peut pas être quantifiée. On affiche la taille des entrées et des sorties, le nombre de paramètres de chaque couche et la taille mémoire utilisée. Les variables θ_1, θ_2, θ_3, θ_4 représentent les paramètres et regroupent les poids W_k et les biais b_k de chaque couche tels que définis dans (7). En pratique, chaque paramètre réel de θ est représenté en *float32*.
Pour chaque expérience, on calcule le taux de compression (19)

$$\tau = \frac{\text{Taille initiale}}{\text{Taille après compression}}, \tag{19}$$

TABLE 3 – Paramètres du réseau CNN entraîné sur MNIST.

Couches	Entrée	Sortie	Paramètres	Taille
Conv 1	28x28x1	26x26x32	320	1,28 KB
Conv 2	26x26x32	24x24x64	18.496	73,9 KB
FC 1	9.216	128	1.179.776	4,71 MB
FC 2	128	10	1.290	5,16 KB
Total	-	-	1.199.882	4,79 MB

l'accuracy (9) sur la base de test, l'erreur quadratique moyenne entre θ et $\hat{\theta}$ (14) et la divergence moyenne de Kullback-Leibler (11). L'accuracy initial du réseau non-compressé est à $99,16\%$. Le but des expérimentations est d'étudier l'évolution des performances du réseau en fonction de la compression.

5.2 Interprétation des résultats

Dans la suite de la section, les abréviations KM et QU indiquent la quantification non-uniforme et la quantification uniforme.

Quantification couche par couche. Pour la première évaluation, on applique la quantification sur une seule couche isolée et les autres couches restent configurées avec les paramètres initiaux non-compressés. On s'intéresse seulement aux résultats obtenus pour 1 à 6 bits, car de 7 à 32 bits la perte d'accuracy est négligeable. La Figure 2 décrit l'accuracy, la Figure 3 la distorsion MSE et la Figure 4 la divergence de KL.
En haut à gauche de chaque figure se trouvent les résultats de la quantification pour CONV 1. Pour la quantification entre 4 et 6 bits, la perte en performance est très faible. À partir de 3 bits, l'accuracy du réseau baisse légèrement de 0,18% pour QU. Avec 2 bits, l'accuracy baisse pour les 2 méthodes d'environ 0,70%. Pour une quantification à 1 bit, le réseau perd 5% d'accuracy. La divergence de KL montre une faible perte d'information pour 2 à 6 bits. Avec 1 bit, la perte d'information de QU est très importante comparée à celle de KM, mais elle n'impacte pas l'accuracy.

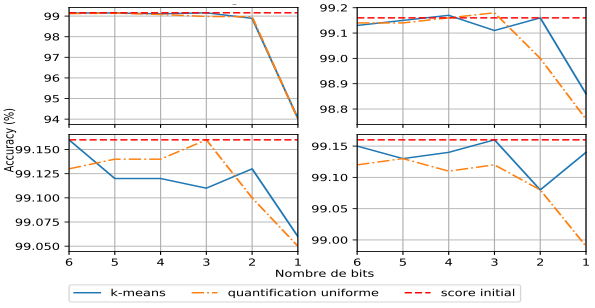

FIGURE 2 – Justesse : CONV 1 (haut gauche), CONV 2 (haut droite), FC 1 (bas gauche), FC 2 (bas droite).

La deuxième couche de convolution est représentée en haut à droite de chaque figure. On remarque qu'elle est moins sensible à la quantification que la première couche. La baisse en accuracy n'est pas très grande. On perd moins de 1% dans les deux cas de compression. La divergence montre une plus grande perte d'information pour KM, contrairement à la distorsion qui indique une plus grande erreur pour QU.

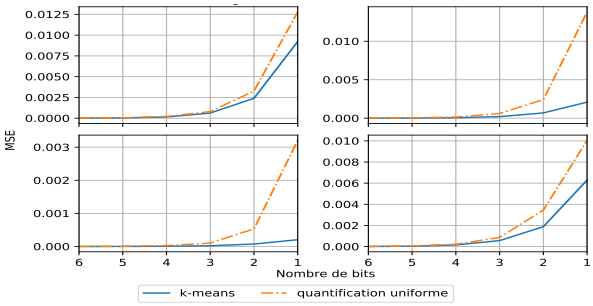

FIGURE 3 – EQM : CONV 1 (haut gauche), CONV 2 (haut droite), FC 1 (bas gauche), FC 2 (bas droite).

Dans le cas des couches FC, la perte est encore plus petite. FC 1, la plus grande couche du réseau avec 1×10^6 paramètres, est affichée en bas à gauche dans les figures. Avec 1 bit, l'accuracy baisse seulement de 0,10%-0,11%. La divergence est jusqu'à 40x plus petite et la distorsion réduite de 4x. La dernière couche, FC 2, placée en bas à droite, a peu de paramètres. Encore une fois, on voit que la perte est négligeable. Dans le cas de KM on perd 0,02% d'accuracy et, dans le cas de QU, on obtient presque la même accuracy que pour FC 1. La distorsion et la divergence indiquent aussi une erreur et une perte d'information très faible.

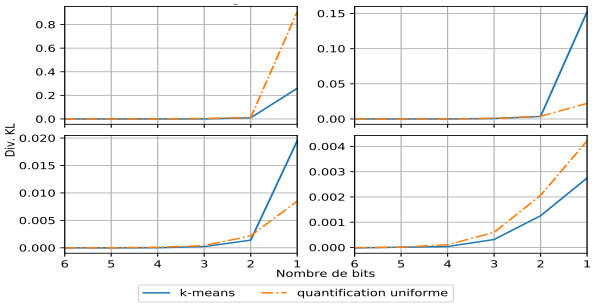

FIGURE 4 – Divergence KL : CONV 1 (haut gauche), CONV 2 (haut droite), FC 1 (bas gauche), FC 2 (bas droite).

Dans cette première évaluation, on constate que la quantification non-uniforme fonctionne mieux, ce qui est normal, car elle doit s'adapter mieux à la distribution des données. Il existe certains cas où la quantification uniforme a des meilleures performances. Globalement, la divergence KL et la distorsion sont plus grandes pour la première couche que pour les autres. Le rôle de la première couche est de construire une bonne base de descripteurs, ce qui pourrait justifier le fait qu'elle soit davantage sensible à la quantification. La Table 4 contient des statistiques de compression pour chaque couche : la méthode utilisée, l'accuracy après compression, le nombre de bits de quantification, la taille de la couche après compression, le taux de compression τ_i de la couche et le taux de compression τ du réseau entier. Pour chaque couche et pour chaque méthode de compression, on indique seulement un résultat, celui qui nous semble avoir le meilleur taux de compression pour un niveau d'accuracy presque inchangé.

Quantification du réseau entier. La deuxième évaluation est faite sur le réseau entier. On quantifie toutes les

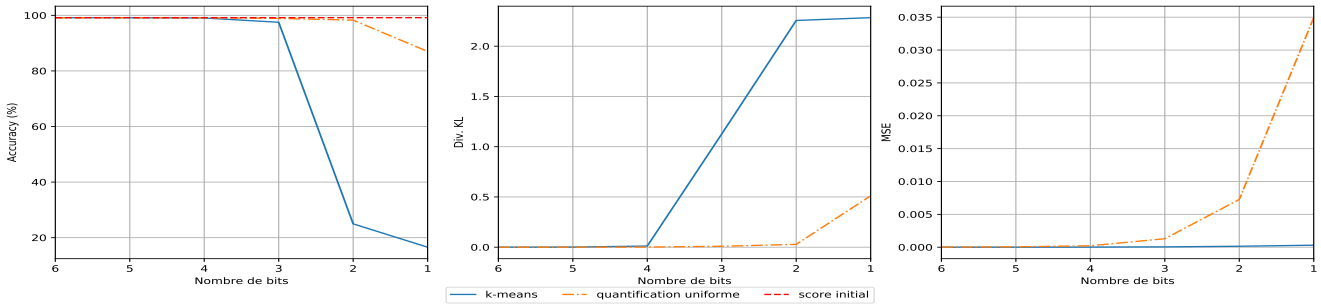

FIGURE 5 – Accuracy, Divergence KL et MSE pour le réseau entier.

TABLE 4 – Statistiques de compression pour chaque couche.

Couche	Méthode	ACC	Bits	Taille	τ_i	τ
CONV 1	QU	98,89 %	3	132B	9,69	1,0001
	KM	99,11 %	3	150B	8,53	1,0001
CONV 2	QU	99,00 %	3	6,94KB	10,64	1,014
	KM	99,16 %	2	4.64KB	15.92	1,014
FC 1	QU	99,05 %	1	147,48KB	31,93	21,025
	KM	99,06 %	1	147,47KB	31,93	21,026
FC 2	QU	98,99 %	1	173B	29,82	1,001
	KM	99,14 %	1	169B	30,53	1,001

couches du réseau avec le même quantificateur. Dans la Figure 5, on voit que le réseau réagit différemment pour les 2 méthodes de quantification. Dans le cas de KM, l'accuracy du réseau descend jusqu'à 17% et QU perd au plus 13% d'accuracy. Même si la distorsion est évidement beaucoup plus grande pour QU, la perte d'information est importante pour KM à la sortie du réseau.

Quantification adaptative du réseau entier. Une dernière évaluation est faite sur le réseau entier en appliquant un quantificateur différent pour chaque couche du réseau. Dans les expériences précédentes, on a observé que, pour 5 ou 6 bits, l'erreur est négligeable. Pour cette évaluation, on a donc appliqué une quantification entre 1 et 4 bits pour chaque couche. On a testé toutes les combinaisons possibles et on a extrait 2 cas pour chaque méthode : le premier cas donne la meilleure accuracy et le second cas présente un bon compromis entre l'accuracy et le taux de compression. Les résultats obtenus sont notés dans la Table 5.

TABLE 5 – Statistiques de compression pour le réseau avec des quantificateurs adaptés à chaque couche.

Méthode	CONV 1	CONV 2	FC 1	FC 2	ACC	τ
QU	4	3	3	4	99,18%	10,66
	3	2	2	2	98,80%	16
KM	4	4	4	3	99,16%	8
	4	2	2	2	98,33%	15,98

Les expérimentations montrent des résultats intéressants. Les deux méthodes peuvent atteindre un bon taux de compression sans une perte significative du niveau de performance. On voit que la quantification uniforme reste tout à fait performante comparée à la quantification non uniforme.

6 Conclusions et perspectives

Cet article présente un état de l'art dans la compression des réseaux de neurones profonds et compare deux méthodes de quantification scalaire, uniforme et non-uniforme, en vue de réduire le stockage en mémoire des paramètres. Le but de ce travail n'est pas de compresser et décompresser les paramètres, mais de reconstruire la sortie du réseau. Nos expériences montrent que les 2 méthodes fonctionnent bien mais, globalement, la quantification uniforme semble donner de meilleurs résultats tout en étant plus facilement manipulable d'un point de vue informatique. De plus, la quantification uniforme passe à l'échelle en temps de calcul. Il apparaît que, quelque soit la méthode utilisée, la sortie du réseau est peu affectée par une quantification grossière des poids (2-4 bits). Avec des poids quantifiés qui occupent 16 fois moins de place mémoire que les poids originaux, on perd moins de 1% en justesse de classification.

Des évaluations numériques sur des réseaux complexes, comme VGG, sont en cours. Les conclusions restent globalement les mêmes. Après compression, les dernières couches du réseau ont des pertes peu significatives, contrairement aux premières couches qui sont très sensibles à l'erreur de quantification. Dans de prochains travaux, on envisage de compléter la méthode de compression proposée avec un élagage du réseau et une compression sans perte pour améliorer encore davantage le taux de compression.

Références

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS 25*, pages 1097–1105, 2012.

[2] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd ICLR*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of IEEE conference on CVPR*, pages 770–778, 2016.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of IEEE conference on CVPR*, pages 1–9, 2015.

[5] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A Horowitz, and W. J Dally. Eie : efficient inference engine on compressed deep neural network. In *ACM/IEEE 43rd Annual ISCA*, pages 243–254. IEEE, 2016.

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

[7] J. Cong and B. Xiao. Minimizing computation in convolutional neural networks. In *Artificial Neural*

Networks and Machine Learning - ICANN 2014 - 24th, pages 281–290, 2014.

[8] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3) :308–323, September 1979. ISSN 0098-3500.

[9] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. In *ICLR*, 2014.

[10] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast convolutional nets with fbfft : A GPU performance evaluation. In Y. Bengio and Y. LeCun, editors, *ICLR*, 2015.

[11] S. Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.

[12] A. Lavin and S. Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

[13] L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31, 1966.

[14] R. Harshman. Foundations of the parafac procedure : Models and conditions for an "explanatory" multimodal factor analysis. 16, 1970.

[15] J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of IT & EE*, 19(1) : 64–77, 2018.

[16] S. Han, H. Mao, and W. J. Dally. Deep compression : Compressing deep neural network with pruning, trained quantization and huffman coding. In Y. Bengio and Y. LeCun, editors, *4th ICLR*, 2016.

[17] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *JETC*, 13(3) :32 :1–32 :18, 2017.

[18] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Yu Wang, and W. J. Dally. Exploring the regularity of sparse structure in convolutional neural networks. *CoRR*, 2017.

[19] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *NIPS 26*, pages 2148–2156, 2013.

[20] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, 2014.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd ICML*, volume 37, pages 1737–1746, 2015.

[22] M. Courbariaux, Y. Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. In Y. Bengio and Y. LeCun, editors, *3rd ICLR*, 2015.

[23] J. Gustafson and I. Yonemoto. Beating floating point at its own game : Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4(2) :71–86, 2017.

[24] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *NIPS 29*, pages 4107–4115, 2016.

[25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net : ImageNet classification using binary convolutional neural networks. In *Comput. Vis. ECCV*, pages 525–542. Springer, 2016.

[26] I. Hubara, M. Courbariaux, Da. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks : Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18 : 187 :1–187 :30, 2017.

[27] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. In *ICLR*, 2017.

[28] F. Borisyuk, A. Gordo, and V. Sivakumar. Rosetta : Large scale system for text detection and recognition in images. In *Proceedings of the 24th ACM SIGKDD, KDD 2018*, pages 71–79, 2018.

[29] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference : A whitepaper. *CoRR*, 2018.

[30] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE CVPR 2018*, pages 2704–2713, 2018.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe : Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*.

[32] CZ Tang and HK Kwan. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Trans. Signal Processing*, 41(8) :2724–2727, 1993.

[33] P. Gysel. Ristretto : Hardware-oriented approximation of convolutional neural networks. *CoRR*, 2016.

[34] P. Merolla, R. Appuswamy, J. V. Arthur, S. K. Esser, and D. S. Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *CoRR*, 2016.

[35] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28 :129–137, 1982.

[36] J. A. Hartigan and M. A. Wong. Algorithm AS 136 : A K-Means clustering algorithm. *Applied Statistics*, pages 100–108, 1979.

[37] D. Arthur and S. Vassilvitskii. k-means++ : the advantages of careful seeding. In N. Bansal, K. Pruhs, and C. Stein, editors, *Proceedings of the 8th Annual ACM-SIAM SODA*, pages 1027–1035. SIAM, 2007.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.