# Introduction aux méthodes d'accélération de réseaux de neurones

Edouard Yvinec Arnaud Dapogny Kévin Bailly

16 octobre 2022

## 1 Introduction

L'objectif de l'apprentissage profond par réseau de neurones est de pouvoir effectuer des prédictions pertinentes dans le cadre d'une tâche donnée. On compte parmi ces tâches, entre autres, celles liées à la vision par ordinateur qui nous serviront d'exemples tout a long de cet article. En particulier, on trouve la classification [1], la détection d'objets dans une image [2] ou encore la segmentation sémantique [3].

Malgré les spécificités liées à chacune de ces tâches, les architectures des réseaux de neurones, utilisées pour obtenir les remarquables performances que l'on reconnaît aujourd'hui à l'apprentissage profond, présentent de nombreuses similarités dont, entre autres, la présence d'une base commune (souvent appelée *backbone*). Pour cette étude, nous allons restreindre le bestiaire des réseaux de neurones aux ResNets [4] (les plus utilisés), aux MobileNets [5] et aux EfficientNets [6] (ces deux derniers exemples étant connus pour leur caractère compact).

Afin de réduire le coût du déploiement de tels modèles, en terme de consommation énergétique ainsi qu'en terme de coût des ressources de calculs, une large gamme de méthodes ont été développées. Dans cet article, nous nous intéresserons aux principales approches présentées dans la figure 1. Nous aborderons tout d'abord les approches par élagage (section 3) qui consistent à supprimer des neurones (élagage structuré) ou des opérations (élagage non structuré). Nous présenterons ensuite les méthodes par décomposition et recomposition des couches du réseau (section 4). Pour finir, nous présenterons l'approche par quantification qui visent à réaliser les opérations arithmétiques à virgule fixe sur un nombre restreint de bits (section 5).

## 2 Notations et Formalisme

Soit $F$ un réseau de neurones artificiels à $L$ couches. L'essentiel des calculs effectués dans un réseau de neurones artificiels $F$ se trouve dans les couches de convolutions et dans les couches densément connectées. Ces couches, ainsi que leurs variantes, (convolutions spatiales, convolutions à trous,...) définissent une multiplication matricielle. Soit $f$ une telle couche, de fonction d'activation $\sigma$, on notera

$$\begin{array}{rccc} f: & \mathcal{A}^{d_e} & \rightarrow & \mathcal{B}^{d_s} \\ & X & \mapsto & \sigma(WX+B) \end{array} \tag{1}$$

où $d_e$ et $d_s$ désignent respectivement les dimensions d'entrée et de sortie, $\mathcal{A}$ et $\mathcal{B}$ définissent chacun un $\mathbb{R}$-espace vectoriel. Dans le cas d'une couche densément connectée on aura $\mathcal{A} = \mathbb{R}$ et dans le cas d'une couche convolutionelle sur une image 224 par 224 pixels on aura $\mathcal{A} = \mathbb{R}^{224\times 224}$. On appellera $W$ le tenseur de poids et $B$ le vecteur de biais.

## 3 Élagage (pruning)

La première catégorie de méthodes permettant d'accélérer les calculs effectués par un modèle $F$ est l'élagage. Cela consiste à retirer des calculs afin de limiter le nombre d'opérations à réaliser. On oppose deux paradigmes et deux implémentations.

**La Structure** Les opérations supprimées du graphe peuvent présenter une cohérence structurelle ou non. En pruning *structuré*, les éléments supprimés sont de l'ordre du neurone (ou de la *feature map* dans le cas d'un réseau convolutif). En reprenant les notations précédentes, le pruning structuré revient à retirer des lignes ou des colonnes de $W$. Avant de décrire le pruning non-structuré, nous vous proposons de détailler un élément important du pruning structuré : *comment éliminer le biais ?*

Soit un réseau $F$ à deux couches tel que

$$F: \quad X \quad \mapsto \quad W^{(2)}\sigma(W^{(1)}X + B^{(1)}) + B^{(2)} \tag{2}$$

Supposons que nous tenons pour certain que la meilleure option est de retirer la seconde ligne de la matrice $W_1$ et notons $\mathfrak{W}^{(1)}, \mathfrak{W}^{(2)}, \mathfrak{B}^{(1)}, \mathfrak{B}^{(2)}$ les paramètres du réseau élagué $\mathfrak{F}$. Pour le moment, on suppose que la seconde ligne de $W_1$ est simplement mise à 0. Dans ce cas, on ne peut pas simplement la retirer et retirer le biais correspondant sans changer la fonction de prédictions.

$$\mathfrak{F}(X) = W^{(2)}\sigma\left(\begin{pmatrix} W^{(1)}_{1,1} & \dots & W^{(1)}_{1,N} \\ 0 & \dots & 0 \\ W^{(1)}_{3,1} & \dots & W^{(1)}_{3,N} \\ \vdots & \ddots & \vdots \\ W^{(1)}_{M,1} & \dots & W^{(1)}_{M,N} \end{pmatrix}\begin{pmatrix} X_1 \\ \vdots \\ X_N \end{pmatrix} + \begin{pmatrix} B^{(1)}_1 \\ \vdots \\ B^{(1)}_M \end{pmatrix}\right) + B^{(2)} \tag{3}$$

Pour déduire les poids élagués, nous allons *propager* le biais correspond au second neurone de la première couche.

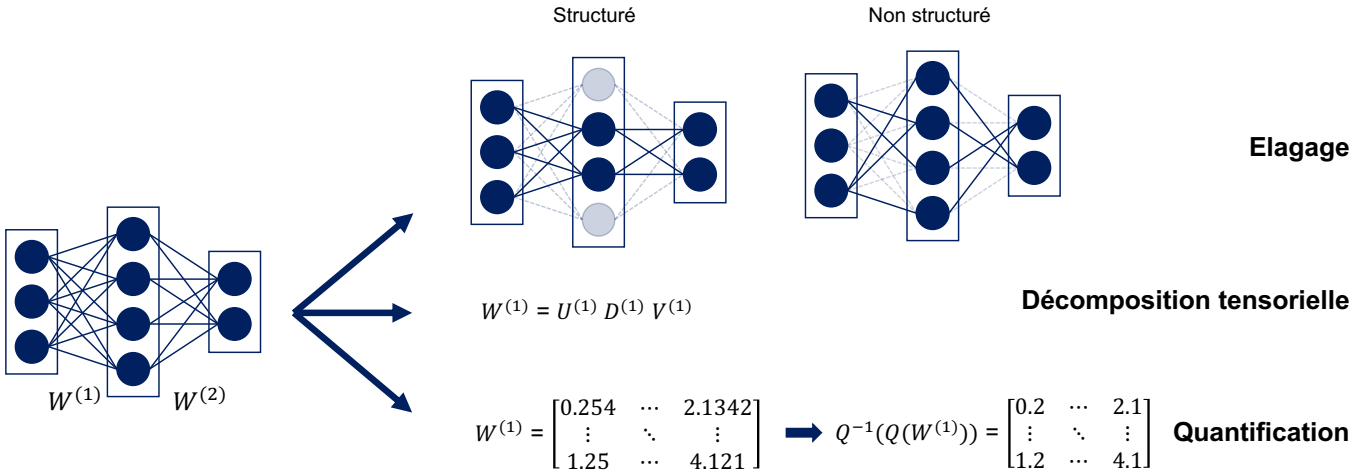


FIGURE 1 – Les 3 grandes catégories de méthodes de compression de réseaux de neurones présentées dans cet article

Ainsi, on obtient

$$\begin{cases} \mathfrak{F}(X) = \mathfrak{W}^{(2)}\sigma(\mathfrak{W}^{(1)}X + \mathfrak{B}^{(1)}) + \mathfrak{B}^{(2)} \\ \mathfrak{W}^{(1)} = \begin{pmatrix} W^{(1)}_{1,1} & \dots & W^{(1)}_{1,N} \\ W^{(1)}_{3,1} & \dots & W^{(1)}_{3,N} \\ \vdots & \ddots & \vdots \\ W^{(1)}_{M,1} & \dots & W^{(1)}_{M,N} \end{pmatrix} \\ \mathfrak{W}^{(2)} = \begin{pmatrix} W^{(2)}_{1,1} & W^{(2)}_{1,3} & \dots & W^{(2)}_{1,M} \\ \vdots & \vdots & \ddots & \vdots \\ W^{(2)}_{K,1} & W^{(2)}_{K,3} & \dots & W^{(2)}_{K,M} \end{pmatrix} \\ \mathfrak{B}^{(1)} = \begin{pmatrix} B^{(1)}_1 \\ B^{(1)}_3 \\ \vdots \\ B^{(1)}_M \end{pmatrix} \\ \mathfrak{B}^{(2)} = \begin{pmatrix} B^{(2)}_1 + W^{(2)}_{2,1}\sigma(B^{(1)}_2) \\ B^{(2)}_3 + W^{(2)}_{2,2}\sigma(B^{(1)}_2) \\ \vdots \\ B^{(2)}_K + W^{(2)}_{2,K}\sigma(B^{(1)}_2) \end{pmatrix} \end{cases} \tag{4}$$

Nous avons donc la possibilité de modifier l'architecture du réseau à l'échelle des neurones en ne travaillant que sur les poids $W$.

Cependant, cette forme d'élagage, bien que simple à exploiter, est très contrainte en terme de nombre d'opérations retirées. En relâchant la contrainte de la structure (élagage non-structuré) des opérations retirées il est possible de retirer beaucoup plus d'opérations. Ce genre méthode repose sur des implémentations efficaces des calculs en matrices creuses. Dans ce cas l'architecture n'est pas modifiée à proprement parler.

**Importance ou Ressemblance** Dans ce qui précède, nous avons supposé connaître la ligne à supprimer dans la matrice $W^{(1)}$. En pratique c'est cette connaissance qui caractérise la difficulté de l'élagage de réseau de neurones. Autrement dit, *comment savoir quel neurones ou opérations peuvent être supprimés sans diminuer significativement la précision du modèle ?*

Il existe alors deux grands paradigmes dans la littérature. Le premier est la mesure de l'importance : il s'agit de déterminer les neurones les moins importants dans la décision du modèle et de les éliminer. La seconde est la mesure de la similarité et consiste à combiner les neurones similaires.

Commençons par la mesure de l'importance. La procédure naïve et qui sert encore souvent de référence consiste à simplement mesurer la norme $L^p$ des poids correspondant à chaque norme pour obtenir un vecteur $I$ de normes :

$$I = \begin{pmatrix} \left\| \begin{pmatrix} W^{(1)}_{1,1} & \dots & W^{(1)}_{1,N} \end{pmatrix} \right\|_p \\ \vdots \\ \left\| \begin{pmatrix} W^{(1)}_{M,1} & \dots & W^{(1)}_{M,N} \end{pmatrix} \right\|_p \end{pmatrix} \tag{5}$$

La ligne à élaguer est alors simplement l'indice du minimum du vecteur d'importance $I$. Dans la littérature, il existe toutefois des critères plus élaborés [7, 8, 9, 10]

Dans le cas de l'élagage par similarité, nous allons grouper les neurones par ressemblance. Une façon de faire consiste à simplement mesurer une matrice de distance $D$ entre les neurones. Par exemple,

$$D = \begin{pmatrix} \left\| \begin{pmatrix} W^{(1)}_{1,1} - W^{(1)}_{1,1} & \dots & W^{(1)}_{1,N} - W^{(1)}_{1,N} \end{pmatrix} \right\|_p & \dots & \left\| \begin{pmatrix} W^{(1)}_{1,1} - W^{(1)}_{M,1} & \dots & W^{(1)}_{1,N} - W^{(1)}_{M,N} \end{pmatrix} \right\|_p \\ \vdots & \ddots & \vdots \\ \left\| \begin{pmatrix} W^{(1)}_{M,1} - W^{(1)}_{1,1} & \dots & W^{(1)}_{M,N} - W^{(1)}_{1,N} \end{pmatrix} \right\|_p & \dots & \left\| \begin{pmatrix} W^{(1)}_{M,1} - W^{(1)}_{M,1} & \dots & W^{(1)}_{M,N} - W^{(1)}_{M,N} \end{pmatrix} \right\|_p \end{pmatrix} \tag{6}$$

Supposons qu'il s'agisse des neurones correspondant aux lignes 1 et 2. Nous allons définir un nouveau neurone fusion qui aura pour poids associé la moyenne des poids des deux

neurones fusionnés. On a alors

$$\begin{cases} \mathfrak{F}(X) = \mathfrak{W}^{(2)}\sigma(\mathfrak{W}^{(1)}X + \mathfrak{B}^{(1)}) + \mathfrak{B}^{(2)} \\ \mathfrak{W}^{(1)} = \begin{pmatrix} \frac{W^{(1)}_{1,1}+W^{(1)}_{2,1}}{2} & \cdots & \frac{W^{(1)}_{1,N}+W^{(1)}_{2,N}}{2} \\ W^{(1)}_{3,1} & \cdots & W^{(1)}_{3,N} \\ \vdots & \ddots & \vdots \\ W^{(1)}_{M,1} & \cdots & W^{(1)}_{M,N} \end{pmatrix} \\ \mathfrak{W}^{(2)} = \begin{pmatrix} W^{(2)}_{1,1} + W^{(2)}_{1,2} & W^{(2)}_{1,3} & \cdots & W^{(2)}_{1,M} \\ \vdots & \vdots & \ddots & \vdots \\ W^{(2)}_{K,1} + W^{(2)}_{K,2} & W^{(2)}_{K,3} & \cdots & W^{(2)}_{K,M} \end{pmatrix} \\ \mathfrak{B}^{(1)} = \begin{pmatrix} \frac{B^{(1)}_1+B^{(1)}_2}{2} \\ B^{(1)}_3 \\ \vdots \\ B^{(1)}_M \end{pmatrix} \\ \mathfrak{B}^{(2)} = \begin{pmatrix} B^{(2)}_1 \\ B^{(2)}_3 \\ \vdots \\ B^{(2)}_K \end{pmatrix} \end{cases} \tag{7}$$

Nous savons donc comment éditer le réseau de telle sorte à fusionner des neurones similaires. Pour aller plus loin, il existe de nombreux travaux comme [11, 12].

**Évaluations Empiriques** Pour l'évaluation des performances de l'élagage, la base de donnée standard est Cifar10. A noter toutefois qu'elle n'offre pas un test réaliste. D'autres bases de classification plus complexes comme ImageNet ainsi que d'autres tâches, sont de plus en plus souvent abordées.

L'architecture standard est ResNet mais également sa variante Wide-ResNet. Depuis peu, on observer des évaluations sur MobileNet. Cependant, sur ce dernier, l'essentiel des paramètres se trouvant dans la tête de prédiction, on mesure le nombre de paramètres retirés parmi les paramètres en dehors de la tête de prédiction.

En matière de métrique, il y a une double mesure de la précision par rapport au nombre de paramètres retirés. Il est courant de rapporter la réduction du nombre de FLOPs (floating points operations) en plus ou place du nombre de paramètres.

Nous vous proposons les résultats suivants : le pourcentage de paramètres retirés selon le réseau de neurone et la tache permettant de préserver la précision du modèle. Ces valeurs sont listées dans le tableau 1. Les points essentiels sont au nombre de deux. Premièrement, les résultats sur Cifar10 ne sont pas de bons indicateurs des performances sur ImageNet (ou sur des cas réels, en général). Deuxièmement, il est difficile d'obtenir des accélérations de l'ordre de $\times 3$ avec de l'élagage comme c'est le cas avec la quantification, comme nous le verrons dans la suite de cet article.

TABLE 1 – Performances en élagage, on rapporte le pourcentage de paramètres retirés qui ne dégrade pas la précision du modèle

| modèle | tache | % paramètres retirés |
|---|---|---|
| ResNet 56 | Cifar 10 | ≈90% |
| Wide ResNet 28-10 | Cifar 10 | ≈75% |
| ResNet 50 | IamgeNet | ≈60% |
| MobileNet V2 | ImageNet | ≈40 % |

# 4 Décompositions et Recompositions

Dans la section précédente, nous avons présenté des solutions permettant d'accélérer les calculs en réduisant le nombre d'opérations à réaliser. Cela s'est traduit notamment par la suppression d'éléments ou de lignes dans la matrice des poids d'une couche. Nous allons maintenant travailler à l'échelle inter-couches des réseaux en présentant deux méthodes : la décomposition de couches (tensor decomposition) et la recomposition de couches (folding).

**Décomposition tensorielle** La décomposition la plus connue est la décomposition en valeur singulière (SVD).

$$W^{(1)} = U^{(1)}D^{(1)}V^{(1)} \tag{8}$$

où $U$ et $V$ sont des matrices unitaires et $D$ est une matrice diagonale. Dans le cas où le rang $r$ de $W^{(1)} \in \mathbb{R}^{M\times N}$ satisfait $r < \frac{MN}{M+N}$ alors la décomposition $U \times (DV \times X)$ nécessite moins de calculs que la couche d'origine $W^{(1)}X$ tout en effectuant exactement le même calcul (aux imprécisions numériques près). Il existe de nombreuses décompositions plus élaborées dans la littérature [13, 14, 15]

**Recomposition de couches** Nous avons vu par le prisme de la SVD, une manière d'augmenter la séquentialité d'un réseau tout en réduisant le nombre total d'opération. Cependant, en pratique, ce qui coûte dans l'inférence des réseaux est le va-et-vient des données, autrement dit : la séquentialité. Les couches que l'on aimerait supprimer sont typiquement les couches effectuant de nombreux calculs en séquences. Le parfait exemple de telles couches est celui des couches de normalisation. En effet, cette couche est définie par $X \mapsto \gamma\frac{X-\mu}{\sigma+\epsilon} + \beta$, requière donc 4 opérations successives.

Heureusement ces couches peuvent être fusionnées dans les couches de convolutions et densément connectées. Ajoutons une telle couche à notre réseau $F$, afin de montrer comment la fusionner.

$$F: \quad X \quad \mapsto \quad W^{(2)}\text{relu}(\text{BN}(W^{(1)}X + B^{(1)})) + B^{(2)} \tag{9}$$

Notons ici que l'on remplace la fonction d'activation pour ne pas avoir de situation de notations malheureuse entre la fonction d'activation et la variance de la normalisation.

Explicitement, on effectue le calcul suivant par neurone :

$$F: \quad X \quad \mapsto \quad W^{(2)}\text{relu}\left(\gamma\frac{W^{(1)}X+B^{(1)}-\mu}{\sigma+\epsilon}+\beta)\right)+B^{(2)} \tag{10}$$

Il est alors possible de réécrire la réseau sans la couche de normalisation pour l'inférence. On obtient alors

$$F: \quad X \quad \mapsto \quad W^{(2)}\text{relu}\left(W^{(1)}\frac{\gamma}{\sigma+\epsilon}\times X+\gamma\frac{B^{(1)}-\mu}{\sigma+\epsilon}+\beta)\right)+B^{(2)} \tag{11}$$

Ce qui correspond bien à un réseau à deux couches sans normalisation. C'est l'occasion de se rendre compte que la normalisation est uniquement une méthode d'apprentissage et non un élément architectural à proprement parler. Pour aller plus loin nous vous proposons la lecture suivante [16].

Ce processus s'applique également aux couches convolutionnelles et densément connectées lorsque celles-ci ne sont pas séparées par une fonction d'activation. Cela peut se produire dans certains réseaux pour réduire le nombre de paramètres. Dans le cas de deux couches densément connectées, il suffit de remplacer les couches par une couches dont les poids sont le produit matriciel des poids des couches d'origine et dont on peut déduire le biais par un simple développement. Pour le cas de couches convolutionnelles successives, il faut établir les dimensions $(k', k')$ du nouveau noyau 2D :

$$k' = k_1 + (k_2 - 1) \times s_1 \quad \text{et} \quad s' = s_1 \times s_2 \tag{12}$$

avec $k_1$ et $k_2$ les dimensions respectives des noyaux de la première et deuxième couche à fusionner et $s_1$ et $s_2$ leurs *strides*. Ensuite, les poids $W'$ sont mis-à-jour avec l'algorithme 1.

**Algorithm 1** Fusion de Convolutions : calcul des poids

**Require:** $W_1 \in \mathbb{R}^{k_1\times k_1\times n_{in}\times n_h}, W_2 \in \mathbb{R}^{k_2\times k_2\times n_h\times n_{out}}$ de strides $s_1$ et $s_2$

$W' \leftarrow \{0\}^{k'\times k'\times n_{in}\times n_{out}}$
**for** $a_{in} \in \{1,\dots,n_{in}\}$ **do**
  **for** $a_{out} \in \{1,\dots,n_{out}\}$ **do**
    **for** $x_1, y_1 \in \{1,\dots,k_1\}^2$ **do**
      **for** $x_2, y_2 \in \{1,\dots,k_2\}^2$ **do**
        $t \leftarrow \langle W_{2[x_1,y_1,n_i n,:]}; W_{2[x_2,y_2,:,n_o ut]}\rangle$
        $W'_{[s_1\times x_2+x_1, s_1\times y_2+y_1, a_{in}, a_{out}]} \leftarrow t$
      **end for**
    **end for**
  **end for**
**end for**
$\rightarrow W'$

# 5 Quantification

Comme précédemment, l'objectif est d'accélérer les calculs coûteux et donc principalement la multiplication matricielle $WX$. Dans les sections précédentes, nous y sommes arrivés en réduisant le nombre d'opérations à effectuer. Dans le cadre de la quantification le gain de performance est obtenu en remplaçant les opérations en arithmétique à virgule flottante sur 32 bits (float32, ou FP pour full-precision), communément utilisées pour l'entraînement des réseaux de neurones artificiels, par des opérations en virgules fixes sur $b$ bits. En général $b$ appartient à $\{1, 2, 4, 8, 16\}$.

Il faut donc convertir les éléments qui composent le calcul de $f$ de telle sorte à les faire *tenir* sur $b$ bits. On appelle $Q$ l'opérateur permettant d'effectuer cette conversion. En conséquence, on obtient

$$Q(X) \in \{-2^{b-1}-1,\dots,2^{b-1}-1\} \quad \text{et} \quad X \in ]-\alpha;\alpha[ \tag{13}$$

On suppose ici que le support de $X$ est symétrique par simplicité (en général on pose $\alpha = \max\{|X|\}$). Par exemple, l'opérateur de quantification *naïf* est définit par

$$Q: \quad X \quad \mapsto \quad \left\lfloor X\times\frac{2^{b-1}-1}{\alpha}\right\rceil \in \{-2^{b-1}-1,\dots,2^{b-1}-1\} \tag{14}$$

où $\lfloor\cdot\rceil$ désigne l'opération d'arrondi. Dans le cas général, l'équation 1 devient alors

$$\begin{array}{rccc} f: & \mathcal{A}^{d_e} & \rightarrow & \mathcal{B}^{d_s} \\ & X & \mapsto & \sigma(Q(W)Q(X)+B) \end{array} \tag{15}$$

À noter ici que le biais $B$ n'est pas quantifié. Dans la plupart des méthodes, il sera traité séparément par le moteur d'inférence qui est en charge de réaliser les traitements pour une architecture matérielle donnée

Se pose alors un problème : *la quantification introduit-elle une erreur conséquente ?* En l'état, si le support de $X$ est très différent de $\{-2^{b-1}-1,\dots,2^{b-1}-1\}$ (ce qui est le cas en pratique), la quantification précédente introduit un changement drastique d'échelle. En réalité, il est nécessaire d'introduire une étape de "dé-quantification".

$$Q^{-1}\circ Q: \quad X \quad \mapsto \quad \frac{\alpha}{2^{b-1}-1}\left\lfloor X\times\frac{2^{b-1}-1}{\alpha}\right\rceil \in ]-\alpha;\alpha[ \tag{16}$$

Dans ce cas, on obtient

$$\begin{array}{rccc} f: & \mathcal{A}^{d_e} & \rightarrow & \mathcal{B}^{d_s} \\ & X & \mapsto & \sigma(Q^{-1}(Q(W)Q(X))+B) \end{array} \tag{17}$$

et l'erreur de quantification correspond bien à une erreur d'arrondi à l'échelle des entrées. Explicitons l'équation précédente :

$$f(X) = \sigma\left(\frac{\max\{|X|\}}{2^{b-1}-1}\frac{\max\{|W|\}}{2^{b-1}-1}\left\lfloor W\times\frac{2^{b-1}-1}{\max\{|W|\}}\right\rceil\times\left\lfloor X\times\frac{2^{b-1}-1}{\max\{|X|\}}\right\rceil+B\right) \tag{18}$$

En observant cette formulation, on constate trois choses. Premièrement, il n'y a aucune raison d'avoir le même nombre de bits utilisés pour quantifier les entrées $X$ et les poids $W$. Dans la littérature, il est classique de noter W4/A8 une quantification sur 4 bits des poids et sur 8

bits des activations (on appelle activation la sortie d'un neurone après passage par la fonction d'activation. Ainsi, les activations des neurones d'une couche, sont les entrées des neurones de la couche suivante). Deuxièmement, on constate que le facteur d'échelle $\lambda_X = \frac{\max\{|X|\}}{2^{b-1}-1}$ appliqué aux entrées est également appliqué aux sorties. En conséquence, il doit être de dimension 1. Cependant, le facteur d'échelle $\lambda_W = \frac{\max\{|W|\}}{2^{b-1}-1}$ est appliqué aux poids et aux sorties et peut donc être un vecteur à $d_s$ dimensions. On parle alors de quantification par-canaux (per-channel). Enfin troisièmement, Si $Q(W)$ et $Q(X)$ tiennent sur $b$ bits, leur produit est sujet à l'*overflow* (modulo). Pour éviter cela, les calculs intermédiaires sont enregistrés sur 32 bits (cela se joue au niveau de la machine). On parle dans ce cas de l'accumulateur en 32 bits.

En pratique, dans les moteurs d'inférence les opérations $Q^{-1}$ et $Q$ sont dissimulées et optimisés afin d'éviter le passage des entiers aux flottants qui aurait un coût non-négligeable. Par ailleurs, la quantification offre un second avantage en dehors de la vitesse d'inférence : la réduction de l'espace mémoire. En effet, l'essentiel du coût en mémoire des réseaux reposent sur le stockage des poids. La quantification en plus de diminuer la taille de leur représentation, augmente la redondance et donc l'efficacité de méthode comme le codage de Huffman utilisé dans les méthodes de compression sans perte.

**PTQ** Supposons que nous avons à disposition un réseau $F$ déjà entraîné (comme il en existe de nombreux disponibles gratuitement en open source). L'opérateur précédent $Q$ peut s'appliquer directement aux poids de ce réseau. Cependant, ce n'est pas le cas pour les entrées des couches intermédiaires. En effet, en l'absence de données, il nous manque la valeur d'$\alpha$ (i.e. $\max\{|X|\}$). Pour y remédier, une solution fut proposée par Nagel *et al.* [17] et consiste à exploiter les statistiques enregistrées dans les couches de normalisation. Dans ce cas, nous avons donc tous les éléments pour faire de la quantification post-entraînement ou PTQ pour *Post Training Quantization*.

Formellement, on note BN une couche de normalisation précédent la couche que nous souhaitons quantifier. Alors

$$\mathrm{BN} : X \mapsto \gamma \frac{X - \mu}{\sigma + \epsilon} + \beta \tag{19}$$

En conséquence, si la couche de normalisation est apprise correctement, nous avons $\mathbb{E}[\mathrm{BN}(X)] = \beta$ et $\mathbb{V}[\mathrm{BN}(X)] = \gamma^2$. On peut donc choisir 6 écart-types (c'est la valeur recommandée par les auteurs mais en pratique n'importe quelle valeur entre 5 et 10 donne des résultats semblable) et avoir $\mathrm{BN}(X) \in ]\beta - 6|\gamma|; \beta + 6|\gamma|[$. Il suffit alors de propager cette information dans le réseau pour obtenir le support de l'entrée de la couche $f$.

**QAT** La méthode précédente a plusieurs mérites. En particulier, elle est simple à implémenter et permet d'obtenir une bonne précision en W8/A8 ce qui correspond à une accélération d'un facteur 3 à 4 à l'inférence. De plus, elle permet une préservation du droit à la vie privée et à la protection des données. Cependant, lorsque les contraintes nécessitent une plus grande compression/accélération du modèle, le PTQ échoue à fournir des modèles précis. Pour cela, il est nécessaire d'utiliser des données. La question devient alors : *peut-on utiliser des méthodes classiques d'apprentissage pour les réseaux quantifiés ?* La réponse rapide est "non mais oui". On parlera alors d'entraînement pour la quantification ou quantization aware training (QAT).

Formellement, les méthodes standards d'apprentissage des réseaux de neurones sont des variantes de la descente de gradient. Or, par la définition de l'opérateur de quantification $Q$, équation 14, ce dernier possède un gradient nul presque partout au sens de la mesure de Lebesgue. Ceci empêche d'appliquer immédiatement les méthodes de descente de gradient. Le problème étant discret, sans a priori, il est NP-difficile. En pratique, les chercheurs ont observé que les méthodes dite par straight through estimator (STE) permettent un apprentissage efficace. Ces méthodes consistent à simplement prétendre que

$$\nabla(Q^{-1} \circ Q) = \mathrm{Id} \tag{20}$$

Ce qui revient à simplement retirer l'opération d'arrondi de l'optimisation. Intuitivement, cela revient à conserver une version à virgule flottante du réseau. Après une première inférence avec le réseau quantifié, on utilise le version à virgule flottante pour rétro-propager l'erreur, avant de se ramener à nouveau à la version quantifiée du réseau pour l'étape suivante du processus d'apprentissage.

**Évaluations Empiriques** Dans cette section, nous allons discuter des méthodes d'évaluations courantes dans le domaine et des résultats à connaître.

La quantification est un domaine très développé avec des évaluations étendues sur de nombreuses bases de données et différentes architectures de réseau. En pratique, l'essentiel des méthodes sont testées sur la tache de classification d'ImageNet qui est le test canonique de la vision par ordinateur. Ce test est très important puisque bon nombre de domaines de la vision par ordinateur réutilisent des modèles pré-entraînés sur ImageNet et partagent une partie non-négligeable de leur architecture. Ce test est en général le plus difficile parmi les plus courant. On notera également Pascal VOC en détection d'objets et en segmentation d'images, MS COCO en détection et CityScapes en segmentation.

Pour ce qui est des architectures, les méthodes sont systématiquement testées sur ResNet 50. Cependant, cette architecture (comme DenseNet et ShuffleNet) est relativement facile à compresser. Depuis quelques années, les modèles compacts représentent le challenge pour la quantification, en particulier MobileNet V2 et V3, ainsi qu'EfficientNet. Par ailleurs, les transformers qui obtiennent une attention croissante en compression ne sont que rarement considérés en quantification à cause du problème suivant : la layer normalization (normalisation utilisée par les transformers). En effet, cette couche n'utilise pas de statistiques

Table 2 – Performances en PTQ sur ImageNet, la précision est donnée en pourcentage de la précision du réseau de départ.

| modèle | W8/A8 | W4/A4 | Ternaire | Binaire |
|---|---|---|---|---|
| ResNet 50 | $\approx 100\%$ | $\approx 90\%$ | - | - |
| DenseNet | $\approx 100\%$ | $\approx 90\%$ | - | - |
| MobileNet V2 | $\approx 100\%$ | $\approx 50\%$ | - | - |
| EfficientNet B0 | $\approx 100\%$ | $\approx 50\%$ | - | - |

Table 3 – Performances en QAT sur ImageNet, la précision est donnée en pourcentage de la précision du réseau de départ.

| modèle | W8/A8 | W4/A4 | Ternaire | Binaire |
|---|---|---|---|---|
| ResNet 50 | $\approx 105\%$ | $\approx 100\%$ | $\approx 92\%$ | $\approx 90\%$ |
| DenseNet | $\approx 105\%$ | $\approx 100\%$ | $\approx 92\%$ | $\approx 90\%$ |
| MobileNet V2 | $\approx 105\%$ | $\approx 90\%$ | $\approx 75\%$ | - |
| EfficientNet B0 | $\approx 100\%$ | $\approx 95\%$ | - | - |

apprises mais est toujours calculée à la volée en flottant. Ceci rend le problème de la quantification très spécifique dans le cas des transformers.

En matière de performance, nous vous proposons les tableaux 2 et 3 donnant les challenges et de l'état de l'art du domaine. Les éléments à retenir sont les suivants. Premièrement, en PTQ, le ternaire et le binaire restent encore hors de portée des méthodes actuelles. Deuxièmement, les réseaux compacts sont significativement plus durs à quantifier. Troisièmement, le QAT permet d'obtenir des résultats parfois supérieurs à ceux du réseau initial en flottant. Enfin quatrièmement, même en QAT le binaire reste extrêmement difficile pour des réseaux comme MobileNet et EfficientNet.

**Subtilités et Astuces** Il est important de préciser plusieurs petites astuces en quantification qui jouent un rôle crucial dans la performance des méthodes !

Il est standard de quantifier la première et la dernière couche du réseau en W8/A8 même si le reste est quantifié en binaire ou ternaire. Autrement dit, personne aujourd'hui ne sait faire de réseau précis entièrement binaires pour des tâches réalistes (comme ImageNet).

Pour des compressions extrêmes (binaire et ternaire), il est standard de modifier l'architecture de réseau en l'élargissant pour compenser la perte d'information due à la quantification.

Tous ces points font de ce domaine un sujet encore ouvert à la recherche ! Voici une liste non-exhaustive de travaux pertinents pour aller plus loin [17, 18, 19, 20, 21, 22]

# 6 Conclusion

Dans cet article nous avons introduit les grands principes de la compression de réseaux de neurones aux travers des trois stratégies les plus courantes : l'élagage, la décomposition et recomposition tensorielles, ainsi que la quantification. Nous avons observé les performances en fonction de l'approche considérée ainsi que de l'information dont nous disposons au moment de la compression du réseau. Ainsi les méthodes qui n'utilisent que peu ou pas de données sont plus respectueuses de la vie privée, en particulier pour des cas d'utilisation sensibles (dans le domaine médical ou militaire par exemple), mais obtiennent des résultats souvent inférieurs aux méthodes qui se permettent un (ré)entraînement du réseau au cours du processus de compression.

Nous avons également veillé à fournir au lecteur quelques références vers des méthodes de la littérature plus élaborées et plus performantes afin qu'il puisse approfondir sa connaissance du domaine. Ces références ne sont toutefois pas exhaustive et certaines méthodes, au delà de la portée de cet article, n'ont pas été abordée, telles que les approches par distillation [23, 24].

# Références

[1] J. Deng, W. Dong, et al. ImageNet : A Large-Scale Hierarchical Image Database. *CVPR*, 2009.

[2] Mark Everingham, Luc Van Gool, CKI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge 2012 (voc2012). In *Results*, 2012.

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CVPR*, pages 3213–3223, 2016.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016.

[5] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2 : Inverted residuals and linear bottlenecks. *CVPR*, pages 4510–4520, 2018.

[6] Mingxing Tan and Quoc V Le. Efficientnet : Rethinking model scaling for convolutional neural networks. *ICML*, pages 6105–6114, 2019.

[7] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. *CVPR*, pages 11264–11272, 2019.

[8] Guiying Li, Chao Qian, Chunhui Jiang, Xiaofen Lu, and Ke Tang. Optimization based layer-wise magnitude-based pruning for dnn compression. In *IJCAI*, pages 2383–2389, 2018.

[9] Congcong Liu and Huaming Wu. Channel pruning based on mean gradient for accelerating convolutional neural networks. *Signal Processing*, 156 :84–91, 2019.

[10] Edouard Yvinec, Arnaud Dapogny, Matthieu Cord, and Kevin Bailly. Singe : Sparsity via integrated gradients estimation of neuron relevance. *arXiv preprint arXiv :2207.04089*, 2022.

[11] Edouard Yvinec, Arnaud Dapogny, Matthieu Cord, and Kevin Bailly. Red : Looking for redundancies for data-freestructured compression of deep neural networks. *NeurIPS*, 34 :20863–20873, 2021.

[12] Edouard Yvinec, Arnaud Dapogny, Kevin Bailly, and Matthieu Cord. Red++ : Data-free pruning of deep neural networks via input splitting and output merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[13] Davide Bacciu and Danilo P Mandic. Tensor decompositions in deep learning. *arXiv preprint arXiv :2002.11835*, 2020.

[14] Anh-Huy Phan, Konstantin Sobolev, Konstantin Sozykin, Dmitry Ermilov, Julia Gusak, Petr Tichavskỳ, Valeriy Glukhov, Ivan Oseledets, and Andrzej Cichocki. Stable low-rank tensor decomposition for compression of convolutional neural network. In *ECCV*, pages 522–539, 2020.

[15] Yinan Wang, Weihong "Grace" Guo, and Xiaowei Yue. Tensor decomposition to compress convolutional layers in deep learning. *IISE Transactions*, 54(5) :481–495, 2022.

[16] Edouard Yvinec, Arnaud Dapogny, and Kevin Bailly. To fold or not to fold : a necessary and sufficient condition on batch-normalization layers folding. In *IJCAI*, 2022.

[17] Markus Nagel, Mart van Baalen, et al. Data-free quantization through weight equalization and bias correction. *ICCV*, pages 1325–1334, 2019.

[18] Guo Cong, Qiu Yuxian, Leng Jingwen, Gao Xiaotian, Zhang Chen, Liu Yunxin, Yang Fan, Zhu Yuhao, and Guo Minyi. Squant : On-the-fly data-free quantization via diagonal hessian approximation. *ICLR*, 2022.

[19] Eldad Meller, Alexander Finkelstein, Uri Almog, and Mark Grobman. Same, same but different : Recovering neural network quantization error through weight factorization. *ICML*, pages 4486–4495, 2019.

[20] Yichi Zhang, Zhiru Zhang, and Lukasz Lew. Pokebnn : A binary pursuit of lightweight accuracy. In *CVPR*, pages 12475–12485, 2022.

[21] Edouard Yvinec, Arnaud Dapgony, Matthieu Cord, and Kevin Bailly. Rex : Data-free residual quantization error expansion. *arXiv preprint arXiv :2203.14645*, 2022.

[22] Edouard Yvinec, Arnaud Dapogny, Matthieu Cord, and Kevin Bailly. Spiq : Data-free per-channel static input quantization. *arXiv preprint arXiv :2203.14642*, 2022.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NeurIPS*, 2014.

[24] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill : Data-free knowledge transfer via deepinversion. *CVPR*, pages 8715–8724, 2020.