Pseudo-Randomisation Partielle et Quantification pour la Compression des réseaux de neurones convolutifs

Florent Crozet[1,2], Stéphane Mancini[1], Marina Nicolas[2]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA
46 avenue Felix Viallet, Grenoble 38000, France

[2]STMicroelectronics
12 rue Jules Horowitz, 38019 Grenoble, France
florent.crozet@st.com, stephane.mancini@univ-grenoble-alpes.fr
marina.nicolas@st.com

Résumé – L'amélioration des performances des CNNs passe, encore aujourd'hui, par une augmentation du nombre de couches du réseau de neurones et/ou du nombre de filtres par couche, induisant une augmentation du nombre de paramètres nécessaires à l'exécution de l'inférence. Pour un système embarqué, ce coût supplémentaire impacte le déploiement de solutions de vision par ordinateur basées sur des CNNs. Nous proposons une méthode de compression des CNNs basée sur le remplacement de poids appris par des poids pseudo-aléatoires qui sont générés à la volée lors de l'inférence ainsi que sur l'utilisation de la quantification pour limiter l'empreinte mémoire du réseau. Avec cette approche, nous divisons par 5 la taille mémoire de MobileNetV2 en impactant sa précision de moins de 5%.

Abstract – To improve the performance of CNNs, the number of layers of the neural network and/or the number of filters per layer must be increased, this leads to an increase in the number of parameters required to perform the inference. For an embedded system, this additional cost impacts the deployment of computer vision solutions based on CNNs. We propose a compression method based on the replacement of learned weights by pseudo-random weights that are generated on the fly during inference and the use of quantization to limit the network memory footprint. With this approach, we divide the memory size of MobileNetV2 by 5 while impacting its accuracy by less than 5%.

1 Introduction

Avec le développement massif d'applications basées sur des réseaux de neurones convolutifs (CNNs), ceux-ci ont évolué vers des architectures très paramétrées. Une des conséquences directes est la taille mémoire des réseaux de neurones qui dépasse le Giga-octet [1]. Cette tendance à toujours plus augmenter la taille des CNNs conduit leurs applications à seulement s'exécuter sur des serveurs avec de très grandes puissances de calcul. Il devient donc de plus en plus difficile de déployer des réseaux de neurones de l'état de l'art sur des systèmes embarqués où la mémoire est la principale contrainte car elle influe directement sur la consommation d'énergie et la taille du circuit.
Une augmentation du nombre de paramètres entraine une augmentation du nombre d'accès mémoire. Chaque accès contribue à une partie non négligeable de l'énergie nécessaire au fonctionnement d'applications embarquées [2], il est donc important de les limiter le plus possible afin d'éviter d'avoir un système embarqué trop énergivore.

Pour éviter d'ajouter de la mémoire sur des systèmes embarqués, des solutions ont été proposées pour réduire le nombre de paramètres, et/ou leur précision, et ainsi la taille mémoire nécessaire à une inférence embarquée. Dans cet article, nous proposons une méthode de compression qui exploite les deux approches d'une part en combinant des générateurs de nombres aléatoires pour réduire le nombre de paramètres à stocker et d'autre part en quantifiant ces paramètres pour réduire la taille mémoire nécessaire. La méthode proposée permet aussi de réduire le nombre de calculs nécessaires pour réaliser une inférence.

L'article commence par un bref état des lieux concernant les méthodes de compression des réseaux ainsi que l'utilisation de nombres aléatoires dans les CNNs. Ensuite, notre méthode de compression ainsi que son implémentation sont présentées dans la section 3. Enfin, les performances post-compression des CNNs sont discutées dans la partie 4.

2 Travaux liés

Le travail présenté dans cet article est basé sur deux champs de recherche impliquant les réseaux de neurones : le premier concerne la compression des CNNs, pour des ap-

plications embarquées ; le second concerne l'utilisation de poids aléatoires dans les réseaux de neurones pour étudier l'impact de l'architecture sur la précision du CNN.

Compression CNN Les méthodes pour compresser des réseaux de neurones sont variées, nous nous concentrons ici sur des techniques permettant de réduire le nombre de paramètres sans modifier profondément l'architecture des réseaux : le Pruning, la Réduction de Dimensionnalité et la quantification.

Le Pruning consiste à réduire le nombre de paramètres nécessaires à l'inférence en selectionant les paramètres influants. On peut distinguer deux approches : mise à zéro de poids, on parle de pruning non-structuré [3] ou mise à zéro de filtres entiers, et on parle de pruning structuré [4]. Une fois un taux de parcimonie atteint, les tenseurs sont compressés en utilisant des méthodes d'encodage tel que le codage de Huffman pour les stocker. La combinaison Pruning/encodage permet une économie de mémoire. Cependant, lors de l'inférence embarquée, les tenseurs sont décompressés et ajoutent un coût calculatoire.

La Réduction de Dimensionnalité consiste à réaliser une projection dans un espace de dimension moindre afin d'obtenir moins de paramètres, et donc d'en stocker moins. On veut ainsi trouver une base d'un sous-espace vectoriel dans lequel la matrice originelle des poids peut être projetée en minimisant l'erreur de reconstruction. La méthode d'analyse en composantes principales (ACP) permet de réduire la dimension en travaillant dans un sous-espace vectoriel construit à partir des vecteurs propres[5].

La quantification permet de réduire la précision des poids à stocker. Les poids d'un réseau de neurones sont généralement représentés par des valeurs flottantes codées sur 32 ou 64 bits. Dans le cadre de la quantification, on vient réduire la cardinalité de l'ensemble des valeurs possibles pour les poids et donc réduire la précision nécessaire pour les stocker.

Nombres aléatoires et CNN L'utilisation de nombres aléatoires dans le domaine des réseaux de neurones intervient de la conception de l'architecture à l'entrainement, avec une initialisation des poids qui se fait de manière aléatoire, voire avec une utilisation lors de l'inférence. Dans notre cas, l'utilisation intéressante est celle où les poids aléatoires sont les mêmes au moment de la compression et à l'inférence. Dans cette optique, nous pouvons souligner le travail de [6] où l'auteur cherche à réduire la proportion de poids appris en laissant la majeure partie des poids initialisés aléatoirement. L'étude [7] montre aussi la possible re-randomisation de certaines couches d'un réseau de neurones affecte ou non la précision du modèle, permettant ainsi de définir des couches critiques, où l'apprentissage est nécessaire, et des couches non critiques, où la réinitilisation ne cause pas de perte de précision du réseau.

Dans notre cas, nous utilisons l'aspect déterministe des générateurs de nombres pseudo-aléatoires comme moyen de compresser les CNNs en générant des filtres pseudo-aléatoires. C'est un point important puisqu'il nous permet d'obtenir les mêmes filtres générés à partir d'une graine et d'un générateur.

3 Méthode de compression

La mémoire étant le facteur limitant pour le déploiement des CNNs sur des cibles embarquées, notre méthode propose d'économiser du silicium en remplaçant du coût mémoire par du coût logique. Plutôt que de stocker une partie des filtres, ceux-ci sont générés à la volée avec des générateurs et des graines lors de l'inférence. Afin de compresser les réseaux de neurones convolutifs, notre méthode remplace les tenseurs de filtres de chaque couche de convolution par un ensemble de filtres principaux quantifiés, un ensemble de coordonnées quantifiés et un ensemble de graines (Seeds).

Notre méthode de compression via l'aléatoire (RCN), ainsi que sa version avec quantification (QRCN), réalise en trois étapes la compression des filtres des couches de convolution. D'une part, nous décomposons les filtres de chaque couche de convolution en une combinaison linéaire de filtres principaux et d'un ensemble de coordonnées en réalisant une analyse en composantes principales (ACP) et un seuillage. D'autre part, nous remplaçons des filtres principaux par des filtres générés de façon pseudo-aléatoire, représentés par leur graine et nous quantifions les filtres principaux conservés. Afin d'anticiper l'étape de quantification, les filtres générés sont directement à valeur entière, permettant ainsi d'avoir un générateur très simple. Finalement, nous réalisons une étape de ré-entrainement pour les coordonnées, afin de corriger la perte de précision induite par les modifications.

3.1 ACP et seuillage

La première étape a pour but d'obtenir une combinaison linéaire décrivant les filtres d'une couche de convolution à partir d'un ensemble de filtres principaux. L'étape d'ACP permet de trouver la combinaison linéaire, alors que l'étape de seuillage qui suit vient réduire le nombre de filtres principaux pour conserver seulement un pourcentage de l'énergie contenue dans ceux-ci. On garde donc seulement les premiers filtres principaux qui sont arrangés de façon décroissante en termes d'importance.

L'analyse en composantes principales permet d'obtenir la combinaison linéaire suivante pour décrire chaque filtre contenu dans la couche de convolution :

$$W = C_{PCA} B_{PCA}^T + \mu \quad (1)$$

Avec W les filtres de la couche de convolutions, C_{PCA} les coordonnées des filtres dans la base des vecteurs propres B_{PCA} et μ la valeur moyenne de W.
Une fois la combinaison linéaire obtenue, on réduit le nombre

de filtres principaux en réalisant un seuillage sur l'énergie, permettant d'écrire les filtres de la manière suivante :

$$\tilde{W} = C_T.B_T^T + \mu \quad (2)$$

Avec $\tilde{W}$ une approximation de W, B_T la base contenant seulement les t premiers vecteurs propres de B_{PCA} et C_T les coordonnées des filtres dans la base B_T extrait de C_{PCA}.

3.2 Filtres pseudo-aléatoires

Dans cette seconde étape, nous venons réduire encore davantage la taille mémoire nécessaire à l'exécution de l'inférence. Le remplacement d'une partie des filtres principaux par des filtres générés pseudo-aléatoirement à valeur entière et la quantification permettent d'économiser plus de mémoire, en n'ayant à stocker que les graines qui génèrent les filtres pseudo-aléatoires et des filtres principaux à valeur entière. Les filtres pseudo-aléatoires sont sélectionnés afin de constituer un sous-espace vectoriel proche de celui obtenu après le seuillage.

Une fois les filtres pseudo-aléatoires sélectionnés, seulement les graines sont conservées, les filtres seront générés à la volée lors de l'inférence. Le nombre de filtres remplacés est donc un paramètre à prendre en compte lors de l'utilisation de la méthode pour optimiser le compromis coût/performance. Nous construisons la base B_R pour approximer la base B_T et l'initialisons avec les e premiers filtres de B_T. Le reste des filtres de la base B_R, soit $t - e$ filtres, sont sélectionnés avec la distance de Grassmann pour minimiser la distance entre le sous-espace vectoriel obtenu avec la base B_T et celui avec la base B_R. La construction de B_R se fait de manière itérative comme présenté dans l'algorithme 1. Le remplacement de filtres principaux par des pseudo-aléatoires entraine une baisse de la précision du CNN.

Algorithm 1 Sélection des filtres pseudo-aléatoires

Input: B_T /* Base issue de l'étape 1 */
  $B_R \leftarrow B_T[1:e]$
  $Seeds \leftarrow []$ /* Table des seeds */
  for $i = e + 1$ to t **do**
    /* Génère X seeds à tester par filtre */
    $Seeds_t \leftarrow GenerateSeeds(X)$
    /* Génère les filtres associés aux seeds */
    $Filters_{Random} \leftarrow PRNG(Seeds_t)$
    $Dist_t \leftarrow []$
    for $filter$ in $Filters_{Random}$ **do**
      $dist \leftarrow GrassmannDistance(B_T[:i], [B_R, filter])$
      $Dist_t.append(dist)$
    end for
    $index_{min} \leftarrow ArgMin(Dist_t)$
    $B_R.append(Filters_{Random}[index_{min}])$
    $Seeds.append(Seeds_t[index_{min}])$
  end for

3.3 Ré-entrainement des coordonnées

Cette dernière étape intervient pour corriger la perte de précision induite par l'introduction des filtres pseudo-aléatoires. Nous ré-entraînons les coordonnées de la combinaison linéaire puisque l'on ne veut pas modifier les filtres de la base B_R.

Le ré-entrainement des coordonnées permet d'obtenir un niveau de précision proche de celui du CNN d'origine. Cependant, une fois cette étape passée, les coordonnées sont à valeur flottante et représente une partie importante de la mémoire. Pour améliorer l'économie de mémoire, leur quantification est possible, mais avant cela, nous venons affiner leur valeur avec une étape d'entrainement pour la quantification [8] pour réduire l'erreur induite par cette étape. Une fois la méthode appliquée, nous obtenons un ensemble de filtres principaux à valeur entière, un ensemble de graines et un ensemble de coordonnées à valeur entière ou flottante.

3.4 Inférence embarquée

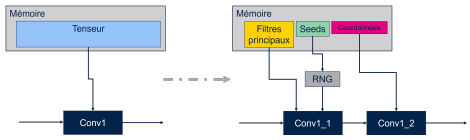

FIGURE 1 – Modification de l'architecture pour une inférence embarquée.

Les filtres principaux, les coordonnées et les graines sont ensuite stockés en mémoire et nous utilisons le même générateur de nombres aléatoires à l'inférence pour obtenir les filtres manquants. Une modification de l'architecture du CNN, décrite sur la figure 1, permet d'éviter de recombiner les filtres :

$$f_{out} = (Coords * B_R) * f_{in} = Coords * (B_R * f_{in}) \quad (3)$$

Avec f_{in} les features maps d'entrée qui sont convolués avec B_R et ensuite avec $Coords$ afin d'obtenir les features maps de sortie f_{out}. Cette modification permet aussi de diminuer le nombre d'opérations pour exécuter une couche de convolution.

4 Expérimentations

Les résultats présentés dans cette partie sont issus de tests réalisés sur différents réseaux de neurones. Dans cet article, nous nous concentrons sur l'évaluation de MobileNetV2 entrainé sur la base de données Cifar10. Nous présentons les métriques permettant de comparer notre méthode avec des méthodes similaires de compression de CNNs, avant de discuter des résultats obtenus.

Les deux métriques utilisées sont la précision et la taille mémoire du réseau. La précision permet de caractériser la

capacité du réseau de neurones à faire la bonne prédiction :

$$Precision = \frac{BonnesPredictions}{PredictionsTotales} * 100 \qquad (4)$$

Tandis que la taille mémoire du réseau permet d'évaluer la compression effective.

Les tests ont été réalisés en utilisant un Registre à décalage à rétroaction linéaire (LFSR) 16 bits où l'on utilise seulement les 8 bits de poids faibles pour obtenir une valeur. Les filtres principaux et pseudo-aléatoires sont à valeur dans l'intervalle [-128, 127].

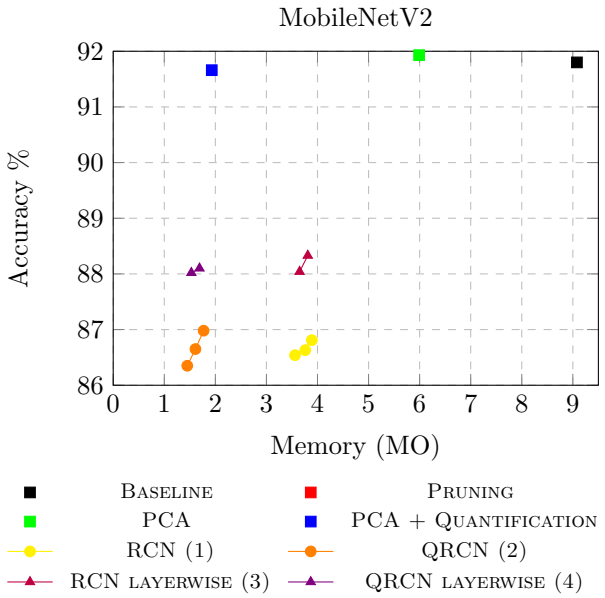

Figure 2 – Diagramme taille mémoire/précision des tests réalisés sur MobileNetV2. Notre méthode présente différents compromis avec des proportions différentes de filtres aléatoires. Pour *RCN (1) et QRCN (2)* les trois points représentent les proportions de filtres pseudo-aléatoires suivantes : 75%, 50% et 25%. Pour *RCN layerwise (3) et QRCN layerwise (4)* les deux points représentent une approche non-uniforme de la méthode avec deux proportions de filtres pseudo-aléatoires majoritaires : 50% et 25%.

Adaptation compromis Précision/Mémoire QRCN permet, en combinant l'utilisation de générateurs de nombres pseudo-aléatoires et la quantification, d'obtenir une taille mémoire inférieure aux autres méthodes comme observé dans la figure 2. L'introduction de filtres pseudo-aléatoires permet de réduire la taille du réseau tout en conservant une précision proche de celle du réseau d'origine. Notre méthode *RCN (1)* quantifie seulement les filtres de la combinaison linéaire alors que *QRCN (2)* quantifie aussi les coordonnées. La quantification des coordonnées a très peu d'impact sur la précision alors que le gain de compression est multiplié par minimum 2 comparé à *RCN (1)*.
Comme présenté en section 3.2, le nombre de filtres pseudo-aléatoires introduit est un point important de la méthode puisqu'il impacte la précision et le gain mémoire. Il est cependant compliqué de définir une bonne valeur pour tout le réseau, c'est pourquoi nous pouvons définir ce paramètre couche par couche. Une approche non-uniforme (layerwise) de la méthode est testée en réduisant le nombre de filtres principaux remplacés dans les premières couches et en l'augmentant pour les dernières couches du réseau. Avec *RCN layerwise (3)* seulement les filtres sont quantifiés alors qu'avec *QRCN layerwise (4)* les coordonnées sont aussi quantifiées. L'approche *layerwise* permet une amélioration concernant la précision tout en maintenant un coût mémoire très proche de la méthode RCN.

5 Conclusion

Nous avons introduit une nouvelle méthode de compression pour les CNNs basée sur l'utilisation de nombres pseudo-aléatoires qui permet de réduire le nombre de paramètres à stocker. Nous avons combiné notre approche avec une méthode de quantification afin d'augmenter davantage le gain mémoire. Cette méthode nous permet de réduire d'un facteur 5 la taille mémoire occupée par MobileNetV2 tout en conservant une précision inférieure à 5% d'erreur. La précision peut être améliorée avec un pourcentage de filtres pseudo-aléatoires non uniforme suivant les couches. Nous pouvons ainsi réduire la taille mémoire nécessaire pour embarquer l'inférence en remplaçant son utilisation par des générateurs de nombre pseudo-aléatoire qui ont un coût logique et énergétique inférieur.

Références

[1] Z. Liu, H. Mao, C. -Y. Wu, C. Feichtenhofer, T. Darrell and S. Xie, A ConvNet for the 2020s, 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 2022, pp. 11966-11976, doi : 10.1109/CVPR52688.2022.01167.

[2] M. Lorowitz. Energy table for 45nm process. Standford VLSI wiki.

[3] S. Han, J. Pool, J. Tran, and W. J. Dally, Learning both weights and connections for efficient neural networks, 2015, In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15). MIT Press, Cambridge, MA, USA, 1135–1143.

[4] D. Mittal, S. Bhardwaj, M. M. Khapra and B. Ravindran, "Recovering from Random Pruning : On the Plasticity of Deep Convolutional Neural Networks," 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 2018, pp. 848-857, doi : 10.1109/WACV.2018.00098.

[5] L. F. Brillet, S. Mancini, S. Cleyet-Merle and M. Nicolas, "Tunable CNN Compression Through Dimensionality Reduction," 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 2019, pp. 3851-3855, doi : 10.1109/ICIP.2019.8803585.

[6] A. Rosenfeld and J. Tsotsos, (2019). Intriguing Properties of Randomly Weighted Networks : Generalizing While Learning Next to Nothing. 9-16. 10.1109/CRV.2019.00010.

[7] C. Zhang, S. Bengio, and Y. Singer, "Are All Layers Created Equal ?", ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena. doi :10.48550/arXiv.1902.01996.

[8] Jacob, Benoit, et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.