Compression par pseudo-randomisation partielle des réseaux de neurones convolutifs sous fortes contraintes mémoire

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

[1]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 – Le nombre de paramètres des réseaux de neurones convolutifs explose créant une contrainte forte pour l'implémentation des algorithmes sur des systèmes embarqués. Dans certains dispositifs électroniques, la capacité mémoire est strictement limitée et il devient nécessaire de remplacer le stockage des données par leur compression préalable, puis la décompression à la volée. Afin d'obtenir un compromis entre la précision et le nombre de paramètres à stocker, nous proposons une méthode de compression basée sur une décomposition linéaire des filtres en vecteurs appris et vecteurs générés. Les vecteurs appris sont stockés alors que les autres sont générés de manière pseudo-aléatoire lors de l'inférence. Avec cette méthode, nous divisons par 8 le nombre de paramètres à stocker pour un réseau de neurones VGG16 en perdant seulement 6% de précision.

Abstract – The number of parameters in convolutional neural networks keeps increasing, creating a strong constraint for the algorithms on embedded devices. In some electronic devices, the storage capacity is strictly limited and it becomes mandatory to replace the data storage by their compression, and then their decompression on the fly. To get a trade-off between the memory footprint and the accuracy, we proposed a compression method based on a linear decomposition of the filters in learned vectors and generated vectors. The learned vectors are stored while the others vectors are generated in a pseudo-random way during the inference. With this method, we divide by 8 the number of parameters to store for a VGG16 neural networks with only 6% of accuracy loss.

1 Introduction

Les CNNs doivent faire face aux contraintes des systèmes IoT où il y a peu de mémoire et une puissance de calcul limitée. Pour gérer le problème de mémoire disponible, plusieurs méthodes de compression ont été proposées avec pour objectif de remplacer la mémorisation des coefficients des CNNs par leur compression avec perte, puis leur décompression, tout en ajustant finement les compromis entre la puissance de calcul, la consommation d'énergie, la surface du circuit et la précision des réseaux de neurones ainsi comprimés. En effet, pour de nombreuses applications, il peut être avantageux de générer les coefficients par calcul matériel plutôt que de les stocker dans une mémoire, elle-même consommatrice d'énergie statiquement (leakage) et dynamiquement (accès mémoires).

Dans cet article, nous proposons une nouvelle méthode de compression pour les CNNs où une partie des poids est stockée en mémoire alors que le reste des poids est généré de façon pseudo-aléatoire pendant l'inférence. Cette approche est basée sur une méthode de réduction de dimensionnalité, en l'occurence l'utilisation de l'analyse en composante principale (ACP), à laquelle est ajoutée la génération des vecteurs pseudo-aléatoires. Le but est de décomposer les poids contenus dans les couches de convolution en une combinaison linéaire faisant intervenir des vecteurs appris et des vecteurs pseudo-aléatoires. De cette façon, à l'inférence, les vecteurs pseudo-aléatoires sont régénérés à partir de leur graine (seed) et il n'y a plus besoin de les stocker. La mise en oeuvre de cette méthode sur des CNNs usuels montrent qu'il est possible d'atteindre un taux de compression d'un facteur 8, pour une perte de seulement 6% de précision du CNN, et que de nombreux compromis sont possibles selon les objectifs du concepteur.

Cet article commence par un bref aperçu des travaux liés aux réseaux de neurones à matrice aléatoire, suivi d'un rappel de l'application de l'ACP à la compression de CNN. Ensuite, la méthode de compression par randomisation est décrite, avec son application à des réseaux de neurones convolutifs classiques. Finalement, après la conclusion, l'article se termine par une évocation des pers-

pectives offertes par cette nouvelle méthode de compression des CNNs.

2 Travaux liés

Cet article est à la jonction entre les réseaux de neurones avec poids aléatoires (ELM) et les méthodes de réduction de dimensionnalité avec l'ACP.

2.1 ELM

L'introduction de nombres aléatoires dans les réseaux de neurones convolutifs a déjà été étudiée au travers des Machines d'Apprentissage Extrême (ELM)[1]. Cet algorithme propose une méthode d'apprentissage où les premières couches des réseaux de neurones sont composées de poids aléatoires et la dernière couche est apprise via une méthode de pseudo-inverse. Cette méthode a été largement appliquée aux réseaux de neurones [2] et aux CNNs [3][4]. Concernant ces derniers, la partie aléatoire concerne l'ensemble des couches de convolutions, représentant la majeure partie des poids, et donc il n'est pas nécessaire de stocker ces valeurs. Cependant, comme mentionné dans [5], la précision des modèles se dégrade sur des tâches de vision par ordinateur plus complexes. L'utilisation des poids aléatoires, pour les ELM, est donc restreinte à des tâches simples.

2.2 Analyse en Composante Principale

l'ACP est utilisée pour réduire la dimensionnalité des poids et ainsi stocker moins de coefficients pour embarquer une inférence. L'idée est de stocker une décomposition de la matrice des poids X et de la reconstituer lors de l'inférence pour ainsi économiser de la place mémoire [6] :

$$\tilde{X} = Y_C.P_i^T + \mu \tag{1}$$

avec P_i qui correspond à la sous-base composée des i premiers vecteurs propres, de la matrice X, exprimant une énergie supérieure à un seuil ($E_{THRESHOLD}$) ; μ le vecteur des valeurs moyennes de X ; et Y_C correspond à la projection de la matrice des poids centrée en zéro X_C dans l'espace décrit par P_i :

$$Y_C = X_C.P_i \tag{2}$$

L'intérêt de cette méthode vient du fait de stocker deux matrices de dimensions moindres Y_C et P_i et qui permettent d'obtenir une approximation de X.

À partir d'un réseau appris avec une méthode classique de descente de gradient, notre méthode propose d'ajouter du pseudo-aléatoire dans les réseaux de neurones convolutifs. Sans les modifier directement, les poids originellement appris sont décomposés en une combinaison linéaire de vecteurs appris et vecteurs pseudo-aléatoires.

3 Méthode de randomisation

La méthode a été construite pour compresser les CNNs en réduisant le nombre de poids stockés et en utilisant un générateur pseudo-aléatoire pour produire l'autre partie des poids. Le déroulement suit l'algorithme 1 : une analyse en composante principale pour réduire la dimension des poids et l'introduction des vecteurs aléatoires dans l'une des deux matrices pour réduire à nouveau le nombre de poids stockés. Une étape de ré-entraînement se fait ensuite pour corriger les écarts entre le sous-espace vectoriel issu de l'ACP et celui généré par tirage aléatoire.

Algorithm 1 Algorithme de pseudo-randomisation des poids

$X_C \leftarrow X - \mu$
$\lambda, V \leftarrow PCA(X_C^T.X_C)$
$P_i \leftarrow ComputeVectorsKept(\lambda, V, E_{THRESHOLD})$
$Y \leftarrow X_C.P_i$
$R \leftarrow RandomNumberGenerator(i-j, SEED)$
$P_i[J:i] \leftarrow R$
$Y_t \leftarrow Training(Model, Y)$
$X_{recombined} \leftarrow Y_t.P_i + \mu$

3.1 Génération des vecteurs aléatoires

On cherche maintenant à remplacer des vecteurs de la base P_i par des vecteurs pseudo-aléatoires afin de réduire le nombre de vecteurs à stocker. L'ensemble des vecteurs aléatoires est généré à partir d'une graine (SEED) et d'un générateur de nombres pseudo-aléatoires. Ainsi, il suffira de ne stocker que la graine et de régénérer les vecteurs manquants.

Les j premiers vecteurs de P_i sont conservés, étant les vecteurs les plus importants extrait par l'ACP. Les $(i-j)$ vecteurs restants sont remplacés par des vecteurs aléatoires. Le paramètre j influencera le taux de compression et la qualité des filtres régénérés. Les filtres contenus dans X sont alors décomposés en une combinaison linéaire de vecteurs stockés et de vecteurs générés, comme l'indique la figure 1. Comme chaque couche a un nombre de vecteurs propres différent, la quantité de vecteurs gardés est exprimée en pourcentage pour l'ensemble du réseau de neurones. p représente le pourcentage de vecteurs propres conservé. Comme les poids de chaque couche sont modifiés, la précision du réseau de neurones va diminuer. Afin de réduire cette perte, une étape de ré-entraînement est réalisée pour ajuster la combinaison linéaire de la figure 1. Seul les coefficients Y_k sont ré-entraînés afin de garder les filtres générés fixes.

La construction de l'ensemble des vecteurs pseudo-aléatoires se fait au travers de deux stratégies naïves. La première méthode consiste à sélectionner, parmi un ensemble plus

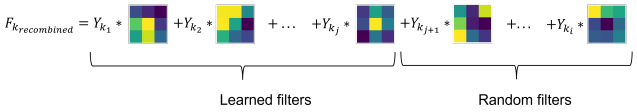

FIGURE 1 – Combinaison linéaire des filtres appris et aléatoires.

grand, des vecteurs aléatoires suivant un critère pour essayer d'obtenir une matrice presque orthogonale. La sélection des vecteurs pseudo-aléatoires intéressants nécessite de stocker les indices des vecteurs choisis afin de pouvoir les sélectionner lors de l'inférence. La deuxième méthode repose sur une orthogonalisation effectuée à chaque inférence sur un nombre de vecteurs aléatoires. On doit cependant réaliser des orthogonalisations à chaque inférence, rajoutant ainsi un coût calculatoire important. Les deux méthodes permettent d'obtenir une matrice P_i avec $i-j$ vecteurs aléatoires ayant le même rang que la matrice P_i composé de i vecteurs propres.

3.2 Pourcentage de poids appris : p

Le pourcentage de vecteurs gardés est un paramètre important de la méthode. L'information gardée permet, suivant la valeur de pourcentage, d'atteindre des valeurs de précision plus ou moins haute. Le cas où l'on garderait $p = 0\%$, c'est-à-dire que tous les vecteurs sont aléatoires n'est pour l'instant pas intéressant puisque l'on cherche à avoir des poids générés et des poids appris, il n'est donc pas traité dans cet article et fait l'objet d'une étude en cours. La méthode propose de remplacer une partie des vecteurs appris par des vecteurs pseudo-aléatoires.

3.3 Nombre de filtres aléatoires : n_r

Pour améliorer la précision du modèle une fois l'introduction de $(i-j)$ vecteurs pseudo-aléatoires, une approche consiste à rajouter plus de vecteurs pseudo-aléatoires pour essayer de reconstituer l'espace vectoriel des poids X. En effet, le simple remplacement des $i-j$ vecteurs de la base P_i par des vecteurs pseudo-aléatoires ne produisant pas nécessairement le même sous-espace vectoriel, générer davantage de vecteurs aléatoires produit une base P_{j+n_r} qui sera un sur-espace vectoriel de P_i.

4 Résultats

L'évaluation de la méthode a été réalisée sur plusieurs réseaux de neurones convolutifs avec le jeu de donnée Cifar10. Le seuil d'énergie pour l'ACP ($E_{THRESHOLD}$) a été fixé à 70%. Les CNNs utilisés sont les suivants :
VGG16 : l'architecture utilisée correspond aux trois premiers blocs de VGG16 connectés à deux couches entièrement connectées qui ont chacune 4096 neurones et la couche de sortie est composée de 10 neurones. Avec un entraînement durant 30 époques, le réseau de neurones atteint 82% de précision sur Cifar10.
ResNet50 : l'architecture utilisée correspond aux 5 blocs convolutifs de ResNet50 avec deux couches entièrement connectées qui ont respectivement 1024 et 512 neurones avec une couche de sortie composée de 10 neurones. Avec 5 époques d'entraînement, le réseau de neurones atteint 92% de précision sur Cifar10.

Dans la partie expérimentale, nous étudions le comportement des CNNs suivant deux paramètres : le pourcentage de vecteurs gardés p et le nombre de vecteurs aléatoires ajoutés n_r. Le pourcentage p permet de calculer le nombre de filtres gardés $j : j = p/100 * i$ et ainsi déduire le nombre de vecteurs $(i-j)$ remplacés par des vecteurs aléatoires. Pour p, les valeurs utilisées sont : 75%, 50% et 25%. Pour le nombre de vecteurs aléatoires, on définit $n_r = f_r * (i-j)$ avec $f_r \in \{0, 1, 2, 4\}$. L'intérêt d'ajouter des vecteurs aléatoires est d'augmenter la précision du modèle par rapport à la référence obtenu avec $f_r = 0$.

4.1 Gain de compression

Comme les vecteurs pseudo-aléatoires sont générés pendant l'inférence, les seuls paramètres à stocker sont : les vecteurs propres appris P, les coordonnées de la combinaison linéaire Y, les graines S (seed), les indices des vecteurs pseudo-aléatoires I, et enfin les poids des couches denses D. Le gain de compression se calcule ainsi :

$$G = N/(P + Y + S + I + D) \tag{3}$$

avec N le nombre de paramètres total du réseau.

4.2 Compromis précision/compression

De façon générale, la méthode permet d'atteindre de très bon taux de compression, avec une large diversité de compromis entre le taux de compression et la précision du CNN obtenu. Par exemple, il est possible de réduire la quantité de mémoire d'un facteur 8, pour une perte de précision de seulement 6%, ou bien un facteur de compression de 12, pour une perte de précision de 11% sur VGG16.

La tendance dégagée dans les figures 2 et 3 est la suivante : l'ajout en sur-nombre de vecteurs aléatoires permet de gagner en précision comparé au cas où il n'y a pas de vecteurs aléatoires ($f_r = 0$). Suivant la valeur du pourcentage de vecteurs gardés, le modèle, avant l'ajout de vecteurs aléatoires, a une précision plus ou moins élévée. Les vecteurs aléatoires viennent ensuite améliorer cette précision mais necessitent l'ajout de coefficients de la combinaison linéaire à stocker, ce qui diminue le gain de compression. Plus le pourcentage de vecteurs gardés est haut, plus vite on atteint un palier où l'ajout de vecteurs aléatoires n'a que très peu d'impact.

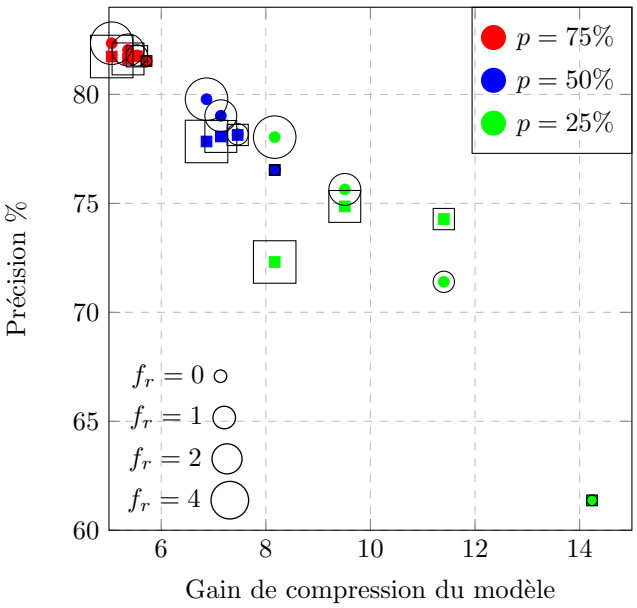

FIGURE 2 – VGG16 précision/compression, □ pour la stratégie de sélection et ◯ pour la stratégie d'orthogonalisation.

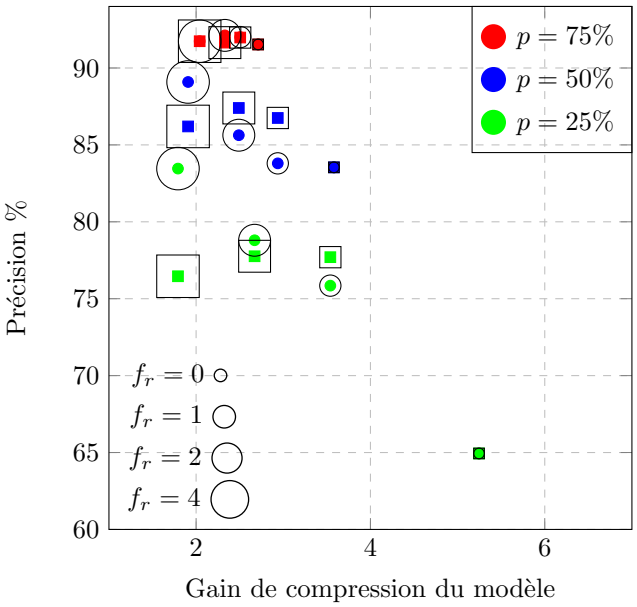

FIGURE 3 – ResNet50 précision/compression, □ pour la stratégie de sélection ◯ pour celle d'orthogonalisation.

En appliquant la méthode proposée par [6], on peut alors comparer les résultats avec notre méthode sur les mêmes CNNs. Pour VGG16, le point de référence est un gain de compression de 4.42 pour une précision de 82.75%. Notre méthode permet d'augmenter le gain de compression en conservant une valeur de précision proche. Elle peut même permettre de doubler le facteur de compression tout en perdant seulement 4% de précision. Pour ResNet50, le point de référence est un gain de compression de 2.19 et une précision de 92.47%. Notre méthode permet d'augmenter le taux de compression sur ce CNN, en proposant différents compromis suivant les besoins.

5 Conclusion

Au travers de cet article, nous avons combiné la réduction de dimensionnalité avec l'utilisation de vecteurs aléatoires pour obtenir une nouvelle méthode de compression. Nous avons réussi à réduire significativement la quantité de paramètres à stocker pour des réseaux de neurones convolutifs de l'état de l'art, cependant, le compromis est d'autant plus interessant que le réseau de départ est surdimensionné.

6 Perspectives

L'amélioration de la méthode proposée se base sur deux aspects. Le premier vise à obtenir une décomposition des filtres basée seulement sur des vecteurs aléatoires permettrait d'obtenir des meilleurs gains de compression. le deuxième concerne l'étude de la construction de l'ensemble des vecteurs aléatoires pour générer directement une base couvrant mieux le sous-espace vectoriel des poids et ainsi augmenter la précision du réseau compressé.

Références

[1] Guang-Bin Huang and Qin-Yu Zhu and Chee-Kheong Siew. *Extreme learning machine : a new learning scheme of feedforward neural networks*. IEEE International Joint Conference on Neural Networks, 2004.

[2] Kasun, Liyanaarachchi and Zhou, Hongming and Huang, Guang-Bin and Vong, Chi-Man. *Representational Learning with ELMs for Big Data*. IEEE Intelligent Systems, 2013.

[3] Huang, Guang-Bin and Bai, Zuo and Kasun, Liyanaarachchi Lekamalage Chamara and Vong, Chi Man. *Local Receptive Fields Based Extreme Learning Machine*. IEEE Computational Intelligence Magazine, 2015.

[4] Pang, Shan and Yang, Xinyi. *Deep Convolutional Extreme Learning Machine and Its Application in Handwritten Digit Classification*. Computational Intelligence and Neuroscience, 2016.

[5] Gallicchio, Claudio and Scardapane, Simone. *Deep randomized neural networks*. Recent Trends in Learning From Data, 2020.

[6] Brillet, Lucas Fernández and Mancini, Stéphane and Cleyet-Merle, Sébastien and Nicolas, Marina. *Tunable CNN Compression Through Dimensionality Reduction*. 2019 IEEE International Conference on Image Processing (ICIP), 2019.