Recherche d'Architecture Neuronale sous Contraintes Matérielles pour un accélérateur de calcul embarqué contraint en mémoire

Andrea CASTAGNETTI[1] Alain PEGATOQUET[1] Léo BERNARD[1] Benoît MIRAMOND[1] Olivier MONTFORT[2] Vincent HUARD[2]

[1]Université Côte d'Azur, LEAT, Sophia Antipolis, France

[2]Dolphin Design, Meylan, France

Résumé – Dans cet article, nous proposons une méthode de recherche d'architecture neuronale (NAS) tenant compte des contraintes matérielles, et axée sur les accélérateurs de réseaux neuronaux à mémoire locale. Notre solution est basée sur la minimisation des transferts de données vers et depuis l'accélérateur tout en maximisant le nombre d'opérations pour conduire à une meilleure efficacité des calculs. Nous introduisons un espace de recherche optimisé intégré dans un algorithme NAS fournissant des réseaux neuronaux qui améliorent la latence tout en minimisant la consommation d'énergie. Ces réseaux atteignent également une meilleure précision par rapport aux architectures embarquées conçues manuellement. Nos résultats expérimentaux, réalisés sur Imagenet, montrent une amélioration de la précision de 1.3% pour une gamme spécifique de latences par rapport à différentes versions de MobileNetV2 redimensionnées manuellement.

Abstract – In this paper we propose a Hardware-Aware Neural Architecture Search method focused on memory constrained neural networks accelerators. Our solution is based on the observation that minimizing data transfers to and from the accelerator while maximizing the number of operations can lead to efficient network design. We introduce an optimized search space integrated into a NAS algorithm providing networks that improve the latency while minimizing the energy consumption. Those networks also achieve a better accuracy when compared to manually designed mobile architectures. Our experimental results, performed on Imagenet, show that the accuracy can be improved by 1.3% for a specific range of latencies compared to different versions of MobileNetV2 rescaled by hand.

1 Introduction

Avec l'avènement de l'IA périphérique, les données peuvent être traitées à proximité des capteurs à l'aide d'algorithmes d'IA. Plusieurs techniques ont été proposées pour concevoir des architectures de réseaux de neurones artificiels (ANN) avec une faible empreinte mémoire ou améliorer l'efficacité des modèles d'apprentissage profond. Les architectures de réseaux neuronaux tels que MobileNet [7] et EfficientNet [8], ont été par exemple spécifiquement conçues pour les plateformes mobiles et à ressources limitées. D'autres techniques, telles que la compression de réseau [4] et le pruning [5], visent par ailleurs à réduire la complexité du réseau par la quantification des poids et des activations et par l'élagage des paramètres redondants. Ces techniques nécessitent toutefois de concevoir manuellement l'architecture de réseaux de neurones qui est ensuite optimisée pour son déploiement sur des dispositifs à ressources limitées. Ces dernières années, le Hardware-Aware Neural Architecture Search (HW-NAS) a été proposé pour concevoir automatiquement des architectures ANN efficaces et optimisées pour un ensemble de contraintes matérielles spécifiques [2]. Plusieurs solutions telles que MCUNet [3] ont été proposées pour les plateformes embarquées à mémoire limitée basées sur des microcontrôleurs (MCU). Dans ce travail nous considérons une plateforme de calcul embarquée composée d'un MCU et d'un accélérateur de réseaux de neurones convolutif (CNN) [6]. Pour cette classe particulière de plateformes embarquées, la taille de la mémoire n'est pas la seule contrainte. Il s'agit en effet aussi d'une question d'organisation de la mémoire. Ensuite, l'utilisation de la mémoire de l'accélérateur doit également être optimisée pour trouver l'architecture CNN la plus efficace sur ce matériel spécifique. Plus précisément, pour améliorer la latence globale et réduire la consommation d'énergie, le transfert des données entre l'accélérateur et le MCU doit être minimisé. Dans cet article, nous proposons une approche NAS tenant compte des contraintes matérielles pour l'accélérateur neuronal intégré Raptor développé par la société Dolphin Design [6]. Notre méthode est basée sur la recherche d'architecture neuronale one-shot [2] et utilise un espace de recherche optimisé dérivé de MobileNetV2 [7].

Les principales contributions de notre travail sont énumérées ci-dessous :

- Nous analysons l'espace de recherche de MobileNetV2 et mettons en évidence deux types de goulots d'étranglement au niveau de la mémoire.
- Nous proposons une optimisation de l'espace de recherche adaptée à la plateforme Raptor MCU + accélérateur CNN.
- Enfin, nous comparons diverses solutions recherchées avec des modèles MobileNetV2 redimensionnés manuellement et montrons que notre approche peut améliorer la précision de 1,3% pour une gamme spécifique de latences.

2 Méthodes

2.1 Architecture matérielle de l'accélérateur CNN Raptor

TinyRAPTOR est un accélérateur matériel programmable spécialisé dans l'inférence de réseaux neuronaux (NN) à très faible consommation [6]. Il peut réaliser jusqu'à 128 opérations MAC par cycle grâce à ses 128 éléments de traitement composés d'unités arithmétiques et logiques à 8 bits et d'unités de génération d'adresses dédiées. Il comprend un système de mémoire locale conçu pour réduire considérablement les transferts de données entre la mémoire du système et l'accélérateur. Cela permet d'augmenter le taux d'utilisation des éléments de traitement et de limiter le gaspillage d'énergie dû aux transferts mémoire. La nature entièrement programmable de Raptor lui permet d'être adapté aux exigences spécifiques de n'importe quelle application.

2.2 Analyse de l'espace de recherche : Goulots d'étranglement de la mémoire

Notre espace de recherche est basé sur l'architecture MobileNetV2 [7]. Cette architecture peut être adaptée à différents points de fonctionnement en modifiant certains hyperparamètres accordables. Comme dans [7], nous choisissons la résolution d'entrée R et le multiplicateur de largeur W_d comme hyperparamètres de compromis précision/performance. En outre, MobileNetV2 utilise la convolution séparable en profondeur (DSC), qui est un élément clé de nombreuses architectures de réseaux neuronaux efficaces [8]. Les convolutions DSC permettent de réduire le nombre total de paramètres ainsi que les coûts de calcul. Par contre, les réseaux qui utilisent les convolutions DSC présentent une disposition de la mémoire fortement déséquilibrée. À titre d'exemple, la figure 1 illustre la distribution des pics d'accès à la mémoire pour un modèle MobileNetV2 ayant une image d'entrée de taille 224×224 pixels. Comme on peut l'observer, le modèle présente une distribution de la mémoire des poids fortement déséquilibrée, puisque la plupart des poids sont alloués sur les six derniers blocs du réseau. La mémoire d'activation, quant à elle, suit une distribution opposée, les premiers blocs nécessitant de 4 à 10 fois plus de mémoire que les derniers blocs.

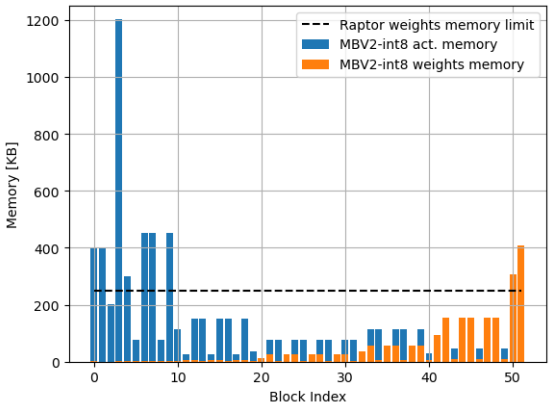

FIGURE 1 : Poids et activations de MobileNetV2 ($W_d = 1.$, $R = 224$), besoins en mémoire par bloc (quantification int8). Total de 3,5 Mo pour les poids.

Cette répartition particulière de la mémoire implique deux types de goulots d'étranglement de la mémoire. D'une part, la répartition de la mémoire d'activation entraîne des échanges de données importants entre l'unité de traitement neuronal (NPU) de l'accélérateur matériel et le MCU. C'est ce qui se produit lorsque la mémoire disponible sur la NPU n'est pas suffisante pour stocker l'ensemble des cartes d'activation. Dans ce cas, les cartes d'activation doivent être divisées en plusieurs parties qui sont ensuite traitées de manière séquentielle. En conséquence, la latence augmente puisque les transferts de données requièrent une part importante des cycles du système. La distribution de la mémoire des poids, d'autre part, limite la taille de la plus grande couche qui peut tenir dans la mémoire du NPU, qui est indiquée par la ligne pointillée sur la figure 1. Comme on peut le voir, la mémoire nécessaire pour stocker les poids des deux derniers blocs dépasse la taille mémoire disponible du NPU. Pour faire face à ces problèmes, la plupart des travaux proposent de rédimensionner l'architecture en ajustant le multiplicateur W_d [3] [8]. Cependant, un rédimensionnement uniforme du réseau [3] entraîne une dégradation des performances. Dans les sections suivantes, nous proposons une approche différente de rédimensionnement du réseaux qui permet de résoudre le problème lié aux deux goulots d'étranglement de la mémoire, améliorant ainsi le compromis global efficacité/précision.

2.3 Mesures d'efficacité matérielle

Dans cette section, nous définissons les paramètres utilisés pour concevoir l'espace de recherche. Pour l'estimation des performances, le nombre d'opérations de multiplication et d'accumulation (MAC) est utilisé comme indicateur de la précision, une relation qui a été établie dans plusieurs travaux antérieurs [8] [3]. Cependant, maximiser uniquement le nombre de MAC fourni par le réseau n'est pas une condition suffisante pour garantir une utilisation efficace de la NPU. Pour prendre en compte les mouvements de données, nous introduisons une métrique d'efficacité, appelée μ, qui est définie comme suit :

$$\mu = \frac{NPU_{cycles}}{SYS_{cycles}} \tag{1}$$

Où NPU_{cycles} représente le nombre total de cycles exécutés par le NPU (dont la plupart sont des opérations MAC), et SYS_{cycles} mesure les cycles exécutés par le MCU (principalement des transferts de données). La mesure d'efficacité définie dans l'équation 1 est quelque peu liée à la notion d'intensité arithmétique, c'est-à-dire le rapport entre les opérations et le nombre total de mouvements de données. Une valeur élevée de μ signifie en effet que l'espace de recherche sélectionné maximise le calcul pour une empreinte mémoire donnée. Dans la section suivante, nous montrons comment la métrique d'efficacité μ peut être maximisée pour améliorer le compromis efficacité/précision.

2.4 Optimisation de l'espace de recherche : Rédimensionnement progressif

Trouver un compromis optimal entre la précision et l'efficacité est un problème complexe. Un exemple est présenté dans le Tab. 1, où nous mesurons à la fois les opérations MAC et

l'efficacité pour différentes configurations de résolution R et de largeur W_d de MobileNetV2.

TABLE 1 : Compromis de MobileNetV2 sur Raptor. Le rédimensionnement uniforme de MobileNetV2 dans l'espace (R, W_d) produit presque deux points de fonctionnement orthogonaux. La configuration (224, 0.6) maximise la précision, tandis que la configuration (96, 0.4) maximise l'utilisation de Raptor.

	$\mathbf{W_d}$					
	0.4		0.5		0.6	
	MACs	μ	MACs	μ	MACs	μ
R						
96	18.4M	0.174	21.5M	0.166	31M	0.14
160	46.6M	0.112	55M	0.115	84M	0.102
224	89M	0.07	105M	0.08	162M	0.071

D'après le Tab. 1, on peut observer que les configurations qui présentent un nombre élevé d'opérations MACs ont une faible efficacité. Pour augmenter le nombre d'opérations, il est en effet nécessaire d'augmenter la taille de la mémoire pour stocker à la fois les poids et les activations. Cependant, cette augmentation de la mémoire entraîne davantage de transferts de données, ce qui réduit l'efficacité.

Pour résoudre ce problème, nous proposons une mise à l'échelle non uniforme de MobileNetV2 que nous appelons *mise à l'échelle progressive*. Contrairement à la méthode de mise à l'échelle existante basée sur un multiplicateur de largeur unique W_d pour l'ensemble de l'architecture, la mise à l'échelle progressive permet à chaque couche d'avoir un multiplicateur de largeur différent W_d^i situé dans l'intervalle $[0.3 - 1.5]$ avec un pas de 0.3.

Cette approche nous permet de couvrir un large spectre de configurations, à la fois en termes de mémoire et d'opérations. Ensuite, nous utilisons la recherche par grille (i.e. *grid search*) pour trouver les configurations de W_d^i qui maximisent à la fois l'efficacité μ et le nombre d'opérations MAC, pour une résolution donnée R.

TABLE 2 : Architectures MobileNetV2 avec mise à l'échelle progressive de la largeur des couches. (*t*=facteur d'expansion, *n*=profondeur du bloc, *s*=couche)

Operator	t	c	W_d	n	s
conv2d 3×3	-	32	0.3	1	2
bottleneck	1	16	0.3	1	1
bottleneck	6	24	0.3	2	2
bottleneck	6	32	0.3	3	2
bottleneck	6	64	1.5	4	2
bottleneck	6	96	1.	3	1
bottleneck	6	160	0.9	3	2
bottleneck	6	320	0.6	1	1
conv2d 1×1	-	1280	0.6	1	1
avgpool	-	-	-	1	-
Linear	-	1K	-	-	-

Le tableau. 2 montre les configurations obtenues pour une résolution d'entrée $R = 224$. La configuration donnée dans le Tab. 2 définit l'espace de recherche d'entrée de l'algorithme NAS qui est décrit dans les sections suivantes.

2.5 NAS pour la spécialisation du modèle

Sur la base de l'espace de recherche proposé à la section 2.4, nous utilisons la technique NAS one-shot proposée dans [2] pour effectuer la spécialisation du modèle. Cette technique NAS est basée sur un super-réseau, entrainé une seule fois, qui contient tous les sous-réseaux spécialisés. Plus précisément, les sous-réseaux sont dérivés de la structure des réseaux présentée dans le tableau. 2, où les facteurs d'expansion t, la profondeur de chaque couche n et la taille des noyaux convolutifs k peuvent être modifiés pour chaque bloc. Notre espace de recherche est configurable comme suit : $t \in [3, 4, 6]$, $n \in [2, 3, 4]$, $k \in [3, 5]$. Nous permettons également à la résolution d'entrée R de varier dans l'intervalle $[128, 160, 192, 224]$.

2.6 Recherche par algorithme évolutionnaire

Un algorithme de recherche évolutionnaire [1] est appliqué afin de trouver un sous-réseau optimisé parmi le super-réseau. Les sous-réseaux sont classés en fonction de leur nombre respectif de MAC, c'est-à-dire NPU_{cycles}. A chaque itération, la population de sous-réseaux courante est divisée en deux, une moitié évolue par mutation et l'autre par croisement. La recherche d'évolution est effectuée pendant 500 itérations.

3 Résultats expérimentaux

3.1 Jeux de données et détails de l'entraînement

Nous entraînons le super-réseau décrit à la section 2.5 sur Imagenet en utilisant SGD (*Stochastic Gradient Descent*) avec un momentum de 0.9 et une décroissance des poids de $3 \cdot 10^{-5}$. Le taux d'apprentissage initial est fixé à $3 \cdot 10^{-2}$ et évolue selon une décroissance cosinus. Un batch size de 64 est utilisé et le modèle est entraîné pour 450 époques. L'entraînement nécessite environ 250 heures en utilisant quatre GPU A40.

3.2 NAS sous contraintes matérielles

Nous effectuons des recherches NAS pour trouver des sous-réseaux efficaces qui répondent à une contrainte de ressource spécifique. Ici, nous nous concentrons sur la latence et nous recherchons des sous-réseaux efficaces dans la plage $[30, 90]$ ms. Les solutions trouvées par le NAS, appelées *RaptorNet*, sont ensuite testées sur l'ensemble de validation Imagenet pour calculer la précision. Le compromis latence/précision est illustré à la figure 2.

Nous présentons également dans la Fig. 2 les résultats obtenus avec deux modèles MobileNetV2 redimensionnés manuellement ayant un W_d de 0.5 et 0.35, et pour trois résolutions d'entrée différentes. Seules deux valeurs de W_d sont considérées car des multiplicateurs plus élevés ne sont pas supportés par le NPU Raptor pour des raisons de limitations mémoire. Les temps de latence de ces modèles ont été mesurés sur la plateforme Raptor afin d'obtenir une comparaison équitable. Comme on peut l'observer, et pour des latences supérieures à

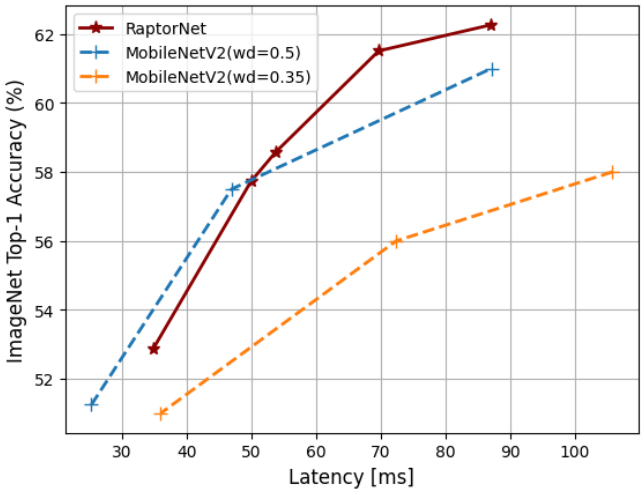

FIGURE 2 : Compromis entre précision et latence sur Imagenet. RaptorNet améliore la précision par rapport à MobileNetV2 pour des latences comprises entre 50 et 100 ms.

50 ms, les sous-réseaux trouvés par le NAS fournissent une meilleure précision par rapport aux modèles rédimensionnés manuellement. À 87 ms par exemple, notre solution fournit une précision supérieure de 1,3% à celle de MobileNetV2 ($W_d = 0.5$). En dessous de 50 ms, nos solutions sont presque équivalentes à MobileNetV2 ($W_d = 0.5$), mais significativement meilleures que des réseaux rédimensionnés avec un multiplicateur W_d de 0.35. Les détails des différents points de fonctionnement sont donnés dans le Tab. 3. Nous indiquons les points de précision les plus élevés pour MobileNetV2 redimensionné manuellement ainsi que les deux points de précision les plus élevés pour RaptorNet.

TABLE 3 : Comparaison de la latence, de l'énergie et de l'efficacité entre RaptorNet, MobileNetV2 et MCUNet [3]. Les résultats de RaptorNet et MobileNet se réfèrent à un déploiement sur Raptor NPU cadencé à 200 MHz. Les résultats de MCUNet se rapportent à un déploiement sur un MCU STM32F412 cadencé à 216 MHz. Les meilleurs résultats sont mis en évidence en gras.

Network	**Latency**	**Energy**	μ	**Acc.**
RaptorNet	87 ms	889 μJ	0.24	**62.3**
RaptorNet	**69.7** ms	**712.7** μJ	**0.26**	61.4
MobileNetV2 ($W_d = 0.5$)	87.1 ms	891 μJ	0.122	61
MobileNetV2 ($W_d = 0.35$)	106 ms	1082 μJ	0.087	58
MCUNet [3] (STM32F412)	200 ms	30240 μJ	N.A.	49.9

Comme on peut le constater, les deux réseaux trouvés par le NAS sont toujours plus performants que les réseaux redimensionnés manuellement. Par exemple, par rapport à MobileNetV2 ($W_d = 0.5$), la deuxième meilleure solution de RaptorNet améliore la précision de 0.4 % tout en réduisant d'environ 20 % la latence et l'énergie. En outre, grâce au rédimensionnement progressif utilisé pour optimiser l'espace de recherche, le NAS est en mesure de découvrir des solutions nettement plus performantes en termes d'efficacité du NPU. En effet, les deux solutions RaptorNet offrent une efficacité μ supérieure de 50 % à celle d'un MobileNet standard, ce qui suggère un ratio plus élevé d'opérations par rapport au nombre total de mouvements de données.

4 Conclusion

La conception d'architectures ANN efficaces pour des cibles embarquées à mémoire limitée pose de nombreux défis. La recherche d'architecture neuronale tenant compte des contraintes matérielles (HW-NAS) est une solution pour concevoir des réseaux neuronaux optimisés pour un ensemble de contraintes matérielles spécifiques avec une intervention humaine limitée. Dans cet article, nous avons montré que pour tirer parti de la recherche d'architecture neuronale, il est nécessaire de prendre en compte les caractéristiques du matériel au plus tôt et d'optimiser l'espace de recherche en conséquence. Sur la base de ces observations, nous avons conçu un espace de recherche optimisé pour l'accélérateur de réseau neuronal Raptor dans le but de minimiser les mouvements de données entre le NPU et le MCU. L'approche proposée permet d'améliorer la précision tout en réduisant la latence, et donc la consommation d'énergie globale.

Références

[1] E. Real et AL. : Regularized evolution for image classifier architecture search. *In Proceedings of the Thirty-Third AAAI Conference*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[2] H. Cai et AL. : Once for All : Train One Network and Specialize it for Efficient Deployment. *In Eighth International Conference on Learning Representations*, avril 2020.

[3] J. Lin et AL. : MCUNet : tiny deep learning on IoT devices. *In Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, pages 11711–11722, Red Hook, NY, USA, décembre 2020. Curran Associates Inc.

[4] Steven K. Esser et AL. : Learned Step Size Quantization. Rapport technique arXiv :1902.08153, arXiv, mai 2020. arXiv :1902.08153 [cs, stat] type : article.

[5] Yihui HE, Xiangyu ZHANG et Jian SUN : Channel Pruning for Accelerating Very Deep Neural Networks. *In 2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, Venice, octobre 2017. IEEE.

[6] Vincent HUARD : Transforming Far-Edge Computer Vision With Energy-Efficient AI, 2023.

[7] Mark SANDLER et AL. : MobileNetV2 : Inverted Residuals and Linear Bottlenecks, mars 2019. arXiv :1801.04381 [cs].

[8] Mingxing TAN et Quoc V. LE : EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks, septembre 2020. arXiv :1905.11946 [cs, stat].