Évaluation de deux architectures matérielles dédiées à l'inférence basée sur des réseaux de neurones convolutifs

Guillaume Devic, Abdoulaye Gamatié, Gilles Sassatelli

Compas'2020 : Parallélisme / Architecture/ Système/ Temps Réel
MILC - Lyon, France, du 30 juin au 3 juillet 2020

Évaluation de deux architectures matérielles dédiées à l'inférence basée sur des réseaux de neurones convolutifs

Guillaume Devic, Abdoulaye Gamatié, Gilles Sassatelli

Université de Montpellier / CNRS,
LIRMM - 161 rue Ada, 34095 Montpellier, Cedex 5 - France
prénom.nom@lirmm.fr

Résumé
Les systèmes embarqués sont de plus en plus utilisés pour exécuter des algorithmes d'intelligence artificielle, qui requièrent beaucoup de ressources matérielles, notamment en mémoire. Ces systèmes étant généralement dotés de ressources limitées, des architectures dédiées commencent à émerger, à l'image de celles proposées dans les deux cartes étudiées dans cet article : GAPuino et Coral Dev Board. Ces dernières s'appuient sur un accélérateur d'algorithmes d'inférence sous forme de réseau de neurones convolutifs (acronyme anglais, CNN). Nous évaluons ces deux cartes en implémentant un CNN entrainé avec la célèbre base de données MNIST. Nous étudions l'impact des architectures mémoires implantées dans les accélérateurs d'inférence, sur la performance et la puissance consommée par l'exécution des CNN.

Mots-clés : Intelligence artificielle ; Réseaux de neurones convolutifs ; Inférence ; Système embarqué ; Architecture mémoire ; GAPuino / GAP8 ; Coral Dev Board / EdgeTPU

1. Introduction

Les objets connectés ont connu une rapide expansion. Ce sont des systèmes embarqués composés de divers capteurs, qui génèrent beaucoup de données généralement centralisées et traitées par des serveurs puissants disponibles dans le"nuage" (paradigme de *cloud computing*). Le traitement de ces données s'effectue avec des algorithmes d'intelligence artificielle (IA), tel que l'apprentissage automatique, permettant d'extraire de la connaissance. L'évolution des objets connectés entraîne de nouveaux usages qui ne peuvent se satisfaire des limitations en sécurité et en bande passante réseau induites par l'utilisation de serveurs distants. Ils doivent avoir la capacité de traiter localement leurs données ou au moins une partie [21, 24, 25, 30].
L'apprentissage automatique sur des systèmes embarqués se confronte aux limitations en ressources matérielles intégrées. En effet, les systèmes embarqués sont souvent soumis à des contraintes de consommation d'énergie et de performance de calcul notamment. Les algorithmes d'apprentissage automatique et d'inférence, notamment basés les réseaux de neurones convolutifs (ou *Convolutional Neural Networks* - CNN) demandent une quantité importante de mémoires et de puissance de calcul. Ces algorithmes sont généralement composés de deux phases : une phase d'*entraînement* et une phase d'*inférence*. La phase d'entraînement traite un grand volume de données pour définir de bons paramètres de réseaux. Ces paramètres permettront une exploitation pertinente du réseau en phase d'inférence. L'inférence est souvent la seule phase gérée par le système embarqué, phase qui s'avère relativement moins gourmande en puissance de calcul et quantité de mémoire.

Rapide état de l'art. Une des voies possibles pour adresser la mise en oeuvre de l'apprentissage automatique dans les systèmes embarqués est l'adoption d'architectures multicoeurs hétérogènes. C'est le cas de la carte Nvidia Jetson Nano [8] qui combine un processeur graphique 128-coeurs Maxwell et un processeur ARM 4-coeurs A57. Une autre approche consiste à utiliser des accélérateurs dédiés à l'apprentissage automatique comme Intel Movidius Myriad 2 [26], Eyeriss V2 [15] et ShiDianNao [18]. Les accélérateurs offrent un meilleur compromis entre rapidité d'exécution et consommation énergétique. Ces architectures conservant de nombreuses contraintes, leurs utilisations se limitent généralement à l'inférence. Les tâches d'entraînement de réseaux de neurones artificiels sont réalisées en amont, sur des serveurs équipés de GPU.

Au-delà de ces approches orientées architecture matérielle, une voie alternative repose sur l'application de techniques d'optimisation de réseaux de neurones, telles que le pruning ou la quantification [27]. Le pruning est basé sur une suppression progressive des connexions entre les neurones d'un réseau. Cela permet une réduction du nombre de calculs à effectuer d'une couche à l'autre. La quantification quant à elle est basée sur une réduction du nombre de bits d'une représentation binaire des paramètres d'un réseau, par exemple de 32 bits à 8 bits. Cela offre une accélération de l'exécution, au prix d'une réduction tolérable de la précision des calculs. Cette technique se rapproche du concept de calcul approximatif [31] qui vise un compromis entre précision et coût de calcul.

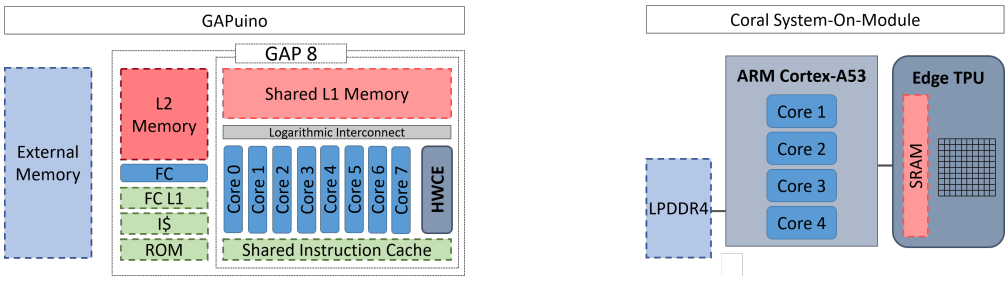
Notre contribution. Dans cet article, nous implémentons un algorithme d'inférence basé sur des CNN en ciblant deux plateformes embarquées, à savoir la carte GAPuino [6] de la société Greenwaves [7] et de la carte Coral Dev Board [2] de Google. Ces dernières sont principalement destinées à exécuter des algorithmes d'apprentissage automatique pour l'inférence. Elles intègrent chacune un accélérateur de CNN. Dans le cas de la Coral Dev Board, l'accélérateur est externe au CPU tandis que pour la Gapuino il est au même niveau que les coeurs. Notre objectif est de comparer les deux plateformes, afin de comprendre l'impact de leurs choix architecturaux, et particulièrement leur organisation mémoire.

2. Approche de l'étude

La mémoire joue un rôle central dans la performance et l'efficacité énergétique des systèmes embarqués dédiés aux tâches d'intelligence artificielle. Elle requiert une organisation et un dimensionnement adéquats pour le stockage des paramètres de réseaux de type CNN, ainsi que des données exploitées (par exemple des images). Ce type de réseau exerce une pression significative sur le sous-système mémoire, de par la nécessité de réaliser un nombre très important d'opérations de convolutions sur l'image dont la taille dépasse typiquement les capacités des caches du système. L'organisation classique de la mémoire repose sur une hiérarchie comprenant des niveaux de caches jusqu'au stockage en passant par la mémoire principale. Plus on s'éloigne des CPU plus les temps d'accès deviennent coûteux. Dans ce contexte, un des enjeux consiste à exploiter au mieux la localité des données. La gestion des mémoires caches pouvant être complexe, d'autres schémas d'organisation de mémoires associent aux CPU des blocs scratchpad, qui rendent les accès aux données plus rapides et prédictibles.

Les systèmes étudiés dans cet article adoptent des organisations différentes concernant la mémoire (cf. Figure 1). La carte GAPuino [19] embarque un processeur dédié, le GAP8, composé d'un cluster principal de huit coeurs RISC-V où chaque coeur partage la mémoire scratchpad L1 de 64Ko (cf. Figure 1a). La mémoire scratchpad L2 de 512Ko est à l'extérieur de cluster contrôlé par un neuvième coeur RISC-V, nommé *Fabric controller* (FC). Ce dernier assure les fonctions classiques d'un MCU [5] tel que les fonctions de contrôle, communications et sécurité. L'accélérateur est contenu dans le cluster avec un accès direct à la mémoire scratchpad

L1. La carte Coral Dev Board, contrairement à la GAPuino, a la capacité d'exécuter Linux [14] grâce au processeur de quatre coeurs A53 de ARM. Sur cette carte, l'accélérateur est un ASIC dit Edge TPU [4] développé par Google (cf. Figure 1b). C'est une version allégée du TPU [1] dédiée principalement aux tâches d'inférence. Il contient une mémoire SRAM de 8 Mo pour stocker des réseaux de neurones. Cette mémoire scratchpad est chargée via le processeur ARM. Une mémoire LPDDR4 est accessible aussi pour étendre les capacités mémoire de l'accélérateur.



(a) GAPuino (b) Coral Dev Board

FIGURE 1 – Architectures des deux cartes électroniques évaluées

Par la suite, nous évaluons les performances des deux cartes en considérant un benchmark typique de réseaux CNN. Nous nous basons sur la variation de certains paramètres de CNN, à savoir le nombre de couches et de filtres de convolutions. L'idée est de comprendre l'impact de l'organisation mémoire choisie dans chaque cas, tout en explorant de possibles pistes logicielles d'amélioration des performances. En particulier, nous considérons la technique du Max-Pooling. Ces deux architectures étant implantées sous forme d'ASIC, nous ne pouvons envisager leur modification matérielle, contrairement à des supports sur FPGA [20,29] ou définis à l'aide de modèles cycle-précis et flexibles [17].

3. Expérimentations

Nous utilisons un modèle de CNN de la base de données MNIST [12]. Ce modèle est composé de deux couches successives de convolutions à deux dimensions (Conv2D), comme illustré dans la Figure 2. La première couche reçoit une image de dimension 28x28 issue de la base de données MNIST [11, 22], contenant des images de chiffres manuscrits en noir et blanc. Cette couche est constituée de 32 noyaux de convolution de dimensions 5x5. Soixante mille images servent à l'entraînement des réseaux et dix mille au test. En plus des couches de convolutions, le réseau comporte une couche de Max-Pooling permettant une réduction de la carte de caractéristiques (*feature map* en anglais), une couche *Flatten* permettant d'aplanir les données 2D en un vecteur 1D requis par les deux dernières couches *Denses*.

Conv2D(32,(5x5)
Conv2D(64,(5x5)
MaxPooling2D
Flatten
Dense(128)
Dense(10)

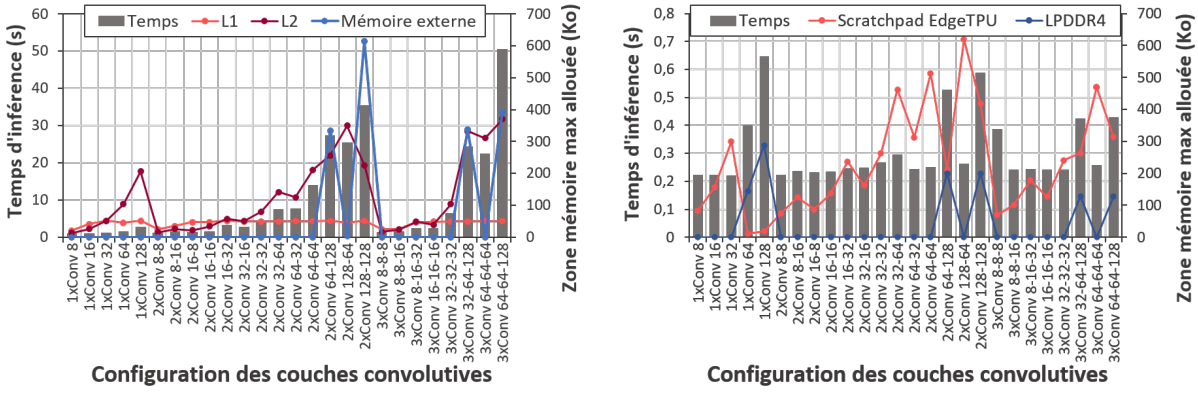
FIGURE 2 – CNN

Ce réseau est entraîné sur un PC tiers via l'API Keras [9] et la libraire Tensorflow [13] écrits en Python. Une fois l'apprentissage terminé, le réseau résultant subit une quantification 8bits ou 16bits et est sauvegardé. Son exécution sur la Coral Dev Board nécessite une étape de compilation [3] afin de déterminer les tâches à exécuter par l'accélérateur EdgeTPU. Le reste sera réservé au CPU. Pour la GAPuino un traitement similaire est effectué avec génération d'un code C compilable et exécutable par la carte.

Nos expérimentations consistent à exécuter sur chaque carte des variantes du modèle de réseau de neurones décrit dans la Figure 2, soit en répliquant soit en supprimant les couches de convolution uniquement. Ce sont ces couches qui ont l'impact le plus important sur le temps

d'exécution des réseaux. De plus, nous faisons varier leur nombre de filtres internes. Pour expliquer la notation utilisée dans les figures de cette section, un modèle à N couches de convolution respectivement de C1, C2, ..., CN filtres est noté "NxConv C1-C2-...-CN". Par exemple, un réseau ayant 2 couches de convolution respectivement de 32 et 64 filtres sera noté "2xConv 32-64". Les autres couches du réseau n'étant pas modifiées, elles ne figurent pas dans la notation. Un appareil d'acquisition de National instrument [10] est utilisé pour évaluer les différentes exécutions de réseaux sur les cartes.

Corrélation entre temps d'inférence et accès mémoires. Nous nous intéressons au lien entre le temps d'inférence et les mémoires utilisées pour stocker le réseau exécuté. Comme illustré sur la Figure 3), le temps d'inférence entre les variantes de réseaux varie différemment selon la carte. Sur la GAPuino, cette variation atteint un facteur 40 entre le temps d'inférence le plus court, i.e. réseau "1xConv 8", et le plus long, i.e. réseau "3xConv 64-64-128" (Figure 3a). Nous remarquons tout de même que les configurations "2xConv 64-128", "2xConv 128-64", "2xConv 128-128", "3xConv 32-64-128","3xConv 64-64-64" et "3xConv 64-64-128" ont un temps d'inférence nettement supérieur à la moyenne de 218%. D'autre part, la zone mémoire à allouer augmente logiquement avec la taille du réseau résultant de la configuration choisie. Les configurations présentant un nombre élevé de filtres requièrent la mémoire externe à la puce.



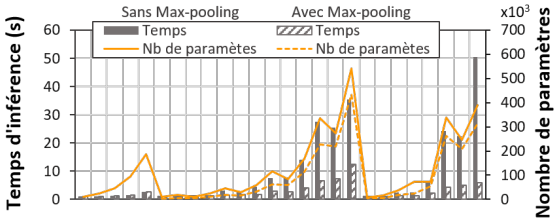
(a) Exécution sur la carte GAPuino.

(b) Exécution sur la Coral Dev Board.

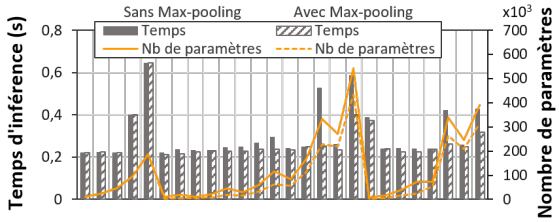
FIGURE 3 – Évolution du temps d'inférence (barres grisées). Les courbes dénotent la taille max des zones mémoire allouées dans la hiérarchie mémoire pour stocker les réseaux sur les cartes.

Sur la Coral Dev Board, les variations du temps d'inférence entre les configurations sont moins importantes que sur la GAPuino. La puce EdgeTPU de la Coral Dev Board profite d'une mémoire interne de 8 Mo qui accélère l'accès aux données. Cependant, lorsque la charge de travail d'un réseau nécessite des accès à la LPDDR4 externe à la puce, cela impacte lourdement le temps d'inférence (jusqu'à 60% par rapport à la majorité des configurations sur puce). Le choix d'utiliser la LPDDR4 est défini par le compilateur du EdgeTPU dans les configurations "1xConv 64", "1xConv 128", "2xConv 64-128", "2xConv 128-128", "3xConv 32-64-128" et "2xConv 64-64-128". Ce résultat interpelle en ce sens qu'il semble que la mémoire interne à la puce n'est pas complètement allouée. Il y aurait donc une marge d'optimisation possible à ce niveau.
De manière globale, on note que la présence d'une mémoire dédiée contribue aux bonnes performances de la Coral Dev Board. Aussi, on peut se demander si une augmentation des mémoire ou bien de l'ajout d'une mémoire *ad hoc* améliorerait les performances de la GAPuino. Étant donné que cette dernière est sous forme ASIC, une telle exploration est hors de portée dans notre étude. Nous privilégions donc une piste logicielle, consistant à adapter les réseaux de façon à prendre en compte les contraintes de l'architecture mémoire de la GAPuino notam-

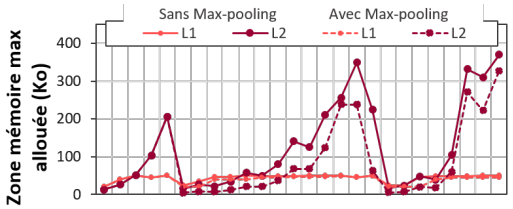
ment. Lors de l'exécution d'un réseau de neurones, la mémoire doit stocker les paramètres du réseau (i.e. poids et biais) et les données se trouvant en entrée et en sortie de chaque couche du réseau. L'ajout d'une couche de Max-Pooling permet de réduire la taille des données, diminuant par la même occasion le nombre de paramètres du réseau.



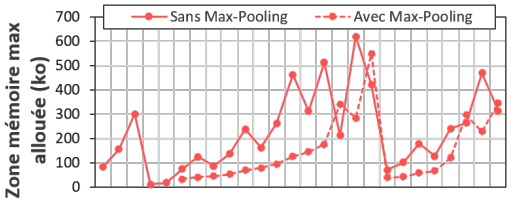
(a) Évolution du temps d'inférence et du nombre de paramètres de chaque CNN sur la GAPuino.



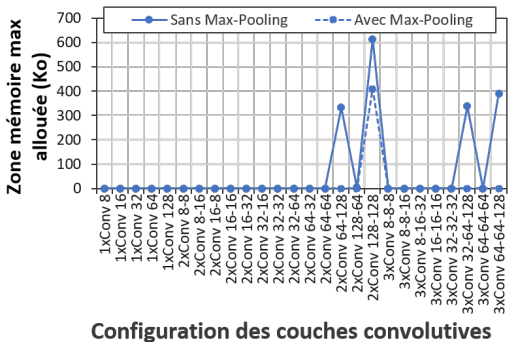
(b) Évolution du temps d'inférence et du nombre de paramètres de chaque CNN sur la Coral Dev Board.



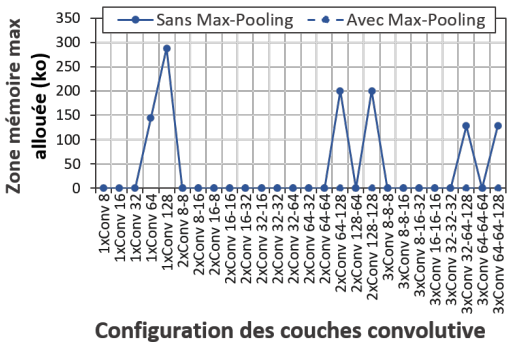
(c) Zone mémoire allouée max sur les niveaux mémoire L1 et L2 de la GAPuino.



(d) Zone mémoire allouée max sur la mémoire scratch-pad de la Coral Dev Board.



(e) Zone mémoire allouée max sur la mémoire externe au processeur GAP8 de la GAPuino.



(f) Zone mémoire allouée max sur la mémoire externe de la Coral Dev Board.

FIGURE 4 – Évaluation de configurations en présence de Max-Pooling.

Effets du Max-Pooling sur le temps d'inférence. L'objectif de la fonction de pooling est de réduire la taille de la carte de caractéristiques issue d'une couche convolutive. Le principe est de définir un filtre qui condense un groupe d'éléments de la carte. Le Max-Pooling retient la plus grande valeur de ces éléments. Il réduit ainsi la taille des données traitées par le réseau, sans endommager la précision de ce dernier, tout en favorisant une réduction du sur-apprentissage. Ici, nous ajoutons une seule couche supplémentaire de MaxPooling2D au début des variantes de réseaux évalués. Dans les Figures 4a et 4b, le nombre de paramètres diminue de 15% à 69% avec une couche de Max-Pooling, réduisant le temps d'exécution de 5% à 88% pour la GAPuino et de 0.5% à 50% pour la Coral Dev Board. Cette amélioration du temps d'inférence est double : moins de paramètres signifient moins de calculs à effectuer, et moins de paramètres signifient moins d'éléments à stocker et à solliciter en mémoire. Dans les Figures 4c, 4d, 4e et 4f nous observons que les niveaux de mémoire les plus éloignés du CPU sont globalement moins

sollicités. Cependant, seulement la configuration "2xConv 128-128" de la GAPuino poursuit l'utilisation de la mémoire externe avec une diminution des accès de 33%. Et quatre configurations de la Coral Dev Board montrent une augmentation jusqu'à 45% de la mémoire scratchpad allouée sur la puce EdgeTPU, avec l'abandon de l'utilisation de la mémoire LPDDR4.

Évaluation de la puissance consommée par l'exécution sur carte. La Figure 5 présente la puissance dynamique et l'énergie consommée pour les variantes de réseaux sans Max-Pooling. Ici, la puissance dynamique est définie par la différence entre les puissances consommées "à vide" et pendant l'inférence. Son évaluation repose sur le même principe que [16]. La GAPuino n'exécutant pas de système d'exploitation, sa consommation à vide est proche de la consommation statique de la carte. Cela n'est pas le cas de la Coral Dev Board qui comporte un système d'exploitation. La puissance varie davantage dans la GAPuino en comparaison à la Coral Dev Board. En moyenne, elle est supérieure de 400% et 18% pour les 2 cartes respectives par rapport à la variante de réseau la plus économe, i.e. 1xConv8. L'énergie suit des tendances similaires.

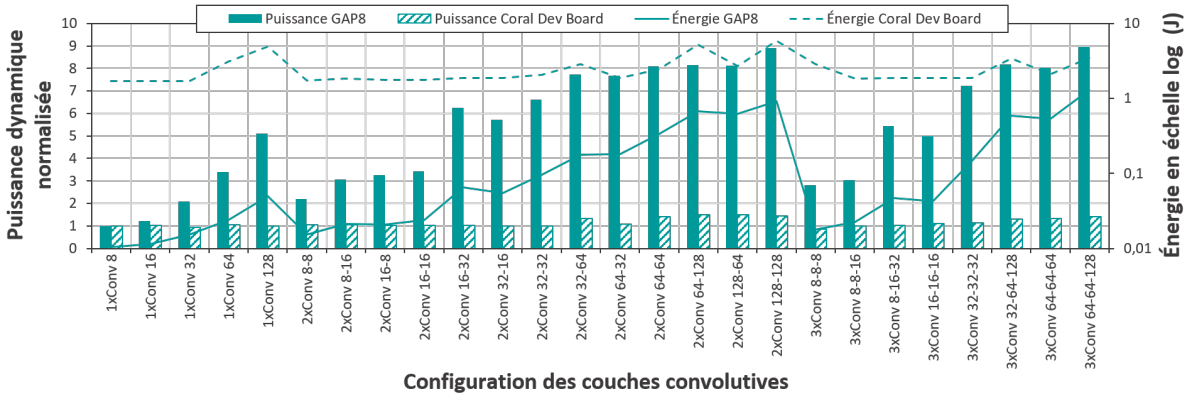


FIGURE 5 – Puissance dynamique et énergie consommée durant l'inférence sans Max-Pooling

Par ailleurs, à travers les variations des puissances nous observons que les cartes n'utilisent qu'une partie de leurs ressources matérielles pour des petits réseaux. Cela est surtout flagrant pour la GAPuino, dont la hiérarchie mémoire est sollicitée davantage dans l'exécution de réseaux de taille importante. La Coral Dev Board quant à elle montre une puissance dynamique variant peu. Cela est dû au dimensionnement de la mémoire scratchpad intégrée à la puce EdgeTPU, permettant de traiter *in-situ* la plupart des réseaux évalués.

4. Conclusion

Nous avons évalué les cartes GAPuino et Coral Dev Board, dédiées à des tâches d'inférence, basée sur des réseaux CNN. Ces cartes s'appuient sur des accélérateurs matériels. Leurs performances sont grandement influencées par la localité des données en mémoire. L'ajout d'une couche de Max-Pooling dans les réseaux a permis de réduire les pénalités liées aux accès mémoires coûteux, améliorant les performances des réseaux. L'organisation de la hiérarchie mémoire exploitée par les différents accélérateurs intégrés semble avoir également un impact sur les puissances et énergies consommées par les deux cartes. En particulier, le schéma basé sur la mémoire scratchpad dans le EdgeTPU offre une opportunité d'optimisation à explorer, afin d'obtenir une dissipation davantage proportionnelle à la complexité des réseaux. Une autre direction à explorer quant à l'amélioration de la consommation énergétique concerne l'intégration adaptée de technologies de mémoires non-volatites émergentes [23, 28].

Bibliographie

1. Cloud tpu : L'entraînement et l'exécution des modèles de machine learning n'ont jamais été aussi rapides. – https://cloud.google.com/tpu. Accessed : 04/03/2020.
2. Coral dev board. – https://coral.ai/products/dev-board/. Accessed : 04/03/2020.
3. Edge tpu compiler. – https://coral.ai/docs/edgetpu/compiler/. Accessed : 04/03/2020.
4. Edge tpu : Puce asic conçue sur mesure par google pour exécuter des inférences en périphérie. – https://cloud.google.com/edge-tpu/. Accessed : 04/03/2020.
5. Gap8 hardware reference manual. – https://gwt-website-files.s3.amazonaws.com/gap8_datasheet.pdf. Accessed : 04/03/2020.
6. Gapuino gap8 development board. – https://greenwaves-technologies.com/product/gapuino/. Accessed : 04/03/2020.
7. Greenwaves technologies. – https://greenwaves-technologies.com/. Accessed : 04/03/2020.
8. Jetson nano developer kit. – https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Accessed : 04/03/2020.
9. Keras : The python deep learning library. – https://keras.io/. Accessed : 04/03/2020.
10. Matériel d'acquisition de données multifonction. – https://www.ni.com/fr-fr/support/model.usb-6009.html. Accessed : 04/03/2020.
11. The mnist database of handwritten digits. – http://yann.lecun.com/exdb/mnist/. Accessed : 04/03/2020.
12. A simple convnet on the mnist dataset. – https://keras.io/examples/mnist_cnn/. Accessed : 04/03/2020.
13. Tensorflow : An end-to-end open source machine learning platform. – https://www.tensorflow.org/. Accessed : 04/03/2020.
14. What is mendel linux? – https://coral.googlesource.com/docs/+/refs/heads/master/ReadMe.md. Accessed : 04/03/2020.
15. Chen (Y.), Emer (J. S.) et Sze (V.). – Eyeriss v2 : A flexible and high-performance accelerator for emerging deep neural networks. *CoRR*, vol. abs/1807.07928, 2018.
16. da Silva (J. C. R.), Pereira (F. M. Q.), Frank (M.) et Gamatié (A.). – A compiler-centric infrastructure for whole-board energy measurement on heterogeneous android systems. – In Niar (S.) et Saghir (M. A. R.) (édité par), *13th Int'l Symp. on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC'18, Lille, France, July 9-11, 2018*, pp. 1–8. IEEE, 2018.
17. Delobelle (T.), Péneau (P.-Y.), Gamatié (A.), Bruguier (F.), Senni (S.), Sassatelli (G.) et Torres (L.). – Magpie : System-level evaluation of manycore systems with emerging memory technologies. – In *2nd International Workshop on Emerging Memory Solutions - Technology, Manufacturing, Architectures, Design and Test at Design Automation and Test in Europe (DATE'2017), Lausanne, Switzerland*, March 2017.
18. Du (Z.), Fasthuber (R.), Chen (T.), Ienne (P.), Li (L.), Luo (T.), Feng (X.), Chen (Y.) et Temam (O.). – Shidiannao : Shifting vision processing closer to the sensor. – In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 92–104, June 2015.
19. Flamand (E.), Rossi (D.), Conti (F.), Loi (I.), Pullini (A.), Rotenberg (F.) et Benini (L.). – Gap-8 : A risc-v soc for ai at the edge of the iot. – In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–4, July 2018.
20. Gamatié (A.), Devic (G.), Sassatelli (G.), Bernabovi (S.), Naudin (P.) et Chapman (M.). – Towards energy-efficient heterogeneous multicore architectures for edge computing. *IEEE*

Access, vol. 7, 2019, pp. 49474–49491.

21. Georgakopoulos (D.), Jayaraman (P. P.), Fazia (M.), Villari (M.) et Ranjan (R.). – Internet of things and edge cloud computing roadmap for manufacturing. *IEEE Cloud Computing*, vol. 3, n4, July 2016, pp. 66–73.
22. Le Cun (Y.), Bottou (L.), Bengio (Y.) et Haffner (P.). – Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, vol. 86, n11, Nov 1998, pp. 2278–2324.
23. Li (H.), Bhargav (M.), Whatmough (P. N.) et Philip Wong (H. .). – On-chip memory technology design space explorations for mobile deep neural network accelerators. – In *56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.
24. Li (H.), Ota (K.) et Dong (M.). – Learning iot in edge : Deep learning for the internet of things with edge computing. *IEEE Network*, vol. 32, n1, Jan 2018, pp. 96–101.
25. Mahdavinejad (M. S.), Rezvan (M.), Barekatain (M.), Adibi (P.), Barnaghi (P.) et Sheth (A. P.). – Machine learning for internet of things data analysis : a survey. *Digital Communications and Networks*, vol. 4, n3, 2018, pp. 161 – 175.
26. Marantos (C.), Karavalakis (N.), Leon (V.), Tsoutsouras (V.), Pekmestzi (K.) et Soudris (D.). – Efficient support vector machines implementation on intel/movidius myriad 2. – In *7th Int'l Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, May 2018.
27. Marculescu (D.), Stamoulis (D.) et Cai (E.). – Hardware-aware machine learning : Modeling and optimization. – In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'18)*, pp. 137 :1–137 :8, New York, NY, USA, 2018. ACM.
28. Senni (S.), Torres (L.), Sassatelli (G.), Gamatie (A.) et Mussard (B.). – Exploring mram technologies for energy efficient systems-on-chip. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, n3, 2016, pp. 279–292.
29. Shawahna (A.), Sait (S. M.) et El-Maleh (A.). – Fpga-based accelerators of deep learning networks for learning and classification : A review. *IEEE Access*, vol. 7, 2019, pp. 7823–7859.
30. Verma (S.), Kawamoto (Y.), Fadlullah (Z. M.), Nishiyama (H.) et Kato (N.). – A survey on network methodologies for real-time analytics of massive iot data and open research issues. *IEEE Communications Surveys Tutorials*, vol. 19, n3, thirdquarter 2017, pp. 1457–1477.
31. Xu (Q.), Mytkowicz (T.) et Kim (N. S.). – Approximate computing : A survey. *IEEE Design Test*, vol. 33, n1, Feb 2016, pp. 8–22.