Zip-CNN

Thomas Garbay

Sorbonne Université

École Doctorale Informatique, Télécommunications et Électronique
ED130

LIP6 – Equipe SYEL

ZIP-CNN

Thèse de Doctorat

Par Thomas GARBAY

Dirigée par Pr Bertrand Granado

Devant un jury composé de :

Guy Gogniat	Université Bretagne Sud	Rapporteur et Président du Jury
Fan Yang	Université de Bourgogne	Rapporteuse
Pierre Langlois	Polytechnique Montréal	Examinateur
Sébastien Pillement	Université de Nantes	Examinateur
Emmanuelle Encrenaz	Sorbonne Université	Examinatrice
Bertrand Granado	Sorbonne Université	Directeur
Khalil Hachicha	Sorbonne Université	Co-Directeur
Andrea Pinna	Sorbonne Université	Encadrant
Wilfried Dron	STMicroelectronics	Encadrant

Remerciements

Je tiens à remercier les membres du jury d'avoir accepté de participer à l'évaluation de mes travaux de thèse, Pierre Langlois, Sébastien Pillement, Emmanuelle Encrenaz et particulièrement Guy Gogniat et Fan Yang pour le temps accordé à l'examen du manuscrit.

Je remercie mon encadrant et directeur de thèse du laboratoire, Andrea Pinna et Khalil Hachicha, pour leur conseil et le temps accordé à m'accompagner dans cette démarche singulière qu'est une thèse. Je souhaite particulièrement remercier mon directeur de thèse Bertrand Granado, pour m'avoir donné la chance de m'exprimer scientifiquement, et avec qui un cours d'électronique a débouché sur une thèse. Je lui en suis profondément reconnaissant.

Je souhaite remercier les membres de l'équipe Wisebatt avec qui j'ai collaboré durant ces travaux. Merci à Marion, Alexis, Loïc, Maher, Jessica pour leur accueil. Je remercie particulièrement Pedro Lusich et Imane Khalis pour leur aide technique précieuse et leur bienveillance. Merci à Wilfried Dron de m'avoir accordé temps et conseils, et d'avoir participé à l'encadrement de mes travaux.

Je remercie les membres de l'équipe SYEL dans laquelle j'ai évolué, Julien Denoulet, Sylvain Feruglio, Farouk Valette et Hichem Sahbi, ainsi que les personnels liés à l'équipe, Manuel Bouyer et Amine Rhouni, pour leur accueil, leurs conseils et leur bienveillance. Je souhaite particulièrement remercier un ancien membre de l'équipe, Orlando Chuquimia, qui m'a accompagné dès mes premiers pas au laboratoire et jusqu'à ces derniers. Je n'oublie pas d'autres anciens pour leurs conseils de doctorants expérimentés quand j'étais néophyte, Olivier Tsiakaka, Faten Sahel, Valentin Rebière. Merci à Petr Dobias avec qui j'ai partagé une année de travail. Merci à mes camarades de thèse qui ont emprunté ce chemin en même temps que moi, Vincent Janiak, Songlin Li, Kahina Khacef, Mathieu Mba, et à ceux qui sont en cours d'exploration, Sylvain Takougang, Victor Enescu, Lokmane Demagh et Fotis Giasemis à qui je souhaite le meilleur.

Je souhaite remercier mes proches avec qui je partage nombre de moments de vie depuis plusieurs années, Mario, Sami, Benoît, André, Judy, Roger.. et bien d'autres, la liste peut être longue. Merci à Axelle d'avoir partagé le quotidien d'un doctorant durant ces années, nous allons pouvoir partir en week-end maintenant. Je souhaite finalement remercier mon frère et mes parents pour leur soutien inconditionnel depuis 28 ans.

Thèse de Doctorat de Sorbonne Université

Résumé :

Les systèmes numériques utilisés pour l'Internet des Objets - *Internet of Things* (IoT) et les Systèmes Embarqués ont connu une utilisation croissante ces dernières décennies. Les systèmes embarqués basés sur des microcontrôleurs - *Microcontroller Unit* (MCU) permettent de résoudre des problématiques variées, en récoltant de nombreuses données. Aujourd'hui, environ 250 milliards de MCU sont utilisés. Les projections d'utilisation de ces systèmes pour les années à venir annoncent une croissance très forte.

L'intelligence artificielle a connu un regain d'intérêt dans les années 2012. L'utilisation de réseaux de neurones convolutifs - *Convolutionnal Neural Network* (CNN) a permis de résoudre de nombreuses problématiques de vision par ordinateur ou de traitement du langage naturel. L'utilisation de ces algorithmes d'intelligence artificielle au sein de systèmes embarqués permettrait d'améliorer grandement l'exploitation des données récoltées. Cependant le coût d'exécution des CNN rend leur implémentation complexe au sein de systèmes embarqués.

Ces travaux de thèse se concentrent sur l'exploration de l'espace des solutions pour guider l'intégration des CNN au sein de systèmes embarqués basés sur des microcontrôleurs. Pour cela, la méthodologie ZIP-CNN est définie. Elle tient compte du système embarqué et du CNN à implémenter. Elle fournit à un utilisateur des informations sur l'impact de l'exécution du CNN sur le système. Un utilisateur peut ainsi estimer l'influence des choix de conception, dans l'objectif de respecter les contraintes de l'application cible. Un modèle fournit quantitativement une estimation de la latence, de la consommation énergétique et de l'espace mémoire nécessaire à une inférence d'un CNN au sein d'une cible embarquée, quelle que soit la topologie du CNN. Ce modèle tient compte des éventuelles réductions algorithmiques telles que la distillation de connaissances, l'élagage ou la quantification. L'implémentation de CNN de l'état de l'art au sein de MCU a permis la validation expérimentale de la justesse de l'approche.

L'utilisation des modèles développés durant ces travaux de thèse démocratise l'implémentation de CNN au sein de MCU, en guidant les concepteurs de systèmes embarqués. De plus, les résultats obtenus ouvrent une voie d'exploration pour appliquer les modèles développés à d'autres matériels cibles, comme les architectures multicœurs ou les FPGA. Les résultats d'estimations sont également exploitables dans l'utilisation d'algorithmes de recherche de réseaux de neurones - *Neural Architecture Search* (NAS). Cela permettrait potentiellement de découvrir de nouvelles topologies de réseaux de neurones, adaptées à l'inférence à faible coût.

Mots clefs : Système Embarqué, intelligence artificielle, Internet des Objets, réseaux de neurones convolutifs, méthodologie, implémentation, distillation de connaissances, élagage, quantification, inférence, exploration d'espace des solutions, microcontrôleurs.

PhD Thesis of
Sorbonne Université

Abstract:

Digital systems used for the Internet of Things (IoT) and Embedded Systems have seen an increasing use in recent decades. Embedded systems based on Microcontroller Units (MCUs) solve various problems by collecting a lot of data. Today, about 250 billion MCUs are in use. Projections in the coming years point to very strong growth.

Artificial intelligence has seen a resurgence of interest in 2012. The use of Convolutional Neural Networks (CNNs) has helped to solve many problems in computer vision or natural language processing. The implementation of CNNs within embedded systems would greatly improve the exploitation of the collected data. However, the inference cost of a CNN makes its implementation within embedded systems challenging.

This thesis focuses on exploring the solution space, in order to assist the implementation of CNNs within embedded systems based on microcontrollers. For this purpose, the ZIP-CNN methodology is defined. It takes into account the embedded system and the CNN to be implemented. It provides an embedded designer with information regarding the impact of the CNN inference on the system. A designer can explore the impact of design choices, with the objective of respecting the constraints of the targeted application. A model is defined to quantitatively provide an estimation of the latency, the energy consumption and the memory space required to perform CNN inference within an embedded target, whatever the topology of the CNN is. This model takes into account algorithmic reductions such as knowledge distillation, pruning or quantization. The implementation of state-of-the-art CNNs within MCUs verified the accuracy of the different estimations through a measurement process.

The use of the models developed during this thesis democratize the implementation of CNNs within MCUs, assisting the designers of embedded systems. Moreover, the results obtained open a way of exploration to apply the developed models to other target hardware, such as multi-core architectures or FPGAs. The estimation results are also exploitable in Neural Architecture Search (NAS). This would potentially allow the discovery of new neural network topologies, suitable for low-cost inference.

Keywords: Embedded Systems, Artificial Intelligence, Internet of Things, Convolutionnal Neural Networks, methodology, implementation, knowledge distillation, pruning, quantization, inference, design space exploration, Microcontroller Unit.

Table des matières

Table des figures

Liste des tableaux

Acronymes

ASIC *Application-Specific Integrated Circuit.*

CNN *Convolutionnal Neural Network.*
CPD *Canonical Polyadic Decomposition.*

DNS *Dynamic Network Surgery.*
DRAM *Dynamic Random Access Memory.*

EST Énergie, Surface, Temps.

FLOPS *Floating-point operations per second.*
FP32 Format binaire en virgule flottante sur 32-bit.
FPGA *Field Programmable Gate Array.*
FPGM *Filter Pruning via Geometric Median.*
FPU *Floating Point Unit.*

G-Ops Giga Opérations.
GPP *General Purpose Processor.*
GPU *Graphical Processing Unit.*

ILSVRC *Imagenet Large Scale Visual Recognition Challenge.*
INT8 Format binaire en entier sur 8-bit.
IoT *Internet of Things.*

KT *Knowledge Transfer.*

LUT *Look Up Table.*

MACC Multiplication accumulation.
MCU *Microcontroller Unit.*

NA Nombre d'applications.
NAS *Neural Architecture Search.*
NPU *Neural Processing Unit.*

OS *Operating System.*

PTQ *Post Training Quantization.*

PUC *Primitive Unit Cost*.
PULP *Parallel Ultra Low Power*.

QAT *Quantization Aware Training*.

RAM *Random Access Memory*.
ReLU *Rectified Linear Unit*.
RGB *Red Green Blue*.
ROM *Read Only Memory*.

SE Système Embarqué.
SRAM *Static Random Access Memory*.
SVD *Singular Value Decomposition*.

TL *Transfert Learning*.
TPU *Tensor Processing Unit*.

Chapitre 1

Systèmes Embarqués et Réseaux de Neurones Convolutifs

Sommaire

1.1 Introduction

Les systèmes numériques sont présents dans une majorité d'applications de la société. Transports, médecine, communication ou encore divertissements, tous font appel aux technologies numériques, particulièrement aux systèmes embarqués. Ces systèmes embarqués collectent des données de toutes natures à l'aide de nombreux capteurs : accéléromètre, caméra, thermomètre, etc ; leur permettant d'être en constante interaction avec leur environnement. Depuis 2012, les algorithmes d'apprentissage profond ont connu un regain d'intérêt grâce à leur capacité à traiter avec succès des tâches complexes. Les réseaux de neurones, et plus particulièrement les réseaux de neurones convolutifs, dominent largement le domaine de la vision par ordinateur ou du traitement du langage naturel. L'intégration des réseaux de neurones au sein de cibles matérielles fortement contraintes, comme les systèmes embarqués, soulève de nombreuses problématiques scientifiques et techniques.

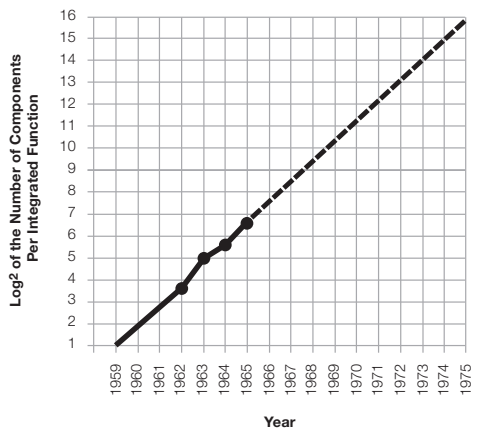
Ce chapitre introduit le rôle des systèmes numériques dans les applications modernes ainsi que l'importance des systèmes embarqués. Puis, les algorithmes d'apprentissage profond sont présentés, particulièrement les réseaux de neurones convolutifs. Les principes de fonctionnement de ces algorithmes sont expliqués. Finalement les problématiques de l'intégration de ces réseaux de neurones au sein de cibles embarquées sont détaillées pour conduire à l'énoncé de la problématique générale.

1.2 Les systèmes embarqués

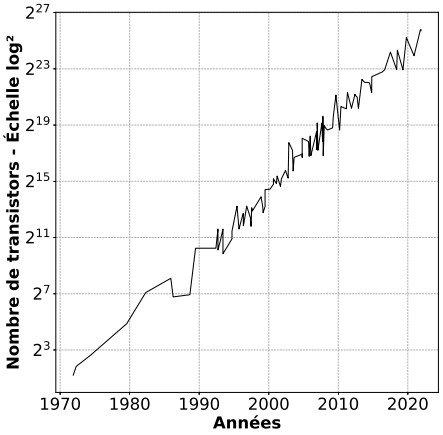
Les architectures numériques ont considérablement évolué depuis la création du premier ordinateur programmable à base de transistors. Aujourd'hui, un téléphone portable coutant moins de 500 dollars est plus performant que l'ordinateur le plus rapide de 1993 qui coûtait 50 millions de dollars [1]. Ces avancées sont dues majoritairement à deux facteurs, l'amélioration des techniques de conception et l'amélioration des technologies de fabrication. Cette tendance de progression dans les performances des systèmes numériques a été définie il y a plusieurs décennies à travers la loi de Moore [2]. En 1965, Gordon Moore prédit que le nombre de transistors par puce va doubler chaque année, avant modification à partir de 1975 pour doubler tous les deux ans. Cette loi est illustrée sur la Figure 1.1a. La comparaison avec la réelle évolution du nombre de transistors au sein des circuits intégrés à travers les dernières décennies est illustrée sur la Figure 1.1b.

Cette tendance définie par Moore a fortement contribué au développement des architectures numériques. Le nombre important de transistors intégrés sur une puce de silicium, des centaines de millions puis des milliards, a permis dans les années 2000 l'avènement sur une même puce des architectures multiprocesseurs, permettant l'amélioration des performances du système en parallélisant l'exécution des instructions.

Cette évolution a aussi permis d'avoir à l'heure actuelle un vaste panel d'architectures matérielles, allant des microcontrôleurs jusqu'aux serveurs de calcul, en passant par les processeurs polyvalents - *General Purpose Processor* (GPP) ou les cartes graphiques *Graphical Processing Unit* (GPU). La nature des applications faisant appel à ses architectures numériques sont également variées : langage, son, audio ou vidéo. Cette variété d'architectures numériques et d'applications a mené à cinq catégories de systèmes résumés dans le tableau Table 1.1.



(a) Loi de Moore : prédiction du nombre de transistors dans les circuits intégrés.



(b) Évolution réelle du nombre de transistors dans les circuits intégrés.

FIGURE 1.1 – (a) Loi de Moore telle que présentée dans l'article de Gordon Moore [2]. (b) Évolution réelle du nombre de transistors dans les circuits intégrés sur les dernières décennies. Ces données sont extraites des travaux de Karl Rupp [3].

TABLE 1.1 – Catégories de systèmes numériques [1]

Systèmes	**Prix du système**	**Prix du processeur**	**Critères conception**
Mobile personnel	100$ à 1000$	10$ à 100$	Coût, Réactivité
Ordinateur personnel	300$ à 2500$	50$ à 500$	Coût/Performance
Serveur de calcul	5000$ à 10M$	200$ à 2000$	Débit, Disponibilité, Adaptabilité
Groupe de serveurs (Cluster)	100k$ à 200M$	50$ à 250$	Coût/Performance, Débit
IoT et embarqué	10$ à 100k$	0,01$ à 100$	Coût, Énergie, Performance spécifique

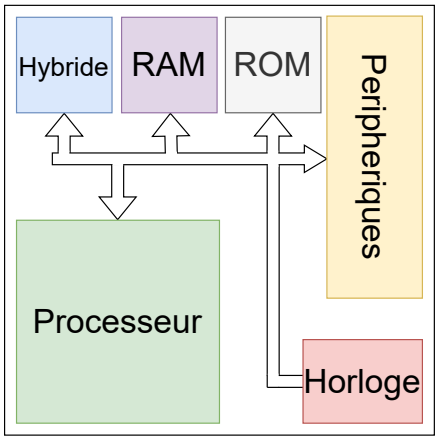
Une catégorie de systèmes, l'Internet des Objets *Internet of Thing* (IoT) et l'embarqué, a connu une très forte croissance ces dernières années. L'internet des objets fait référence aux systèmes embarqués connectés entre eux. La majorité de ces systèmes sont basés sur l'utilisation des unités microcontrôleurs *Microcontroller Unit* (MCU). Grâce à un ensemble de capteurs, ces MCU collectent des données et interagissent avec le monde physique. Aujourd'hui, environ 250 milliards de MCU sont déployés sur la planète [4]. Un acteur industriel majeur du domaine est la société ARM. Elle a vendu 4,4 milliards de puces basées sur un cœur de processeur Arm Cortex-M, très utilisé dans le domaine de l'embarqué et l'IoT, durant le dernier trimestre de 2021 uniquement [5]. Les projections [6] pour les années à venir annoncent un nombre croissant de systèmes embarqués déployés (plus de 30 à 40 milliards de plus par an) ou de données générées (des dizaines de zetta octets, un zetta octet $= 10^{21}$ octets). Les projections des acteurs du domaine varient, ce qui rend difficile une estimation précise de l'impact des systèmes embarqués sur le monde industriel et domestique. Cependant, tous s'accordent sur un point : la croissance serait très forte dans les années à venir sur ce secteur.

Les microcontrôleurs sont des circuits intégrés contenant un microprocesseur, de l'espace mémoire, des périphériques tels que des liaisons séries ou des convertisseurs

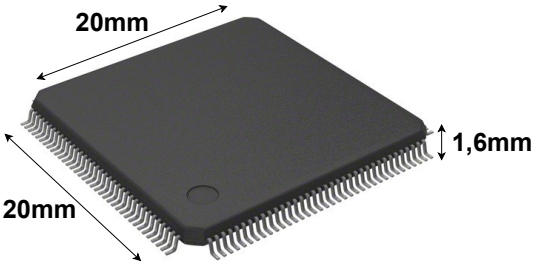
analogique-numérique. Les MCU sont caractérisés par un haut degré d'intégration. Leur consommation énergétique est faible par rapport aux autres architectures matérielles embarquables (GPP, GPU, etc), tout comme leur fréquence de fonctionnement allant de quelques kilos Hertz jusqu'à plus d'un giga Hertz. Les systèmes basés sur MCU sont conçus spécifiquement par rapport aux contraintes de l'application cible. Par exemple, un MCU ayant pour finalité une application de commande de moteur industriel [7] n'aura pas les mêmes caractéristiques matérielles (fréquence de fonctionnement, taille mémoire, etc.) qu'un MCU adapté à des applications de localisation géographique [8]. Les éléments principaux d'un MCU sont listés ci-dessous.

- Un processeur : interprète les instructions du programme.
- Une horloge : pour cadencer le microcontrôleur.
- Des espaces mémoire : non volatile appelé Read Only Memory - ROM, volatile appelé Random Access Memory - RAM et hybride par exemple Flash.
- Des périphériques : pour interagir avec l'environnement extérieur.

Ces éléments sont schématisés sur la Figure 1.2.



(a) Schéma simplifié des éléments principaux d'un microcontrôleur



(b) Représentation physique d'un MCU : le STM32L496ZG

FIGURE 1.2 – (a) Schéma simplifié d'un microcontrôleur. Les éléments de base sont un processeur, une mémoire ROM, RAM et hybride, une horloge ainsi que des périphériques. (b) Exemple d'un microcontrôleur, le STM32L496ZG. Tous les éléments du microcontrôleur sont contenus dans ce circuit intégré de dimension L : 20mm, l : 20mm, H : 1,6mm maximum.

La Figure 1.3 représente une vue détaillée de la composition du microcontrôleur STM32L496ZG [9]. Ce schéma est extrait de la notice technique du MCU. Les éléments schématisés dans la Figure 1.2a qui composent les microcontrôleurs sont présents sur la Figure 1.3, le processeur est représenté en vert, les mémoires Flash et RAM sont respectivement en bleu et violet, l'horloge est en rouge et les périphériques sont en jaune. Des bus de communications internes assurent le bon fonctionnement de l'ensemble. Tous ces composants sont contenus dans le boitier représenté en Figure 1.2b. Pour le STM32L496ZG, le boîtier fait 20 millimètres de longueur sur 20 millimètres de largeur et au maximum 1,6 millimètre d'épaisseur. La partie jaune de la Figure 1.3, les périphériques, est très représentée. C'est un point fort du microcontrôleur d'avoir de nombreux périphériques permettant une interaction variée avec son environnement extérieur. Par exemple, le STM32L496ZG est

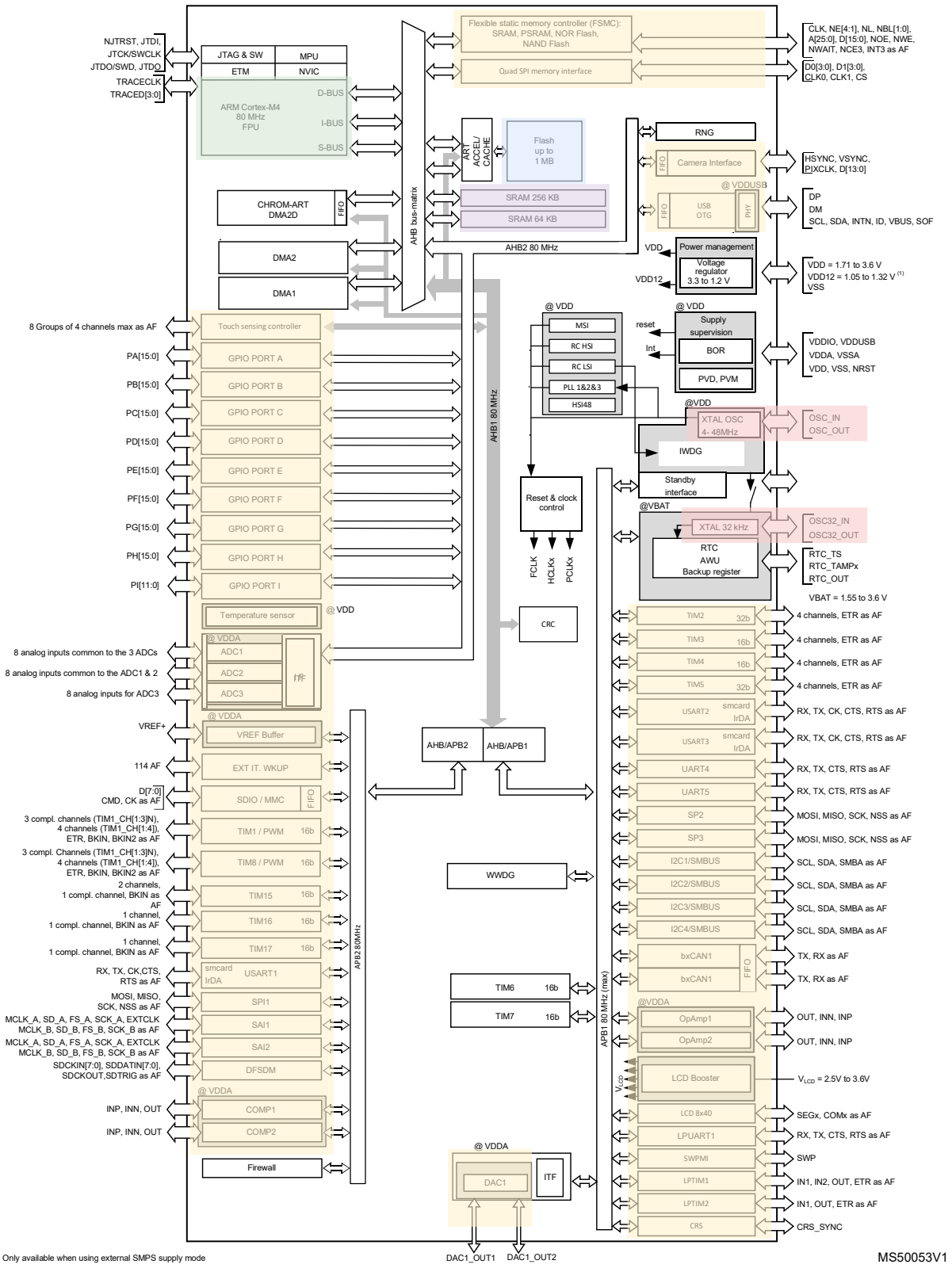


FIGURE 1.3 – Exemple de la composition d'un MCU : le STM32L496ZG. Le processeur est représenté en vert, les mémoires hybrides, ici mémoire Flash, et RAM sont respectivement en bleu et violet, l'horloge est en rouge et les périphériques sont en jaune.

équipé de périphériques pour la vidéo, l'USB, les mémoires externes, les convertisseurs analogique-numérique, les liaisons de communications séries pour l'audio, l'électronique domestique, les mémoires externes, d'autres microcontrôleurs, etc.

Ces interfaces communiquant avec des capteurs assurent l'acquisition de nombreuses données. Une partie des applications actuelles traitent ces données à l'extérieur du système embarqué, via l'utilisation de serveurs dans le *Cloud*. Ces derniers sont plus puissants et offrent un traitement précis et rapide. Par exemple, la célèbre application "Keyword Spotting" [10] pour "Détection de mots clés", consiste en la détection d'un mot ou expression, "Ok Google" ou "Hey Siri". Une fois le mot clé détecté, la chaîne complète de reconnaissance vocale est activée et transmet la demande de l'utilisateur sur le serveur. Ce type d'application requiert cependant une connexion et une bande passante suffisante pour traiter la nature et le nombre des demandes. Ce mode de fonctionnement avec une partie des traitements sur le système embarqué et une autre, plus chronophage, sur un serveur *Cloud*, est envisageable pour des applications au sein d'environnements connectés. Mais il existe de nombreuses applications embarquées qui évoluent dans un environnement isolé, sans connexion. Un exemple illustrant cette contrainte est le domaine spatial, où les données ne peuvent être envoyées sur Terre depuis un satellite que lors d'une fenêtre de temps très précise. Le traitement sur un Cloud nécessite de communiquer les données. Cette communication a un impact sur l'autonomie du système embarqué qui dépend fortement de la durée de vie de la batterie associée. La consommation énergétique du transfert de données est importante pour un budget d'énergie contraint, une puissance de plusieurs dizaines de milliwatts est nécessaire pour les protocoles basse consommation [11]. Le traitement externe des données sur des serveurs soulève également une problématique de confidentialité importante. Le traitement au plus près du système des nombreuses données récoltées par les capteurs pourrait apporter une solution à ces problématiques.

Cette acquisition de données de toute nature permet l'utilisation des microcontrôleurs dans de nombreux domaines comme l'automobile [12], la maintenance prédictive [13], la classification d'image [14], la détection d'activité [15] ou de signal audio [16]. Aujourd'hui, une des solutions utilisées par la recherche pour analyser ce nombre important de données est l'utilisation d'algorithme d'apprentissage profond. Ces algorithmes sont réputés gourmands à la fois en termes de puissance de calcul, mais aussi en termes d'énergie consommée et d'empreinte mémoire utilisée. Les processeurs utilisés dans les microcontrôleurs ont une puissance de calcul faible comparée à ceux utilisés dans les ordinateurs ou les serveurs *Cloud*. L'espace mémoire est limité que cela soit pour la mémoire ROM et la mémoire RAM. Les systèmes embarqués fonctionnent à l'aide d'une batterie, imposant un budget énergétique limité. Ces contraintes matérielles fortes rendent difficile le déploiement d'algorithmes d'analyse de données à l'aide de l'apprentissage profond et au plus près du système.

1.3 Les réseaux de neurones convolutifs

Les réseaux de neurones convolutifs *Convolutional Neural Network* (CNN) font partie de la famille des algorithmes d'apprentissage profond. L'apprentissage profond est une branche de l'intelligence artificielle. Yann Lecun décrit l'intelligence artificielle comme un ensemble de techniques qui permettent aux machines de réaliser des actions ou de résoudre un problème qui, normalement, est réservé aux humains ou aux animaux [17]. L'intelligence artificielle, à travers des techniques d'apprentissage, doit permettre de s'adapter à une majorité de situations sans pour autant les avoir décrites préalablement. Par exemple en reconnaissance d'images, après avoir montré à une machine des millions d'images étiquetées, c'est à dire chacune appartenant à une catégorie, la machine peut classer une nouvelle image qu'elle n'a jamais vue auparavant et qui appartient à l'une des catégories apprises. Ce phénomène est appelé la capacité de généralisation. Les algorithmes d'apprentissage profond sont composés de deux blocs majeurs, un extracteur de caractéristiques et un classifieur. Ces algorithmes doivent être entrainés et sont constitués de plusieurs modules en série représentant chacun une étape de traitement. Pour un exemple donné, tous les paramètres de chaque module sont ajustés de telle façon que la valeur de sortie du système se rapproche le plus possible de la valeur de sortie souhaitée. Cette succession de modules, appelés également "couches", a amené le qualificatif de profond dans l'apprentissage profond.

L'apprentissage profond a connu trois vagues de développement. La première, nommée cybernétique, s'est déroulée entre les années 1940 et 1960. Durant cette période, le neurone formel de McCulloch-Pitts [18] définit un calcul logique pour caractériser le fonctionnement d'un neurone. Le perceptron de Rosenblatt [19] est le premier modèle capable d'apprendre des poids depuis des exemples. Une dernière réalisation notable est celle de Widrow et Hoff [20] qui est ADALINE - Adaptative Linear Neuron. Il s'agit d'un réseau de neurones simple couche dont l'algorithme d'entrainement utilisé pour adapter les poids des neurones est la descente de gradient stochastique. Puis la cybernétique a laissé place au connexionnisme entre les années 1980 et 1990. Une réalisation majeure de cette époque est l'utilisation de la rétropropagation du gradient, développée pour l'entrainement de réseaux de neurones profonds par Hinton [21]. Cette technique a été appliquée pour l'entrainement automatique d'un réseau de neurones convolutifs nommé LeNet, développé par Lecun [22], pour classifier les images basse résolution de chiffres écrits à la main. Ce CNN a été déployé pour la lecture automatique des chèques de banques. Il lisait environ 10% de tous les chèques émis aux États-Unis à la fin des années 1990. Malgré ce succès, le manque de données d'entrainement et de puissance de calcul des ordinateurs de l'époque rend les réseaux de neurones convolutifs difficiles à entrainer. Finalement le connexionnisme laisse sa place à l'apprentissage profond à partir de 2006. Le succès majeur de cette ère toujours en cours est celui de Krizhevsky. En 2012, cette équipe de recherche a gagné la compétition ILSVRC [23] - Imagenet Large Scale Visual Recognition Challenge - avec un réseau de neurones convolutifs, AlexNet [24]. Cette compétition consiste en la classification de millions d'images en mille catégories. L'entrainement de ce CNN a été possible grâce aux progrès technologiques dans le domaine des GPU. Depuis cette victoire, les CNN dominent largement le domaine de la vision par ordinateur.

L'architecture des CNN est inspirée de l'architecture du cortex visuel des mam-

mifères, à travers les travaux de Hubel et Wiesel [25] en 1959. Le principe est basé sur la capacité d'apprendre à représenter le monde de manière hiérarchique. Les premières couches des CNN extraient des caractéristiques basiques, comme la présence ou non d'un contour. Ces caractéristiques simples sont ensuite combinées par les couches suivantes qui amènent la formation de concepts de plus en plus complexes et abstraits. Un avantage majeur des CNN est leur propriété d'invariance par translation. Prenons l'exemple d'une tâche de classification d'images de visage humain. L'algorithme a besoin de savoir qu'un nez est situé vers le milieu du visage, mais n'a pas besoin de le situer au pixel près. Deux phases majeures caractérisent le développement de ces réseaux. Dans un premier temps, l'entrainement. Il s'agit de l'ajustement de chaque paramètre de chaque module du CNN. Pour ce faire, la technique de rétropropagation du gradient est utilisée. Cette technique permet de calculer un gradient, représentant la quantité par laquelle l'erreur en sortie va augmenter ou diminuer si le paramètre d'une quantité donnée est modifié. Dans un second temps, l'inférence. Le CNN est déployé sur un système réel et va classer une entrée qu'il n'a encore jamais vue. La composition d'un CNN est représentée schématiquement sur la Figure 3.5. Sur la partie gauche de la Figure 3.5 est représenté l'extracteur de caractéristiques, dont le rôle est d'extraire les caractéristiques de l'entrée, par exemple des contours. Puis, sur la partie droite de la Figure 3.5 est représenté le classifieur, qui basé sur les caractéristiques précédemment extraites, donne une probabilité en sortie de ce qu'est l'entrée.

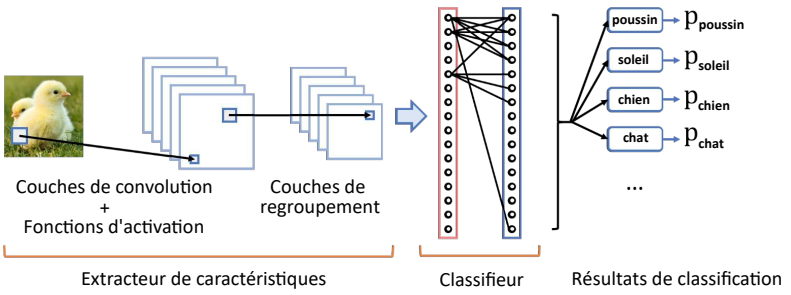


FIGURE 1.4 – Composition générale des CNN. La partie gauche représente l'extracteur de caractéristiques composé de couches de convolution, de fonctions d'activation et de couches de regroupement. La partie droite représente le classifieur composé de couches entièrement connectées et de fonctions d'activation. La sortie du classifieur est une probabilité entre 0 et 1 de la classe à laquelle appartient l'entrée.

Les CNN sont composés de plusieurs couches dont les plus communes sont :

- La couche de convolution : opération de base de l'extraction de caractéristiques. Ces couches appliquent un filtrage par convolution en parcourant l'image avec une fenêtre qui représente le filtre. Le produit de convolution entre la fenêtre et la portion d'image recouverte est ainsi réalisé pour mettre en avant les caractéristiques de l'image, par exemple un contour ou une ligne. Pour chaque paire image/filtre est générée une carte de caractéristiques. L'opération de convolution est mathématiquement définie avec l'Équation 1.1. Pour I image bidimensionnelle, K noyau bidimensionnel, et S la fonction de convolution :

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) + b \qquad (1.1)$$

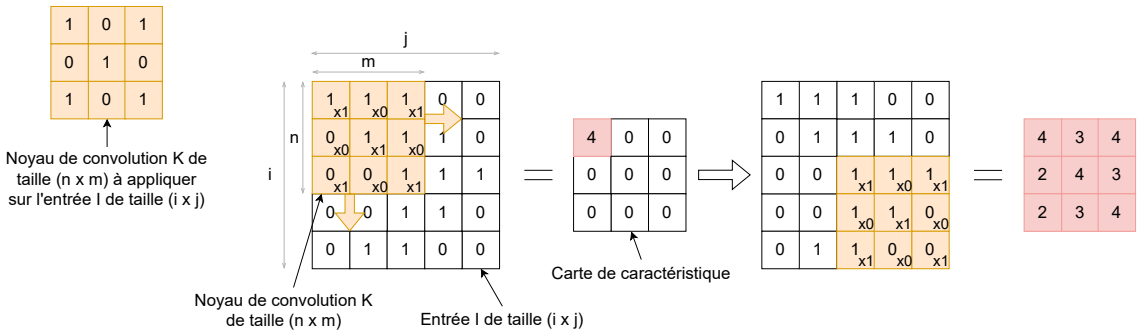


FIGURE 1.5 – Représentation schématique d'une convolution. L'entrée de la convolution est de taille i par j, ici 5×5. Le noyau de convolution est de taille n par m, ici 3×3

Le principe d'une opération de convolution est représenté sur la Figure 1.5.

- La couche de regroupement : plus connu sous le nom de pooling en anglais. Les opérations de regroupement les plus communes sont le regroupement par valeur maximale - Maximum Pooling - et le regroupement par valeur moyenne - Average Pooling. Le principe de ces opérations est illustré sur la Figure 1.6. L'un des avantages à utiliser la couche de regroupement est d'amener une invariance à de légères translations de l'entrée.

(a) Représentation schématique d'un regroupement par valeur maximale - Maximum Pooling

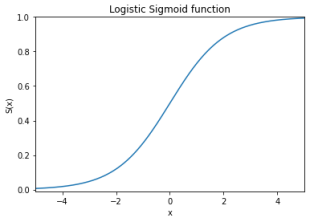
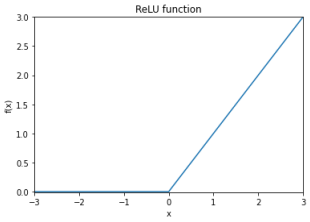
(b) Représentation schématique d'un regroupement par valeur moyenne - Average Pooling

FIGURE 1.6 – Le noyau de regroupement représenté ici est de taille 2×2. (a) Le maximum pooling va garder la valeur maximale recouverte par le noyau de pooling. (b) L'average pooling va faire la moyenne des valeurs recouvertes par le noyau de pooling.

- La couche d'activation : introduit des non-linéarités dans la fonction de classification du réseau. Trois exemples de fonction d'activation sont présentés dans le Tableau 1.2. De nombreuses fonctions d'activations existent. Historiquement, la fonction sigmoïde, exprimée dans l'Équation 1.2, a été très utilisée. Aujourd'hui la fonction d'activation la plus communément utilisée est la fonction ReLU de l'Équation 1.3 - Rectified Linear Unit - pour l'accélération de la phase d'entrainement du réseau qu'elle apporte par rapport à la sigmoïde. De plus, la fonction softmax de l'Équation 1.4 est très utilisée pour normaliser la sortie d'un CNN entre 0 et 1.
- La couche entièrement connectée : placée à la fin des réseaux de neurones convolutifs. Il s'agit d'une combinaison linéaire qui produit un vecteur de taille N en sortie après avoir reçu un vecteur en entrée. La dernière couche entièrement connectée est généralement suivie d'une fonction d'activation softmax. Dans ce cas, N représente le nombre de classes du problème. Chaque composante du vecteur de sortie indique la probabilité que l'image appartienne à telle classe.

Le réseau de neurones convolutifs AlexNet [24] qui a gagné la compétition ILS-

TABLE 1.2 – Exemples de fonctions d'activation

Fonction	Définition mathématique	Représentation graphique	
Sigmoïde	$sigmoid(x) = \frac{1}{1+e^{(-x)}}$		(1.2)
ReLU	$f(x) = \begin{cases} 0, & \text{si x} < 0 \\ x, & \text{si x} \geq 0 \end{cases}$		(1.3)
Softmax	$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$	N.A.	(1.4)

VRC en 2012 est composé de 5 couches de convolution, 3 couches entièrement connectées, des couches de regroupement par valeur maximale (Maximum Pooling) et des couches d'activation. Le nombre de paramètres de ce réseau est de 60 millions. En comparaison, le premier CNN à succès entrainé par Yann Lecun, LeNet5 [22], pour reconnaître les chiffres manuscrits comporte 66 000 paramètres.

Après 2012, la communauté de l'apprentissage profond s'est concentrée sur la création de nouveaux CNN. Ils ont remarqué que plus le réseau de neurones était profond (plus il avait de couches), meilleur était la précision. La Figure 1.7 illustre cette direction de recherche qui a donné lieu à des CNN de plus en plus profond et de plus en plus précis. Sur ce graphique est représentée la précision Top-1 de plusieurs CNN sur la base de données ImageNet utilisée lors du concours ILSVRC. Deux types de métriques de précision de classification sont utilisés dans le domaine de l'apprentissage profond. La précision Top-1 signifie que la réponse du modèle, la probabilité la plus élevée donnée par le classifieur du réseau de neurones, est la bonne réponse. La précision Top-5 signifie que parmi les 5 réponses les plus importantes du modèle, les 5 probabilités les plus élevées données par le classifieur du réseau de neurones, se trouve la bonne réponse.

Comme illustré sur la Figure 1.7, AlexNet avec ses 60 millions de paramètres, obtient moins de 55% de précision Top-1 et environ 2,5 Giga-Opérations (G-Ops) par inférence. Il a été largement dépassé par d'autres CNN. L'architecture VGGNet [27] a eu un grand succès en 2014 avec ses variantes à 16 couches, VGG-16, et à 19 couches, VGG-19. La précision Top-1 est de 71,3% sur ImageNet pour 138 millions de paramètres et un peu plus de 30 G-Ops pour VGG-16. Sa variante à 19 couches, VGG-19, a 144 millions de paramètres et un peu moins de 40 G-Ops pour une inférence. Puis, l'architecture de réseau de neurones ResNet [28] est sortie en 2016. Elle encore très utilisée aujourd'hui. Sous plusieurs variantes en fonction du nombre de couches empilées dans le réseau, de ResNet-18 à ResNet-152, cette architecture offre un meilleur compromis au regard de la précision par rapport au nombre de paramètres ou du nombre d'opérations. En effet, pour 25 millions de paramètres et

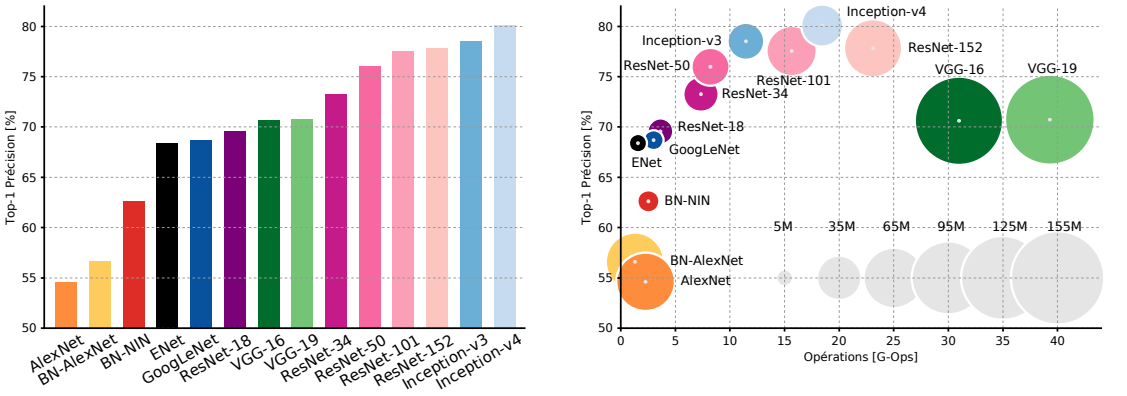


FIGURE 1.7 – Précision *vs.* Nombre de paramètres et Giga opérations pour une inférence [26]. Les couleurs des CNN de (a) correspondent aux couleurs des CNN de (b). (a) Précision Top-1 de plusieurs CNN sur la base de données ImageNet du ILSVRC. (b) Précision Top-1 vs. nombre d'opérations et de paramètres. Le nombre d'opérations en Giga Opérations [G-Ops] correspond au nombre d'opérations nécessaires pour une inférence. La taille des cercles est proportionnelle au nombre de paramètres du réseau, allant de 5 millions à 155 millions.

environ 8 G-Ops, ResNet-50 présente une précision Top-1 de 74,9%. Une spécificité de cette architecture est la connexion résiduelle. Plus de détails sont fournis dans la suite du manuscrit concernant cette architecture. En effet, l'architecture ResNet a été utilisée durant les travaux de recherche présentés dans les chapitres suivants. À la date d'écriture du manuscrit, le CNN le plus performant sur la base de données ImageNet est l'architecture EfficientNet [29]. Avec la variante proposée par Pham et al. [30], la précision Top-1 est de 90,2% pour 480 millions de paramètres.

La course à la précision n'a pas été sans conséquences. La grande majorité des CNN développés après 2012 l'ont été sans se soucier du coût impliqué par une inférence. Si ce coût peut-être acceptable sur du matériel comme des serveurs de calculs ou des GPU, il ne l'est pas sur du matériel embarqué. Les systèmes embarqués présentent des contraintes fortes en termes de latence pour une inférence, de consommation énergétique et d'espace mémoire. Depuis quelques années, une communauté de chercheurs nommée TinyML [31] s'est formée autour de la problématique d'intégration des CNN au sein de ces cibles embarquées, fortement contraintes.

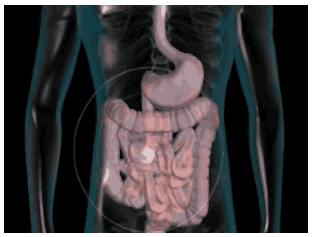
1.4 Une intégration complexe

L'entrainement d'un nouveau réseau de neurones sur une nouvelle tâche de classification est un processus complexe et coûteux. Cela nécessite une base de données importante et équilibrée. La création d'une telle base de données est souvent très difficile et chronophage. Une méthode très simple d'utilisation pour entrainer un CNN à une nouvelle tâche de classification est le transfert d'apprentissage - *Transfert Learning* (TL) [32]. Un CNN existant ayant déjà appris à reconnaitre un ensemble de catégories, par exemple celles de la base ImageNet, est adapté au nouveau problème de classification en pratiquant un ré-apprentissage. Ce ré-apprentissage est effectué en initialisant le CNN avec ses paramètres, appris sur la base de données originale. Certains paramètres sont ré-entrainés, souvent les dernières couches du CNN, pour les spécialiser à une tâche de classification précise. Cette technique permet à tous les utilisateurs d'entrainer un CNN pour le spécialiser à une application, même avec une base de données contenant un faible nombre d'échantillons. Avec des librairies d'apprentissage profond comme TensorFlow [33] ou Pytorch [34], cette méthode d'apprentissage est aujourd'hui utilisable en quelques lignes de code. Cependant, la seule ambition de cette technique est la satisfaction de la contrainte fonctionnelle de l'application, à savoir un fort taux de reconnaissance en obtenant une bonne précision. Aucune considération pour l'intégration dans un système embarqué n'est analysée.

Les systèmes embarqués sont conçus en fonction des contraintes de l'application ciblée. Chaque application a ses propres contraintes. Ainsi, chaque système embarqué doit avoir ses propres spécifications pour répondre aux besoins de l'application. La Figure 1.8 représente trois systèmes embarqués dans trois domaines différents. Un affichage tête haute à embarquer dans un casque de moto. Un drone utilisable pour analyser les champs d'agriculture. Et une capsule vidéo endoscopique appliquée dans la gastroentérologie.

(a) Transport [1]

(b) Agriculture [2]



(c) Medecine [3]

FIGURE 1.8 – (a) Affichage tête haute dans un casque de moto : L*l = 6x4cm, E = 3W, (b) Un drone pour analyser un champ d'agriculture : L*l = 15x6x8cm, E = 60W, (c) Une capsule vidéo endoscopique pour analyser automatiquement la présence de polypes intestinaux : L*l = 28x11mm, E = 1mW

Ces trois différents systèmes ont des contraintes similaires : l'espace mémoire, la consommation énergétique et la latence. Ces contraintes sont cependant quantitativement différentes. La surface du système limite la taille de l'espace mémoire

1. http ://www.lerepairedesmotards.com/dossiers/hud-moto-affichage-tete-haute.php
2. https ://leshorizons.net/cest-quoi-agriculture-de-precision/
3. https ://www.gastroenterologue-montpellier.fr/les-specialites/capsule-endoscopique/

disponible. Par exemple, concernant le drone, il n'est pas simple d'ajouter de la surface. Cela incomberait une modification de la conception de l'ensemble du système, et pourrait impacter sur des contraintes mécaniques liées au vol de l'appareil par exemple. Il faut donc que le réseau de neurones soit suffisamment léger en mémoire pour être embarqué. De plus, un système ne repose pas que sur l'utilisation du réseau de neurones. L'algorithme d'apprentissage profond n'est qu'une partie de la chaîne de traitement du système embarqué. Il faut donc pouvoir embarquer le reste du programme, en charge de la communication avec d'autres systèmes ou bien de la gestion d'une caméra. La consommation énergétique impacte la durée de vie de la batterie du système. Dans l'exemple de la capsule vidéo endoscopique, la durée de vie de la batterie doit être d'un trajet de digestion intestinale, au risque de manquer une information cruciale pour le gastroentérologue. Concernant la latence du système, la contrainte de temps réel est récurrente dans les systèmes embarqués. Par exemple concernant l'affichage tête haute pour le casque de moto, les données doivent être traitées, puis envoyées au pilote pour qu'il ait le temps de prendre une décision avant qu'un virage arrive, et non après. Outre ces trois contraintes propres à tous les systèmes embarqués, il est nécessaire de prendre en compte les spécificités de la cible matérielle. En effet, les performances et possibilités peuvent varier énormément d'un MCU à l'autre. De plus, le choix du CNN à embarquer est important. De nombreux réseaux existent, chacun avec leur composition. Le nombre de paramètres peut grandement varier d'un réseau à un autre. Ces différentes configurations peuvent fortement impacter le système embarqué. Des détails sur ces différentes contraintes sont exposés dans les points suivants.

1.4.1 La latence

Les contraintes de latence sont très fortes sur les systèmes embarqués. Particulièrement lorsque des contraintes de temps réels sont à prendre en compte dans la conception du système. Ces contraintes sont de deux natures. Le temps réel dur : dans le cas du non-respect de cette contrainte, le système est en faute et cela peut amener à des conséquences graves, par exemple sur un système de contrôle aérien. Le temps réel souple : le non-respect des échéances n'occasionne pas de conséquences graves, par exemple un décalage entre le son et l'image lors d'une projection. L'exécution d'un réseau de neurones nécessite de nombreux calculs. Il est nécessaire de prendre en compte la latence engendrée par l'inférence du réseau de neurones, pour s'assurer du respect des contraintes de latence de l'application cible.

1.4.2 La consommation d'énergie

La consommation d'énergie est critique dans un système embarqué. Cette contrainte impacte directement la durée de vie de la batterie qui alimente le système. En fonction des applications, la batterie doit pouvoir alimenter le système embarqué entre quelques jours à quelques années. L'énergie consommée par un système est donnée par l'Équation 1.5, où E est l'énergie consommée, P la puissance moyenne consommée et T le temps d'exécution du programme.

$$E = P \times T \tag{1.5}$$

La puissance P est modélisée comme la somme de deux puissances, suivant

l'Équation 1.6, où P_{total} est la puissance totale consommée, P_{stat} est la puissance statique consommée et P_{dyn} est la puissance dynamique consommée.

$$P_{total} = P_{stat} + P_{dyn} \tag{1.6}$$

La puissance statique peut-être définie suivant l'Équation 1.7 où I_{stat} est le courant statique et V la tension d'alimentation du système.

$$P_{stat} \propto I_{stat} \times V \tag{1.7}$$

La puissance dynamique majoritaire dans le budget énergétique [1], [35]. Elle peut-être exprimée suivant l'Équation 1.8, où V^2 est la tension d'alimentation du système au carré, f_{clk} est la fréquence de l'horloge du système et C_{EFF} est une constante exprimant la capacité effective de commutation des portes logiques du système.

$$P_{dyn} \propto V^2 \times f_{clk} \times C_{EFF} \tag{1.8}$$

L'Équation 1.5 souligne l'influence de la latence du programme à exécuter, notée T, dans la consommation d'énergie du système embarqué. La latence d'exécution d'un réseau de neurones va donc impacter la consommation d'énergie nécessaire à son exécution. L'Équation 1.8 souligne l'influence de la fréquence de l'horloge du système, notée f_{clk}, sur la consommation d'énergie à travers la puissance dynamique consommée. Il est possible d'augmenter la fréquence d'horloge f_{clk} pour augmenter la vitesse de traitement du système, et donc diminuer la latence T d'exécution du CNN. Cependant, augmenter la fréquence d'horloge impacte directement la puissance dynamique consommée et donc la consommation d'énergie.

1.4.3 L'espace mémoire

Les mémoires embarquées sont de deux natures. La ROM, non volatile, elle permet de stocker le code à exécuter ou des valeurs importantes pour l'exécution du programme. La RAM, volatile, permet de stocker les variables intermédiaires nécessaires au bon fonctionnement de l'application. L'espace mémoire disponible sur les MCU est de l'ordre de quelques centaines de kilo-octets. Le Tableau 1.3 référence trois exemples de MCU équipés de mémoires différentes. La conception d'une application embarquée est fortement influencée par l'espace mémoire nécessaire au bon fonctionnement de l'application. Par exemple, un programme nécessitant de nombreux calculs intermédiaires à stocker dans la RAM, par exemple 150 ko, pourra donc s'exécuter sur les MCU STM32F779II et Kinetis K80-150 mais pas sur le PIC32MM0064. Même si l'espace mémoire ROM est suffisant pour stocker le programme. Il est possible d'étendre l'espace mémoire ROM à travers l'ajout d'une mémoire externe. Cela nécessite une communication et une synchronisation entre le système et la mémoire externe. Outre la complexité de programmation, la communication engendrée par un tel ajout augmente significativement les temps d'accès à l'espace mémoire, et donc la latence du programme.

TABLE 1.3 – Exemples d'espaces mémoire disponibles sur des MCU : le STM32F779II de chez STMicroelectronics, le Kinetis K80-150U de chez NXP et le PIC32MM0064 de chez Microchip

MCU	Constructeur	ROM	RAM
STM32F779II [36]	STMicroelectronics	2 Mo	512 ko
Kinetis K80-150 [37]	NXP	256 ko	256 ko
PIC32MM0064 [38]	Microchip	64 ko	8 ko

1.4.4 La diversité de matériel

Les composants présents au sein d'un MCU, en plus de la mémoire, sont un processeur et une horloge (*c.f.* Section 1.2). Pour compléter le Tableau 1.3, les processeurs et fréquences d'horloge de chacun des MCU sont ajoutés dans le Tableau 1.4. La nature du processeur va impacter les performances globales du système. La fréquence de fonctionnement va impacter la latence d'exécution du programme ainsi que l'énergie consommée lors de cette exécution (*c.f.* Sous-Section 1.4.2).

TABLE 1.4 – Exemples de configurations matérielles disponibles sur des MCU : le STM32F779II de chez STMicroelectronics, le Kinetis K80-150U de chez NXP et le PIC32MM de chez Microchip

MCU	Constructeur	ROM	RAM	Processeur	Fréquence d'horloge
STM32F779II [36]	STMicroelectronics	2 Mo	512 ko	Cortex-M7	216 MHz
Kinetis K80-150 [37]	NXP	256 ko	256 ko	Cortex-M4	150 MHz
PIC32MM0064 [38]	Microchip	64 ko	8 ko	MIPS32	25 MHz

1.4.5 La diversité de réseau de neurones convolutifs

De nombreux CNN existent, certains ont été présentés dans la Section 1.3. Chaque CNN a sa propre composition. Le nombre de couches et la taille de noyau des opérations à effectuer (convolution, regroupement) peuvent varier d'un CNN à l'autre et également au sein même du réseau. La taille et la nature des images à traiter peuvent être différentes, ce qui modifie les premières couches d'un CNN. Ces variations ont un impact sur les performances du matériel cible pour exécuter le réseau. En effet, le nombre d'opérations à exécuter ou l'espace mémoire pour stocker le réseau et les résultats intermédiaires nécessaires à une inférence peuvent grandement varier d'un réseau à un autre. La Table 1.5 référence la composition de trois réseaux, LeNet5, ResNet-50 et VGG-16.

TABLE 1.5 – Exemples de compositions de trois CNN, LeNet5, ResNet-50 et VGG-16.

Réseau	Nombre de paramètres	Nombre de couches	Taille des noyaux de convolution	Nature de l'entrée
LeNet5	66 000	7	5×5	Image 32×32, niveau gris
ResNet-50	25 000 000	50	7×7 et 3×3 et 1×1	Image 224×224, niveau RGB
VGG-16	138 000 000	16	7×7 et 3×3 et 1×1	Image 224×224, niveau RGB

1.5 Synthèse et Problématique

Au cours de ce chapitre, les éléments nécessaires à la définition d'une problématique ont été présentés. Les applications embarquées ont un fort impact sur notre société. Les microcontrôleurs sont très utilisés dans le domaine des systèmes embarqués et de l'IoT. Ils permettent d'aborder des problématiques variées grâce à leur capacité de s'interfacer avec de nombreux capteurs, récoltant un nombre important de données. Cependant, les systèmes embarqués basés sur des MCU sont spécifiques à l'application cible. Leurs ressource énergétique, puissance de calcul ou encore espace mémoire sont limités. L'apprentissage profond a connu une forte croissance cette dernière décennie. Les réseaux de neurones convolutifs ont été la solution à des problématiques complexes. Le développement de nouveaux CNN a été fait au détriment du coût d'exécution de ces algorithmes. Cela est peu impactant si le réseau est exécuté sur des serveurs, ayant accès à de nombreuses ressources. Cependant, ce coût d'exécution rend leur intégration complexe lorsque la cible matérielle est contrainte, comme un système embarqué. De plus, la variété de MCU et de composition de réseaux de neurones rend difficile la normalisation d'étapes de conception. De ce constat, une première problématique peut-être résumée sous la forme de l'interrogation suivante.

Comment intégrer des réseaux de neurones convolutifs au sein d'une cible fortement contrainte telle qu'un microcontrôleur ?

Le chapitre suivant présente les solutions de l'état de l'art permettant potentiellement d'apporter une réponse à cette problématique.

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

La problématique de réduction du coût d'inférence des réseaux de neurones convolutifs est apparue suite à l'explosion du nombre de paramètres de ces modèles. Denil et al. [39] ont mis en avant la surparamétrisation des réseaux de neurones. Pour une précision donnée, de nombreux paramètres sont redondants dans les CNN. Il est possible de réduire le coût d'inférence du réseau en limitant la redondance des paramètres. Ces efforts d'optimisation ont d'abord été orientés vers la conception de CNN dont le nombre de paramètres réduits et la structure du réseau rendent l'inférence moins coûteuse. Des techniques de réduction à appliquer sur le CNN ont été proposées. Ces optimisations algorithmiques sont certes efficaces, mais difficiles à déployer manuellement. Ainsi, des environnements de développement et des librairies spécialisées ont été créés pour faciliter l'application de ces méthodes d'optimisation. Certains processus d'implémentation basés sur une optimisation algorithmique ont été proposés pour simplifier la tâche d'implémentation à l'utilisateur. Finalement, des cibles matérielles spécialisées ont rendu plus efficace l'exécution des CNN par rapport à une exécution sur une cible matérielle généraliste. Cette démarche de réduction du coût et d'optimisation de l'inférence d'un CNN est illustrée sur la Figure 2.1.

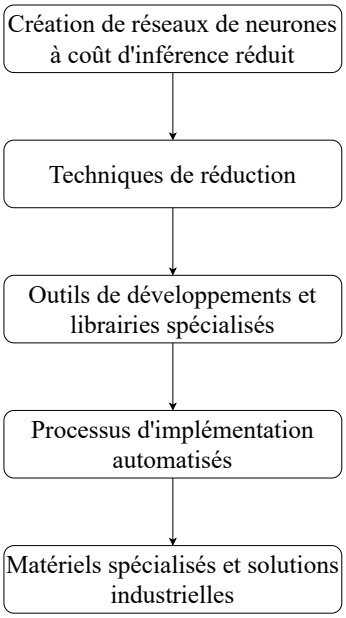


FIGURE 2.1

L'étude de l'état de l'art aborde en premier lieu les aspects de conception de CNN à coût d'inférence réduit. Les techniques de réduction de CNN sont abordées ensuite. Puis les environnements de développement et librairies spécialisées sont présentés. Les méthodes d'évaluation et d'intégration sont détaillées. Finalement, les cibles matérielles spécialisées dans l'exécution des CNN sont abordées, ainsi que les solutions industrielles proposées. Suite à l'analyse de cet état de l'art, les besoins non couverts par les solutions introduites sont identifiés pour affiner la problématique initiale de la thèse.

2.2 Création de réseaux de neurones convolutifs

Cette section présente des CNN de l'état de l'art. La première partie présente des CNN conçus manuellement et détaille les mécanismes de réduction du coût d'inférence mis en jeu dans ces réseaux. La seconde partie de cette section introduit le concept de recherche automatique d'architectures de réseau de neurones - *Neural Architecture Search* (NAS).

2.2.1 Réseaux de neurones à faible coût d'inférence

Cette partie présente des architectures de réseaux de neurones spécifiquement conçues pour réduire le coût d'exécution d'une inférence. Trois architectures de l'état de l'art sont présentées : SqueezeNet [40], MobileNet [41] et ResNet [28].

Le réseau de neurones SqueezeNet a pour objectif de maintenir une précision en classification similaire aux réseaux de neurones comme AlexNet, mais avec moins de paramètres. Pour ce faire, Iandola et al. [40] ont déployé trois stratégies de conception.

1. Remplacer une majorité des filtres de convolution de taille 3×3 par des filtres de taille 1×1 pour un nombre de filtres de convolution donné. Un filtre de taille 1×1 ayant 9 fois moins de paramètres qu'un filtre de taille 3×3.
2. Réduire le nombre de canaux d'entrée des filtres de convolution de taille 3×3 à l'aide des filtres de taille 1×1. Dans une image couleur RGB, les canaux d'entrées sont au nombre de 3, un canal R - *Red*, un canal G - *Green* et un canal B - *Blue*. Le nombre de canaux d'entrées varie en fonction des couches du réseau. Ainsi, le nombre de paramètres nécessaires à une convolution de taille 3×3 est réduit. Le nombre de paramètres dans une couche de convolution de filtres de taille 3×3 est défini suivant l'Équation 2.1. En diminuant le nombre de canaux d'entrées de la couche, le nombre total de paramètres nécessaires est donc réduit.

$$\text{Nbr paramètres} = \text{Nbr canaux d'entrées} \times \text{Nbr filtres} \times (3 \times 3) \tag{2.1}$$

3. Sous-échantillonner tard pour que les couches de convolution aient des cartes de caractéristiques larges. Des couches d'activation larges doivent permettre d'obtenir une précision de classification plus élevée [42].

Les deux premières stratégies ont pour objectif de réduire le nombre de paramètres du réseau tout en maintenant la précision de classification. La troisième stratégie a pour objectif de maximiser la précision de classification avec un budget de paramètres donné. L'application de ces trois stratégies de conception a donné lieu à un réseau SqueezeNet basé sur un module appelé *Fire*. Ce module est représenté schématiquement sur la Figure 2.2.

Le traitement appliqué dans ce module est divisé en deux parties. Le premier est une convolution en compression - *squeeze*. Il s'agit de l'application de la stratégie 1 et 2. Les filtres de taille $n \times m$, ici 1×1, et de profondeur $d2$ sont appliqués sur une entrée de taille $i \times j$ et de profondeur d. La profondeur $d2$ doit être inférieure à la profondeur d pour ainsi réduire le nombre de canaux d'entrée des filtres suivants et respecter la stratégie de conception 2. Puis, des filtres de taille 1×1 et de taille 3×3 sont

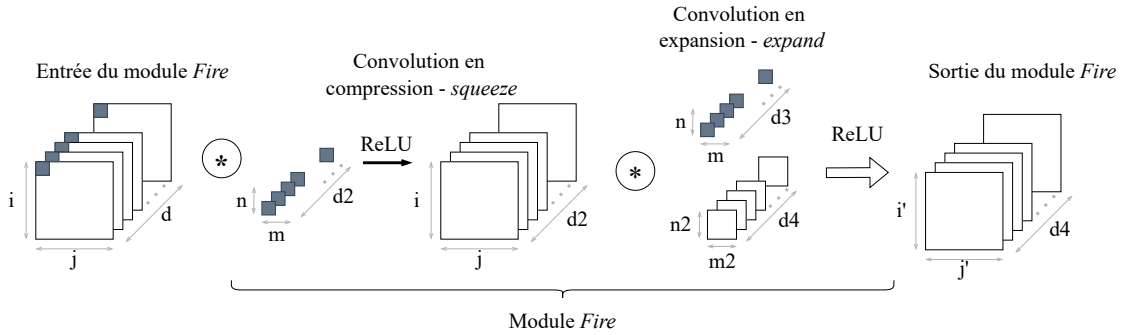


FIGURE 2.2 – Représentation schématique d'un module *Fire* utilisé dans le réseau de neurones SqueezeNet. Ce module *Fire* applique 2 traitements sur l'entrée. Le premier est une convolution en compression. Le second est une convolution en expansion.

TABLE 2.1 – Comparaison de la complexité de SqueezeNet et AlexNet [24] sur ImageNet.

Réseau	**Nombre de paramètres**	**Précision Top-1**	**Précision Top-5**
SqueezeNet	1 250 000	57.5%	80.3%
AlexNet	60 000 000	57.2%	80.3%

appliqués pour extraire les caractéristiques de l'entrée. Pour assurer une limitation du nombre de paramètres dans le réseau, la profondeur $d2$ doit être inférieure à la profondeur $d3 + d4$. Grâce à l'application de ces trois stratégies de conception, le réseau SqueezeNet réduit considérablement le nombre de paramètres nécessaires avec une précision en classification similaire à celle d'un réseau comme AlexNet. Le Tableau 2.1 illustre le rapport entre la précision des réseaux SqueezeNet et AlexNet et leur nombre de paramètres.

Le réseau MobileNet est basé sur une extraction des caractéristiques appelée convolution séparable profonde - *depthwise separable convolution*, introduite par Sifre [43]. Cette opération appliquée au sein des réseaux de neurones convolutifs est représentée schématiquement sur la Figure 2.3. Il s'agit d'une factorisation de la convolution standard en deux étapes, une première pour appliquer le filtre permettant d'extraire des caractéristiques et une seconde pour combiner les caractéristiques extraites précédemment.

L'objectif d'une telle factorisation est de réduire le coût de calcul des extractions

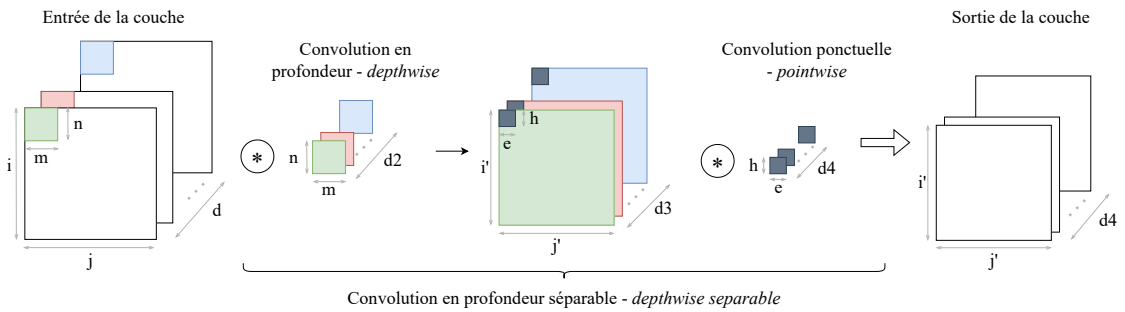


FIGURE 2.3 – Représentation schématique de la convolution séparable profonde utilisée dans le réseau MobileNet. La première partie de l'extraction de caractéristiques est une convolution en profondeur. La seconde partie est une convolution ponctuelle.

de caractéristiques faites avec une convolution classique. En effet, le coût de calcul d'une convolution classique est défini suivant l'Équation 2.2 où $i \times j$ est la taille de l'entrée, d est le nombre de canaux de l'entrée, $n \times m$ est la taille des noyaux de convolution et $d2$ est le nombre de noyaux.

$$i \times j \times d \times n \times m \times d2 \tag{2.2}$$

En décomposant l'étape de convolution en deux étapes, le coût de calcul de la convolution en profondeur séparable est défini suivant l'équation 2.3.

$$(i \times j \times d \times d2) + (n \times m \times i \times j \times d) \tag{2.3}$$

Ainsi la réduction en coût de calcul pour l'extraction de caractéristiques faite avec la convolution en profondeur séparable est définie suivant l'équation 2.4.

$$\frac{(i \times j \times d \times d2) + (n \times m \times i \times j \times d)}{i \times j \times d \times n \times m \times d2} = \frac{1}{d2} + \frac{1}{(n \times m)} \tag{2.4}$$

Outre cette réduction du coût de calcul des opérations d'extraction de caractéristiques, deux multiplicateurs sont introduits pour réduire la complexité du réseau. Il s'agit d'un multiplicateur de profondeur noté α ainsi que d'un multiplicateur de résolution noté ρ. Le multiplicateur de profondeur est défini tel que $\alpha \in (0, 1]$ avec des valeurs standard de 1, 0.75, 0.5 et 0.25. L'architecture de base du réseau MobileNet est pour un multiplicateur $\alpha = 1$. Le multiplicateur de résolution est défini tel que $\rho \in [0, 1]$. Il est appliqué sur l'entrée de la couche et les calculs internes de la couche. Les valeurs de ρ permettent d'avoir une résolution de l'entrée de 224, 192, 160 ou 128. En reprenant l'équation 2.3 et en y intégrant les multiplicateurs, le coût de calcul de l'extraction de caractéristiques du réseau MobileNet est maintenant défini par l'équation 2.5. α étant appliqué à d et $d2$, ρ étant appliqué à i et j, les deux multiplicateurs ont pour effet de réduire le coût de calcul par α^2 et par ρ^2.

$$(\rho.i \times \rho.j \times \alpha.d \times \alpha.d2) + (n \times m \times \rho.i \times \rho.j \times \alpha.d) \tag{2.5}$$

Le réseau MobileNet obtient une précision en classification supérieure à des réseaux comme VGG16, SqueezeNet ou AlexNet. Les auteurs ont comparé le coût d'exécution de MobileNet en nombre de multiplications-accumulations (MACs) nécessaires à une inférence et en nombre de paramètres, comme dénotés dans le Tableau 2.2. Aujourd'hui, une version de MobileNet améliore l'équilibre précision en classification et coût d'une inférence, il s'agit de MobileNetV2 [44]. Une version de MobileNetV3 [45] a été proposée, utilisant les techniques NAS présentées dans la partie suivante.

Le réseau ResNet a été introduit avec le concept de connexions résiduelles. Après la victoire du réseau AlexNet à la compétition ImageNet en 2012, les réseaux de neurones ont été conçus de plus en plus profonds dans l'objectif d'augmenter la précision en classification. Cependant, l'augmentation en profondeur des réseaux de neurones a mis en avant un problème lors de la phase d'entrainement de ces réseaux : la disparition ou l'explosion du gradient - *vanishing or exploding gradient* [46]. Pour éviter ce problème lors de l'entrainement, He et al. [28] ont introduit le concept de bloc résiduel au sein du réseau ResNet. Ces blocs résiduels sont constitués de plusieurs couches et d'une connexion résiduelle, connectant une couche à une autre en court-circuitant les couches intermédiaires. Un bloc résiduel est schématisé sur la

TABLE 2.2 – Comparaison des performances de MobileNet, VGG16, SqueezeNet et AlexNet sur ImageNet. α et ρ sont les multiplicateurs du réseau MobileNet.

Réseau	**Nombre de paramètres**	**MAC pour une inférence**	**Précision Top-1**
$\alpha = 1$ et $\rho = 224$ MobileNet	4 200 000	569 millions	70.6%
$\alpha = 0.5$ et $\rho = 160$ MobileNet	1 320 000	76 millions	60.2%
VGG-16	138 000 000	15300 millions	71.5%
SqueezeNet	1 250 000	1700 millions	57.5%
AlexNet	60 000 000	720 millions	57.2%

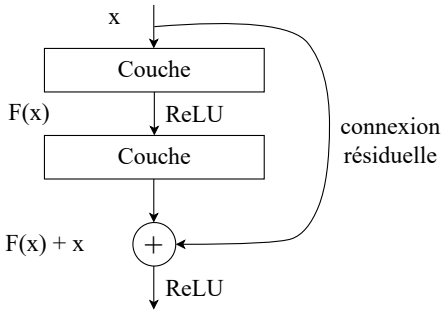


FIGURE 2.4 – Représentation schématique d'une connexion résiduelle utilisée dans le réseau ResNet. Cette connexion apporte une forme de régularisation pour réduire les problèmes de disparition ou d'explosion du gradient lors de la phase d'entrainement du réseau de neurones.

Figure 2.4. Un réseau ResNet est conçu en empilant ces blocs résiduels. Ainsi, des profondeurs variables de ResNet ont donné lieu à plusieurs réseaux, de ResNet152 pour 152 couches à ResNet18 pour 18 couches. ResNet152 obtient une précision en classification supérieure à VGG16 avec un nombre de paramètres moindre, comme dénoté dans le Tableau 2.3. Les auteurs ont comparé le coût de calcul de leur réseau pour une inférence en nombre d'opérations flottantes par seconde (FLOPS).

Ces nouvelles architectures de réseaux de neurones convolutifs ont marqué l'état de l'art. MobileNet et ResNet sont encore très populaires aujourd'hui. Cependant le nombre d'applications embarquées étant très variés et spécifiques aux contraintes de chaque application, il serait nécessaire de créer autant de réseaux de neurones que d'applications embarquées. La création d'un réseau de neurones est chronophage et manuelle. De plus, aucune règle de conception de réseau de neurones n'existe, ce qui rend la conception de nouveaux réseaux difficile. La partie suivante présente une technique de conception automatique de CNN qui tend à résoudre ces problématiques.

TABLE 2.3 – Comparaison de la complexité de ResNet152 et VGG16 sur ImageNet.

Réseau	**Nombre de paramètres**	**FLOPS pour une inférence**	**Précision Top-1**	**Précision Top-5**
ResNet152	60 000 000	11.3 milliards	78.57%	90.67%
VGG-16	138 000 000	15.3 milliards	71.93%	94.29%

2.2.2 Recherche de nouvelles architectures de réseau de neurones - *Neural Architecture Search* - NAS

La recherche d'architectures de réseau de neurones - *Neural Architecture Search* (NAS) est une méthode pour automatiser la conception d'un réseau de neurones en tenant compte de contraintes et d'une base de données. Ces méthodes sont basées sur l'utilisation d'algorithmes comme l'optimisation Bayésienne, les méthodes évolutives telles que les algorithmes génétiques, la descente de gradient ou encore l'apprentissage par renforcement. Les méthodes NAS peuvent être formulées comme un problème où il faut trouver un réseau de neurones α qui appartient à l'espace de recherche $\mathcal{A}$ et qui maximise une fonction objectif nommée $\mathcal{L}$. Cette fonction objectif peut être la précision en classification ou encore des contraintes de ressources matérielles comme l'espace mémoire. La maximisation de cette fonction objectif peut-être faite à travers une stratégie d'estimation des performances du réseau. Une vue schématique illustre le fonctionnement des méthodes NAS sur la Figure 2.5. L'espace de recherche contient les représentations possibles des modèles de réseaux de neurones. Par exemple, le bloc résiduel du réseau ResNet présenté précédemment (*c.f. Section 2.2.1*) peut faire partie de cet espace de recherche. Ainsi le réseau de neurones résultant de la méthode NAS sera basé sur le bloc résiduel. L'avantage d'incorporer des propriétés de modèles de réseaux de neurones connus est de profiter des améliorations apportées par ces types d'architectures. Cependant, cela introduit un biais humain dans le processus de conception et peut donc limiter la découverte de nouveaux blocs architecturaux. La stratégie de recherche détaille comment explorer l'espace de recherche. Cette stratégie doit être équilibrée pour trouver rapidement un modèle de réseau de neurones, mais éviter des convergences trop rapides vers des modèles non optimisés. La stratégie d'estimation des performances doit estimer les performances du modèle de réseau de neurones trouvé, par exemple la précision en classification sur des données jamais analysées par le réseau, ou le besoin mémoire pour une inférence du réseau.

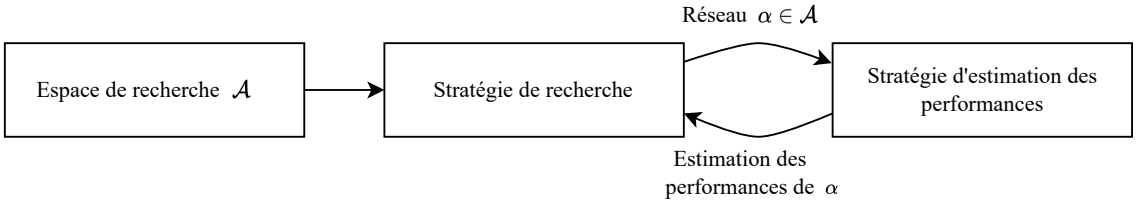


FIGURE 2.5 – Représentation schématique du fonctionnement des méthodes NAS [47].

Les méthodes NAS ont permis de découvrir des réseaux de neurones plus précis que certains réseaux de neurones conçus à la main, pour des tâches de classification d'images [48] ou de traduction [49]. Une possibilité de la méthode NAS est de prendre en compte les caractéristiques de la cible matérielle, par exemple un microcontrôleur. Lin et al. [50] ont proposé MCUNet, qui combine une méthode NAS nommée TinyNAS, à un moteur d'inférence nommé TinyEngine. TinyNAS permet de trouver un réseau de neurones et TinyEngine d'optimiser l'inférence du réseau de neurones trouvé. Le réseau MCUNet, codé en 8-bit entier, a été implémenté dans le microcontrôleur STM32F746. Sur la base de données ImageNet, MCUNet présente une précision en classification de 70,7%. Les besoins en mémoire FLASH et en

mémoire RAM sont respectivement 6 fois et 13,8 fois moins importants pour MCUNet par rapport à MobileNetV2 avec un coefficient $\alpha = 0,75$, pour une précision en classification similaire sur ImageNet. De plus, l'utilisation du moteur d'inférence TinyEngine a accéléré l'inférence de 1,7 fois et 3,3 fois respectivement par rapport à une implémentation faite en utilisant la librairie CMSIS-NN et TensorFlow Lite for Microcontrollers. Lin et al. [51] ont ensuite proposé une autre version de leurs travaux, nommée MCUNetV2. En analysant des réseaux de neurones existants, comme MobileNetV2, MCUNetV2 apporte des modifications dans l'architecture du réseau analysé pour réduire sa consommation en espace mémoire.

Banbury et al. [52] ont introduit un réseau de neurones convolutifs, MicroNets. Le modèle NAS utilisé a conçu un réseau de neurones adaptés aux microcontrôleurs, basé sur des contraintes en latence et en espace mémoire ainsi que sur des architectures de réseaux de neurones existants. Plusieurs réseaux ont été créés en fonction des microcontrôleurs cibles à savoir STM32F446RE, STM32F746ZG et STM32F767ZI. Les CNN ont été implémentés avec la librairie TensorFlow Lite for Microcontrollers. Les mesures ont été faites au regard de trois applications de références de l'état de l'art : détection visuelle - *visual wake words*, détection audio de mots clés - *audio keyword spotting* et détection d'anomalie - *anomaly detection*. Le réseau résultant de MicroNet a présenté l'optimal de Pareto sur ces trois applications au regard de la latence, l'espace mémoire nécessaire à l'inférence du modèle et la précision en classification.

Fedorov et al. [53] ont pris en compte la précision en classification et la taille mémoire du réseau de neurones avec leur algorithme SpArSe. Cette approche prend en compte une technique de réduction de la taille des réseaux de neurones (*c.f. Section 2.3*) : l'élagage. Basé sur des opérations de convolutions et de regroupement par valeur maximale - *maximum pooling*, les réseaux de neurones résultants de SpArSe ont une précision en classification de l'ordre de l'état de l'art sur des bases de données comme MNIST ou CIFAR10.

Liberis et al. [54] ont introduit un modèle NAS nommé μNAS. Leur approche cible les microcontrôleurs avec des espaces mémoire allant de 0.5kB à 64kB. μNAS prend en compte les contraintes de mémoire et de latence pour une inférence. Le réseau résultant et ensuite quantifié en format binaire 8-bit (*c.f. Section 2.3*). Les réseaux obtenus sont évalués sur 5 bases de données, dont MNIST et CIFAR10. Grâce à leur algorithme, ils ont pu soit obtenir une meilleure précision en classification avec le même nombre de paramètres que l'état de l'art, soit obtenir une précision en classification similaire avec un nombre réduit de paramètres par rapport aux modèles de l'état de l'art.

Les méthodes NAS équilibrent les compromis entre les contraintes de précision, de latence ou d'espace mémoire nécessaire à l'inférence d'un CNN. Cependant le temps de recherche nécessaire à une nouvelle architecture de CNN peut-être très important, plusieurs centaines d'heures d'exécution sur des GPU [55]. Cette contrainte rend le processus couteux pour satisfaire le nombre important d'applications embarquées avec des contraintes différentes. De plus les architectures des réseaux de neurones trouvés par les méthodes NAS sont parfois complexes à implémenter au sein d'une cible embarquée [56]. Finalement, aucune approche basée sur NAS n'a permis de concevoir un CNN basé sur des contraintes énergétiques ou de durée de vie de la batterie du système cible.

Synthèse de la conception de CNN à coût d'inférence réduit :

Cette section présente la conception de CNN à coût d'inférence réduit, basée sur une approche manuelle ou une approche automatique de type NAS. Malgré la conception de ce type de CNN, il peut être nécessaire d'appliquer des techniques de réduction au CNN. En effet, devant la variété des contraintes des systèmes embarqués, concevoir un CNN pour chaque application n'est pas envisageable. La conception manuelle de CNN est très chronophage. La conception de CNN basée sur NAS nécessite une très forte puissance de calcul, pour un résultat parfois complexe à implémenter au sein d'un MCU. La possibilité d'adapter un CNN existant pour une application cible permettrait d'éviter des conceptions de CNN nombreuses et chrono phages. Ces techniques de réduction permettant d'adapter le CNN aux contraintes de l'application cible sont présentées dans la Section suivante.

2.3 Techniques de réduction

Devant la taille importante de certains CNN, une réduction de leur taille tout en maintenant une précision en classification acceptable a été envisagée. Pour cela, des techniques de réduction des CNN ont été développées. Ces techniques de réduction peuvent être catégorisées suivant la Figure 2.6. La factorisation matricielle consiste à décomposer les calculs mis en jeu dans les CNN pour rendre leur exécution moins coûteuse. La quantification des poids du réseau exprime des poids dans un format binaire donné vers un format binaire moins demandeur en espace mémoire et en ressources de calculs. L'élagage consiste en la suppression des éléments du réseau ne contribuant pas ou peu à la classification fournie en sortie du CNN. La distillation de connaissances permet d'entraîner un réseau de neurones en tenant compte de la fonction de classification d'un autre réseau de neurones. Une réduction hybride correspond à l'application de plusieurs techniques de réduction comme par exemple l'élagage et la quantification appliqués sur le même CNN. Ces techniques de réduction sont détaillées dans la suite de cette section.

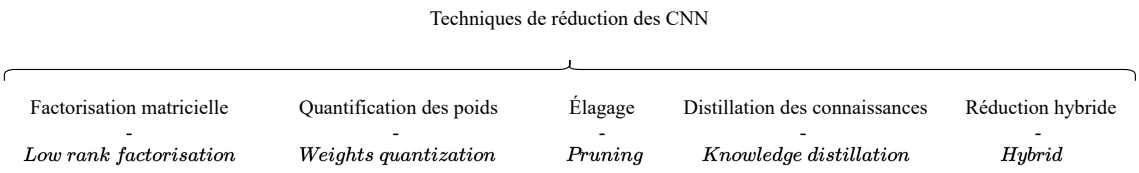


FIGURE 2.6 – Catégorie des techniques de réduction.

2.3.1 Factorisation

Les opérations sur des tenseurs ou des matrices sont la base des calculs des réseaux de neurones. Les techniques de factorisation peuvent être appliquées sur les structures tensorielles et/ou matricielles des CNN. La factorisation en matrices plus petites permet d'éliminer les calculs redondants. C'est ce qui a inspiré la convolution en profondeur séparable du réseau MobileNet ou le module *Fire* du réseau SqueezeNet présentés précédemment (*c.f. Section 2.2.1*). Ces techniques peuvent amener à des réductions de taille de réseau et de coût d'inférence important.

Denton et al. [57] ont analysé plusieurs décompositions sur des matrices de poids préentrainées, comme la décomposition en valeur singulière - Singular Value Decomposition (SVD) ou la décomposition polyadique canonique - Canonical Polyadic (CPD). Sur un CNN de 15 couches et sur la base ImageNet, la vitesse d'exécution des couches de convolution a été multipliée par 2-3x et le nombre de paramètres dans les couches entièrement connectées a été réduit par 5-10x. Tai et al. [58] ont également appliqué une décomposition SVD et ont compressé AlexNet sur la base d'images ImageNet. Avec une compression de 5x, le débit du réseau a été multiplié par 1,8 avec une perte en précision de classification de 0,5%. Hayashi et al. [59] ont appliqué sur des opérations de convolutions une CPD et ont mis en avant que de manière constante cette décomposition offre un bon équilibre entre la complexité de calcul et la précision. Certaines d'entre elles cependant ne permettent pas de fournir un équilibre entre la complexité de calcul résultante de la décomposition et la précision en classification [60]. La décomposition de Tucker-2 d'un réseau de neurones n'offre pas une précision en classification au niveau de l'état de l'art [61]. La CPD ou encore la décomposition en Batch Normalization (BMD) semble offrir la meilleure précision en classification.

Les factorisations réduisent le coût d'inférence d'un CNN tout en maintenant une précision en classification correcte. Cependant, l'état de l'art s'accorde sur un défaut majeur de ces méthodes : le manque de compréhension théorique pour expliquer pourquoi certaines décompositions fonctionnent mieux que d'autres. De plus, le coût de calcul supplémentaire pour réaliser une décomposition compense parfois le gain amené par la décomposition. Finalement les décompositions actuelles sont plus adaptées aux couches entièrement connectées qui présentent plus de redondances par rapport aux couches de convolutions [62].

2.3.2 Quantification

La quantification est un sujet très actif de la recherche. Cette technique est appliquée au réseau de neurones, mais a trouvé de nombreuses applications dans d'autres domaines. Le lecteur peut se référer aux travaux de Gray et Neuhoff pour un résumé de l'historique de la quantification [63]. La quantification appliquée aux réseaux de neurones a pour objectif de diminuer la précision de la représentation binaire utilisée pour modéliser les données mises en jeu par un réseau de neurones. Par défaut, le format binaire utilisée est à virgule flottante sur 32 bits (FP32). L'objectif de cette réduction est de diminuer l'espace mémoire nécessaire à une inférence ainsi que de simplifier les opérations à réaliser dans l'objectif d'accélérer l'inférence. Plusieurs paramètres peuvent être quantifiés au sein d'un réseau de neurones : les paramètres du réseau comme les poids, les données de propagation dans le réseau comme les cartes de caractéristiques ou les données de mis à jour comme les gradients [62]. Deux catégories de quantification existent, la quantification uniforme et la quantification non uniforme. La différence entre ces deux quantifications est illustrée sur la Figure 2.7. Les valeurs réelles du domaine continu r sont converties en valeurs discrètes et moins précises du domaine quantifié Q. La quantification non uniforme permet de mieux capturer l'information du signal quantifié. Cependant, elle est difficile à déployer. Aujourd'hui, la quantification uniforme est majoritairement utilisée de par sa facilité d'utilisation et son efficacité une fois déployée.

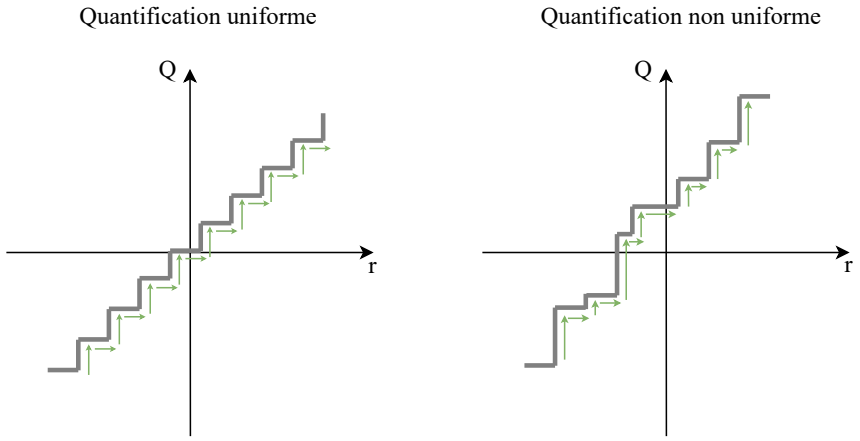


FIGURE 2.7 – Différence entre une quantification uniforme à gauche et non uniforme à droite. Les valeurs réelles du domaine continu r sont converties en valeurs discrètes et moins précises du domaine quantifié Q. La distance entre les valeurs quantifiées est la même pour une quantification uniforme et est différente pour une quantification non uniforme.

Un schéma de quantification classique [64] peut-être défini suivant l'équation 2.6, où Q représente la fonction de quantification, r représente la valeur réelle à quantifier, S est le facteur de mise à l'échelle, Z est la valeur quantifiée représentant le 0 réel et Int est une fonction d'arrondi.

$$Q(r) = Int(\frac{r}{S}) + Z \tag{2.6}$$

Le schéma inverse de la quantification exprimé dans l'équation 2.6 est possible. Il s'agit de la déquantification, définie dans l'équation 2.7 où $Q(r)$ représente la valeur quantifiée. La valeur réelle $\tilde{r}$ ne sera pas exactement la même que la valeur d'origine r de l'équation 2.6.

$$\tilde{r} = S \times (Q(r) - Z) \tag{2.7}$$

Une quantification uniforme telle que définie ci-dessus peut-être symétrique ou asymétrique. Cela dépend du paramètre de mise à l'échelle S, définie dans l'équation 2.8. Les paramètres α et β définissent la plage d'écrêtage des valeurs réelles, il s'agit de la calibration, et b représente le nombre de bit représentant la valeur quantifiée.

$$S = \frac{\beta - \alpha}{2^b - 1} \tag{2.8}$$

Un choix classique pour un schéma de quantification asymétrique correspond à $\alpha = r_{min}$ et $\beta = r_{max}$ alors que pour un schéma de quantification symétrique $-\alpha = \beta$. Un avantage indéniable de la quantification symétrique est que la valeur de Z de l'équation 2.6 est 0, simplifiant ainsi les calculs de quantification. La Figure 2.8 schématise un schéma de quantification symétrique et asymétrique.

De nombreux schémas de quantification existent. La plage des valeurs d'écrêtage peut-être dynamique ou statique. Les cartes de caractéristiques d'un réseau de neurones différent en fonction de l'entrée du réseau. Ainsi, la quantification avec une plage d'écrêtage dynamique offre une meilleure précision en classification, la plage

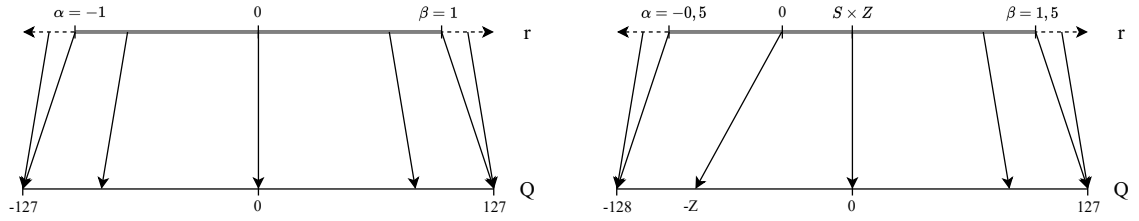


FIGURE 2.8 – Différence entre une quantification symétrique à gauche et une quantification asymétrique à droite [65] pour le cas d'une quantification en format binaire entier sur 8 bits d'un réseau de neurones.

d'écrêtage s'adaptant à chaque entrée du CNN. Cependant le coût en calcul à exécuter à chaque inférence est important. De ce fait, en pratique une quantification avec une plage d'écrêtage statique, quelle que soit l'entrée est favorisée. Les schémas de quantification peuvent également être appliqués à différentes granularités au sein du réseau comme la quantification par couche, par groupe ou par canal. Ces schémas de quantification peuvent être appliqués de deux manières pour obtenir un réseau de neurones quantifié : l'entrainement tenant compte de la quantification - Quantization Aware Training (QAT) ou la quantification après l'entrainement - Post Training Quantization (PTQ). QAT offre souvent la meilleure précision en classification, mais le coût de l'entrainement du réseau de neurones est plus important. PTQ quantifie des réseaux de neurones un peu moins précis en classification qu'avec QAT, mais il s'agit de la quantification la plus simple et rapide à appliquer, nécessitant peu ou pas d'entrainement supplémentaire pour le réseau de neurones.

Dans le cadre de cible matérielle de type microcontrôleurs, une quantification du réseau de neurones à implémenter est aujourd'hui incontournable. Certains MCU n'étant pas équipés d'unité de calcul pour les formats binaires à virgule flottante, Zhou et al. [66] ou Chen et al. [67] ont concentré les efforts de quantification sur les poids du réseau de neurones. Cela a pour effet de diminuer les besoins en espace mémoire pour stocker le réseau, mais ne diminue pas ou très peu le coût d'exécution d'une inférence. Jacob et al. [64] ont développé une technique de quantification et d'entrainement d'un modèle quantifié, en poids et en activation, pour effectuer l'inférence complète d'un réseau de neurones en format binaire entier sur 8-bits (INT8). Sur la base ImageNet, les réseaux ResNet en INT8 perdent 2% de précision en classification pour un besoin en espace mémoire divisé par 4. Certaines études ont appliqué une quantification vers un format binaire plus petit que INT8 [68], [69]. Zhuo et al. [70] ont étudié l'équilibre coût d'une inférence, espace mémoire économisé et perte en précision pour 4 réseaux de neurones sur 4 bases de données, comme ResNet8 sur la base CIFAR10 ou MobileNetV1 sur la base Visual Wake Words (VWW). Ils ont fait varier le format binaire de 8-bits à 2-bits. Par rapport à une inférence en 8-bits, descendre jusqu'à 4-bits permet d'économiser plus de 50% de ressources mémoire et de calculs sans perdre trop de précision en classification. Aller en dessous de 4-bits dégrade fortement la précision en classification. Cependant, les auteurs n'ont pas implémenté les modèles réduits pour faire des mesures sur le matériel. Les gains liés à la réduction du format binaire pourraient être différents au sein des MCU. Le format 8-bits étant le plus petit format supporté, passer vers un format binaire plus petit nécessite d'émuler ce format en 8-bits. Des quantifications vers des formats binaires extrêmement réduits ont été explorées. La quantification

binaire proposée par Courbariaux et al. [71] transforme les valeurs réelles en deux valeurs possibles : +1 ou -1. Cette quantification extrême peut-être une binarisation déterministe, comme définie dans l'équation 2.9, ou stochastique, comme définie dans l'équation 2.10. w est la valeur réelle, w_b la valeur binaire et σ est la fonction *hard sigmoid*.

$$w_b = \begin{cases} +1 & \text{si } w \geq 0 \\ -1 & \text{sinon.} \end{cases} \tag{2.9}$$

$$w_b = \begin{cases} +1 & \text{avec une probabilité } p = \sigma(w) \\ -1 & \text{avec une probabilité } 1 - p \end{cases} \tag{2.10}$$

Ces quantifications binaires offrent des résultats en précision de classification cohérents avec plus de 90% de précision sur la base CIFAR10 et plus de 98% de précision sur la base MNIST. Rastegari et al. [72] ont implémenté un réseau de neurones binaires nommé XNOR-Net, mettant en avant que les opérations binaires peuvent être effectuées sur le matériel cible avec des opérations XNOR [73]. Cette implémentation leur a permis de réduire le besoin mémoire d'un facteur 32 et le temps d'inférence d'un facteur 58 par rapport au modèle en FP32. Przewlocka-Rus et al. [74] ont étudié une quantification logarithmique de base-2. Cette quantification peut également être effectuée avec de simple décalage de bit au sein du matériel cible, ce qui diminue grandement le coût d'exécution du réseau de neurones. Li et al. [75] ont introduit une quantification ternaire, basée sur trois valeurs possibles : -1, 0 et +1. Cette ternarisation est basée sur la réduction de la distance euclidienne entre la valeur du poids en précision initiale W et la valeur du poids en valeur ternaire W^t. Cette fonction de ternarisation peut-être définie suivant l'équation 2.11 où Δ est un paramètre positif de seuil.

$$W_i^t = \begin{cases} +1 & \text{si } W_i \sup \Delta \\ 0 & \text{si } |W_i| \leq \Delta \\ -1 & \text{si } W_i \inf -\Delta \end{cases} \tag{2.11}$$

Sur les bases MNIST et CIFAR10, la précision en classification est légèrement meilleure que la binarisation réalisée par Courbariaux et al. [71]. Sur la base ImageNet, la ternarisation de ResNet18 a permis de réduire les besoins mémoires par 16 par rapport au modèle en FP32, offrant cependant une précision en classification Top-1 de 65,3% et Top-5 de 86,2%.

Dans son ensemble les techniques de quantification sont nombreuses et incontournables lorsqu'il s'agit de réduire la taille des CNN. Les avantages indéniables de ces techniques sont la réduction de l'espace mémoire nécessaire et parfois la diminution de la latence pour une inférence. Cependant ces techniques demandent régulièrement un ré-entrainement du CNN ce qui peut être couteux. De plus, concernant les quantifications extrêmes comme la binarisation, la ternarisation ou une quantification en dessous de 4 bits, leur gain en termes de latence ou de consommation énergétique a été peu exploré sur des MCU. Ainsi le gain théorique de ces techniques pourrait être réduit en pratique, les MCU ne supportant pas les formats binaires en dessous de 8 bits.

2.3.3 Élagage

L'élagage - *pruning* en anglais, consiste en la suppression des connexions redondantes ou des neurones qui ne contribuent pas ou peu à la précision du résultat dans un réseau de neurones. Ces techniques sont inspirées des travaux de LeCun et al. [76] et Hassibi et al. [77]. Les méthodes d'élagage d'aujourd'hui peuvent être catégorisées en deux types. Les méthodes d'élagage non structurées - *Fine-grained pruning*, illustrées sur la Figure 2.9, et structurées - *Coarse-grained pruning*, illustrées sur la Figure 2.10.

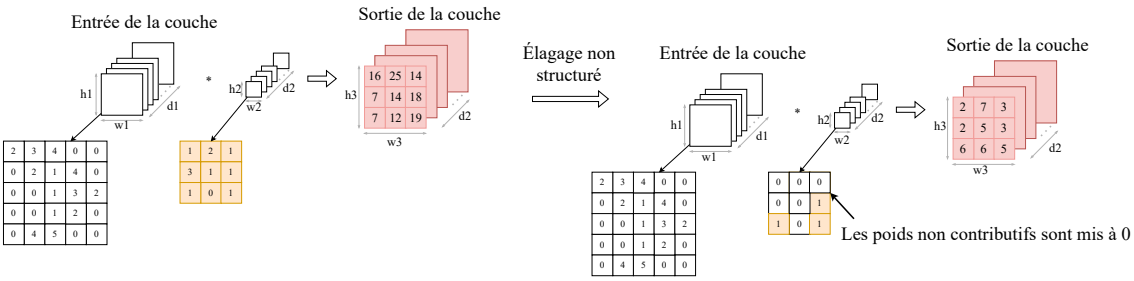


FIGURE 2.9 – Schéma d'un élagage non structuré sur des noyaux de convolution.

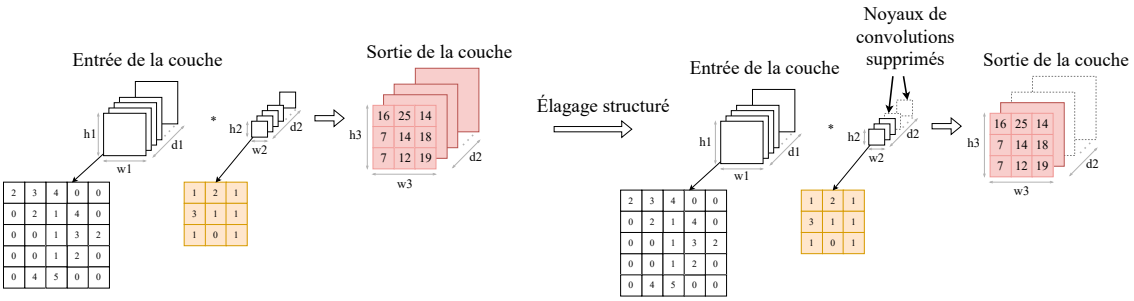


FIGURE 2.10 – Schéma d'un élagage structuré sur des noyaux de convolution.

Han et al. [78] ont proposé un processus d'élagage en itération. Après une étape d'entrainement du réseau de neurones, les poids dont la valeur est en dessous d'une valeur seuil sont élagués. Puis le réseau est ré-entrainé pour adapter les poids restants à l'élagage. Avec cette technique, sur la base ImageNet les réseaux de neurones AlexNet et VGG16 ont été compressés respectivement d'un facteur 9 et d'un facteur 13. Guo et al. [79] ont appliqué leur méthode nommée Dynamic Network Surgery (DNS). L'élagage appliqué dans DNS est divisé en deux étapes appliquées en boucle, l'élagage puis le recollage - *pruning and splicing*. Une fois les connexions non importantes élaguées, si certaines d'entre elles semblaient importantes elles sont recréées. Cette méthode résulte en une compression des réseaux de neurones LeNet5 de 108 fois par rapport au réseau original, et AlexNet de 17,7 fois. Carreira-Perpinan et al. [80] ont défini l'élagage comme un problème d'optimisation où les poids doivent minimiser la fonction perte du réseau tout en satisfaisant une condition de coût de l'élagage. Sur la base d'images CIFAR10 et avec un réseau comme ResNet, ils ont réussi à maintenir la précision en classification du réseau de base avec un réseau réduit gardant entre 5% et 10% des poids initiaux. Ces techniques d'élagage non structurées autorisent un élagage très agressif sans perte de précision en classifica-

tion significative. Cependant, cela se traduit souvent par des structures matricielles résultantes creuses, qui sont difficiles à implémenter efficacement [81].

Avec une approche d'élagage structurée, un groupe d'éléments du réseau de neurones est supprimé, par exemple un filtre de convolution ou une carte de caractéristique. Élaguer un filtre revient à supprimer également la carte caractéristique et le filtre associé suivant. Li et al. [82] ont utilisé la norme l_1 pour élaguer des filtres entiers au sein du réseau. Sur la base CIFAR10 et avec les réseaux de neurones VGGNet et ResNet, ils ont pu réduire de 30% le coût de calcul nécessaire à une inférence. He et al. [83] ont introduit le concept d'élagage doux - *soft pruning*. Les filtres du réseau de neurones sont élagués basé sur la norm l_2 mais sont toujours mis à jour pendant l'entrainement, ce qui résulte en une meilleure précision en classification du réseau réduit. Sur la base ImageNet et avec le réseau ResNet, l'accélération pour une inférence a été de 30%. Ye et al. [84] ont élagué les canaux d'un réseau se basant sur le paramètre d'échelle des couches de Batch-Normalization du réseau ResNet, sur la base CIFAR10. Ils ont élagué jusqu'à obtenir un réseau correspondant à 39% du réseau de base, tout en maintenant une précision cohérente. He et al. [85] ont élagué les filtres de convolution en se basant sur la valeur de la médiane géométrique de tous les filtres - *Filter Pruning via Geometric Median* (FPGM). Tout en maintenant une précision en classification correcte sur CIFAR10, FPGM a réduit le nombre de FLOPS nécessaire à une inférence de ResNet110 de 52%. Sur la base ImageNet, FPGM a réduit le nombre de FLOPS nécessaire à une inférence de 42%. Dans le cas d'élagage structuré, la réduction de la mémoire est effective concernant les poids du filtre supprimé, mais également les cartes de caractéristiques liées à ce filtre. De plus, la latence des calculs à exécuter concernant ce filtre est supprimée. Cependant, avec cette approche un élagage trop agressif peut résulter en une dégradation importante de la précision en classification.

Les techniques d'élagage ont montré une réduction de la complexité des CNN sur lesquels elles sont appliquées. Cependant, les techniques d'élagage nécessitent souvent un réentrainement du réseau de neurones, qui parfois peut-être couteux. De plus les techniques d'élagage non structurées résultent en un modèle avec une structure creuse qui peut-être dure à exécuter si le matériel cible n'est pas spécifique à l'algorithme. Ainsi, les gains en latence ou consommation d'énergie ne seront pas évidents. Les techniques à élagage structurées résultent souvent en des réseaux de neurones plus simples à implémenter et dont le gain en complexité de calcul est significatif. Cependant, bien que l'état de l'art soit développé sur les techniques d'élagage, peu d'études ont été faites concernant la caractérisation de la réduction apportée par l'élagage sur les paramètres de latence ou de consommation d'énergie.

2.3.4 Distillation des connaissances

L'entrainement d'un CNN est une étape clé qui définit la précision du réseau. Une précision adéquate à l'application cible est parfois difficile à obtenir, particulièrement sur les réseaux de petite taille offrant peu de paramètres à entrainer. Les CNN de grande taille, comportant un grand nombre de paramètres, peuvent apprendre des fonctions complexes [86]. Apporter cette capacité à des petits réseaux de neurones permettrait d'intégrer des CNN de grande précision et de faible taille au sein de systèmes embarqués contraints. Ainsi, cela permettrait d'éviter des phases complexes de réduction du coût d'inférence de réseau de grande taille. Pour ce faire,

la distillation de connaissances - *Knowledge Distillation* (KD) fonctionnant sur un paradigme "enseignant-élève", permet à un petit réseau de neurones (l'élève) de mimer la fonction de classification apprise par un réseau de neurones de profondeur plus importante (l'enseignant). Le mécanisme de KD fonctionne sur 3 aspects : la modélisation des connaissances à distiller, le schéma de distillation et les réseaux de neurones "enseignant-élève". La Figure 2.11 schématise l'interaction entre ces 3 aspects.

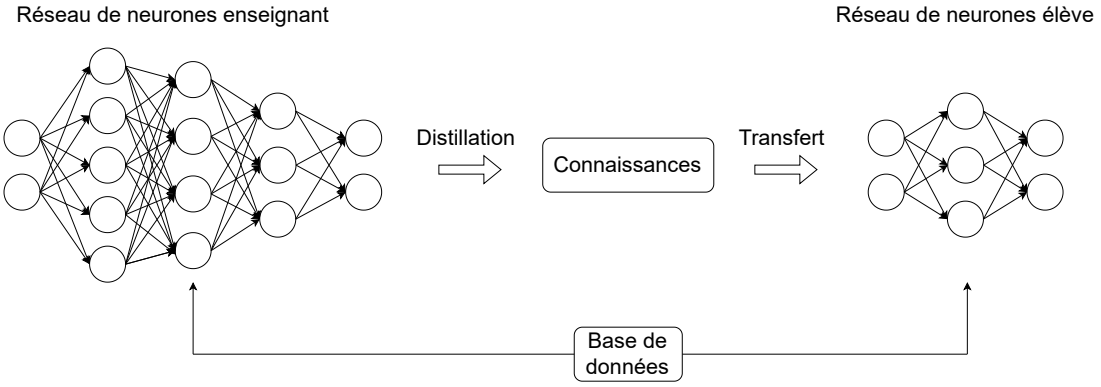


FIGURE 2.11 – Principe de la distillation de connaissances - *Knowledge Distillation*, schéma inspiré des travaux de Gou et al. [87].

Les premiers travaux appliquant cette politique de compression sont ceux de Bucilua et al. [88]. Appelé au début Transfert de connaissances - *Knowledge Transfer* (KT), le principe était d'entrainer le petit réseau de neurones sur des données labellisées par le gros réseau de neurones. L'idée derrière cette démarche est que des informations supplémentaires se trouvent dans ces données labellisées par le réseau de neurones profond. Par exemple, une entrée ayant plusieurs probabilités de sorties élevées partagerait donc des caractéristiques similaires entre ces sorties. En mimant la fonction de classification du grand réseau, le petit réseau peut atteindre une précision en classification plus importante que si l'entrainement avait été effectué sur les données originales d'entrainement.

Différents types de connaissances peuvent être distillées d'un réseau de neurones à un autre. Hinton et al. [89], ayant démocratisé l'appellation de KD, ainsi que Ba et al. [90] ont utilisé les "logits", les sorties basées sur des couches Softmax des réseaux enseignants et élèves. Une fonction "Distillation Loss" modélise la différence entre la réponse de l'étudiant et la réponse de l'enseignant. Cette fonction est à minimiser pour que le réseau étudiant apprenne. Les sorties des couches intermédiaires d'un réseau peuvent également être utilisées en tant que connaissances à distiller. Le concept de représentation intermédiaire dans la distillation de connaissances a donc été introduit par Romero et al. [91]. Les sorties des couches intermédiaires de l'enseignant ont été utilisées pour distiller la connaissance vers les sorties des couches intermédiaires du réseau élève. Ainsi sur CIFAR10 une compression de 10,4x a été appliquée sur un réseau. Le réseau réduit surpasse la précision en classification du réseau de base. Pour aller plus loin, la relation entre des couches ou des caractéristiques du réseau enseignant et du réseau élève ont été exploré. Yim et al. [92] ont proposé une matrice de processus d'élaboration de solution - *flow of solution process* (FSP), basée sur la matrice de Gram entre une paire de couches. Le réseau étudiant est ensuite entrainé pour que ses matrices FSP se rapprochent le plus possible des

matrices FSP du réseau enseignant.

Une fois la nature de la connaissance identifiée et générée, un schéma de distillation doit être mis en place. Ces schémas de distillation peuvent être dit hors ligne - *offline*, en ligne - *online*, et autodistillé - *self distillation*. Dans le cas d'une distillation hors ligne, le réseau enseignant est entrainé, puis la connaissance est extraite de ce réseau et ensuite transférée à l'étudiant pour entrainement [89], [91]. Ce schéma de distillation est le plus simple à utiliser. Dans le cas d'une distillation en ligne, les réseaux de neurones s'entrainent ensemble et travaillent en "collaboration" [93]. Dans le cas d'une autodistillation, un seul réseau joue le rôle de l'enseignant et de l'élève. Zhang et al. [94] ont proposé un schéma de distillation où les couches les plus profondes du réseau distillent les connaissances dans les couches les moins profondes.

Finalement, les modèles de réseau de neurones mis en jeu dans la distillation de connaissances sont importants. Tous les réseaux de neurones ne peuvent apprendre l'un de l'autre. Le modèle de réseau élèves doit être choisi en fonction du modèle de réseau enseignant et vice versa. De nombreux travaux de l'état de l'art ont utilisé les modèles de réseaux ResNet pour appliquer la distillation [87]. Par exemple, un ResNet110 en tant qu'enseignant et un ResNet56 en tant qu'élève. Cho et al. [95] ont étudié la politique de "plus le réseau enseignant est profond, plus l'apprentissage pour le réseau élève est effectif". Ils ont mis en avant qu'à un certain niveau de précision du réseau enseignant, la précision en classification du réseau élève commence à baisser. Le ratio de précision en classification réseau élève/réseau enseignant est donc important. Ainsi ils ont pu augmenter la précision de plusieurs réseaux ResNet sur la base CIFAR de quelques pour cent.

L'état de l'art présente de nombreux travaux concernant la distillation. Cette technique permet d'augmenter la précision en classification des réseaux de neurones. Cela permet de rendre envisageable l'implémentation de petit réseau de neurones, ayant désormais une précision en classification acceptable. Cependant, la compréhension théorique et les preuves empiriques de l'efficacité de la distillation sont encore aujourd'hui insuffisantes.

2.3.5 Approche Hybride

Les approches de réduction hybrides consistent en l'application de plusieurs techniques de réduction sur le même CNN. En combinant les bénéfices de plusieurs techniques de réduction détaillées dans les sous-sections précédentes, la réduction appliquée aux CNN peut s'en retrouver efficace.

Mishra et al. [96] ont appliqué la quantification et la distillation de connaissances. Le CNN enseignant a distillé sa connaissance vers un CNN élève qui était quantifié. Une quantification ternaire et sur 4-bits a été appliquée sur le CNN élève. Le modèle de CNN ResNet a été utilisé et plusieurs variantes de ce réseau ont été exploitées, à savoir ResNet18-34-50-101. La profondeur du CNN enseignant étant toujours supérieure ou égale à la profondeur du CNN élève. Ainsi, ils ont remarqué que l'efficacité de la distillation est effective en fonction de la précision en classification du réseau enseignant par rapport au réseau élève, et non pas en fonction de la différence de profondeur entre le réseau enseignant et le réseau élève. Les réseaux ResNet distillés et quantifiés ont ainsi fourni une précision en classification supérieure de quelques pourcentages en étant quantifiés à 8-bits pour les activations et 4-bits pour les poids. Dans le même registre de combinaison de techniques de réduction, Polino et al. [97]

ont appliqué la distillation et la quantification. Sur la base CIFAR10, ils ont quantifié un CNN à 2, 4 et 8-bits. En 8-bits et 4-bits et en maintenant la même précision en classification, la réduction par rapport au CNN original est respectivement de 5,3 fois et 17,7 fois. La réduction du modèle 2-bits est de l'ordre de 53 fois, mais la précision en classification se dégrade de manière significative. Alemdar et al. [98] ont entrainé avec la technique de distillation des modèles de CNN quantifiés en format binaire et ternaire. Sur un CNN dérivé du modèle de VGGNet, et sur les bases de données MNIST, CIFAR10 et CIFAR100, ils ont obtenu une précision en classification similaire à l'état de l'art des réseaux de neurones binaires et ternaires.

Li et al. [99] ont introduit une méthode pour combiner les techniques d'élagage et de factorisation. Leur méthode a été testée sur plusieurs CNN dont ResNet et VGGNet ainsi que sur les bases CIFAR10, CIFAR100 et ImageNet. Sur CIFAR10, un réseau ResNet20 a été réduit à 44,55% de son nombre de paramètres de base tout en maintenant une bonne précision en classification.

Aghli et al. [100] ont appliqué la distillation et l'élagage. Le réseau enseignant est élagué, basé sur l'analyse des activations les moins significatives. Puis, la distillation est appliquée vers un réseau élève de taille inférieure au réseau enseignant. Sur la base CIFAR10, ResNet110 et ResNet164 ont respectivement étaient réduit de 4,7 fois et 3,63 fois en perdant seulement 1,27% et 0,57% en précision de classification.

Hawks et al. [101] ont exploré l'élagage combiné à QAT, nommé l'élagage tenant compte de la quantification - *Quantization Aware Pruning* (QAP). Cette technique a été appliquée sur des réseaux de neurones entièrement connectés. Sur un modèle élagué et quantifié en 6-bits, une réduction de 80% a été faite par rapport au réseau de base.

Finalement, Han et al. [102] ont appliqué la technique d'élagage, de quantification et une compression basée sur le codage de Huffman. Cette combinaison de techniques a été appliquée sur AlexNet et VGGNet sur la base ImageNet. Une quantification des couches entièrement connectées a été faite sur 5-bits et des couches de convolutions sur 8-bits. Sans perte de précision en classification, le réseau AlexNet a été réduit d'un facteur 35 et le réseau VGGNet d'un facteur 49. Les auteurs ont également mis en avant que l'élagage ne dégrade pas les performances d'un réseau quantifié. La précision en classification commence à chuter au même nombre de bits appliqués pour la quantification, que le réseau soit élagué ou non.

Synthèse des techniques de réduction :

Cette section présente 5 politiques de réduction pour les CNN. La factorisation des matrices et/ou des tenseurs permet d'éliminer des opérations redondantes. La quantification diminue la précision de la représentation binaire utilisée pour modéliser les données mises en jeu dans le CNN. L'élagage supprime les connexions ou neurones ne contribuant pas ou peu à la précision du CNN. La distillation de connaissances permet à un réseau de neurones de taille modeste d'augmenter sa précision en mimant la fonction de classification d'un réseau de neurones de taille plus importante. Finalement, une approche hybride consiste en l'application de plusieurs de ces techniques de réduction. L'ensemble des politiques de réductions présentées diminue le coût d'inférence des CNN. Cependant, la réduction du coût d'inférence obtenue varie beaucoup en fonction des ou de la technique de réduction appliquée, du CNN à réduire et de la cible matérielle. Il est nécessaire d'appliquer les tech-

niques de réduction pour caractériser le coût d'inférence du CNN réduit. L'étude de la réduction de la consommation énergétique est rarement effectuée dans l'état de l'art. De plus, ces techniques de réduction sont complexes à appliquer. Pour faciliter l'application des techniques de réduction sur les CNN, des environnements de développement et des librairies spécialisées ont été créés. La Section suivante introduit ces outils.

2.4 Environnements de développement et librairies spécialisées

FIGURE 2.12 – Environnements de développement et librairies pour l'apprentissage profond en jaune et les systèmes embarqués en rouge. Des outils de développement ont été créés pour répondre à des problématiques au croisement de ces deux domaines.

Les environnements de développement et les librairies logicielles ont permis la démocratisation de l'utilisation de certaines plateformes. Les outils d'aide au développement comme Keil [103], STM32Cube [104], MCUXpresso [105] ou des librairies adaptées à certaines applications embarquées comme CMSIS [106] accélèrent et simplifient le développement d'une application embarquée sur des cibles de type MCU. Cependant, aucun outil de développement embarqué n'est initialement prévu pour faciliter l'intégration des réseaux de neurones dans les cibles matérielles.

Du côté de l'apprentissage profond, des outils comparables ont permis la démocratisation de l'utilisation des réseaux de neurones. Aujourd'hui en une dizaine de lignes de codes il est possible de lancer un entrainement d'un réseau de neurones de l'état de l'art. Des outils comme Tensorflow [33] de Google et Pytorch [34] de Meta ont ajouté à leur librairie une surcouche de développement simplifiant le développement de réseaux de neurones, respectivement Keras [107] et Caffe2 [108]. Apache Software a également sorti sa librairie MXNet [109] utilisée par Microsoft. La librairie ONNX [110] permet d'adapter un modèle d'une librairie à une autre, par exemple un modèle développé sous Caffe2 utilisable avec Keras. Cependant ces outils nécessitent un développement en langage interprété comme du Python. Ce type de langage est très difficilement embarquable. Les librairies d'apprentissage profond

citées ci-dessus sont inutilisables dans les MCU dont la majorité n'ont pas d'OS ou d'interpréteur.

Pour faciliter l'implémentation d'applications embarquées incorporant des algorithmes d'intelligence artificielle, des librairies et des environnements de développement spécialisés ont été créés. La librairie CMSIS-NN [111], développée par ARM fournit des fonctions en langage C embarqué permettant de créer un réseau de neurones plus simplement. STM32CubeAI [112] est une extension de l'environnement de développement STM32Cube. Cette extension reçoit en entrée un modèle de réseaux de neurones développé sous Keras ou Caffe2, et génère automatiquement un projet C embarqué pour les microcontrôleurs de type STM32. Tensorflow Lite for Microcontrollers [113] a pour objectif de porter des modèles développés sur Keras ou Tensorflow vers tout type de microcontrôleurs.

Ces outils facilitent le développement et le déploiement de réseaux de neurones convolutifs au sein de MCU. Cependant, peu d'informations sur les contraintes non fonctionnelles du CNN sont données au concepteur de système embarqué. Il est nécessaire d'aller loin dans le processus de conception pour caractériser la latence, la consommation énergétique ou l'espace mémoire pour une inférence. De plus, ces outils sont des boîtes noires et il est difficile de contrôler les optimisations faites sur le réseau de neurones implémenté. Les librairies ou environnements de développement comme STM32Cube ou MCUXpresso des constructeurs STMicroelectronics et NXP respectivement sont spécifiques à leurs produits et non adaptables à d'autres microcontrôleurs.

Dans l'objectif de développer le déploiement de CNN au sein de cibles matérielles, des méthodes d'intégration automatique ont été créées. La Section suivante présente ces méthodes.

2.5 Méthodes d'évaluation et d'intégration

Des méthodes d'intégration simple à mettre en place pour l'utilisateur sont proposées dans l'état de l'art. Des méthodes de compressions sont appliquées automatiquement et une implémentation au sein de la cible matérielle est automatiquement proposée.

Sankaran et al. [114] introduisent un module sous forme de boîte noire pour l'optimisation des modèles d'apprentissage profond, nommé Deeplite Neutrino. Ce module est à ajouter au processus de développement classique des CNN, basé sur l'entrainement puis l'implémentation du réseau, comme représenté sur la Figure 2.13.

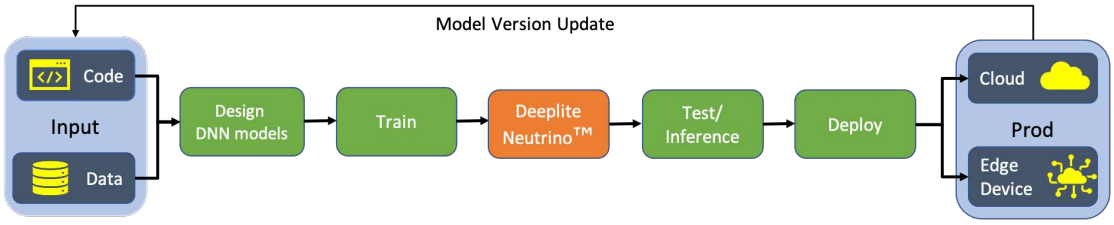


FIGURE 2.13 – Processus d'intégration du module Deeplite Neutrino proposé par Sankaran et al. [114].

Un utilisateur fourni des contraintes à respecter pour le modèle final, comme une

limite en précision de classification ou un espace mémoire maximal à respecter. Le module "Deeplite Neutrino" est intégrable dans les chaînes de développement d'un modèle d'apprentissage profond basé sur les outils Pytorch ou TensorFlow. L'utilisateur doit fournir au processus Deeplite Neutrino un CNN pré-entrainé, un jeu de données et des contraintes de réduction. Le CNN pré-entrainé doit faire partie des CNN acceptés par l'outil, entre une vingtaine de CNN comme les architectures ResNet, VGGNet ou MobileNet. Le conducteur a ensuite pour objectif de collecter les entrées et les contraintes fournies par l'utilisateur pour ensuite exécuter le processus en inférant le modèle. Puis les deux étapes d'optimisation et de compression du CNN ciblé sont effectuées. Le processus d'optimisation des CNN a montré des taux de compression variés sur plusieurs métriques : espace mémoire nécessaire au réseau, nombre de paramètres, nombre de MAC pour une inférence, etc. Ce processus proposé sous forme de boîte noire offre l'avantage d'être utilisable par de nombreuses personnes. Cependant le concept de boîte noire incombe de ne pas maîtriser la chaîne d'optimisation. De plus aucune exploration n'est effectuée sur la consommation en énergie du modèle optimisé. Pendant le processus d'optimisation, les optimisations sont appliquées à l'aveugle, il est donc nécessaire de tester plusieurs versions d'optimisation avant éventuellement de trouver une solution acceptable. Ce processus peut-être très long et ne fournit aucune certitude sur la faisabilité de l'implémentation dans le respect des contraintes de l'application.

Deutel et al. [115] proposent un processus pour compresser et intégrer un CNN dans des cibles basées sur un cœur de processeur ARM Cortex-M. Ils appliquent les techniques d'élagage et de quantification. Ce processus d'intégration est basé sur deux étapes principales. La première pendant l'étape d'entrainement du modèle est d'appliquer les techniques de compression. La seconde étape est l'implémentation du réseau de neurones, représenté par un graphe de haut niveau, en du code de bas niveau. Cette seconde étape fait appel à un générateur de code et une librairie externe. Un modèle de réseau de neurones est généré en format ONNX puis transcrit en langage C. Pour évaluer ce processus, il a été appliqué à 3 CNN : AlexNet et ResNet sur CIFAR10 et LeNet5 sur MNIST. L'implémentation de ces réseaux a été faite dans un Rasberry Pi Pico basé sur un cortex M0+ et un Arduino Nano basé sur un cortex M4. Des compressions de plus de 95% ont été faites sur les 3 CNN tout en gardant une précision de classification proche de celle du modèle de base.

Heim et al. [116] introduisent une chaîne d'outils pour l'intégration de réseaux de neurones optimisés au sein de matériel basé sur le cœur ARM Cortex-M. Leur processus est divisé en plusieurs étapes, représentées sur la Figure 2.14.

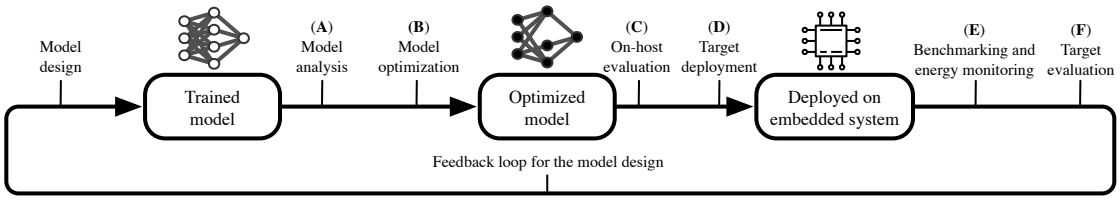


FIGURE 2.14 – Processus d'intégration de CNN au sein de cibles embarquées basées sur des cœurs de processeur Cortex-M, proposé par Heim et al. [116].

La première est une analyse du réseau de neurones à implémenter, basée sur son nombre de paramètres ou sa précision en classification. Puis une optimisation est appliquée, à savoir une quantification en format INT8. Une évaluation du modèle

quantifié est effectuée sur un système à faibles contraintes. Puis le modèle est déployé sur la cible matérielle. Des mesures sont ensuite effectuées sur la cible matérielle pour déterminer la latence du réseau de neurones ou encore la consommation en énergie de la plateforme exécutant le réseau de neurones. Finalement la précision du réseau implémenté est évaluée sur la plateforme cible. Ce processus utilise les librairies TensorFlow Lite pour la quantification et CMSIS-NN pour l'implémentation dans la cible matérielle, profitant des avantages de ces solutions, mais aussi de leurs inconvénients. Leur processus a été testé sur les réseaux LeNet5 et ResNet20 et implémenté dans les cibles STM NUCLEO L496ZG, STM DISCO F496NI, STM NUCLEO F767ZI. La latence de ces réseaux sur les différentes cibles matérielles a été réduite d'un facteur 15 pour LeNet5 et d'un facteur 30 pour ResNet. Les défauts de ce processus sont les mêmes que ceux pour deeplite à savoir un bouclage jusqu'à trouver la bonne solution sans garantie ou estimation de départ.

Les réseaux de neurones sont souvent évalués sur leur précision en classification. La prise en compte de la complexité algorithmique du réseau à implémenter ou de l'espace mémoire dont il a besoin fait rarement lieu à une représentation quantitative. Quelques métriques ou méthodes donnant des indications sur l'efficacité des réseaux de neurones à implémenter ont été proposées. Wong [117] propose une métrique appelée "NetScore" qui fournie une information quantitative de l'équilibre entre la précision, la complexité de calcul et la complexité du modèle du réseau de neurones évalué. Cette métrique donne un ordre d'idée des réseaux de neurones à cibler si l'on souhaite privilégier une efficacité de l'exécution du réseau sur une cible matérielle. La métrique NetScore est définie suivant l'équation 2.12, où $a(\mathcal{N})$ est la précision en classification du réseau, $p(\mathcal{N})$ est le nombre de paramètres du réseau de neurones et $m(\mathcal{N})$ est le nombre de MAC [1] nécessaire à une inférence du réseau de neurones. Les coefficients α, β, γ pondère l'influence de chaque métrique.

$$\Omega(\mathcal{N}) = 20log\left(\frac{a(\mathcal{N}^{\alpha})}{p(\mathcal{N}^{\beta}).m(\mathcal{N}^{\gamma})}\right) \tag{2.12}$$

Wong propose que le coefficient alpha soit mis à 2 pour mettre en avant l'importance de la précision sur les autres métriques mises en jeu dans le NetScore, et que les coefficients β et σ soient eux à 0,5. Un total de 60 CNN sur la base ImageNet ont été caractérisés avec cette métrique. Les réseaux conçus en vue d'une efficacité d'exécution sont ainsi classés premiers par le NetScore, par exemple SqueezeNet, MobileNetV2 ou MobileNetV1. VGG16, étant un réseau très profond et comportant énormément de paramètres redondants, est dernier de ce classement.

Pour prédire les performances d'un réseau de neurones une fois implémenté, Velasco-Montero et al. [118] ont proposé PreVIous. Il s'agit d'une méthodologie pour prédire l'inférence et la consommation en énergie du système embarquant un CNN pour une inférence. L'évaluation de ces métriques est faite par couche. Deux CNN sont utilisés dans l'objectif de faire cette évaluation. Ces CNN ont pour objectif unique de mesurer le coût d'inférence de certaines couches. Un CNN est utilisé pour évaluer les couches entièrement connectées et Softmax. Un second CNN est utilisé pour les autres types de couches comme les couches de convolution ou de moyennage. Pour évaluer le coût d'inférence d'un CNN, le coût unitaire de chaque couche qui compose le CNN à implémenter est additionné. Ce modèle de prédictions est simple

1. Multiply–ACcumulate (MAC) est une instruction qui réalise le produit de deux nombres est l'ajoute dans un accumulateur.

à calculer, mais difficile à mettre en œuvre. En effet l'évaluation par couche nécessite un temps de caractérisation élevé au regard du nombre de configurations possibles pour les couches d'un CNN.

Williams et al. [119] introduisent le modèle *Roofline*, offrant des directives en matière de performance. Les cibles matérielles sont des cibles multicœurs. Ils basent leur évaluation sur trois métriques. La première est l'intensité opérationnelle caractérisée par le nombre d'opérations par octets de DRAM exprimé en FLOPS/octets. La seconde est la performance en nombre flottant exprimée en Giga FLOPS et la troisième la performance mémoire. Ces trois métriques sont combinées dans un seul graphe pour mettre en avant la limite haute des performances de calcul en nombre flottant qu'une cible matérielle peut atteindre, ainsi que la limite haute des performances de calcul que la mémoire permet d'atteindre. Un exemple d'un graphique *Roofline* est représenté sur la Figure 2.15. Avec ce modèle, la performance de plusieurs noyaux de calcul est analysée. Les CNN et les MCU ne sont pas les cibles premières de ce modèle. Cependant, cette approche d'évaluation est remarquable en termes de modélisation des limites d'une cible matérielle exécutant un algorithme basé sur des matrices.

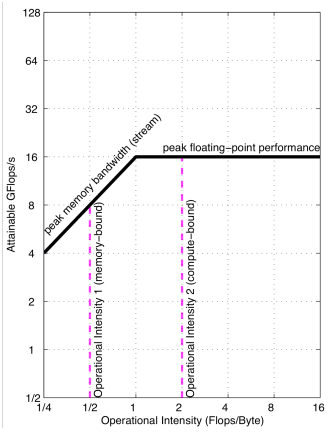


FIGURE 2.15 – Exemple d'un graphique *Roofline* de Williams et al. [119].

Les méthodes d'intégrations ou d'analyses présentées fournissent des indications à l'utilisateur pour implémenter un CNN dans une cible matérielle. La section suivante introduit des cibles matérielles spécifiquement conçues pour l'exécution des réseaux de neurones.

2.6 Unités matérielles dédiées et solutions industrielles

Cette section présente dans un premier temps des unités matérielles spécifiquement conçues pour optimiser l'exécution des CNN. Dans l'histoire de l'apprentissage profond, les unités matérielles ont toujours été développées de pair avec les avancées algorithmiques. Dans un second temps, cette section présente des solutions industrielles pour simplifier l'implémentation des CNN dans des cibles matérielles.

Unités matérielles dédiées :

Les calculs mis en jeu lors de l'exécution des CNN sont des calculs matriciels et/ou tensoriels. Les architectures numériques généralistes ne sont pas équipées d'instructions spécialisées pour l'exécution de ce type de calcul et ne sont pas optimales dans leur exécution. Une unité de calcul dédiée aux opérations des CNN est une amélioration matérielle possible d'un système. Ces unités ont toujours étaient conçues en parallèle des avancées algorithmiques des réseaux de neurones. Le Perceptron [19] de Rosenblatt est associé à l'unité matérielle Mark I [120]. Le CNN LeNet5 [22] de Yann Lecun a été déployé sur l'unité matérielle ANNA [121]. Concernant l'ère de l'apprentissage profond actuelle, autour des années 2010, les accélérateurs matériels modernes d'algorithmes d'apprentissage profond sont utilisés au sein de serveurs de calcul. L'unité de calcul de tenseurs - Tensor Processing Unit - TPU [122], est un ASIC (Application-Specific Integrated Circuit). Il s'agit d'un circuit électronique conçu sur mesure pour une tâche. Cette architecture a été déployée dans les serveurs de calculs de Google à partir de 2015. Google a également introduit l'accélérateur Pixel Visual Core, conçu pour le calcul d'image et de vision par ordinateur pour des systèmes mobiles comme des téléphones ou des tablettes. Microsoft a également sorti sa solution d'accélérateur matériel pour ses serveurs : Catapult [123]. Cet accélérateur est basé sur un FPGA (Field Programmable Gate Array). Ces deux accélérateurs ont été les premiers dans leur genre, et conçus pour des serveurs de calculs. L'accélérateur EyerissV2 [124] a apporté une solution d'accélération pour des applications mobiles en tenant compte de la diversité de réseaux de neurones à exécuter. Cependant ce type de solution matérielle reste difficile à embarquer dans un système contraint. Un accélérateur plus adapté aux contraintes des systèmes basés sur des microcontrôleurs est l'architecture Parallel Ultra Low Power - PULP [125] basée sur le cœur de processeur RISC-V. L'environnement autour de l'architecture PULP comprend des librairies spécialisées [126] et plusieurs architectures ASIC [127]. La société GreenWaves Technologies distribue certaines puces basées sur l'architecture PULP [128]. Une solution récente proposée par la société ARM en 2019 est une unité de calcul neuronal - Neural Processing Unit (NPU), nommée Ethos [129]. La famille Ethos se divise en deux séries, la série N et la série U. Les EthosU, introduits en 2020, sont conçus pour travailler de concert avec des microcontrôleurs de type Cortex-M. Pour l'exécution des algorithmes de types réseau de neurones, l'augmentation des performances annoncées par le constructeur par rapport à une exécution sur un Cortex-M est de 480.

L'introduction d'architectures dédiées aux calculs des réseaux de neurones a augmenté l'efficacité des systèmes les exécutant. Cependant, mis à part certaines architectures très récentes basées sur PULP, comme PULPino [127], ou la famille Ethnos de chez ARM, très peu sont adaptées aux systèmes basés sur des microcontrôleurs.

Solutions industrielles : La complexité de déploiement des réseaux de neurones est un frein pour un déploiement à grande échelle. Des solutions industrielles ont été déployées pour faciliter l'accès à un nombre plus important de concepteurs de systèmes embarqués.

Fondée en 2018, alwaysAI [130] propose une plateforme permettant de développer un modèle de réseau de neurones, l'entrainer et le déployer sur une plateforme. Les réseaux de neurones disponibles dans l'outil sont applicables sur des problématiques

de classification, de détection d'objet ou encore de segmentation. Une dizaine de plateformes matérielles sont utilisables. Sur le même modèle, Qeexo AutoML [131] facilite l'accès au déploiement de réseaux de neurones sur cibles embarquées. Basés sur le format ONNX du réseau, ils génèrent un code embarquable, ciblant 5 MCU basés sur des Cortex M4 et des Cortex M0+. Fondé en 2017, l'outil SensiML [132] propose de créer des modèles sur une cible matérielle, comme des MCU de chez Nordic Semi conductor, NXP ou encore STMicroelectronics. Fondée en 2019, la plateforme EdgeImpulse [133] simplifie le développement d'un modèle de son entrainement à son déploiement. Leur outil simplifie la collecte et mise en forme des données d'entrainement pour le réseau de neurones. L'exportation du réseau de neurones est ensuite possible à travers une librairie pour être exploitable sur des plateformes embarquées. Ils offrent également des solutions adaptées à une vingtaine de plateformes allant de l'Arduino Nano 33 BLE Sense au Raspberry Pi RP2040 en passant par un Nordic Semi nRF52840 DK.

Malgré ce déploiement facilité des réseaux de neurones dans des cibles matérielles contraintes, les solutions proposées par ces industrielles sont déployables sur quelques dizaines de cibles matérielles. De plus, l'analyse du réseau de neurones à embarquer ne tient pas compte des contraintes de consommation d'énergie. Dans de rares cas, l'information de consommation ou de latence peut-être fournie.

2.7 Synthèse et Problématique

L'état de l'art présenté ci-dessus expose les différentes solutions pour l'intégration des CNN au sein de cibles embarquées. Des solutions algorithmiques se concentrent sur la conception de CNN à faible coût d'inférence. D'autres approches algorithmiques ont pour objectif de réduire la taille ou la complexité des réseaux. Certains travaux proposent également des modèles ou des processus d'intégration, utilisant des librairies spécialisées et facilitant l'implémentation des CNN. Le matériel dédié à l'exécution des réseaux a pour objectif de rendre plus efficace l'exécution des réseaux de neurones sur la cible matérielle. La réduction algorithmique de la complexité du CNN semble une solution adéquate pour réduire le coût d'inférence du réseau. Cependant, peu de travaux ont mis en avant l'impact de ces réductions sur la cible matérielle exécutant le CNN. Le taux de compression du réseau obtenu suite à l'application des techniques de réduction est souvent mis en avant. Mais l'impact de cette compression sur la consommation énergétique, la latence ou encore l'espace mémoire du CNN est rarement étudié. Ce manque d'analyse rend le choix des optimisations à appliquer sur le CNN difficile à faire. La nature des applications embarquées, et donc leurs contraintes, est très variable. La diversité de cible matérielle et de modèle de CNN rend la politique de réduction et d'optimisation difficile à déterminer. Toutes les solutions de l'état de l'art tiennent compte de la précision du CNN. En effet, il s'agit du critère principal d'intégration d'un réseau de neurones. De plus avec la méthode de *Transfert Learning*, l'adaptation en précision d'un CNN est simple à réaliser. L'étude des solutions proposées de l'état de l'art a mis en avant plusieurs critères discriminants concernant les avantages et inconvénients de ces solutions. La précision du CNN pouvant être respectée simplement, quelle que soit la politique d'intégration choisie, n'est pas un critère discriminant. Ces critères sont associés aux solutions de l'état de l'art dans le Tableau 2.4. Les mots clefs définis dans ce tableau font référence aux critères suivants :

- Implémentation : nécessite une implémentation du CNN au sein du matériel cible pour évaluer l'impact d'une inférence. Le besoin d'implémenter un CNN est un effort d'ingénierie non négligeable et aucune garantie n'est fournie en amont sur la faisabilité de l'implémentation. Le critère peut prendre la valeur "Nécessaire" ou "Non nécessaire".
- Complexité : complexité de mise en œuvre de la solution. Si la complexité est trop élevée et que la solution est utilisable que par certains utilisateurs, cela peut-être un frein à un déploiement de masse. Le critère peut prendre la valeur "Forte" ou "Faible".
- Déploiement : effort nécessaire pour déployer une solution. Même si une solution d'implémentation de CNN est simple à utiliser, cette solution peut être chrono phage ou nécessité beaucoup de ressources. Le critère peut prendre la valeur "Importants" ou "Faibles".
- Latence, Énergie et Mémoire : la solution d'implémentation tient compte des contraintes non fonctionnelles du CNN à savoir sa latence, sa consommation d'énergie et son espace mémoire nécessaire sur la cible matérielle. Le critère peut prendre la valeur "Oui", "Non", "Possible".

TABLE 2.4 – Comparaison des solutions proposées par l'état de l'art. La couleur verte signifie que c'est un avantage, la couleur rouge signifie que c'est un inconvénient, la couleur orange signifie que le critère peut-être respecté sous certaines conditions.

Solution	**Implémentation**	**Complexité**	**Efforts de Déploiement**	**Pour un CNN cible, caractérise la :**		
				Latence	**Énergie**	**Mémoire**
Matériel dédié	Nécessaire	Forte	Importants	Non	Non	Non
Env. de dév. et librairies	Nécessaire	Faible	Faibles	Possible	Non	Possible
Création de CNN	Nécessaire	Forte	Importants	Non	Non	Oui
NAS	Non nécessaire	Forte	Importants	Oui	Non	Oui
Techniques de réduction	Nécessaire	Forte	Importants	Non	Non	Possible
Processus d'intégration	Nécessaire	Faible	Faibles	Non	Non	Non
Solutions Industrielles	Nécessaire	Faible	Faibles	Possible	Non	Possible

La majorité des solutions proposées nécessitent l'implémentation du CNN dans le matériel pour déterminer si les contraintes d'une application embarquée seront respectées. Cela engendre un coût en temps de développement et en ressources important, pour au final aucune garantie sur l'implémentation finale. Des solutions comme l'implémentation de matériel dédié à l'exécution du CNN, ou la création de CNN manuellement et à travers NAS sont des procédés complexes et le temps de déploiement de cette solution est très long. De plus, peu d'utilisateurs peuvent mettre en place de telles solutions. Les processus d'intégration ou les solutions industrielles proposées sont conçus pour être simples à utiliser par l'utilisateur. Cependant, outre l'implémentation nécessaire du CNN, peu de détails sont donnés sur les optimisations appliquées et la consommation énergétique est un critère qui n'est pas pris en compte. Avec le *Transfert Learning*, la précision du réseau peut être adaptée à la problématique cible. En combinant cela aux techniques de réduction, l'utilisateur pourrait adapter l'impact du CNN à implémenter en fonction des contraintes de l'application cible. Il existe de nombreuses techniques de réduction et donc de nombreuses configurations de réduction possibles. La latence et l'espace mémoire nécessaire à une inférence sont parfois impactés par certaines réductions du réseau. Cependant, peu d'implémentations ont été faites dans du matériel embarqué pour vérifier les gains réels annoncés. De plus, la consommation énergétique est rarement étudiée. Le manque de visibilité sur le respect des contraintes de l'application cible durant le processus d'intégration peut amener à de nombreuses tentatives d'implémentation infructueuses. Les démarches proposées dans l'état de l'art avancent à l'aveugle. Elles peuvent être chronophages et mobiliser d'importantes ressources humaines et matérielles de par la variété de politiques d'implémentations à tester. De plus, avant une quelconque tentative d'implémentation, aucune garantie sur la possibilité d'implémenter le CNN ciblé dans le système embarqué n'est faite. Pour vérifier si les contraintes de l'application cible sont respectées, il est nécessaire d'appliquer la politique d'implémentation choisie, incombant parfois des réductions algorithmiques, puis d'essayer d'implémenter le CNN résultant. La possibilité de valider les contraintes de l'application cible est uniquement possible après l'implémentation du

CNN au sein du matériel cible. La démarche des méthodes de l'état de l'art est résumée et schématisée sur la Figure 2.16.

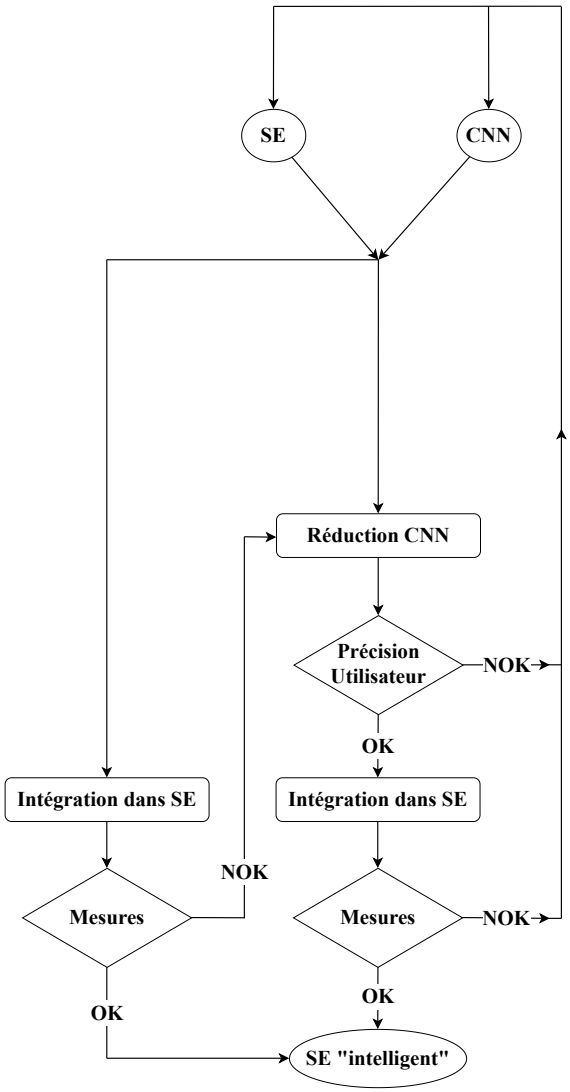


FIGURE 2.16 – Principe des processus d'intégration de l'état de l'art. "SE" signifie système embarqué, "CNN" signifie Convolutionnal Neural Network. La démarche est soit d'essayer d'implémenter le CNN sans réduction, puis de mesurer si les contraintes de l'application sont respectées. Soit d'appliquer des politiques de réductions jusqu'à respecter les contraintes de l'application.

De cette synthèse de l'état de l'art, la problématique initiale peut-être affinée comme suit :

Comment explorer rapidement l'espace des solutions pour embarquer un CNN dans un MCU en considérant les différentes techniques de réduction ?

Le chapitre suivant a pour objectif de décrire la méthode d'intégration répondant à la problématique proposée dans cette thèse.

Chapitre 3

Méthodologie ZIP-CNN

Sommaire

3.1 Introduction

L'étude menée au cours des chapitres précédents a mis en avant la problématique d'implémentation de CNN au sein de MCU. Le manque de visibilité sur le respect des contraintes de l'application cible durant le processus d'intégration peut amener à de nombreuses tentatives d'implémentation infructueuses. La possibilité de valider les contraintes de l'application cible est uniquement possible après l'implémentation du CNN au sein d'une cible matériel.

Pour pallier ces désavantages, une méthode d'exploration de l'espace des solutions pour intégrer des CNN au sein de cibles MCU est proposée. Dans un premier temps, la démarche globale de la méthodologie est présentée. Dans un deuxième temps, un modèle caractérisant les spécificités du matériel cible ainsi que la variété des CNN est détaillé. Dans un troisième temps, l'influence des réductions appliquées sur le CNN est étudiée, en caractérisant le coût d'inférence du CNN réduit sur le matériel cible. L'un des objectifs principaux de la méthodologie est de guider l'utilisateur en estimant les performances du MCU cible dans lequel le CNN est implémenté.

3.2 La méthodologie ZIP-CNN

Dans cette section, la méthodologie ZIP-CNN, dont le principe est représenté sur la Figure 3.1, est expliquée. Dans un premier temps, une étape d'estimation va quantitativement estimer la latence, la consommation en énergie et l'espace mémoire nécessaire à une inférence du CNN sur le système embarqué cible. L'estimation quantitative informe l'utilisateur du respect, ou non, des contraintes de l'application cible. L'utilisateur peut ensuite choisir d'implémenter le CNN.

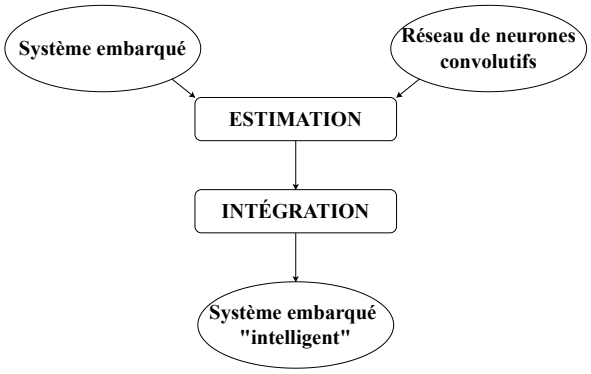


FIGURE 3.1 – Principe général de la méthodologie ZIP-CNN proposée.

Le principe présenté sur la Figure 3.1 est détaillé dans la suite du chapitre.

3.2.1 Étape de l'estimation

Deux étapes sont ajoutées pour obtenir la méthodologie ZIP-CNN complète. La première étape est une estimation. Les hypothèses de départ sont d'avoir un Système Embarqué (SE) et un Réseau de Neurones Convolutifs (CNN) dont la précision respecte les contraintes de l'application cible. Cette précision peut-être obtenue à travers un entrainement classique du CNN, ou bien à travers l'utilisation du *Transfert Learning*.

Les démarches d'intégration utilisées dans l'état de l'art ne fournissent aucune garantie sur la possibilité d'implémentation du CNN. Aucune indication n'est donnée sur l'éventuel impact de la réduction appliquée sur les contraintes de latence, de consommation énergétique et d'espace mémoire. Pour pallier ces défauts, la première étape consiste en un bloc d'estimation nommé "Estimation EST", *E - Énergie, S - Surface, T - Temps*, ainsi qu'une vérification nommée "Contraintes". Cette étape est détaillée dans la suite du chapitre. L'ajout de cette première étape d'estimation est schématisé en rouge sur la Figure 3.2.

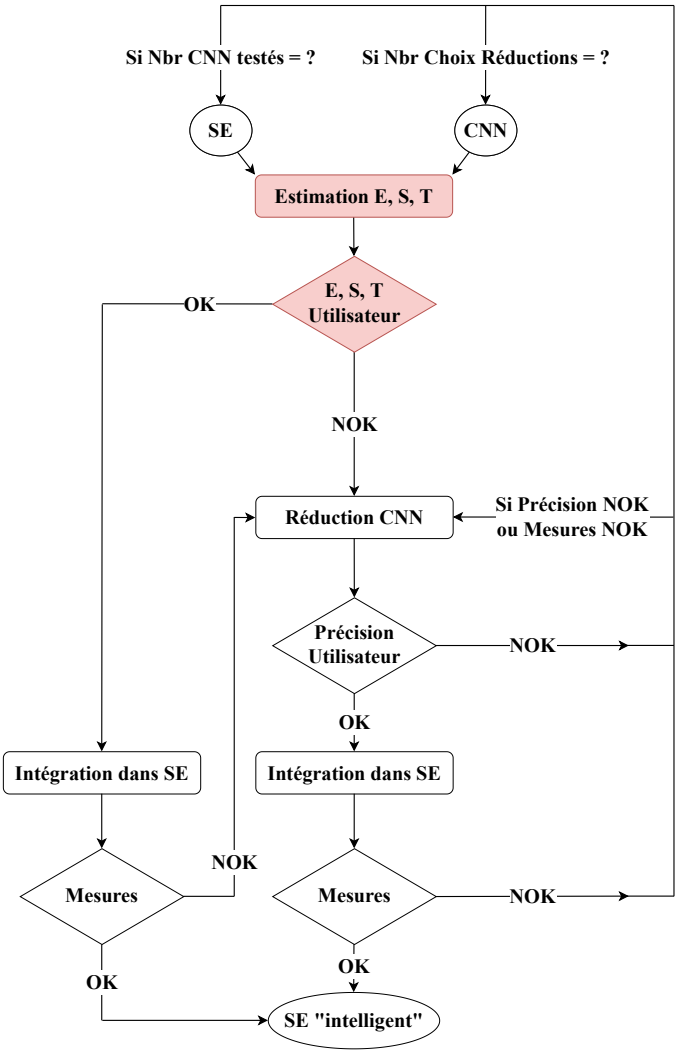


FIGURE 3.2 – Une étape "Estimation EST" est ajoutée au processus de l'état-de-l'art. Elle permet d'estimer le coût de l'inférence du CNN sur le système embarqué. Elle ne nécessite aucune intégration du CNN. Une vérification des contraintes et ensuite effectuée pour orienter l'utilisateur vers une étape d'intégration immédiate ou une étape de réduction.

L'étape d'estimation EST tient compte de la configuration matérielle cible et du modèle de CNN à implémenter. Elle offre l'avantage de ne pas nécessiter l'implémentation du CNN pour estimer son coût d'inférence. Le modèle d'estimation utilisé est introduit dans la suite du chapitre. Cette première estimation a plusieurs objectifs :

1. Déterminer si le CNN peut être embarqué sans réduction.
2. Estimer quantitativement la latence, la consommation énergétique et l'espace

mémoire nécessaire à une inférence du CNN dans le matériel cible, pour vérifier quelle(s) contrainte(s) non fonctionnelle(s) ne sont pas satisfaite(s).

3. Quantifier l'effort de réduction à appliquer au CNN.
4. S'adapter simplement aux nombreuses topologies de CNN.

En estimant quantitativement le coût d'inférence du CNN, il est possible de déterminer rapidement sans mise en œuvre matérielle si le CNN peut être embarqué ou s'il est nécessaire de réaliser une réduction algorithmique. Suite à cette estimation, deux cas sont envisageables. Dans le premier cas, les contraintes de l'application peuvent être respectées avec un réseau de neurones non réduit. Ainsi une étape d'intégration peut commencer, avant une étape de mesures pour vérifier en pratique le respect des contraintes de l'application. Dans un second cas, les contraintes de l'application ne sont pas respectées. L'étape du choix des techniques de réduction à appliquer est à effectuer. Cette étape est définie dans la suite du chapitre. Une fois que la politique de réduction du CNN est choisie, elle peut-être appliquée pour réduire le réseau. Un ré entrainement du CNN est parfois nécessaire en fonction de la technique de réduction appliquée, par exemple pour toutes les techniques d'élagage de l'état-de-l'art et parfois pour certaines techniques de quantification. Ce ré entrainement est effectué par l'utilisateur en fonction du besoin de précision pour l'application cible. Une étape de validation de la précision du CNN réduit est ensuite effectuée, puis si la réduction est validée, le réseau est intégré. Finalement une étape de mesures vient valider les contraintes non fonctionnelles de l'application. Si l'étape de validation de la précision du CNN réduit, ou les mesures des contraintes sur le système embarqué ne sont pas validées, plusieurs choix sont possibles. Retravailler sur la phase de réduction du réseau, pour identifier une autre politique de réduction respectant les contraintes de l'application, si cette politique existe. Changer le CNN à implémenter pour choisir une topologie de réseau dont l'inférence est moins coûteuse, tout en respectant la contrainte de précision de l'application. Finalement, si aucune configuration d'intégration n'est trouvée, il pourra être nécessaire de changer de matériel cible pour réitérer le processus d'estimation.

3.2.2 Modèle d'estimation

Cette partie détaille le modèle mis en jeu pour l'étape "Estimation EST". Le modèle d'estimation est basé sur une décomposition des CNN en plusieurs primitives. Une primitive de calcul d'un CNN est une fonction de base mise en jeu lors de l'inférence d'un CNN, par exemple une convolution ou une activation. La granularité de cette décomposition en primitives est au niveau des motifs de calcul. Un motif de calcul est une primitive composant les CNN. En connaissant le coût d'inférence unitaire d'une primitive sur un MCU, ainsi que le nombre de primitives au sein du CNN à implémenter, l'impact de l'inférence du CNN est estimé. Dans la suite de cette partie, la granularité de décomposition des CNN est d'abord détaillée. Puis, la caractérisation des primitives est explicitée, ainsi que le processus d'estimation complet.

La décomposition en motifs de calcul

Le modèle d'estimation est basé sur une décomposition des CNN. Un aspect central dans cette démarche est la granularité appliquée à la décomposition. La granularité proposée est au niveau des motifs de calcul - *computation patterns*. Le niveau d'abstraction de cette granularité de décomposition des CNN est représenté sur la Figure 3.3 par rapport à d'autres niveaux d'abstraction. La granularité en motifs de calcul se situe entre la granularité par couches et la granularité par opérations ou par *proxys* (MAC ou FLOP). Un *proxy* est une donnée de substitution pour représenter une variable parfois non observable ou difficilement mesurable.

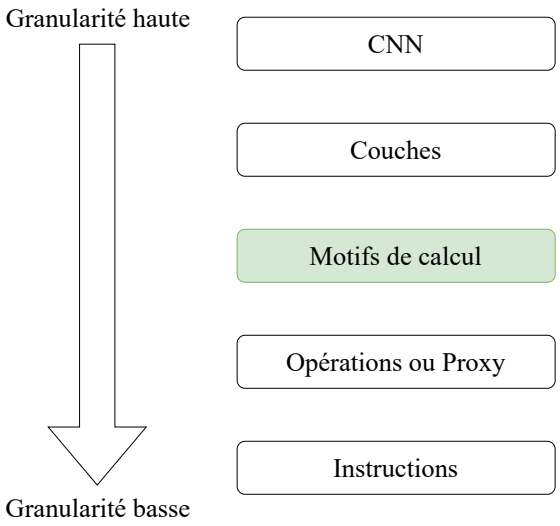
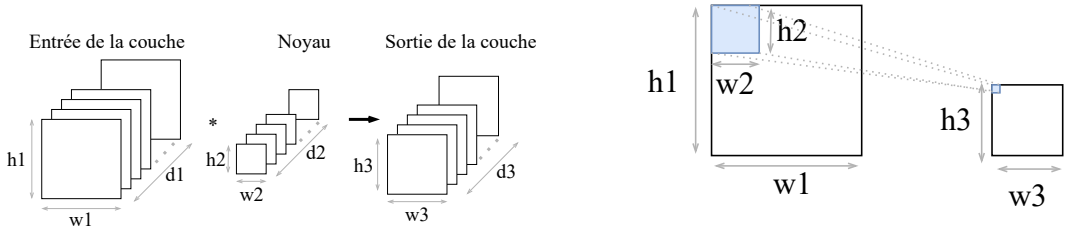


FIGURE 3.3 – Niveau d'abstraction des décompositions possibles des CNN. La granularité de décomposition appliquée dans ces travaux est au niveau des motifs de calcul.

Une granularité de décomposition trop basse, comme celle des opérations ou des instructions, ne permet pas de capturer des comportements essentiels d'une cible matérielle, comme les accès mémoire ou les transferts de données. Pourtant, ces accès mémoires et transfert de données sont très importants lors de l'inférence d'un CNN pour accéder aux poids entrainés du réseau permettant de faire une prédiction. De plus, Yang et al. [134] ont démontré que les *proxys* comme les MAC ou les FLOP ne reflètent pas toujours correctement certaines métriques comme la latence.

Une granularité de décomposition trop haute rend difficile et coûteuse la prise en compte des nombreuses topologies de CNN. L'avantage de la granularité en motif de calcul proposée dans ce modèle d'estimation par rapport à une granularité par couche est illustré sur la Figure 3.4.

L'estimation avec une granularité par couche nécessite une caractérisation dépendante des paramètres d'entrées de la couche et des noyaux appliqués. La hauteur, la largeur et la profondeur des couches peuvent être sensiblement différentes selon le CNN. Cela implique un effort de caractérisation important pour représenter la variété des configurations de couches possibles dans les modèles de CNN. En comparaison, la granularité du motif de calcul proposée implique un effort de caractérisation bien moins important. Uniquement la hauteur (h2) et la largeur (w2) d'un noyau sont nécessaires. Si deux tailles de noyau différentes $k = h2 \times w2$ sont appli-



(a) Granularité par couche.

(b) Granularité par motifs de calcul proposée dans ce modèle d'estimation.

FIGURE 3.4 – Caractérisation basée sur une granularité par couche *vs.* une granularité par motifs de calcul. (a) Décomposition des CNN à une granularité par couche comme proposée par Velasco-Montero et al. [118]. La granularité par couche est une fonction de la hauteur, de la largeur et de la profondeur de l'entrée de la couche et des noyaux (h1, w1, d1, h2, d2, w2). (b) Décomposition des CNN à une granularité par motifs de calcul proposée dans ce modèle d'estimation. La granularité par motif de calcul est seulement fonction de la hauteur et de la largeur d'un noyau (h2, w2).

quées dans un CNN, par exemple $k = 5 \times 5$ et $k = 3 \times 3$, une seule caractérisation pour chaque noyau est nécessaire pour estimer le coût de l'ensemble du réseau.

Pour résumer, une granularité trop basse ne tient pas compte des accès mémoires et des transferts de données. Les proxys ne sont parfois pas adaptés pour refléter des métriques comme la latence de l'inférence d'un CNN. Cependant, une granularité basse offre une certaine souplesse pour caractériser les différentes topologies. Une granularité trop haute tiendra compte des accès mémoires ou transfert de données, mais l'adaptation aux topologies de CNN et de MCU est complexe et coûteuse. La granularité en motif de calcul présente un juste équilibre entre la granularité par couche et la granularité par opérations ou par proxy. Elle permet une caractérisation plus simple des éléments d'un CNN pour effectuer une estimation du coût d'inférence total du CNN. Cette caractérisation tient compte des accès mémoire et transfert de données pour accéder aux poids entrainés du réseau. L'adaptation aux différentes topologies de CNN et de MCU est simplifiée de par la nature des primitives à caractériser. Ces primitives sont présentées en suivant.

Un CNN est composé de deux parties : un extracteur de caractéristiques et un classifieur, illustrés sur la Figure 3.5.

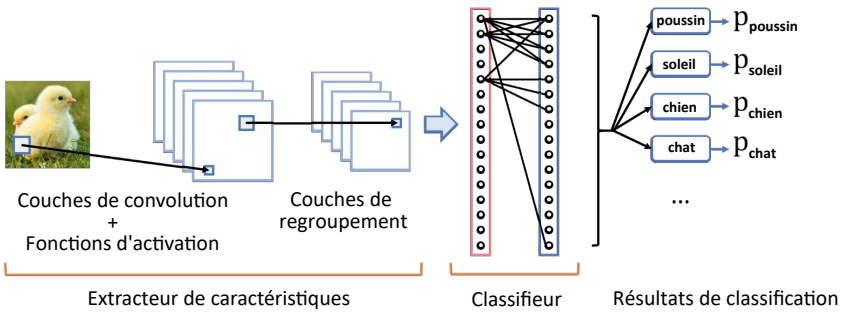


FIGURE 3.5 – Composition générale des CNN. La partie gauche représente l'extracteur de caractéristiques composé de couches de convolution, de fonctions d'activation et de couches de regroupement. La partie droite représente le classifieur composé de couches entièrement connectées et de fonctions d'activation.

Sur la partie gauche de la Figure 3.5, l'extracteur de caractéristiques est composé de :

- primitives de convolution : pour extraire les caractéristiques des données.
- primitives de regroupement : pour sous-échantillonner les caractéristiques.
- primitives de régularisation : fonction d'activation pour introduire des non-linéarités dans les traitements des CNN.

Sur la partie droite de la Figure 3.5, le classificateur est composé de :

- primitives entièrement connectées : pour réaliser les combinaisons linéaires.
- primitives de régularisation.

Tous les CNN sont caractérisées par le type, les paramètres, l'ordre et le nombre d'applications des primitives. Cependant, tous les CNN sont basés sur des primitives similaires. La caractérisation unitaire de ces primitives permet d'estimer le coût d'inférence d'une majorité de CNN sur le matériel ciblé, quelle que soit la topologie du réseau de neurones. Ces primitives sont ainsi regroupées en quatre familles, à savoir :

1. les primitives d'extraction de caractéristiques : convolution, convolution en profondeur séparable, etc.
2. les primitives de sous-échantillonnage : regroupement par valeur moyenne, regroupement par valeur maximale, etc.
3. les primitives de normalisation : fonction d'activation, *Batch Normalization*, etc.
4. les primitives de combinaison linéaire : couches entièrement connectées, etc.

Processus d'estimation

Basé sur la décomposition en primitives présentées précédemment, le processus d'estimation est basé sur l'Équation 3.1 où PUC est le coût unitaire d'une primitive - *Primitive Unit Cost* et NA est le Nombre d'Applications de la primitive au sein du CNN.

$$CNN_{estimation} = \sum PUC \times NA \tag{3.1}$$

La détermination du PUC est basée sur des mesures en latence, en consommation énergétique et en espace mémoire de la primitive. Le PUC permet ainsi de caractériser la cible matérielle. Le NA permet de prendre en compte la variété de modèles de CNN. Par exemple, dans le cas du réseau ResNet, différents modèles de ce réseau existent (*c.f. Sections 2.2.1* et *1.4.5*). Pour tenir compte de cette variété de configurations , les CNN étant basés sur les mêmes primitives, la variable NA caractérisera le nombre de primitives au sein du CNN à implémenter.

3.2.3 Caractérisation des primitives

La caractérisation du coût de l'inférence des primitives a pour objectif de construire une table de correspondance - *Look Up Table* (LUT) des coûts associés. La caractérisation des primitives est une étape unique pour un MCU donné et les mêmes LUT sont réutilisables pour tous les CNN. Celle-ci est basée sur des mesures de

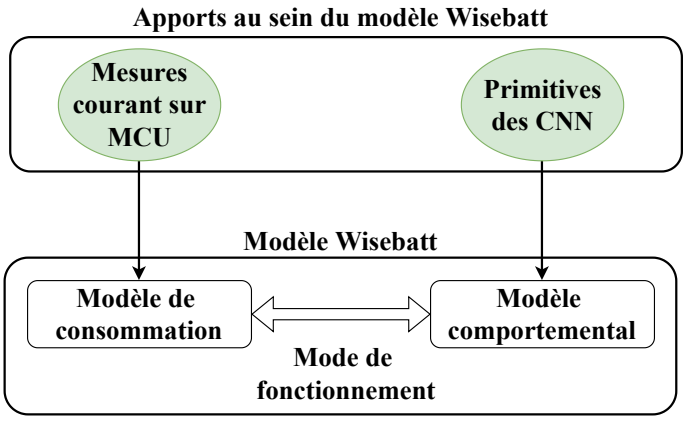


FIGURE 3.6 – Interaction entre le modèle de l'outil de Wisebatt et le modèle des primitives de CNN. Les mesures de courant sur le MCU au sein du modèle de consommation permettent de reproduire la consommation induite par le comportement des CNN. L'ajout des primitives au sein du modèle comportemental permet de reproduire le comportent de la cible matérielle, lorsqu'elle traite des données liées à l'inférence d'un CNN.

métriques perceptibles, ainsi des connaissances poussées sur la cible matérielle ne sont pas nécessaires. Ces métriques perceptibles sont à opposer aux proxys couramment utilisés dans l'état de l'art [54], [135] pour l'estimation de la latence d'une inférence d'un CNN. Heim et al. [116] ont introduit cette notion de métriques perceptibles pour caractériser le système tel qu'il est vécu par l'utilisateur, par exemple la consommation d'énergie ou la latence pour une inférence. Dans la méthodologie ZIP-CNN, la latence, la consommation d'énergie et l'espace mémoire nécessaire à chaque primitive sont déterminés de la façon suivante :

- La consommation d'énergie : un profil de consommation au sein du MCU est décrit pour chaque primitive. Ce profil est défini comme un macro-état. Ce macro-état est caractérisé par (1) les mesures de courant du MCU, (2) la fréquence d'horloge du cœur de processeur du MCU et (3) l'algorithme de la primitive évaluée. Pour mesurer uniquement le courant du MCU, la sonde UlinkPlus de ARM-Keil [136] est utilisée. Une fois caractérisé, le macro-état de la primitive est implémenté dans l'outil de la société Wisebatt [137]. Cet outil est basé sur les travaux de recherche de Dron et al. [138], [139] et a donné lieu à un brevet [140]. Wisebatt prend en compte trois aspects clés : (1) la plateforme matérielle cible (2) l'algorithme exécuté sur la plateforme matérielle cible et (3) la batterie qui l'alimente. Il introduit une approche de modélisation décrivant le transfert d'énergie entre la batterie et les composants matériels. Il propose un modèle réaliste reproduisant le comportement non linéaire d'une batterie réelle. Le modèle basé sur les primitives développé durant ces travaux est utilisé au sein du modèle de l'outil de Wisebatt. L'interaction entre les deux modèles est représentée sur la Figure 3.6. L'objectif est de pouvoir décrire le comportement des CNN durant une simulation de consommation énergétique de la plateforme matérielle. Le modèle de consommation caractérise la consommation de la cible matérielle. Le modèle comportemental restitue le comportement fonctionnel du composant. Le mode de fonctionnement assure la synchronisation entre les deux entités. L'ajout des mesures de courant du MCU cible au sein du modèle de consommation per-

met de reproduire la consommation induite par le comportement des CNN. L'ajout des primitives au sein du modèle comportemental permet de reproduire le comportent de la cible matérielle lorsqu'elle traite des données liées à l'inférence d'un CNN.

- La latence : un registre incrémental - *timer* matériel fonctionnant à la même fréquence d'horloge que l'horloge du cœur du processeur est utilisé. Le registre incrémental mesure le nombre de cycles nécessaires au calcul de l'algorithme d'une primitive. Ensuite, ce nombre de cycles nécessaire à l'exécution d'une primitive est multiplié par la période de l'horloge du *timer* pour obtenir une latence.
- L'espace mémoire : un MCU est équipé de au minimum deux types d'espace mémoire, un espace non volatile et un espace volatile. Les MCU utilisés durant ces travaux sont équipés de mémoire non volatile de type Flash et de mémoire volatile de type SRAM ou RAM. La mémoire Flash est utilisée pour stocker le code et les poids du CNN. Il est donc nécessaire d'estimer cet espace mémoire pour être sur que le CNN dans son entièreté puisse être stocké. Pour estimer l'espace mémoire Flash nécessaire au stockage du CNN, la taille du code et des données sont mesurées pour chaque primitive. Cette information est générée par le processus de compilation dans un fichier d'organisation mémoire.

  De plus, durant l'inférence du CNN, des calculs intermédiaires sont nécessaires. Ces calculs sont majoritairement liés aux cartes de caractéristiques entre les couches du réseau et sont stockés dans la RAM durant l'inférence. Le principe est d'allouer de l'espace mémoire correspondant au besoin temporaire des cartes de caractéristiques de la couche à calculer. Puis, il est nécessaire de libérer cet espace mémoire une fois les calculs effectués. Il est donc possible que l'espace mémoire Flash soit suffisant pour stocker l'ensemble du CNN, mais que l'espace mémoire RAM ne soit pas suffisant pour permettre une inférence du CNN, les calculs intermédiaires à stocker temporairement étant trop nombreux. Ainsi, l'estimation de l'espace mémoire RAM nécessaire à l'inférence du CNN est un paramètre essentiel. Pour l'estimation de ce paramètre avec les primitives, une analyse par couche du CNN est nécessaire. Il est possible d'identifier les calculs intermédiaires nécessitant le plus d'allocations d'espace mémoire RAM, en identifiant la couche du CNN, notée $\mathcal{L}$, où le nombre de primitives utilisées est le plus important. Pour cela il faut suivre les deux règles suivantes :
  - Tenir compte du nombre de primitives en entrées et en sortie de chaque couche.
  - Si une couche est suivie par une normalisation (ReLU, Sigmoïde, etc.) il faut prendre en compte le nombre de primitives de normalisation.

  Les calculs de la couche $\mathcal{L}$ ne peuvent se faire sans les résultats de la couche $\mathcal{L}-1$. Ce processus d'estimation de l'espace RAM grâce aux primitives est représenté sur la Figure 3.7. Plus le nombre de primitives utilisées est important, plus les données générées en entrée et en sortie de $\mathcal{L}$ sont importantes, plus l'espace mémoire RAM nécessaire au stockage de ces données sera important. Ensuite, il suffit de comparer cette estimation à l'espace mémoire RAM disponible sur le MCU pour assurer qu'une inférence s'exécutera correctement.

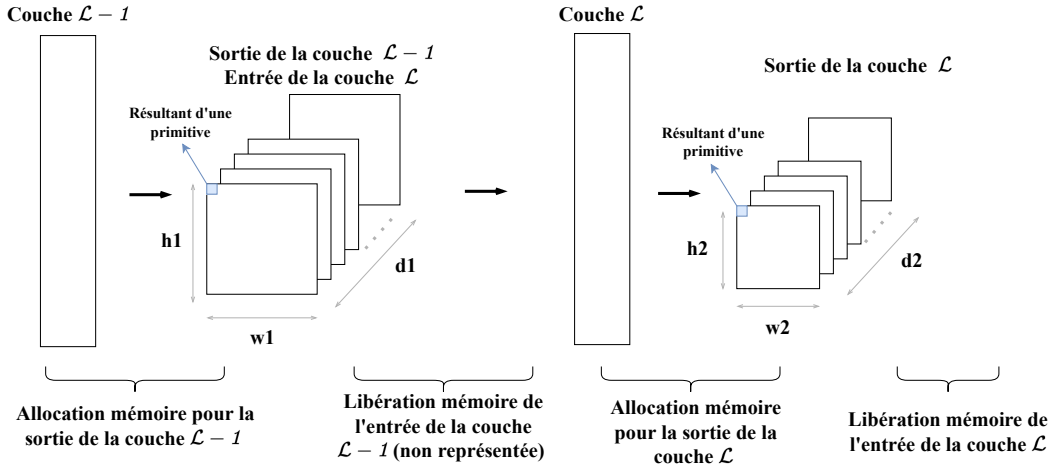


FIGURE 3.7 – L'espace mémoire RAM nécessaire à une inférence d'un CNN est estimé en identifiant les couches du CNN nécessitant le plus l'utilisation des primitives. Ici, le calcul de la sortie de la couche $\mathcal{L}$ nécessite $h_1 \times w_1 \times d_1 + h_2 \times w_2 \times d_2$ primitives. Chaque primitive résultante en une donnée codée en un format binaire, il est ensuite nécessaire de multiplier $h_1 \times w_1 \times d_1 + h_2 \times w_2 \times d_2$ par le nombre d'octets du format binaire, par exemple 4 dans le cas d'un format binaire en virgule flottante sur 32 bits ou 1 dans le cas d'un format binaire en entier sur 8 bits.

Le processus complet de caractérisation des primitives et d'estimation du coût d'inférence total d'un CNN est représenté sur la Figure 3.8. Ce modèle d'estimation est validé dans le chapitre suivant.

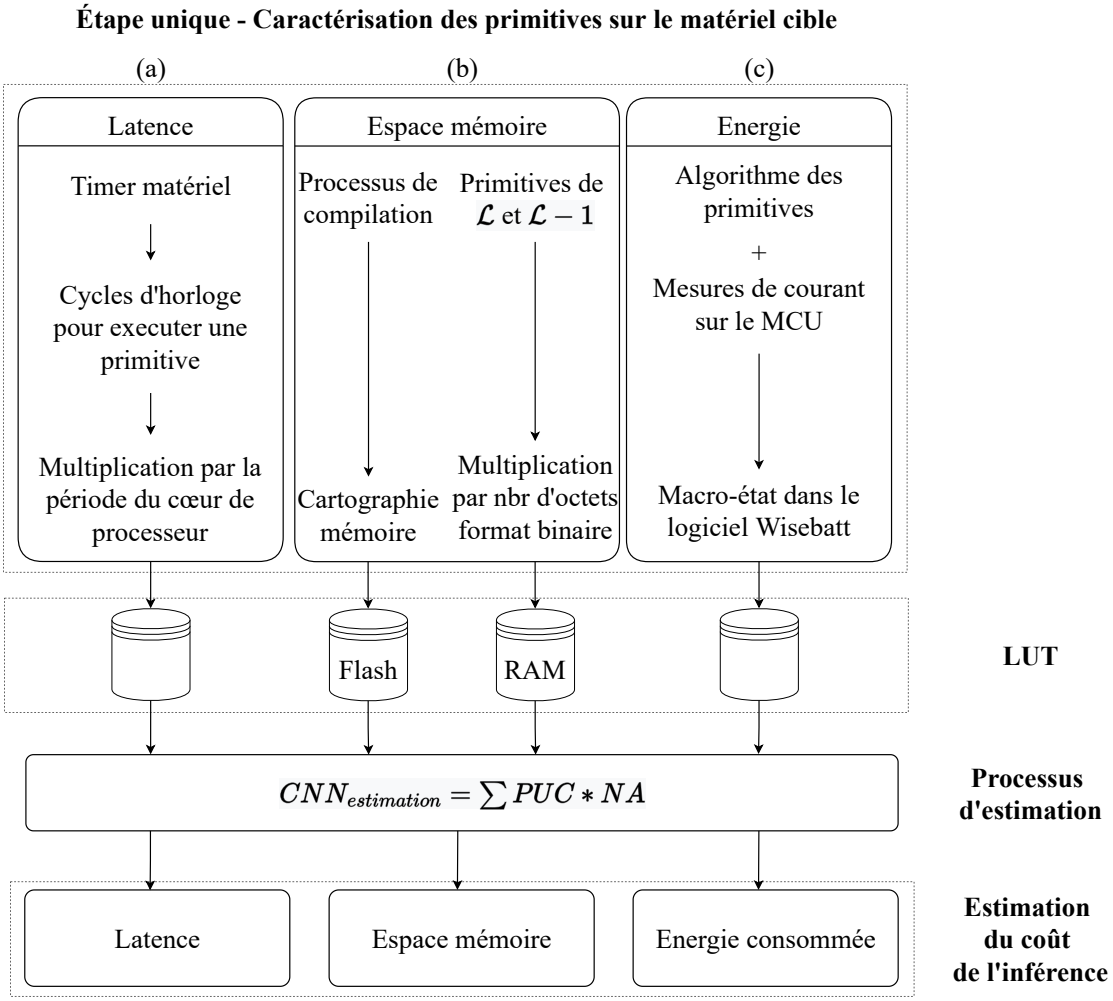


FIGURE 3.8 – Méthode d'estimation et de caractérisation des primitives proposées. (a) La latence est mesurée avec un registre incrémental matériel. Les cycles d'horloge nécessaires au calcul d'une primitive sont multipliés par la période d'horloge du cœur de processeur. (b) L'espace mémoire Flash est estimé avec la taille du code et des données requise par une primitive, générée lors du processus de compilation dans un fichier de cartographie mémoire. L'espace mémoire RAM est estimé en fonction du format binaire utilisé et avec le nombre de primitives nécessaires pour générer la sortie de la couche $\mathcal{L}$ et la sortie de la couche précédente $\mathcal{L}-1$. (c) La consommation énergétique des primitives est dérivée des mesures de courant, et de l'algorithme des primitives combiné au logiciel Wisebatt. Ensuite, le coût unitaire d'une primitive est enregistré dans une LUT. Enfin, il est multiplié par le nombre d'applications de chaque primitive appliquée dans le CNN à embarquer.

3.2.4 Jeu de primitives

Cette partie se concentre sur l'introduction des primitives mises en jeu pour estimer le coût d'inférence d'un CNN. Les primitives présentées ont toutes été caractérisées et utilisées durant ces travaux de thèse. L'étape de caractérisation, présentée sur la Figure 3.8, ne dépend pas des primitives à caractériser, ainsi d'autres primitives peuvent être définies en fonction du besoin. Certains paramètres des primitives peuvent varier. Ainsi différentes tailles de noyau de convolution et de regroupement ont été caractérisées. Finalement, un paramètre influençant les performances d'exécution de la cible matérielle est la fréquence de fonctionnement du cœur de pro-

cesseur. Les étapes de caractérisation des primitives ont été effectuées sur la plage de fréquence $[f_{min}; f_{max}]$ des MCU ciblés. Il est ainsi possible d'estimer la fréquence du MCU la plus appropriée pour une inférence du CNN, dans l'objectif de respecter les contraintes de l'application cible. L'étape de caractérisation des primitives, en faisant varier la taille de noyau des primitives ainsi que la fréquence de fonctionnement du MCU, est un processus qui a été automatisé. Ce processus est représenté sur la Figure 3.9. Un code en langage Python est utilisé pour écrire le code à mesurer dans le *main.c* du MCU. Des commandes de Keil μVision, appelées dans le code Python, permettent ensuite de lancer la compilation du code, de le stocker en mémoire Flash dans le MCU et d'ouvrir le logiciel. Le mode *Debug* est ensuite lancé. Un fichier d'initialisation exécute des commandes dès l'ouverture du mode *Debug*. Ces commandes permettent de placer un point d'arrêt dans le code, exécuter le code jusqu'au point d'arrêt et récupérer les données mesurées dans un fichier .log. Ce fichier est ensuite traité par le code en Python. Finalement, un nouveau code peut être écrit dans le *main.c* du MCU et mesuré, en remplaçant l'ancien. Il est donc possible de mesurer différentes primitives ou CNN en exécutant une seule fois le code d'automatisation des mesures. Le changement de fréquence du MCU est également possible en remplaçant certains paramètres dans la fonction configurant l'horloge du MCU. Cette automatisation a permis d'effectuer suffisamment de mesures pour quantifier la précision du modèle d'estimation. Elle permet également une reproductibilité des résultats. L'automatisation mise en place est applicable avec des fonctions de bases, en Python et avec Keil μVision, accessibles à tous. Finalement, ce processus d'automatisation assure une caractérisation rapide des primitives, diminuant fortement l'effort de caractérisation pour permettre un déploiement du modèle d'estimation sur de nombreux MCU.

Les primitives développées sont référencées dans la Table 3.1. Les algorithmes de chaque primitive sont présentés en Annexe A.1. Ces primitives ont été utilisées pour le développement de plusieurs CNN, dans l'objectif de tester la méthodologie proposée. Les paramètres des primitives pouvant varier sont la taille des noyaux, dans le cas de convolution ou de regroupement, ou le nombre d'entrées et de sorties dans le cas de certaines fonctions d'activations ou de couches entièrement connectées. La primitive de sigmoïde est très coûteuse en calcul. Pour réduire son coût d'exécution, les valeurs possibles de la sigmoïde sont pré-calculées et stockées en mémoire au sein du MCU. Puis, la valeur à filtrer est utilisée pour accéder aux valeurs correspondantes de la sigmoïde. Cette pratique est régulièrement utilisée lors de l'implémentation d'un CNN pour réduire le coût des opérations complexes à effectuer sur des architectures non spécialisées comme les MCU.

Le coût d'inférence des primitives présentées ci-dessus est caractérisé dans la cible matérielle avec le processus de caractérisation présenté en Figure 3.8. La caractérisation de ces primitives est la base de l'estimation du coût d'inférence d'un CNN, mais également la base de la seconde étape de la méthodologie ZIP-CNN consistant au choix des méthodes de réductions à appliquer.

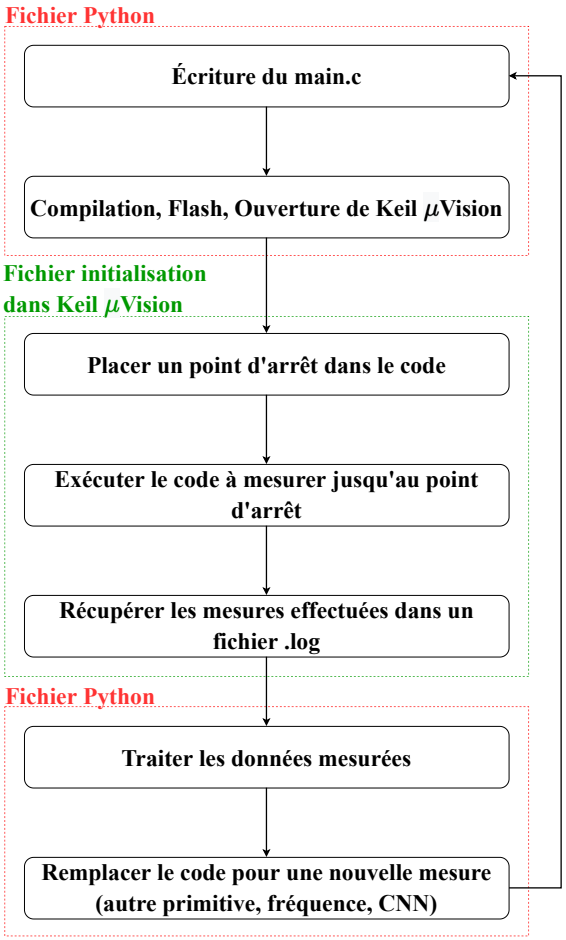


FIGURE 3.9 – Processus d'automatisation des mesures de latence et de consommation d'énergie.

TABLE 3.1 – Résumé des primitives et des paramètres associés développées pendant la thèse.

Famille de primitives	**Primitive**	**Paramètre : taille noyau k ou Entrées/Sorties**	**Algorithme Annexe A.1**
Extraction de caractéristiques	Convolution 2D	k = 1×1, 3×3, 5×5, 7×7, 9×9	Algorithme 1
Extraction de caractéristiques	Convolution 2D RGB	k = 1×1, 3×3, 5×5, 7×7, 9×9	Algorithme 2
Sous-échantillonnage	Regroupement par valeur moyenne	k = 2×2, 4×4, 6×6, 8×8, 10×10	Algorithme 3
Normalisation	Sigmoïde	NA	Algorithme 4
Normalisation	Softmax	10/10	Algorithme 5
Normalisation	ReLU	NA	Algorithme 6
Normalisation	*Batch Normalization*	NA	Algorithme 7
Normalisation	Connexion résiduelle	NA	Algorithme 8
Combinaison linéaire	Entièrement connectée	400/120, 120/84, 84/10, 64/10	Algorithme 9

3.2.5 Étape du choix

Le bloc d'estimation présenté précédemment nécessite une caractérisation des primitives. Cette étape de caractérisation est à faire une seule fois par MCU. Les résultats d'estimation sont ensuite utilisés une première fois pour estimer l'impact d'un CNN non réduit. Puis, grâce à la même caractérisation des primitives, le coût d'inférence du CNN s'il est réduit par de l'élagage ou de la quantification est estimé. Ainsi, l'impact des techniques de réduction sur le coût d'inférence du CNN est quantitativement estimé, sans avoir besoin d'appliquer ces techniques au préalable. Si l'estimation du coût d'une inférence du CNN réduit respecte les contraintes de l'application, les techniques de réduction concernées peuvent être appliquées. L'estimation de trois techniques de réduction a été étudiée, à savoir la quantification, l'élagage et la distillation de connaissances. La suite de cette sous-section décrit comment l'étape de caractérisation des primitives permet d'estimer le coût d'inférence d'un CNN réduit. Les aspects théoriques des techniques de réduction utilisées dans ces travaux sont ensuite présentés. De plus, l'étape de choix des techniques de réduction est ajoutée et détaillée au schéma de la méthodologie ZIP-CNN.

L'ajout de la seconde étape de la méthodologie ZIP-CNN, nommée "Choix Réduction", est schématisé en bleu sur la Figure 3.10. Cette étape offre l'avantage de ne pas avoir besoin d'appliquer les techniques de réduction sur le CNN puis de l'implémenter et d'effectuer des mesures, pour déterminer son coût d'inférence. L'étape "Choix Réduction" a plusieurs objectifs à savoir :

1. Choisir la politique de réduction la plus adaptée aux contraintes de l'application cible.
2. Estimer quantitativement le coût d'une inférence du CNN réduit en ce qui concerne la latence, la consommation d'énergie et l'espace mémoire nécessaires à une inférence.
3. Estimer quantitativement la réduction du coût d'inférence obtenue en appliquant une technique de réduction.

La vue détaillée de l'étape "Choix Réduction" est représentée sur la Figure 3.11. Pour estimer l'impact des trois techniques de réduction étudiées, à savoir la quantification, l'élagage et la distillation de connaissances, la consultation de la caractérisation des primitives faites précédemment est nécessaire. Puis l'estimation du coût d'inférence peut-être effectuée. Le modèle EST fourni une estimation quantitative du coût de l'inférence du CNN si la technique de réduction évaluée est appliquée. Si l'utilisateur valide le respect des contraintes de l'application cible avec un CNN réduit, la réduction peut-être appliquée au CNN. Dans le cas où aucune politique de réduction ne peut fournir un CNN respectant les contraintes de l'application cible, le choix d'un autre CNN dont la topologie offre une inférence moins coûteuse est nécessaire.

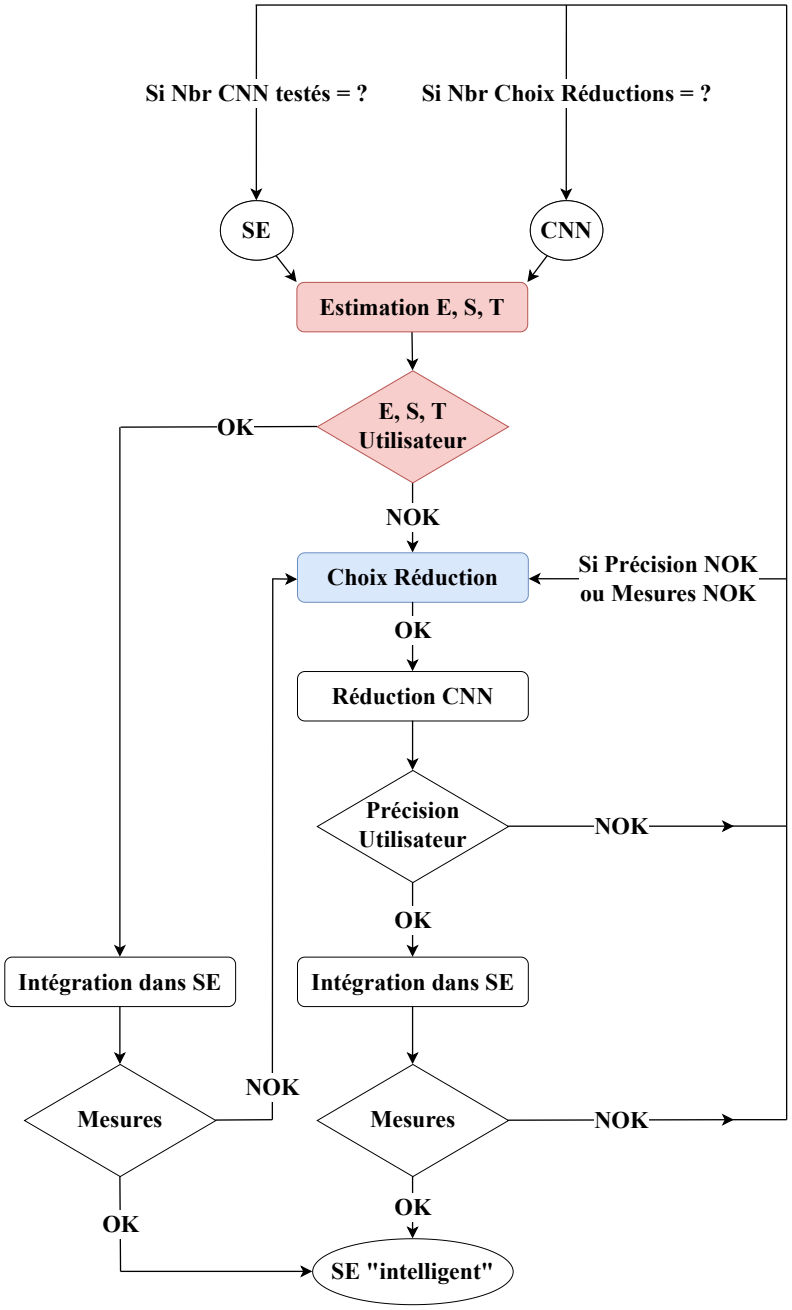


FIGURE 3.10 – Démarche d'intégration ZIP-CNN. La seconde étape "Choix Réduction" propose une analyse de l'impact des techniques de réduction sur le coût d'inférence du CNN au sein du système embarqué. Cette analyse est basée sur la consultation des caractérisations des primitives faites précédemment.

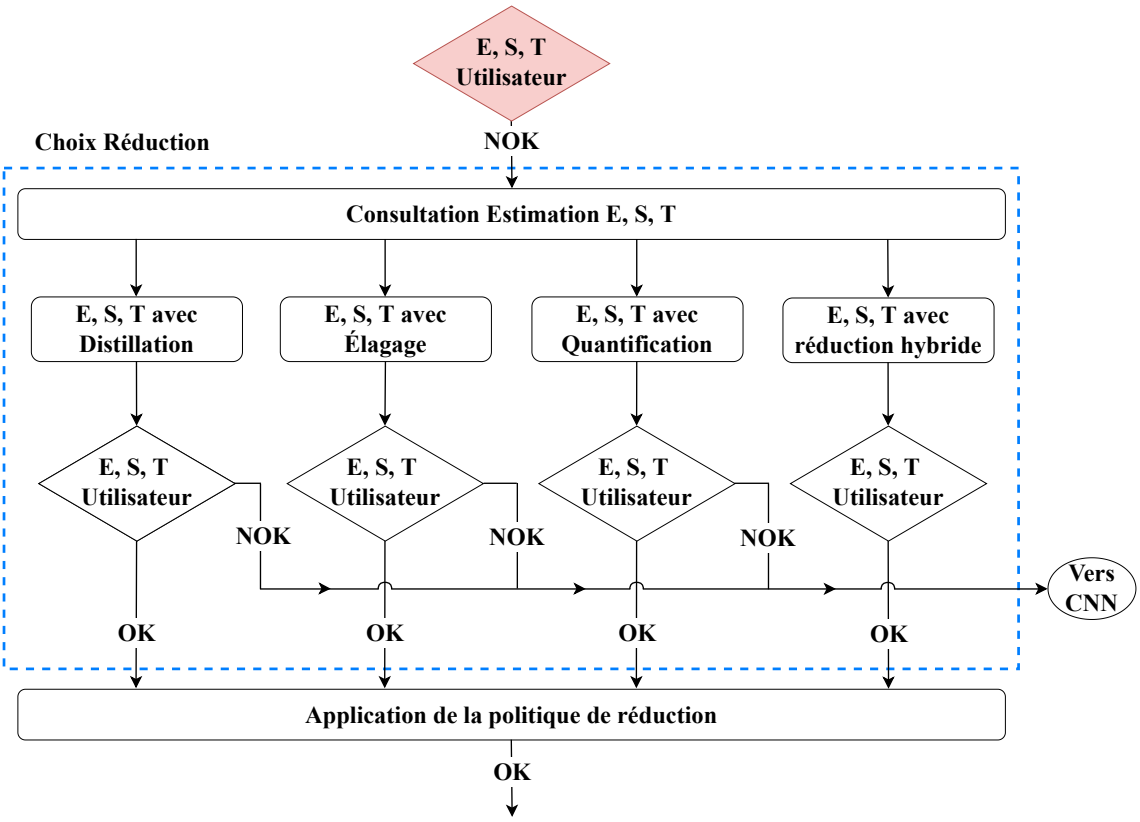


FIGURE 3.11 – . Vue détaillée du module "Choix Réduction". Trois techniques de réduction de l'état de l'art peuvent être estimées actuellement à savoir la quantification, l'élagage et la distillation de connaissances.

L'impact de l'application d'une seule technique de réduction ou l'application d'une stratégie hybride peut-être estimée. Une stratégie hybride comme un mélange d'élagage et de quantification peut-être plus efficace du point de vue des contraintes de l'application, mais demande également plus d'efforts de réductions. L'Équation 3.1, caractérisant l'estimation de l'impact du CNN non réduit et en FP32, est modifiée suivant l'Équation 3.2 où β est l'indice de format binaire et δ est l'indice d'élagage.

$$CNN_{estimation} = \sum PUC_{\beta} \times NA_{\delta} \tag{3.2}$$

En fonction du format binaire à appliquer durant la quantification ou du format binaire du réseau étudiant lors d'une distillation, le coefficient β fera correspondre la valeur du coût unitaire des primitives dans la LUT. En fonction du taux d'élagage appliqué, le coefficient δ fera correspondre le nombre d'applications de la primitive. Par exemple, dans le cas d'une quantification vers un format binaire INT8 et un élagage structuré de 50% des filtres de convolutions, l'Équation 3.2 devient :

$$CNN_{estimation} = \sum PUC_{INT8} \times NA_{50} \tag{3.3}$$

Ainsi, la LUT correspondant au coût unitaire des primitives au format binaire INT8 sera consultée, et le nombre d'applications passera du nombre d'applications de base de la primitive vers un nombre d'applications correspondant au taux d'élagage de 50%. Les aspects théoriques des techniques de réduction utilisées dans ces travaux sont présentés dans les sections suivantes.

3.2.6 Réductions algorithmiques appliquées

Cette section présente les trois techniques de réduction algorithmique étudiées pour réduire le coût d'inférence d'un CNN sur une cible matérielle embarquée. La liste des techniques de réduction n'est pas exhaustive. Comme présenté dans l'état de l'art, de nombreuses techniques de réduction existent et plusieurs approchent de quantification, d'élagage ou de distillation de connaissances sont possibles. Une présentation théorique des algorithmes utilisés durant ces travaux est faite, à savoir :

- la quantification d'un CNN au format binaire FP32 vers un format binaire INT8.
- l'élagage structuré des filtres de convolution d'un CNN au format binaire FP32.
- la distillation de connaissances d'un CNN en format binaire FP32 vers un CNN en format binaire FP32.

a. La quantification

Les techniques de quantification appliquées aux CNN, présentées dans l'état de l'art (*c.f.Section 2.3.2*), ont pour objectif de diminuer la précision de la représentation binaire utilisée lors d'une inférence. Par défaut, les outils d'apprentissage profond comme TensorFlow utilisent le format binaire à virgule flottante sur 32 bits (FP32). Une quantification en un format binaire moins lourd permettrait de réduire le coût d'inférence du CNN. Le schéma de quantification choisi est une quantification uniforme et symétrique. Le processus de quantification appliquée est celui de Jacob et al. [64], utilisé dans la librairie *TensorFlow Lite*. Le processus de quantification a été appliqué en PTQ - *Post Training Quantization*. Il est donc nécessaire d'entrainer un CNN en FP32 (un entraînement classique), puis d'appliquer le schéma de quantification pour quantifier les poids entrainés d'un format binaire FP32 vers un format binaire INT8. L'inférence complète du CNN est ainsi effectuée en format INT8, les poids et les activations étant quantifiés. Le schéma de quantification appliqué est défini suivant l'Équation 3.4, où Q représente la fonction de quantification, r représente la valeur réelle à quantifier, S est le facteur de mise à l'échelle, Z est la valeur quantifiée représentant le 0 réel et Int est une fonction d'arrondi.

$$Q(r) = Int(\frac{r}{S}) + Z \tag{3.4}$$

Le schéma inverse de la quantification est défini suivant l'Équation 3.5 où *Q(r)* représente la valeur quantifiée.

$$\tilde{r} = S \times (Q(r) - Z) \tag{3.5}$$

Une fois les poids quantifiés en INT8, il est nécessaire d'adapter le schéma d'inférence du CNN pour tenir compte de cette quantification. Cette adaptation est représentée sur la Figure 3.12 dans le cas d'une convolution entre deux matrices.

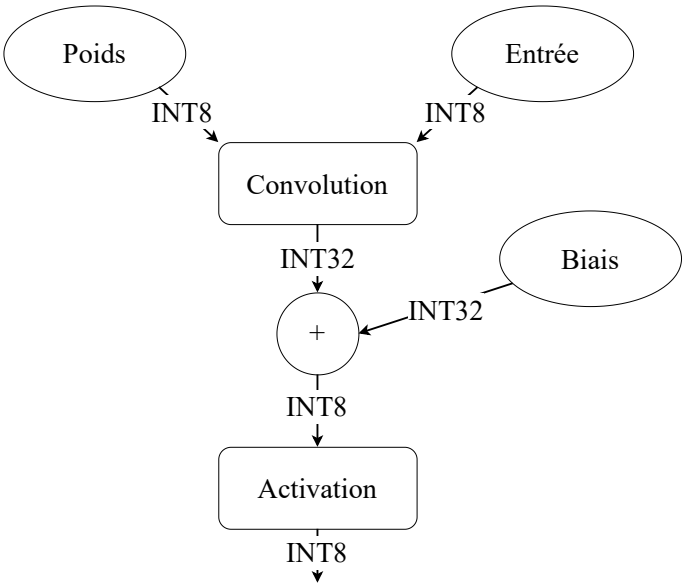


FIGURE 3.12 – Représentation fonctionnelle du schéma d'inférence d'un CNN quantifié en INT8 dans le cas d'une convolution entre deux matrices.

Le schéma d'inférence en INT8 est détaillé mathématiquement comme suit. La multiplication de deux matrices de nombre réel, nommées $r1$ et $r2$ et de taille $N \times N$ est notée $r3 = r1 \times r2$, où $r_\alpha(\alpha = 1, 2 \text{ ou } 3)$ est définie comme $r_\alpha^{(i,j)}$ pour $1 \leqq i, j \leqq N$. Les paramètres de quantification (S et Z) sont notés (S_α, Z_α). La valeur quantifiée est notée $q_\alpha^{(i,j)}$. Avec cette notation, l'Équation 3.5 est exprimée suivant l'Équation 3.6.

$$r_\alpha^{(i,j)} = S_\alpha(q_\alpha^{(i,j)} - Z_\alpha) \tag{3.6}$$

En suivant les règles d'une multiplication de deux matrices, l'Équation 3.6 est ré écrite suivant l'Équation 3.7 où le coefficient M est définie comme $M := \frac{S_1 S_2}{S_3}$.

$$q_3^{(i,k)} = Z_3 + M \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2) \tag{3.7}$$

Dans l'objectif de réduire le coût de calcul de l'inférence, Jacob et al. [64] ont proposé de ré écrire l'Équation 3.7 suivant l'Équation 3.8 avec $a_2^{(k)}$ et $\bar{a_1}^{(i)}$ défini suivant l'Équation 3.9.

$$q_3^{(i,k)} = Z_3 + M\left(NZ_1Z_2 - Z_1a_2^{(k)} - Z_2\bar{a_1}^{(i)} + \sum_{j=1}^{N} q_1^{(i,j)}q_2^{(j,k)}\right) \tag{3.8}$$

$$\bar{a_1}^{(i)} := \sum_{j=1}^{N} q_1^{(i,j)},\ a_2^{(k)} := \sum_{j=1}^{N} q_2^{(j,k)} \tag{3.9}$$

En pratique, l'implémentation de ce schéma d'inférence en INT8 est décrite comme suit. La multiplication des matrices est effectuée et le résultat est stocké dans un accumulateur au format INT32. Le biais, quantifié en INT32 est ensuite ajouté. Les coefficients de quantification du biais sont définis comme $S_{biais} = S1 \times S2$ et $Z_{biais} = 0$. Puis, il est nécessaire de faire une remise à l'échelle du format INT32 vers

un format INT8. Cette remise à l'échelle est effectuée en multipliant la valeur accumulée par le coefficient M. Finalement, la fonction d'activation est appliquée. Certaines fonctions d'activations étant très complexes à implémenter en INT8, comme la Sigmoïde ou la Tangente Hyperbolique, la fonction d'activation utilisée dans la suite de ces travaux est la fonction ReLU (*c.f. Tableau 1.2 Section 1.3*).

b. L'élagage

Les techniques d'élagage, présentées dans l'état de l'art (*c.f. Section 2.3.3*), ont pour objectif de supprimer les connexions redondantes ou des neurones qui ne contribuent pas ou peu à la précision du résultat dans un réseau de neurones. La technique d'élagage appliquée est un élagage structuré. Notre choix s'est porté sur ce type d'élagage plutôt que sur un élagage non structuré, parce que des matrices creuses résultent d'un élagage non structuré. Ce type de matrice n'est pas efficace à exécuter sur un MCU. Pour appliquer un élagage sur un CNN il est d'abord nécessaire de définir un critère d'élagage. La technique d'élagage appliquée dans ces travaux est celle de He et al. [85]. Basés sur le critère de médiane géométrique, les filtres de convolutions présentant des informations redondantes dans le CNN sont élagués. L'idée principale d'utiliser la médiane géométrique est décrite suivant l'Équation 3.10. $\mathcal{F}_{i,j}$ représente le j_{eme} filtre de la i_{eme} couche et de dimension $N_i \times K \times K$. Le nombre de canaux d'entrée et de sortie de la couche sont notés N_i et N_{i+1}. Soit un jeu de points n, $a^{(1)}...a^{(n)}$ avec $a^{(i)} \in \mathbb{R}$, il est nécessaire de trouver un point $x^* \in \mathbb{R}$ qui minimise la distance Euclidienne par rapport à ces points.

$$x^* = argmin\ f(x) \text{ où } f(x) = \sum_{i \in [1,n]} \parallel x - a^{(i)} \parallel_2 \tag{3.10}$$

Appliquée aux filtres de convolution d'un CNN, la médiane géométrique notée x^{GM}, défini suivant l'Équation 3.11, est utilisée pour caractériser les informations de tous les filtres de convolution, par couche de convolution notée i.

$$x^{GM} = \underset{x \in \mathbb{R}^{N_i \times K \times K}}{\operatorname{argmin}} \sum_{j' \in [1,N_{i+1}]} \parallel x - \mathcal{F}_{i,j'} \parallel_2 \tag{3.11}$$

Ensuite, en suivant l'Équation 3.12, le filtre de la couche de convolution i le plus proche de la médiane géométrique de la couche est déterminé.

$$\mathcal{F}_{i,j^*} = \underset{\mathcal{F}_{i,j'}}{\operatorname{argmin}} \parallel \mathcal{F}_{i,j'} - x^{GM} \parallel_2 \text{ , avec } j' \in [1, N_{i+1}] \tag{3.12}$$

Fi, j^* est un filtre de convolution présentant des informations redondantes par rapport aux informations des autres filtres de la couche. Ainsi, élaguer ce filtre n'aura que peu ou pas d'impact sur la précision finale du réseau, l'information supprimée existant toujours à travers les autres filtres de convolution de la couche.

c. La distillation de connaissances

La distillation de connaissances, présentée dans l'état de l'art (*c.f. Section 2.3.4*), est basée sur un principe enseignant-élève. Le schéma de distillation mis en place est une distillation hors-ligne, inspirée des travaux de Hinton et al. [89]. Ce schéma de distillation est l'un des plus répandus de l'état de l'art et offre un bon équilibre entre

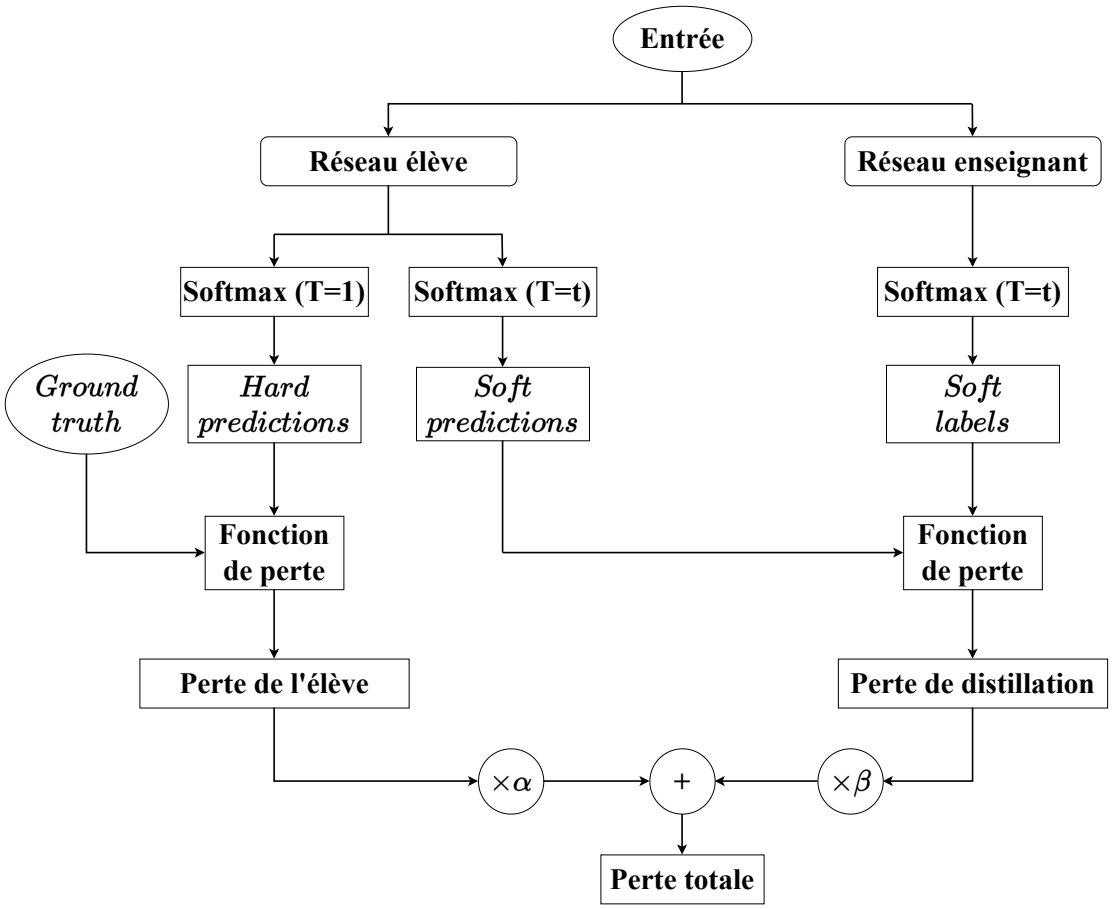


FIGURE 3.13 – Représentation fonctionnelle de la distillation de connaissances utilisée. La "Perte totale" permet d'entrainer le CNN élève. Cette perte résulte de la combinaison entre la "Perte de l'élève" et la "Perte de distillation".

l'efficacité des résultats et la complexité de mise en œuvre. La vue fonctionnelle de ce schéma de distillation de connaissances est illustrée sur la Figure 3.13.

Le CNN enseignant est d'abord entrainé pour fournir une précision en classification correspond aux attentes de l'utilisateur. Ensuite, la distillation lors de l'entrainement du réseau élève peut être mise en place. Les CNN enseignant et élève doivent se terminer par une fonction d'activation *Softmax*, la majorité des CNN de l'état de l'art remplissent cette condition. Ainsi, la probabilité d'appartenance aux différentes classes peut être exploitée. Tout d'abord, sur la partie droite de la Figure 3.13, les prédictions du CNN enseignant - *Soft labels* - sont combinés aux prédictions faites par le CNN étudiant - *Soft predictions* - pour créer une fonction de perte, la perte de distillation. Cette perte de distillation est multipliée par un coefficient β. Sur la partie gauche de la Figure 3.13, les prédictions du CNN étudiant - *Hard predictions* - en comparaison avec les étiquettes des données - *Ground truth* - génère une fonction de perte, qui devient la perte de l'élève. Multipliée par un coefficient α, cette perte de l'élève est additionnée à la perte de distillation pour créer la fonction de perte totale permettant d'entrainer le CNN étudiant. Les coefficients α et β pondèrent la contribution des fonctions de pertes de l'enseignant et de l'élève dans la fonction de perte totale. La fonction de perte de l'élève est définie suivant l'Équation 3.13, où L_S est la fonction de perte de l'élève - *Loss Student*, p_s est la *soft predictions* de l'élève et g_t l'étiquette des données - *Ground truth*.

$$L_S(p_s, g_t) = -\frac{1}{N}\sum^{N} g_t \times log(p_s) \tag{3.13}$$

La fonction de perte de distillation est définie suivant l'Équation 3.14, où L_D est la fonction de perte de distillation - *Loss distillation*, p_s est la *soft predictions* de l'élève et avec p_t les *soft labels* du CNN enseignant définis suivant l'Équation 3.15. Dans l'Équation 3.15, x est la sortie du CNN enseignant et T est un paramètre de distillation nommé *Temperature*.

$$L_D(p_t, p_s) = \sum p_t \times log(\frac{p_t}{p_s}) \tag{3.14}$$

$$p_t = Softmax(\frac{x}{T}) = \frac{\exp(-\frac{x}{T})}{\sum \exp(-\frac{x}{T})} \tag{3.15}$$

Ainsi, la fonction de perte totale nommée L_{total} est définie suivant l'Équation 3.16, avec $\alpha = 1 - \beta$.

$$L_{total} = \alpha \times L_S + \beta \times L_D \tag{3.16}$$

Le processus de distillation dépend ainsi de trois paramètres : la *Temperature* T et les coefficients de pondération α et β. L'effet de ces paramètres sur l'apprentissage n'est pas clairement défini dans l'état de l'art. Ainsi, une étude paramétrique a été encadrée et menée lors du stage de fin d'études de master de Karim Hocine. Des entrainements ont été effectués en faisant varier la valeur des paramètres α et β, entre 0,1 et 0,9 avec un pas de 0,1 et une température T fixée à 5. La meilleure précision en classification obtenue correspond à un $\alpha = 0,1$ et un $\beta = 0,9$. Cela signifie que la fonction de perte de l'élève contribue à 10% dans la fonction de perte totale, et la fonction de perte de distillation contribue à 90%. L'amélioration en précision de classification par rapport à un entrainement sans distillation est de +2%. Il a également été observé que la distillation de connaissances permet de faire converger la fonction de classification du CNN plus rapidement lors de l'entrainement.

3.2.7 Représentation complète de la méthodologie

Les parties précédentes ont détaillé les ajouts faits dans la méthodologie ZIP-CNN par rapport à la démarche d'intégration classique de l'état de l'art. La méthodologie complète est illustrée sur la Figure 3.14. Cette représentation de la méthodologie intègre les deux étapes présentées précédemment. Un utilisateur pourra être informé de la possibilité d'implémentation d'un CNN au sein d'un MCU dès le début du processus de conception. Sans implémentation, il pourra caractériser quantitativement la latence, la consommation énergétique et l'espace mémoire nécessaire à l'inférence de nombreux CNN, quelle que soit leur topologie. Si nécessaire, l'estimation du coût d'inférence du CNN réduit avec des techniques de réduction de l'état de l'art est possible. Cette estimation ne nécessite ni implémentation du CNN réduit, ni application des techniques de réduction évaluées. Ainsi, l'utilisateur pourra concentrer ses efforts sur une politique d'implémentation, avec ou sans réductions, en ayant une estimation quantitative de l'impact de l'inférence du CNN sur la cible matérielle. De nombreuses tentatives d'implémentation infructueuses peuvent ainsi être évitées, en guidant l'utilisateur dans les choix les plus judicieux pour respecter les contraintes en latence, en consommation énergétique et en espace mémoire de l'application cible. Dans le cas où l'estimation fournirait à l'utilisateur un coût d'inférence du CNN trop élevé, le choix d'une autre topologie de CNN devra être effectué. Le même processus d'estimation pourra être immédiatement appliqué avec ce nouveau CNN, sans coût de caractérisation supplémentaire grâce aux primitives. Il est possible qu'aucun CNN ne puisse respecter les contraintes de l'application cible. Dans ce cas, l'utilisateur devra changer de MCU. Si les primitives ont déjà été caractérisées une fois sur le nouveau MCU, l'estimation peut immédiatement être faite, sans coût de caractérisation supplémentaire. Dans le cas contraire, les primitives devront être caractérisées, suivant la Figure 3.8, sur le nouveau MCU pour suivre la méthodologie d'intégration.

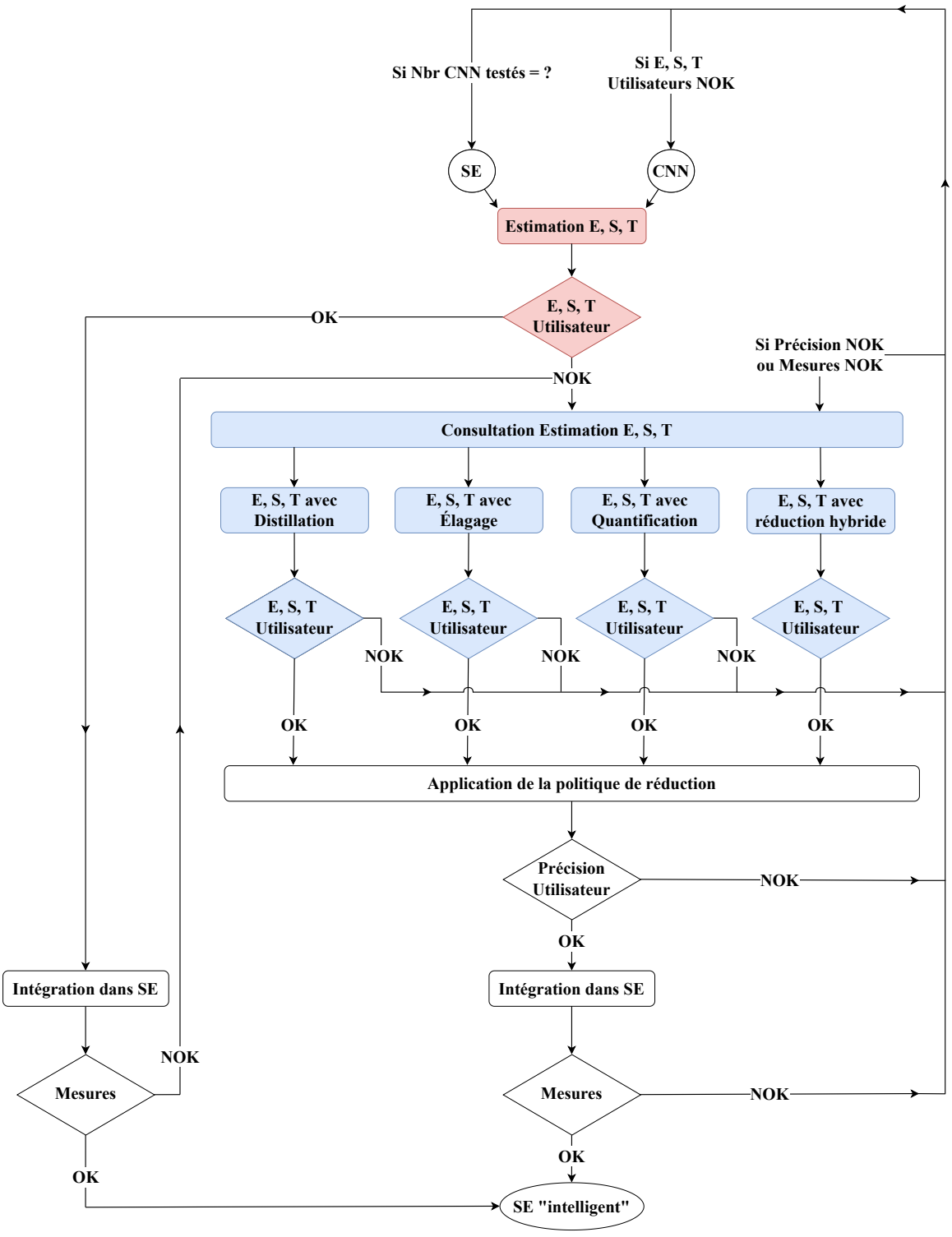


FIGURE 3.14 – Vue complète de la méthodologie ZIP-CNN. Les blocs en rouge correspondent au premier apport de la méthodologie à savoir l'Estimation EST. Les blocs en bleu correspondent au second apport de la méthodologie à savoir le Choix des techniques de réduction à appliquer.

3.3 Synthèse

Ce chapitre introduit la méthodologie ZIP-CNN. Deux étapes sont proposées pour guider un utilisateur dans les choix d'implémentation du CNN au sein d'un MCU, dans l'objectif de respecter les contraintes de l'application embarquée.

L'étape d'estimation du coût d'inférence du CNN dans la cible matérielle permet de déterminer rapidement et simplement si le CNN peut-être implémenté. De plus la latence, la consommation énergétique et l'espace nécessaire à l'inférence sont quantitativement estimés. L'estimation est basée sur une décomposition des CNN en plusieurs primitives, à la granularité des motifs de calcul. Cette granularité induit un effort de caractérisation et de mesure faible, tout en assurant une adaptation aux différentes topologies de CNN. Cette étape ne nécessite pas l'implémentation du CNN dans la cible matérielle.

L'étape du choix des optimisations algorithmiques à appliquer au CNN est basé sur l'estimation du coût d'inférence du CNN réduit. Basé sur les mêmes caractérisations que l'étape d'estimation, le coût d'inférence du CNN réduit est quantitativement estimé en ce qui concerne la latence, la consommation énergétique et l'espace mémoire nécessaires à une inférence. Cette étape ne nécessite pas l'application des optimisations algorithmiques ni l'implémentation du CNN réduit au sein de la cible matérielle.

Le chapitre suivant se concentrera sur la validation du modèle d'estimation, basé sur la granularité de décomposition des CNN par motif de calcul. Ce modèle sera confronté à des mesures validant quantitativement la justesse des estimations en latence, consommation énergétique et espace mémoire nécessaires à une inférence.

Chapitre 4

Validation du modèle d'estimation

Sommaire

4.1 Introduction

Ce chapitre présente la validation du modèle d'estimation du coût d'inférence d'un CNN. Ce modèle est central dans la méthodologie ZIP-CNN. Il donne une estimation quantitative à un utilisateur sur la latence, la consommation énergétique et l'espace mémoire d'un CNN au sein d'une cible matérielle. L'utilisateur peut ainsi facilement déterminer si les contraintes de son application embarquée seront respectées. L'approche de validation du modèle d'estimation est d'abord présentée. Puis, une première estimation du coût d'inférence d'un CNN est faite. Les techniques de réduction de quantification et d'élagage présentées précédemment ont été appliquées pour réduire le coût d'inférence du CNN testé. Finalement, le coût d'inférence du CNN réduit est estimé et la précision de l'estimation est présentée en comparaison à des mesures.

4.2 Approche de validation du modèle d'estimation

Cette section présente la validation mise en place pour évaluer la justesse du modèle d'estimation. Le processus d'implémentation d'un modèle python vers un MCU est d'abord décrit. Puis, le CNN utilisé pour cette preuve de concept est introduit. Les cibles matérielles et le processus de prise de mesures de courant sont finalement présentés.

4.2.1 Implémentation d'un CNN au sein d'un MCU

Une problématique récurrente dans l'implémentation d'un CNN au sein d'un MCU est d'exprimer un modèle décrit en Python en un modèle décrit en langage C embarqué. Certains outils présentés dans l'état de l'art simplifient cette traduction. Cependant, ces outils sont des boîtes noires et donc, ne sont pas adaptés à des problématiques de recherche où une exploration plus fine des modèles utilisés est nécessaire. Ainsi, les modèles de CNN utilisés dans la suite du manuscrit ont été codés et déployés manuellement. Pour avoir la maîtrise complète de l'implémentation effectuée, la procédure décrite sur la Figure 4.1 a été appliquée durant ces travaux.

Le CNN considéré est d'abord décrit avec un environnement de développement d'apprentissage profond. Les outils TensorFlow et Keras ont été utilisés dans ces travaux. Cette description au sein de ces outils a trois objectifs. L'entrainement du CNN est fait pour obtenir une précision en classification cohérente par rapport à l'état de l'art ou à la contrainte de l'application cible le cas échéant. Les techniques de réduction comme la quantification, l'élagage, ou la distillation de connaissances sont également mises en place durant cette étape. Finalement, une fois l'entrainement et/ou la réduction du CNN terminé(s), la sauvegarde du CNN et de ses poids entrainés est faite dans le format HDF - Hierarchical Data Format, dont l'extension est *.h5*. Le format HDF [141] est un conteneur de fichier, permettant de sauvegarder et de structurer de grandes quantités de données. Il s'agit d'un format de sauvegarde utilisé par TensorFlow et Keras. Ensuite, les poids du réseau entrainé sont extraits du fichier de sauvegarde du CNN. Ces poids sont utilisés dans le même CNN décrit en langage C. Cette description est effectuée à l'aide des primitives décrites dans la section précédente. L'intérêt de décrire le CNN en langage C avant de l'implémenter dans le MCU est de confirmer le fonctionnement de la topologie du réseau. Cela

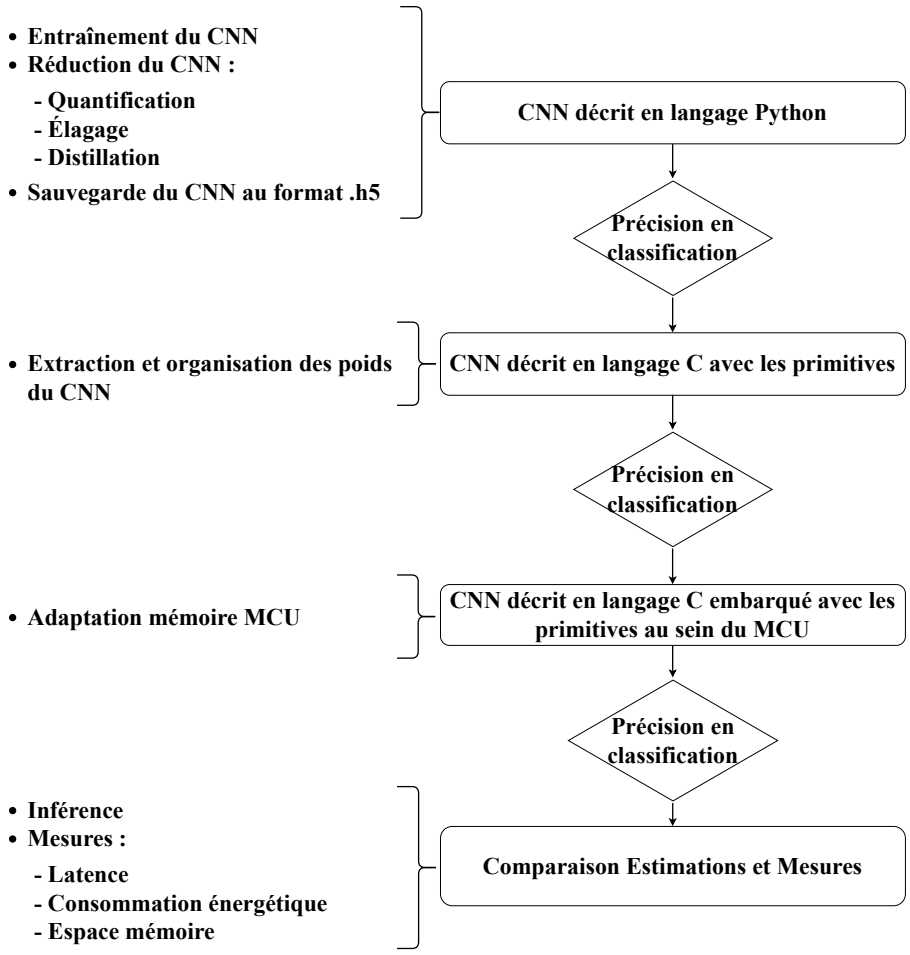


FIGURE 4.1 – Procédure d'implémentation d'un CNN décrit en langage Python vers un CNN implémenté dans un MCU pour y mesurer son coût d'inférence.

permet d'isoler les erreurs d'exécution du code liées à la topologie du CNN, de celles liées aux problématiques d'exécution sur MCU. Des tests en précision sont effectués sur ce modèle. Puis, ce modèle en langage C est compilé pour le MCU cible. Des ajustements sont nécessaires entre le modèle en langage C et le modèle en langage C embarqué, dont le principal est l'adaptation à la mémoire limitée du MCU. De nouveau, un test en précision est effectué pour confirmer le fonctionnement de la topologie du CNN au sein d'une cible matérielle contrainte. Le résultat en précision du CNN au sein du MCU est validé s'il est similaire à celui du modèle en langage C décrit sur ordinateur. Une fois ces étapes validées, des mesures sur des inférences sont effectuées en ce qui concerne la latence, la consommation énergétique et l'espace mémoire. Ces mesures sont finalement comparées aux estimations pour quantifier la précision du modèle d'estimation.

4.2.2 CNN utilisé pour la première estimation : LeNet5

Pour valider le modèle basé sur les primitives, nous utilisons le CNN LeNet5 développé par Lecun [22]. Ce réseau, représenté sur la Figure 4.2, est composé de 2 couches de convolution de taille de noyau $k = 5 \times 5$, 2 couches de regroupement par valeur moyenne de taille de noyau $k = 2 \times 2$ et 3 couches entièrement connectées.

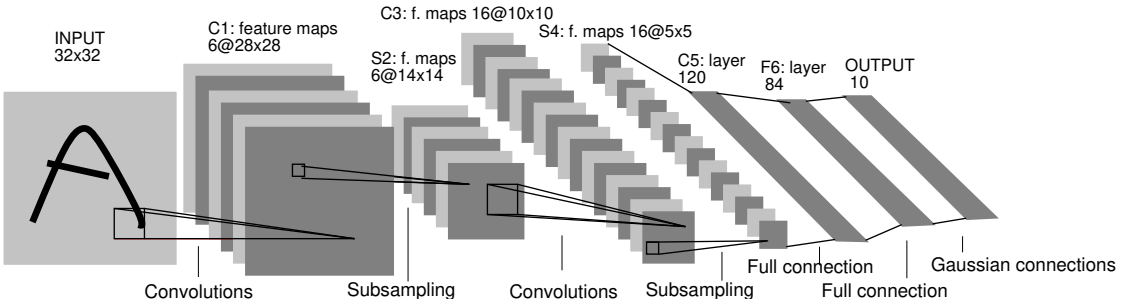


FIGURE 4.2 – Représentation fonctionnelle du CNN LeNet5.

Les fonctions d'activation appliquées sont la sigmoïde et le softmax (*c.f. Tableau 1.2 Section 1.3*). Le réseau LeNet5 est un CNN de l'état de l'art. Ses exigences en matière d'espace mémoire et de calcul illustrent bien les contraintes de déploiement des modèles d'apprentissage profond dans les MCU [10], [113]. Ce réseau a été utilisé en tant que preuve de concept. L'entrainement de LeNet5 a été effectué sur TensorFlow, en format binaire FP32, avec la base d'image MNIST [142]. Cette base d'images contient 60 000 images d'entrainement et 10 000 images de test. Chaque image est en niveau gris, de taille 28×28 et correspond à un chiffre manuscrit compris entre 0 et 9. Pour limiter les effets de bord, un remplissage - *padding* - de 2 est ajouté sur les images [22], résultant en une image d'entrée de taille 32×32. La sortie du CNN LeNet5 est un vecteur de taille 10, chaque sortie correspond à une classe, à savoir un chiffre entre 0 et 9. Après l'entrainement de LeNet5, la précision en classification obtenue sur la base des 10 000 images de test est de 97,86%.

La décomposition en primitives décrites précédemment, appliquée au réseau LeNet5, est référencée dans le Tableau 4.1.

TABLE 4.1 – Décomposition en primitives du CNN LeNet5.

Famille de primitives	**Primitive**	**Paramètre : taille noyau k ou Entrées/Sorties**	**Nombre d'exécutions**
Extraction de caractéristiques	Convolution 2D	$k = 5 \times 5$	14 304
Sous-échantillonnage	Regroupement par valeur moyenne	$k = 2 \times 2$	1 576
Combinaison linéaire	Entièrement connectée	400/120	1
Combinaison linéaire	Entièrement connectée	120/84	1
Combinaison linéaire	Entièrement connectée	84/10	1
Normalisation	Sigmoïde	NA	6,508
Normalisation	Softmax	10/10	1

L'architecture de LeNet5 définit le nombre d'applications pour chaque primitive, c'est-à-dire la variable NA dans l'Équation 3.1 (*c.f. Section 3.2.1*). Par exemple, LeNet5 est basé sur 2 couches de convolution. Le nombre d'applications pour la primitive de convolution est de 14 304. En effet, pour la première couche de convolution, 6 noyaux de taille 5×5 avec un pas de 1 sont appliqués sur l'entrée de la couche, c'est-à-dire une entrée de taille 32×32. Cela conduit à 6 cartes de caractéristiques de taille 28×28, soit $6 \times 28 \times 28 = 4\ 704$ applications de la primitive. Pour la deuxième couche de convolution, 16 noyaux de taille 5×5 avec un pas de 1 sont appliqués à l'entrée de la couche, c'est-à-dire une entrée de 6 cartes de caractéristiques de taille 14×14. Cela conduit à 16 cartes de caractéristiques de taille 10×10, soit $16 \times 6 \times 10 \times 10 = 9\ 600$ applications de la primitive. Ainsi, le nombre total d'applications de la primitive de convolution de taille de noyau 5×5 au sein du CNN LeNet5 est de $4704 + 9600 = 14\ 304$.

L'implémentation de ce modèle a été faite au sein de 3 MCU.

4.2.3 Cibles matérielles et outils

Les développements et mesures effectuées ont été implémentés en utilisant l'IDE Keil μVision [103]. La raison principale de ce choix est l'implémentation des configurations de MCU de plusieurs constructeurs au sein de cet outil. Ainsi, les travaux décrits dans ce manuscrit peuvent être appliqués sur des MCU de différents fabricants de puces. L'objectif initial était de tester les modèles sur des MCU des sociétés STMicroelectronics et NXP. Cependant, devant les délais très longs de commande de matériels électronique sur la période de la thèse, certaines cibles matérielles n'ont pas pu être testées. De ce fait, la validation des modèles a été effectuée sur des MCU de chez STMicroelectronics.

Plateformes matérielles cibles :

Les estimations et les mesures ont toutes été faites sur 3 plateformes matérielles, alimentées par une tension $V = 3.3V$. Elles intègrent l'architecture ARM Cortex-M, de la mémoire Flash et de la mémoire SRAM ou RAM. Les cibles matérielles sont référencées dans le Tableau 4.2, allant d'un MCU basse consommation (STM32L496ZG), d'un MCU haute performance (STM32F746ZG) et d'un MCU équilibré (STM32F446ZE). Toutes les expériences sont réalisées sur puce. Le compilateur utilisé est le compilateur ARM version 5.06 avec une optimisation de niveau 3(-O3).

TABLE 4.2 – MCU utilisés durant ces travaux.

MCU	Cœur de processeur	Mémoire Flash/S-RAM	Plage de fréquence
STM32L496ZG	Cortex M4	1Mo / 320ko	100kHz-80MHz
STM32F446ZE	Cortex M4	512ko / 132ko	16MHz-180MHz
STM32F746ZG	Cortex M7	1Mo / 320ko	200kHz-216MHz

Plus précisément, les MCU présentés ci-dessus sont implémentés au sein de carte NUCLEO-144. Les références des cartes utilisées sont une NUCLEO-L496ZG [143], une NUCLEO-F446ZE [144] et une NUCLEO-F746ZG [145]. Cette dernière est représentée sur la Figure 4.3.

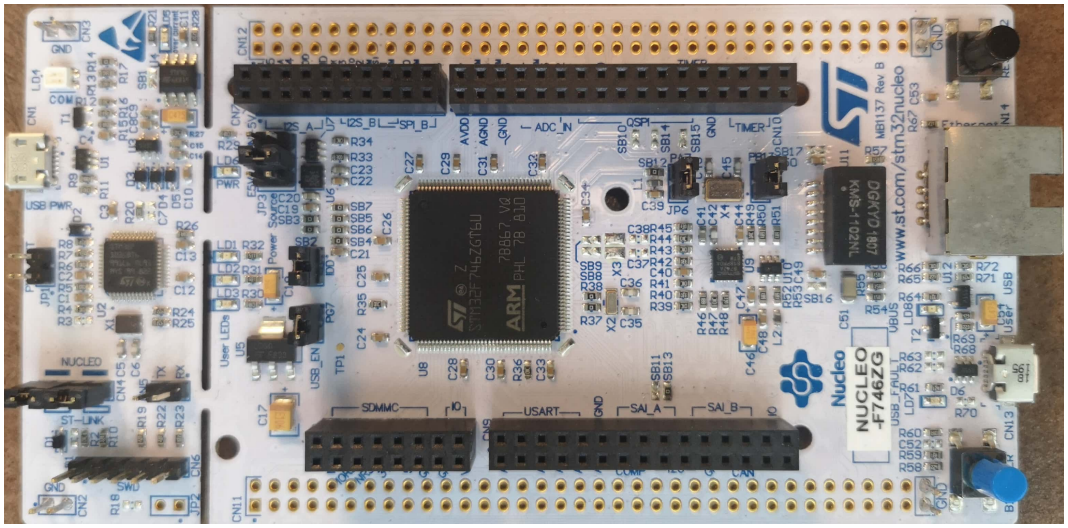


FIGURE 4.3 – Illustration d'une carte NUCLEO-144 utilisée dans ces travaux : la carte NUCLEO-F746ZG.

Outil de mesures de courant :

Concernant les mesures des trois métriques présentées précédemment, seul le courant est mesuré avec un outil extérieur au MCU. La sonde utilisée est une UlinkPlus de ARM-Keil [136]. Cette sonde offre une fréquence d'échantillonnage de 20MHz basée sur la technologie delta-sigma 16 bits, une précision sur la mesure du courant de 2% et une résolution de 200nA. Une fois connectée au port IDD (*jumper JP5*) des cartes NUCLEO-144, la sonde UlinkPlus permet de mesurer le courant consommé par le MCU. La connexion entre la sonde et une carte NUCLEO-144 est illustrée sur la Figure 4.4.

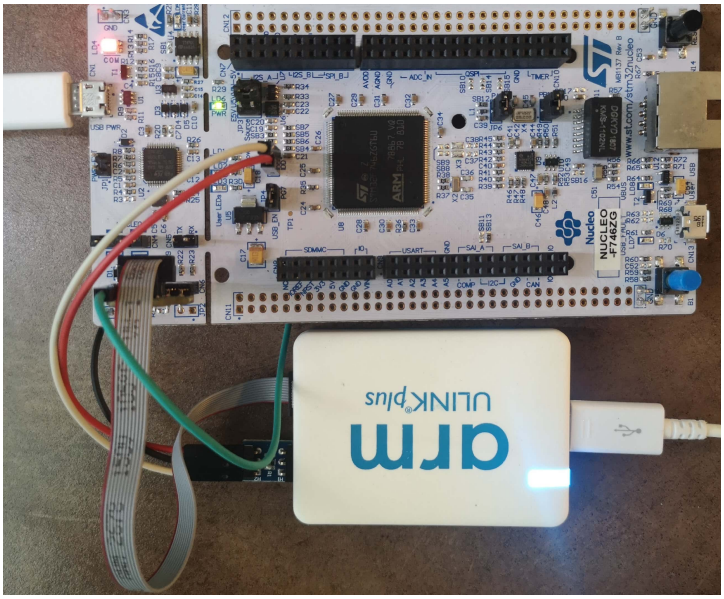


FIGURE 4.4 – Carte NUCLEO-F746ZG connectée à la sonde ARM-Keil UlinkPlus pour une mesure de courant. Les câbles rouge et blanc sont reliés au port IDD de la carte pour la mesure de courant du MCU. Le câble noir est relié à la masse et le câble vert est relié à une broche 3V3 pour alimenter l'adaptateur de la connexion de débogage de la sonde.

Des mesures en latence, consommation énergétique et espace mémoire du coût de l'inférence de LeNet5 visent à valider l'estimation basée sur les primitives. La section suivante présente ces résultats d'estimation et la comparaison avec les mesures.

4.3 Estimation du coût d'inférence de LeNet5

Les résultats d'estimation du coût d'inférence d'un CNN ont tous été comparés à des résultats de mesures du CNN implémenté au sein du MCU cible. Les primitives composant le CNN LeNet5 ont toutes été caractérisées suivant le processus décrit précédemment (*c.f. Figure 3.8 Section 3.2.1*). Les mesures du coût d'inférence de LeNet5 ont été réalisées suivant le même protocole que celui des mesures utilisées pour caractériser les primitives. Les mesures ont été faites sur la plage de fréquence respective des MCU. Le CNN LeNet5 est en format binaire 32 bits en flottant (FP32), ce format étant celui par défaut sur les outils d'apprentissage profond utilisés, à savoir TensorFlow et Keras.

Estimation de la latence :

La comparaison entre l'estimation et la mesure de la latence pour une inférence de LeNet5 est représentée sur la Figure 4.5. La latence exprimée en *ms/inférence* est représentée sur le graphique du haut de la Figure 4.5. La Figure 4.6 représente les mêmes données avec une échelle logarithmique sur l'axe des abscisses représentant les fréquences de fonctionnement des microcontrôleurs. Sur les 3 MCU et les 14 fréquences respectives, l'erreur entre l'estimation et la mesure de la latence d'une inférence est en moyenne de 5,78%. Sur chaque MCU, l'erreur est en moyenne de 7,54% pour le STM32L496ZG, de 6,52% pour le STM32F446ZE et de 3,29% sur le STM32F746ZG. Le détail quantitatif de ces résultats est disponible en Annexe B.2. La latence estimée de LeNet5 par le modèle de primitives suit le comportement de la latence mesurée, avec un coefficient de corrélation $r = \frac{Cov(X,Y)}{\sigma_X.\sigma_Y}$ supérieur à 0,99, où $Cov(X,Y)$ désigne la covariance des variables X et Y et σ_X et σ_Y désignent leurs écarts types. Cette corrélation entre la courbe d'estimation et la courbe de mesure est particulièrement visible aux fréquences 0,8MHz et 1MHz. Le graphique du bas de la Figure 4.5 représente la différence exprimée en % entre l'estimation et la mesure. La différence minimale observée sur toutes les mesures est de 0,47% et la différence maximale est de 15,49%. De plus, la différence sur toutes les fréquences est constante concernant le STM32L496ZG et le STM32F446ZE. Il est remarquable que la différence entre la prédiction et la mesure est du même ordre de grandeur concernant ces deux MCU. Cependant à partir de 80MHz, le pourcentage de différence varie entre la mesure et l'estimation concernant le STM32F746ZG. Ceci peut être dû à plusieurs effets liés à l'architecture du MCU. Le STM32L496ZG et le STM32F446ZE sont équipés du cœur de processeur Cortex M4. Le STM32F746ZG est équipé du Cortex M7. Les différences architecturales engendrent des accès aux données différents. Par exemple, les Cortex M4 et M7 sont équipés d'une matrice de bus Advanced High-performance Bus (AHB), assurant l'interconnexion entre les éléments maitres et esclaves. Pour le STM32F746ZG, les mémoires Flash et SRAM sont connectées avec le Cortex M7 via une interface Advanced eXtensible Interface (AXI)/multi-AHB. Cette interface communique avec une mémoire cache de niveau

L1 à travers un bus AXIM. Or, pour les STM32L496ZG et STM32F446ZE, ces connexions entre les mémoires Flash et SRAM avec le Cortex M4 n'existent pas. De plus, un Adaptive Real-Time (ART) Accelerator assure une performance architecturale optimale entre le cœur du processeur équipé d'une unité de calcul en virgule flottante - *Floating Point Unit (FPU)* et la technologie Flash. Dans le cas des hautes fréquences d'exécution du MCU, le ART Accelerator implémente des instructions de file d'attente de prélecture. Ces phénomènes architecturaux peuvent faire varier la justesse des prédictions de latence. La granularité de décomposition des CNN en motifs de calculs offre un bon équilibre entre la mise en place du modèle et sa justesse d'estimation. Cependant cela implique des limites sur la prise en compte de phénomènes visibles avec une granularité au niveau des réseaux de neurones ou des couches. Des investigations supplémentaires sur l'effet de ces mécanismes permettraient de préciser les limites du modèle d'estimation. Il reste remarquable que les prédictions rendent fidèlement le comportement en latence de l'inférence du CNN.

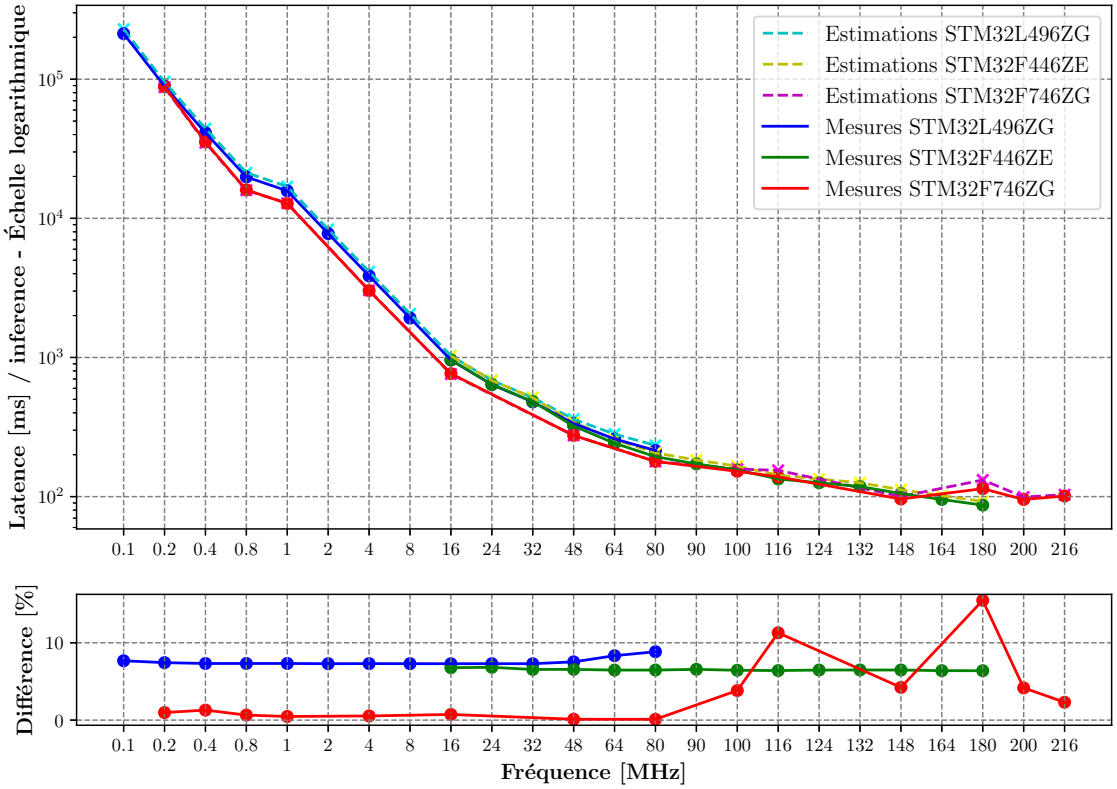


FIGURE 4.5 – En haut : estimations de la latence *vs.* mesures de la latence pour une inférence de LeNet5 au format binaire FP32 au sein des STM32L496ZG (cyan *vs.* bleu), STM32F446ZE (jaune *vs.* vert), et STM32F746ZG (magenta *vs.* rouge). Échelle logarithmique sur l'axe Y. En bas : différence entre la latence estimée pour une inférence et la latence mesurée.

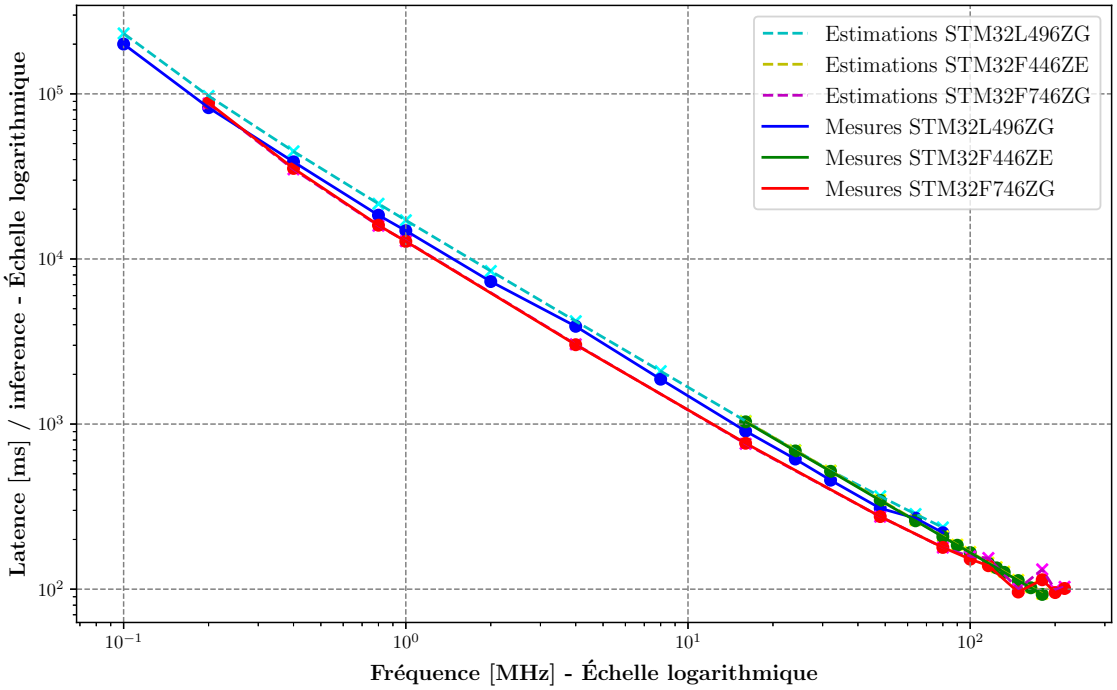


FIGURE 4.6 – Estimations de la latence *vs.* mesures de la latence pour une inférence de LeNet5 au format binaire FP32, échelle logarithmique

Estimation de la consommation d'énergie :

La comparaison entre l'estimation et la mesure de la consommation d'énergie pour une inférence de LeNet5 est représentée sur la Figure 4.7. La Figure 4.8 représente les mêmes données avec une échelle logarithmique sur l'axe des abscisses. La consommation énergétique exprimée en *J/inférence* est représentée sur le graphique du haut de la Figure 4.7. Sur les 3 MCU et les 14 fréquences respectives, le coefficient de corrélation est supérieur à 0,99 et l'erreur moyenne entre l'estimation de la consommation énergétique et la mesure est de 4,85%. Le graphique du bas de la Figure 4.7 représente la différence exprimée en % entre l'estimation et la mesure. La différence minimale observée sur toutes les mesures est de 0,039% et la différence maximale est de 17,51%. La différence entre les estimations et les mesures est plus marquée concernant le STM32L496ZG. De 100kHz à 32MHz la différence passe de 6,14% à 17,18% avant de retomber à 1,50% dans les hautes fréquences du MCU. Ce comportement peut être dû à plusieurs raisons. Il est nécessaire de tenir compte de l'erreur de l'estimation de la latence, qui impacte directement la consommation énergétique. De plus, le STM32L496ZG est conçu pour la basse consommation. Des mécanismes architecturaux assurent une consommation faible dans les basses fréquences, par exemple avec des sources d'horloge différentes. Cette hypothèse nécessite plus d'expériences concernant l'impact des mécanismes architecturaux du MCU cible. Malgré ces variations, l'erreur de la prédiction est en moyenne de 10,34% pour le STM32L496ZG. Concernant le STM32F446ZE elle est de 5,32% et pour le STM32F746ZG, de 3,12%. Le détail de ces résultats est disponible en Annexe B.2. Il est remarquable, de par l'étude du coût d'inférence sur l'ensemble des fréquences de fonctionnement des MCU, que le modèle d'estimation permet d'identifier la fréquence de fonctionnement offrant la plus faible consommation d'énergie.

Il est observable qu'une fréquence de fonctionnement trop basse consomme plus d'énergie qu'une fréquence de fonctionnement élevée. Ceci est contre intuitif. Ce comportement est dû à la prise en compte de la latence. La consommation énergétique estimée est liée au courant estimé, à la tension alimentant le MCU et à la latence estimée d'une inférence. La latence estimée impacte directement la consommation énergétique estimée. La latence d'une inférence est très élevée aux basses fréquences. Ainsi, les fréquences de fonctionnement les plus efficaces sont celles en milieu de plage de fréquences pour chaque MCU.

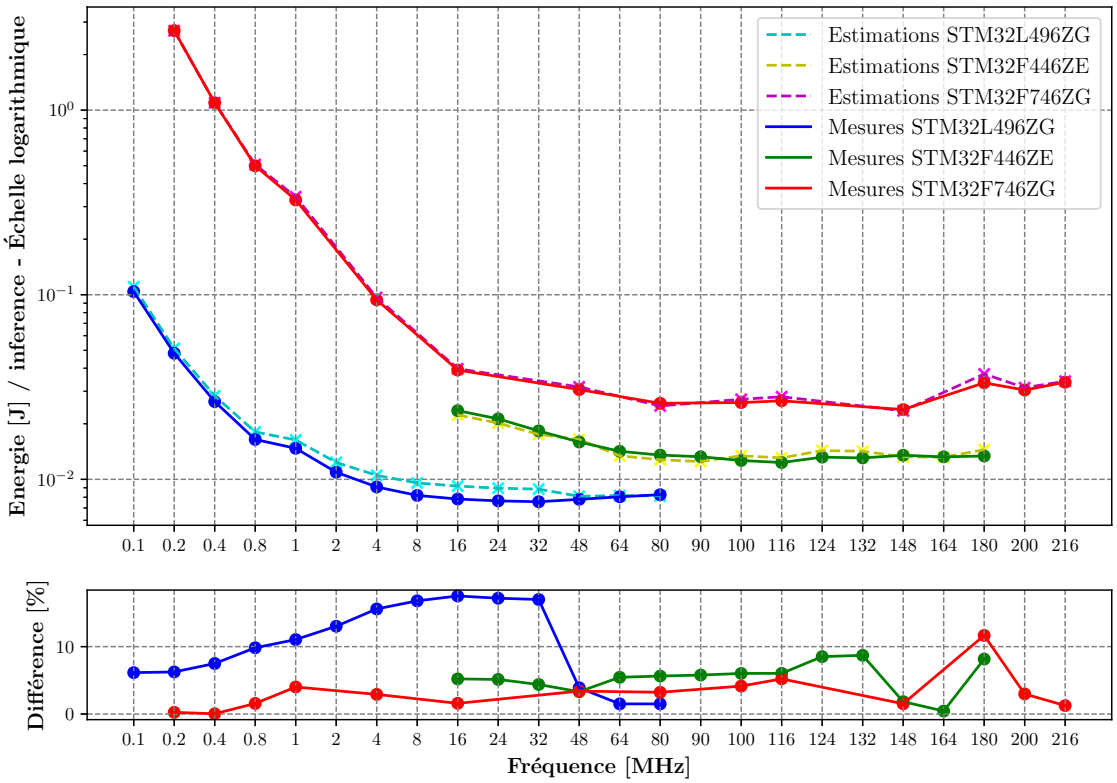


FIGURE 4.7 – En haut : estimations de la consommation d'énergie *vs.* mesures de la consommation d'énergie pour une inférence de LeNet5 au sein des STM32L496ZG (cyan *vs.* bleu), STM32F446ZE (jaune *vs.* vert), et STM32F746ZG (magenta *vs.* rouge). Échelle logarithmique sur l'axe Y. En bas : différence entre la consommation énergétique estimée pour une inférence et la consommation énergétique mesurée.

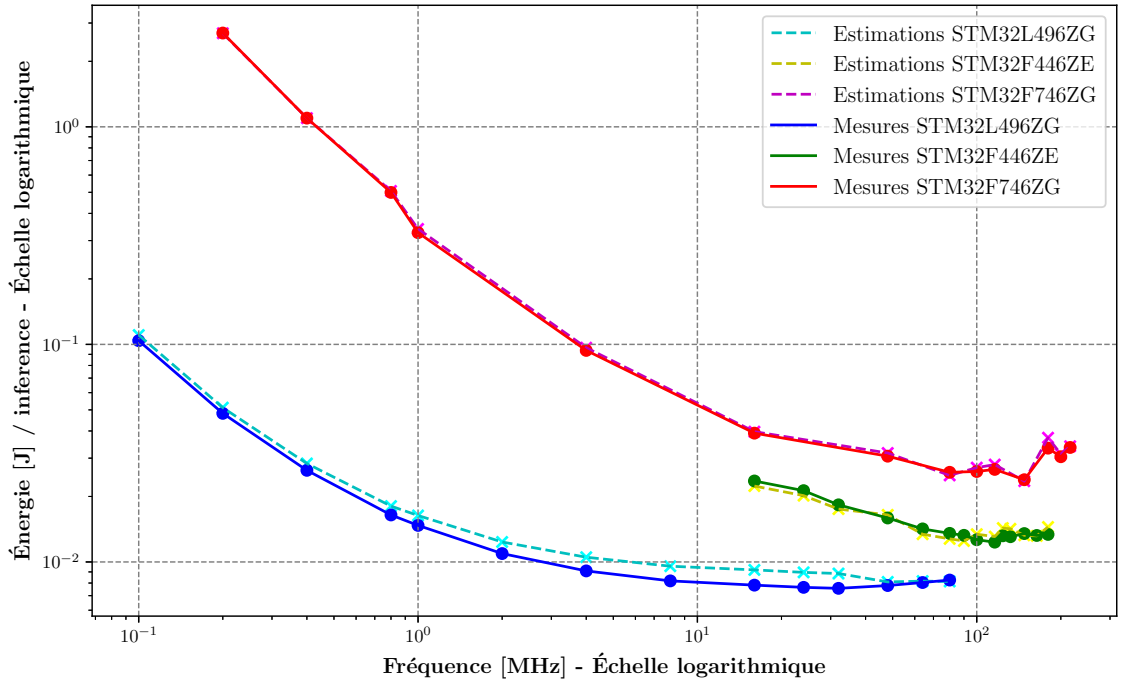


FIGURE 4.8 – Estimations de la consommation d'énergie *vs.* mesures de la consommation d'énergie pour une inférence de LeNet5, échelle logarithmique

La latence impactant directement l'estimation de la consommation énergétique, il est judicieux d'analyser également l'estimation de la puissance. Ces résultats permettent de mettre en avant la justesse du modèle de consommation des CNN. Le Tableau 4.3 indique la différence entre la mesure et l'estimation pour la puissance, la consommation d'énergie et la latence. Concernant le STM32F446ZE et le STM32F746ZG, l'ordre de grandeur entre l'estimation de la puissance et de la consommation d'énergie est similaire. Cependant, pour le STM32L496ZG, l'erreur d'estimation passe de 10,34% pour la consommation énergétique à 5,52% pour l'estimation de la puissance.

TABLE 4.3 – Comparaison entre les estimations et les mesures de la puissance et de l'énergie consommée pour une inférence de LeNet5 au format binaire FP32 au sein des MCU. La différence de l'estimation de la latence est également référencée, l'énergie étant directement liée à ce paramètre.

MCU	Diff. Puissance [%]	Diff. Énergie [%]	Diff.Latence [%]
STM32L496ZG	5,52%	10,34%	7,54%
STM32F446ZE	6,44%	5,32%	6,52%
STM32F746ZG	2,56%	3,12%	3,29%

Estimation de l'espace mémoire nécessaire :

La comparaison entre l'estimation et la mesure de l'espace mémoire Flash nécessaire pour implémenter LeNet5 est présentée sur le Tableau 4.4. Sur les 3 MCU, l'erreur moyenne entre l'estimation de l'espace mémoire Flash et la mesure est de 1,82%. Pour chaque MCU la différence est de 2,08% pour le STM32L496ZG, 1,06% pour le STM32F446ZE et 2,31% pour le STM32F746ZG. La faible différence entre

l'espace Flash mesuré pour les 3 MCU est due au code de configuration du système qui est différent entre tous les MCU.

TABLE 4.4 – Estimation de l'espace mémoire Flash (Code+Poids)

MCU	Mémoire (Code+Poids) estimée [ko]	Mémoire (Code+Poids) mesurée [ko]	Différence [%]
STM32L496ZG	248.31	254.02	2.25%
STM32F446ZE	248.31	253.24	1.95%
STM32F746ZG	248.31	258.44	3.92%

Le Tableau 4.5 présente l'estimation de la mémoire RAM nécessaire à l'inférence de LeNet5. La couche nécessitant le plus de primitives et donc le plus de données intermédiaires à stocker est la couche $\mathcal{L}2$. Cette couche prend en entrée 4 704 primitives de normalisation, à savoir la fonction d'activation sigmoïde, et génère en sortie 1176 données via la primitive de regroupement par valeur moyenne. L'espace RAM minimum nécessaire à une inférence de LeNet5 en FP32 est ainsi de $(4704 + 1176) \times 4 = 23520$ octets, soit 23,52 ko. En comparaison, l'espace mémoire RAM disponible sur les MCU ciblés, à savoir 132 et 320 ko, est largement supérieur aux besoins pour l'inférence de LeNet5.

TABLE 4.5 – Estimation de l'espace mémoire RAM nécessaire au stockage des calculs intermédiaires.

Couche	Primitive à considérer	Primitives en entrée	Primitives en sortie	Total primitives	Total mémoire [ko]
$\mathcal{L}1$	Normalisation	NA	4 704	4 704	18,82
$\mathcal{L}2$	Sous-échantillonnage	4 704	1 176	5 880	23,52
$\mathcal{L}3$	Normalisation	1 176	1600	2 776	11,10
$\mathcal{L}4$	Sous-échantillonnage	1600	400	2 000	8,00
$\mathcal{L}5$	Normalisation	400	120	520	2,08
$\mathcal{L}6$	Normalisation	120	84	204	0,82
$\mathcal{L}7$	Normalisation	84	10	94	0,38

Synthèse de la première estimation :

Les premières estimations du coût d'inférence de LeNet5 ont caractérisé la latence et la consommation énergétique à 14 fréquences de fonctionnement différentes, respectives à chaque MCU, ainsi que l'espace mémoire sur les 3 MCU. Sur toutes ces estimations, l'erreur moyenne est de 5,78% pour la latence, 4,85% pour la consommation d'énergie et 1,82% pour l'espace mémoire. Cette première estimation a permis de caractériser quantitativement la précision du modèle basé sur les primitives. Avec le même modèle, la latence, la consommation énergétique et l'espace mémoire Flash et RAM nécessaires à une inférence d'un CNN peuvent être quantitativement estimés. Il est maintenant requis d'appliquer le même processus d'estimation pour l'inférence d'un CNN résultant d'une réduction algorithmique. L'objectif est de caractériser la précision du même modèle, pour estimer le coût d'inférence d'un CNN réduit par une technique d'élagage, de quantification ou de distillation de connaissances.

4.4 Estimation du coût d'inférence de LeNet5 réduit

Cette section présente les résultats d'estimation de LeNet5 réduit. L'impact de la quantification est estimé, en quantifiant LeNet5 d'un format binaire FP32 vers un format binaire INT8. Le réseau au format binaire FP32 est également élagué à 50%, 60%, 70%, 80% et 90% de ses noyaux de convolutions. La distillation de connaissances utilisée dans ses travaux est appliquée depuis un CNN en FP32 vers un CNN en FP32. L'estimation d'un CNN en FP32 a été présentée précédemment. Ainsi, la distillation n'a donc pas été ré appliquée à LeNet5, l'estimation de son coût d'inférence étant déjà présentée dans la section précédente.

La fonction d'activation sigmoïde est complexe à implémenter en INT8. Elle a été remplacée par la fonction d'activation ReLU, plus utilisée dans l'état de l'art. La décomposition en primitive de LeNet5 est ainsi définie suivant le Tableau 4.6. Le nombre de primitives reste inchangé, seulement la fonction sigmoïde a été remplacée par la fonction ReLU.

TABLE 4.6 – Décomposition en primitives du CNN LeNet5, la fonction d'activation ReLU a remplacé la fonction d'activation sigmoïde.

Famille de primitives	**Primitive**	**Paramètre : taille noyau k ou Entrées/Sorties**	**Nombre d'exécutions**
Extraction de caractéristiques	Convolution 2D	$k = 5 \times 5$	14 304
Sous-échantillonnage	Regroupement par valeur moyenne	$k = 2 \times 2$	1 576
Combinaison linéaire	Entièrement connectée	400/120	1
Combinaison linéaire	Entièrement connectée	120/84	1
Combinaison linéaire	Entièrement connectée	84/10	1
Normalisation	ReLU	NA	6,508
Normalisation	Softmax	10/10	1

Le même processus d'estimation et de mesure précédemment présenté a été utilisé pour caractériser le coût d'inférence de LeNet5 réduit. Entre la quantification et l'élagage, cela représente 6 réseaux LeNet5 différents. Pour comparer quantitativement la précision de l'estimation à la mesure de l'inférence, chaque LeNet5 réduit a été implémenté au sein des 3 MCU. Les mesures ont également été réalisées sur les 14 fréquences de fonctionnement de chaque MCU. Les résultats présentés en suivant présentent un coefficient de corrélation supérieur à 0,99 entre les données d'estimation et les données mesurées. Les résultats de ces mesures et de ces estimations sont disponibles en Annexe B.2

4.4.1 LeNet5 quantifié en INT8

Les primitives et le réseau LeNet5 ont été quantifiés en format binaire INT8. Les primitives ont été modifiées pour prendre en compte les coefficients mis en jeu lors de l'inférence d'un CNN quantifié en INT8, à savoir les coefficients de quantification

S, *Z* et le coefficient de mise à l'échelle *M*, définis *Section 3.2.6*. La précision en classification de LeNet5 quantifié en INT8 est 98,2% sur la base MNIST.

Estimation de la latence :

La comparaison entre l'estimation et la mesure de la latence pour une inférence de LeNet5 quantifié est représentée sur la Figure 4.9. La Figure 4.10 représente les mêmes données en échelle logarithmique concernant la représentation des fréquences de fonctionnement du microcontrôleur. La latence est exprimée en *ms/inférence*. Sur les 3 MCU et les 14 fréquences respectives, l'erreur entre l'estimation de la latence d'une inférence est la mesure est en moyenne de 5,79%. Sur chaque MCU, l'erreur est en moyenne de 3,14% pour le STM32L496ZG, de 3,36% pour le STM32F446ZE et de 10,88% sur le STM32F746ZG. Le détail quantitatif de ces résultats est disponible en Annexe B.2.

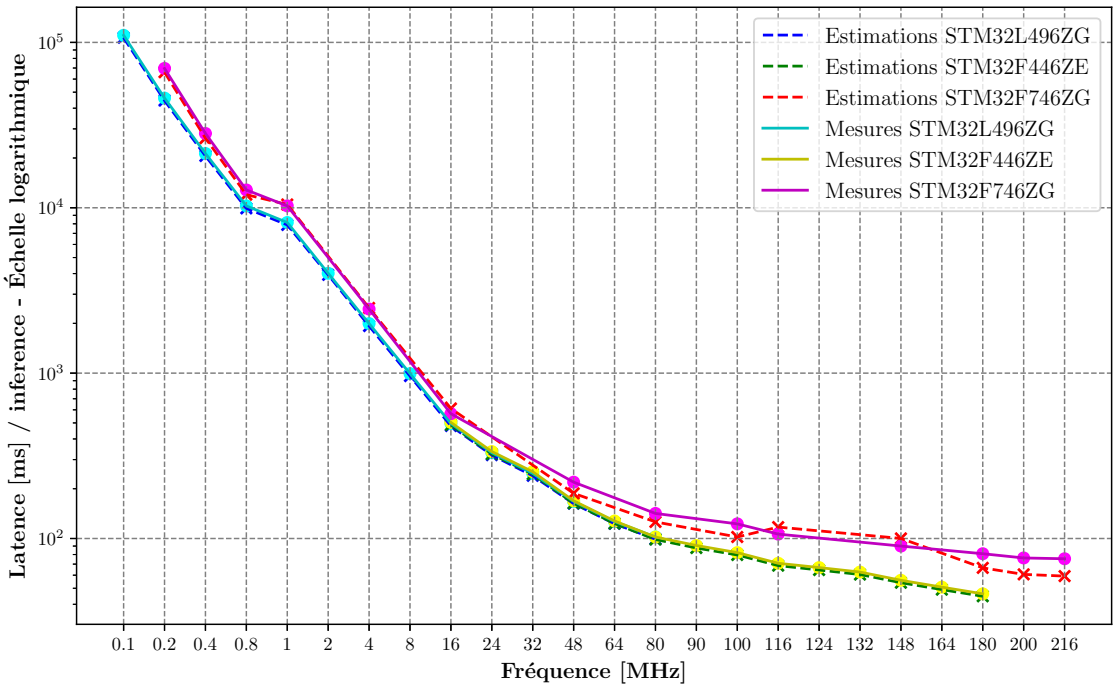


FIGURE 4.9 – Estimations de la latence *vs.* mesures de la latence pour une inférence de LeNet5 au format binaire INT8 au sein des STM32L496ZG (bleu *vs.* cyan), STM32F446ZE (vert *vs.* jaune), et STM32F746ZG (rouge *vs.* magenta). Échelle logarithmique sur l'axe l'axe Y.

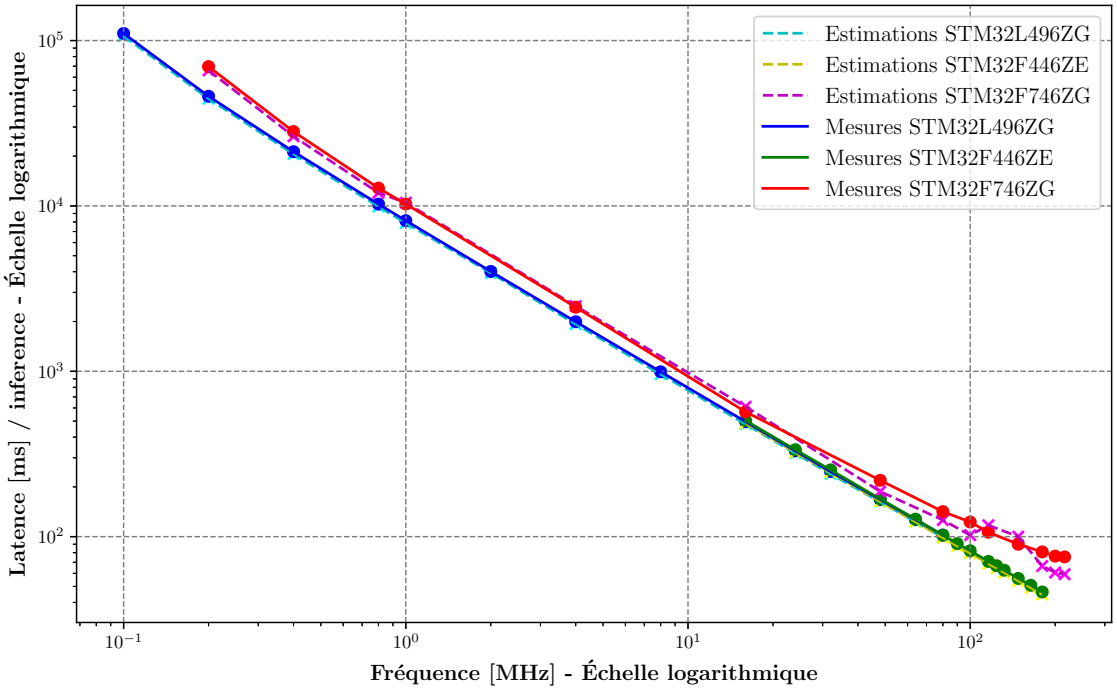


FIGURE 4.10 – Estimations de la latence *vs.* mesures de la latence pour une inférence de LeNet5 au format binaire INT8, échelle logarithmique

L'allure de la courbe de la latence de LeNet5 en INT8 est très proche de celle de LeNet5 en FP32. La quantification vers un format binaire INT8 n'influe pas le comportement en latence d'un CNN implémenté dans un MCU. Il est cependant nécessaire de nuancer cette observation. Les MCU utilisés sont équipés d'une FPU. Une unité matérielle est dédiée aux calculs en FP32. Ainsi, le gain en latence concernant une exécution en INT8 et en FP32 est faible. Dans le cas où le même test serait mené sur un MCU n'étant pas équipé de FPU, le gain en latence attendue en quantifiant vers un format binaire INT8 serait plus élevé. Le modèle d'estimation met ainsi en avant que si la contrainte à améliorer est la latence, une quantification en INT8 n'est pas efficace si le MCU cible est équipé d'une FPU. De plus, le modèle d'estimation doit être capable de prédire la réduction du coût d'inférence apportée par la quantification. La réduction mesurée et estimée en latence, entre le format binaire FP32 et le format binaire INT8 est référencée sur le Tableau 4.7 et illustrée sur la Figure 4.11. Concernant le STM32F446ZE, une différence d'une quinzaine de pour cent apparaît entre la réduction estimée et la réduction mesurée. Cette différence est due à l'erreur d'estimation en latence pour LeNet5 en FP32 sur le STM32F446ZE. Pour les STM32L496ZG et STM32F746ZG, l'estimation de la réduction en latence est très précise avec respectivement 2,82% et 0,84% d'erreur.

TABLE 4.7 – Réduction en latence de LeNet5 en INT8 par rapport à LeNet5 en FP32.

MCU	Réduction mesurée	Réduction estimée	Différence
STM32L496ZG	5,48%	8,30%	2,82%
STM32F446ZE	21,77%	4,96%	16,81%
STM32F746ZG	12,76%	13,60%	0,84%

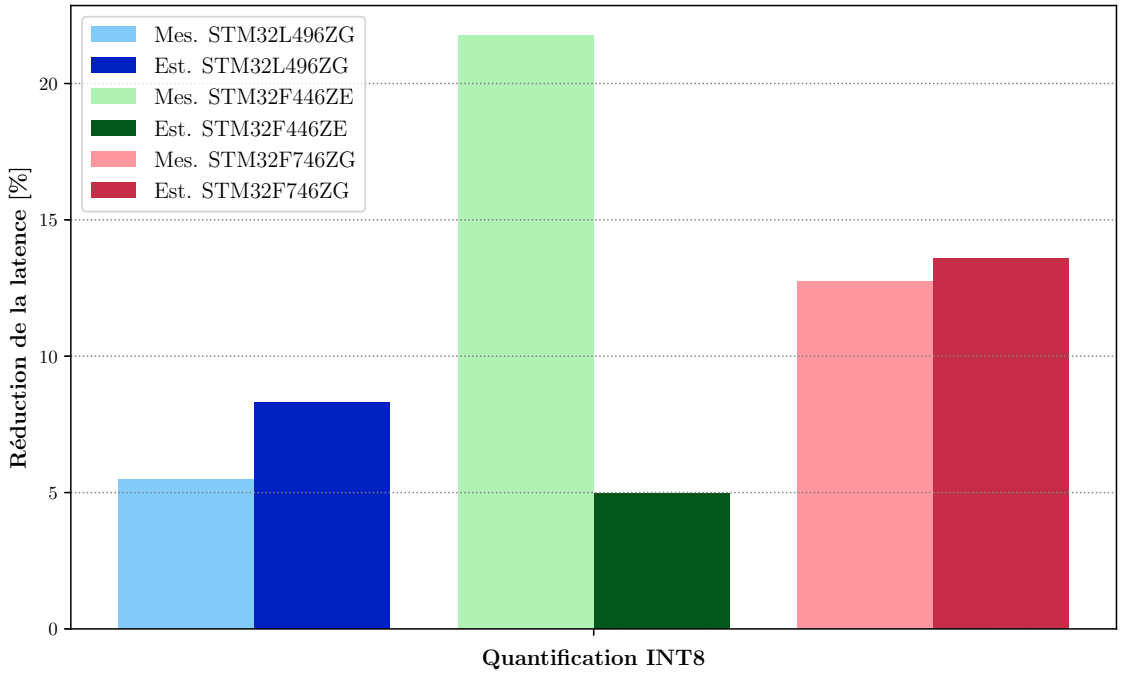


FIGURE 4.11 – Réduction en latence estimée et mesurée de LeNet5 en INT8 par rapport à LeNet5 en FP32.

Estimation de la consommation d'énergie :

La comparaison entre l'estimation et la mesure de la consommation d'énergie pour une inférence de LeNet5 en INT8 est représentée sur la Figure 4.12. La Figure 4.13 représente les mêmes données en échelle logarithmique. La consommation énergétique est exprimée en *J/inférence*. Sur les 3 MCU et les 14 fréquences respectives, l'erreur moyenne entre l'estimation de la consommation énergétique et la mesure est de 9,32%. Sur chaque MCU, l'erreur est de 4,65% pour le STM32L496ZG, de 11,74% pour le STM32F446ZE et de 11,57% sur le STM32F746ZG. Le détail de ces résultats est disponible en Annexe B.2.

Le Tableau 4.3 indique la différence entre la mesure et l'estimation pour la puissance, la consommation d'énergie et la latence. Concernant le STM32F446ZE l'ordre de grandeur entre l'estimation de la puissance et de la consommation est similaire. Pour le STM32L496ZG et le STM32F746ZG, une différence d'un facteur 2 existe entre l'estimation de la consommation énergétique et l'estimation de la puissance. Ces résultats montrent de nouveau une influence de la latence dans l'estimation de la consommation énergétique.

TABLE 4.8 – Comparaison entre les estimations et les mesures de la puissance et de l'énergie consommée pour une inférence de LeNet5 au format binaire INT8 au sein des MCU. La différence de l'estimation de la latence est également référencée, l'énergie étant directement liée à ce paramètre.

MCU	Diff. Puissance [%]	Diff. Énergie [%]	Diff.Latence [%]
STM32L496ZG	1,56%	4,65%	3,14%
STM32F446ZE	9,26%	11,74%	3,36%
STM32F746ZG	6,21%	11,57%	10,87%

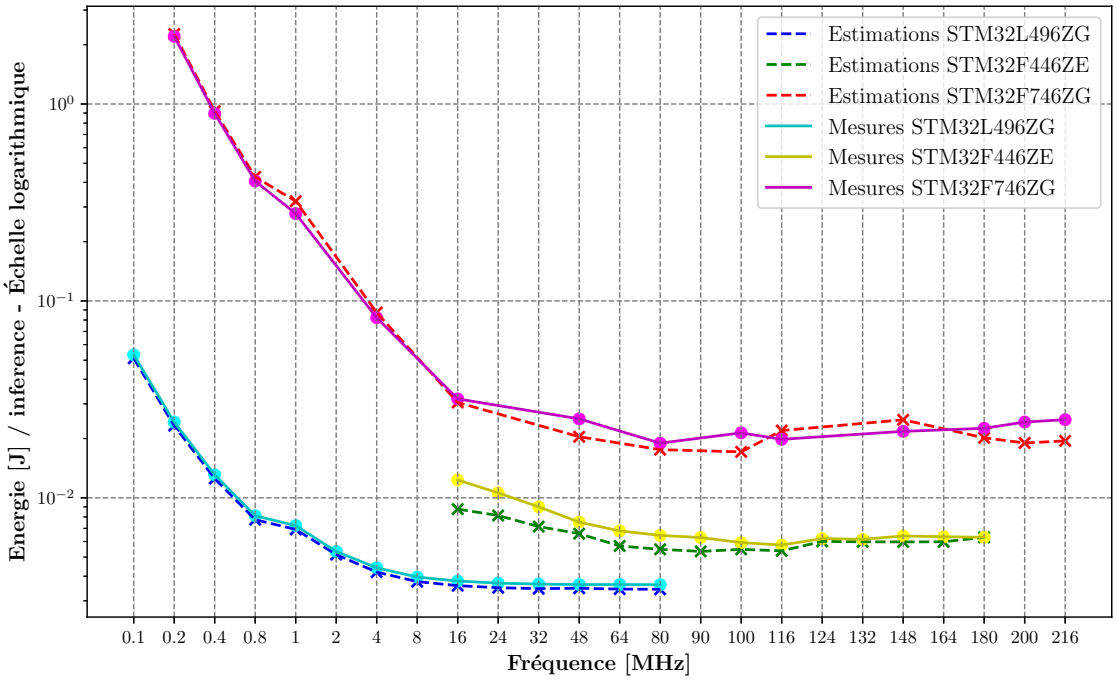


FIGURE 4.12 – Estimations de la consommation d'énergie *vs.* mesures de la consommation d'énergie pour une inférence de LeNet5 au format binaire INT8 au sein des STM32L496ZG (bleu *vs.* cyan), STM32F446ZE (vert *vs.* jaune), et STM32F746ZG (rouge *vs.* magenta). Échelle logarithmique sur l'axe l'axe Y.

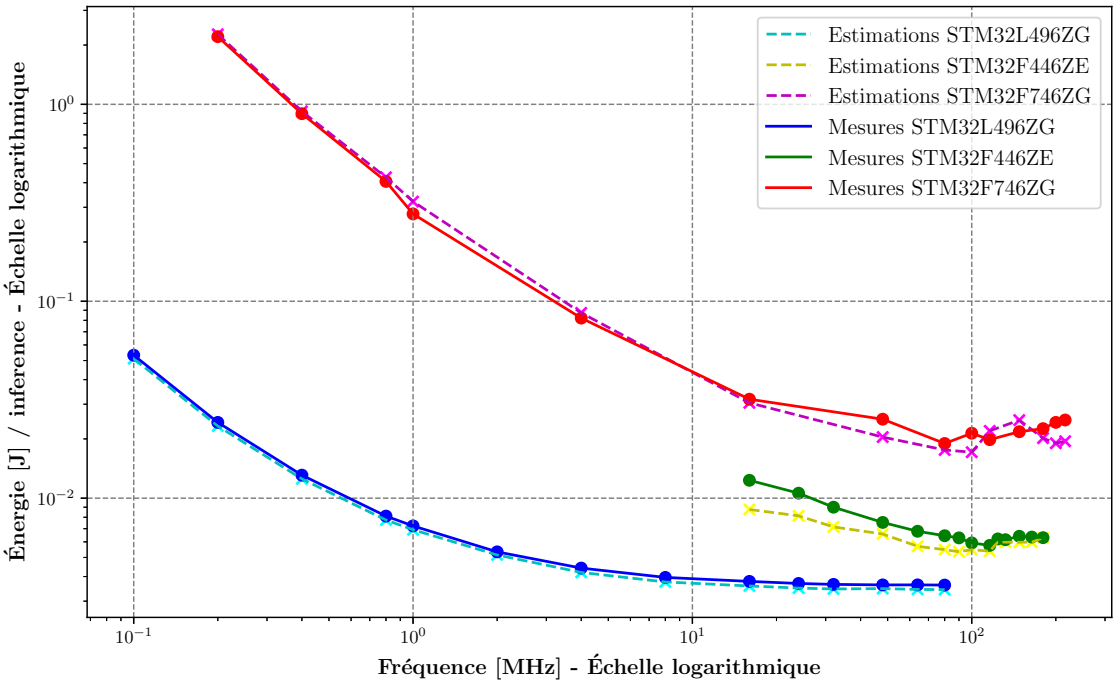


FIGURE 4.13 – Estimations de la consommation d'énergie *vs.* mesures de la consommation d'énergie pour une inférence de LeNet5 au format binaire INT8, échelle logarithmique

L'estimation et la mesure de la réduction en consommation énergétique liée à la quantification en INT8 sont détaillées pour chaque MCU sur le Tableau 4.9 et illustrées graphiquement sur la Figure 4.14. La réduction sur le STM32L496ZG est d'environ 8%. Une réduction plus importante sur le STM32F446ZE et STM32F746ZG, environ 20%, est observable. Le STM32L496ZG étant conçu pour avoir une faible consommation énergétique, l'optimisation sur cette caractéristique est moindre comparée aux deux autres MCU. La présence d'une FPU au sein des MCU utilisés influe également sur la réduction en consommation énergétique amenée par le INT8, comme expliqué précédemment concernant la réduction en latence. Il est de nouveau notable que le modèle d'estimation caractérise correctement la réduction en consommation énergétique apportée par la quantification en INT8. L'erreur entre l'estimation et la mesure est de 1,26% pour le STM32L496ZG, 5,90% pour le STM32F446ZE et 0,28% pour le STM32F746ZG.

TABLE 4.9 – Réduction en consommation énergétique de LeNet5 en INT8 par rapport à LeNet5 en FP32.

MCU	Réduction mesurée	Réduction estimée	Différence
STM32L496ZG	8,56%	7,30%	1,26%
STM32F446ZE	22,97%	17,07%	5,90%
STM32F746ZG	18,90%	19,18%	0,28%

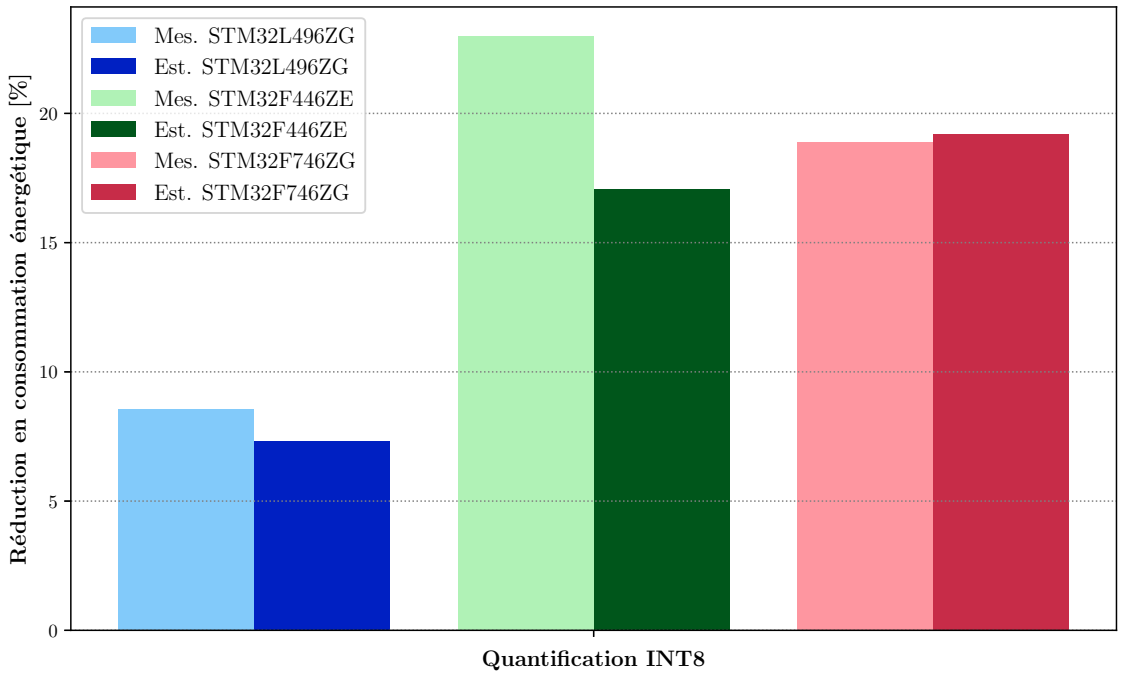


FIGURE 4.14 – Réduction en consommation d'énergie estimée et mesurée de LeNet5 en INT8 par rapport à LeNet5 en FP32.

Estimation de l'espace mémoire :

La comparaison entre l'estimation et la mesure de l'espace mémoire Flash pour implémenter LeNet5 est présentée sur le Tableau 4.10. Sur les 3 MCU, l'erreur

moyenne entre l'estimation de l'espace mémoire Flash et la mesure est de 1,5%. Pour chaque MCU la différence est de 1,52% pour le STM32L496ZG, 1,57% pour le STM32F446ZE et 1,43% pour le STM32F746ZG. Le gain en espace mémoire obtenu avec une quantification d'un format FP32 vers un format INT8 est d'environ un facteur 4. Cette valeur correspond bien au passage d'un format binaire nécessitant 4 octets vers un format binaire nécessitant qu'un seul octet.

TABLE 4.10 – Estimation de l'espace mémoire Flash (Code+Poids)

MCU	**Mémoire (Code+Poids) estimée [ko]**	**Mémoire (Code+Poids) mesurée [ko]**	**Différence [%]**
STM32L496ZG	63,90	62,94	1,52%
STM32F446ZE	63,90	62,92	1,57%
STM32F746ZG	63,90	63,00	1,43%

Le Tableau 4.11 présente l'estimation de la mémoire RAM nécessaire à l'inférence de LeNet5 en INT8. Cette estimation est basée sur le même nombre de primitives par rapport à l'estimation pour le réseau en FP32, le nombre de primitives ne changeant pas lors d'une quantification. Cependant, il est nécessaire de multiplier le nombre de primitives par le nombre d'octets du format binaire INT8, à savoir 1 octet. L'espace RAM minimum nécessaire à une inférence de LeNet5 en INT8 est ainsi de $(4704 + 1176) \times 1 = 5880$ octets, soit 5,88 ko, ce qui correspond à une réduction de 75% par rapport au réseau non quantifié en FP32. Cet espace mémoire RAM minimum correspond à l'exécution de la couche $\mathcal{L}2$ de LeNet5.

TABLE 4.11 – Estimation de l'espace mémoire RAM nécessaire au stockage des calculs intermédiaires de LeNet5 en format binaire INT8.

Couche	**Primitive à considérer**	**Primitives en entrée**	**Primitives en sortie**	**Total primitives**	**Total mémoire [ko]**
$\mathcal{L}1$	Normalisation	NA	4 704	4 704	4,70
$\mathcal{L}2$	Sous-échantillonnage	4 704	1 176	5 880	5,88
$\mathcal{L}3$	Normalisation	1 176	1600	2 776	2,78
$\mathcal{L}4$	Sous-échantillonnage	1600	400	2 000	2,00
$\mathcal{L}5$	Normalisation	400	120	520	0,52
$\mathcal{L}6$	Normalisation	120	84	204	0,20
$\mathcal{L}7$	Normalisation	84	10	94	0,94

Synthèse de l'estimation du coût d'inférence d'un CNN quantifié :

Le modèle d'estimation basé sur les primitives a été utilisé pour estimer le coût d'inférence de LeNet5 quantifié en format binaire INT8. Les estimations présentées ont été comparées à des mesures. Les primitives ont été légèrement adaptées pour tenir compte des coefficients de quantification nécessaires à une inférence en format binaire INT8. Sur les 3 MCU et les 14 fréquences de fonctionnement des MCU, le modèle d'estimation présente une précision moyenne de 94,2% pour estimer la latence, 90,68% pour estimer la consommation énergétique et 98,5% pour estimer l'espace mémoire nécessaire à une inférence d'un CNN. La précision du modèle d'estimation

passe de 90,68% pour estimer la consommation énergétique à 94,33% pour estimer la puissance lors d'une inférence. Finalement, la réduction du coût d'inférence apportée par la quantification est correctement estimée par le modèle, la réduction estimée étant proche de la réduction mesurée. La quantification réduisant directement la taille du format binaire représentant les données, la réduction de l'espace mémoire Flash et RAM nécessaire à une inférence en INT8 est significative, 4 fois moins qu'un format binaire en FP32. La réduction en latence et en consommation énergétique est plus modérée, de l'ordre de 10% à 20%.

4.4.2 LeNet5 élagué

Cette partie présente les résultats sur LeNet5 élagué à 50%, 60%, 70%, 80% et 90% de ces filtres de convolution. Chaque réseau élagué a été implémenté au sein des MCU pour mesures, et ainsi valider les estimations. Le format binaire utilisé est le format FP32. Les primitives étant inchangées et déjà caractérisées sur les MCU, l'estimation est faisable immédiatement. Les résultats présentés en suivant sont également comparés au réseau sans modification (Élagage 0%), c'est à dire non élagué. Ainsi, l'impact de l'élagage peut être clairement visible. Un total de 12 courbes, 6 estimations et 6 mesures sont représentées sur les figures suivantes. Les résultats sont donc présentés par MCU, pour une meilleure visibilité des données. De plus, pour améliorer la lecture des résultats, les graphiques de latence et de consommation énergétique concernant le STM32F446ZE et le STM32F746ZG sont présents en Annexe B.1.

Le nombre de primitives utilisées est directement impacté par le taux d'élagage appliqué au CNN. La technique utilisée impacte les filtres de convolution. De ce fait, le nombre de primitives *Convolution 2D* du Tableau 4.6 est directement impacté. Les primitives *Regroupement par valeur moyenne* et *ReLU* sont indirectement impactées. Le Tableau 4.12 détaille le nombre de primitives à prendre en compte pour l'estimation de la latence, de la consommation énergétique et de l'espace mémoire Flash et RAM en fonction du taux d'élagage appliqué.

TABLE 4.12 – Décomposition en primitives de LeNet5 en fonction du taux d'élagage appliqué, 50%, 60%, 70%, 80%, 90%.

Primitive	**Nbr. de primitives en fonction du taux d'élagage**					
	0%	50%	60%	70%	80%	90%
Convolution 2D	14 304	4 752	4 452	2568	2 368	984
Regroupement par valeur moyenne	1 576	813	763	517	492	246
Entièrement connectée	1	1	1	1	1	1
Entièrement connectée	1	1	1	1	1	1
Entièrement connectée	1	1	1	1	1	1
ReLU	6 508	3 356	3 256	2 262	2 172	1 188
Softmax	1	1	1	1	1	1

Le Tableau 4.13 indique la précision en classification des LeNet5 élagués. Le réseau LeNet5 non élagué avec la fonction d'activation ReLU a une précision sur la base MNIST de 99,02%. L'élagage diminue légèrement la précision en classification, dans le cas de l'élagage le plus agressif, 90%, la précision en classification de LeNet5 est de 98,25%

TABLE 4.13 – Précision en classification de LeNet5 sur la base MNIST, en fonction du taux d'élagage appliqué, 50%, 60%, 70%, 80%, 90%.

LeNet5 à taux d'élagage variable	Précision en classification
LeNet5 0%	99,02%
LeNet5 50%	98,93%
LeNet5 60%	98,18%
LeNet5 70%	98,67%
LeNet5 80%	98,84%
LeNet5 90%	98,25%

Estimation de la latence :

La comparaison entre l'estimation et la mesure de la latence de LeNet5 en FP32 aux différents pourcentages d'élagage est présentée sur la Figure 4.15 et en échelle logarithmique sur la Figure 4.16 pour le STM32L496ZG. Concernant les Figure B.1 et B.2 pour le STM32F446ZE et Figure B.3 et B.4 pour le STM32F746ZG, elles sont présentées en Annexe B.1.

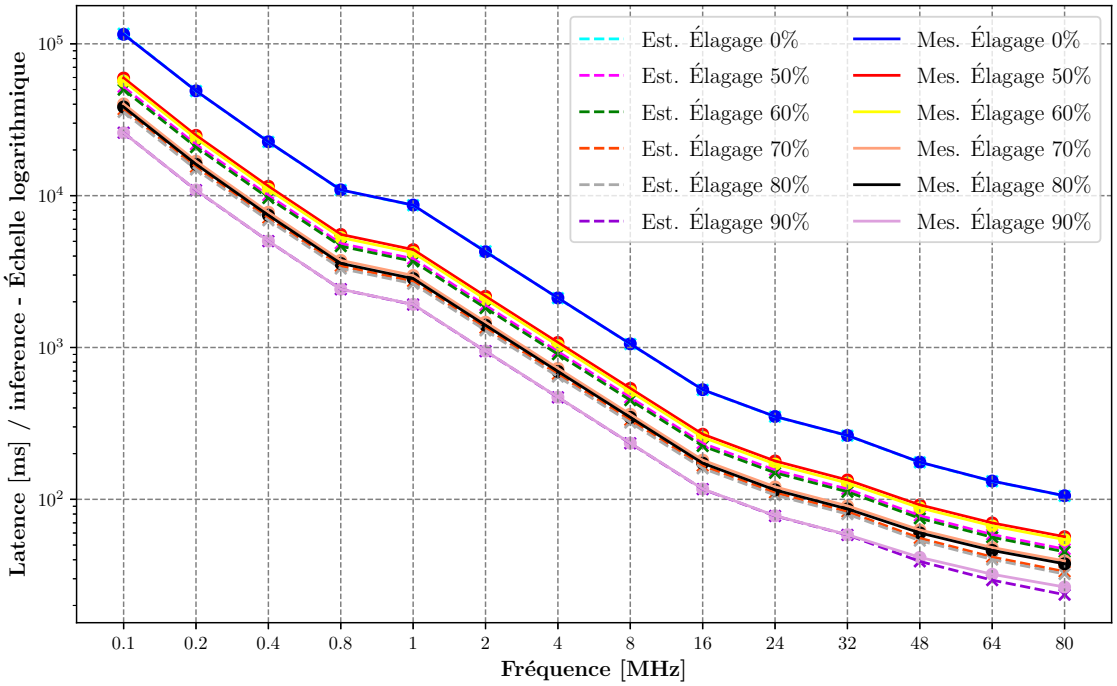


FIGURE 4.15 – Estimations *vs.* mesures de la latence de LeNet5 en FP32 au sein du STM32L496ZG. Élagage à : 0% (cyan *vs.* bleu), 50% (rose *vs.* rouge), 60% (vert *vs.* jaune), 70% (orange *vs.* saumon), 80% (gris *vs.* noir) et 90% (violet foncé vs violet clair). Échelle logarithmique sur l'axe Y.

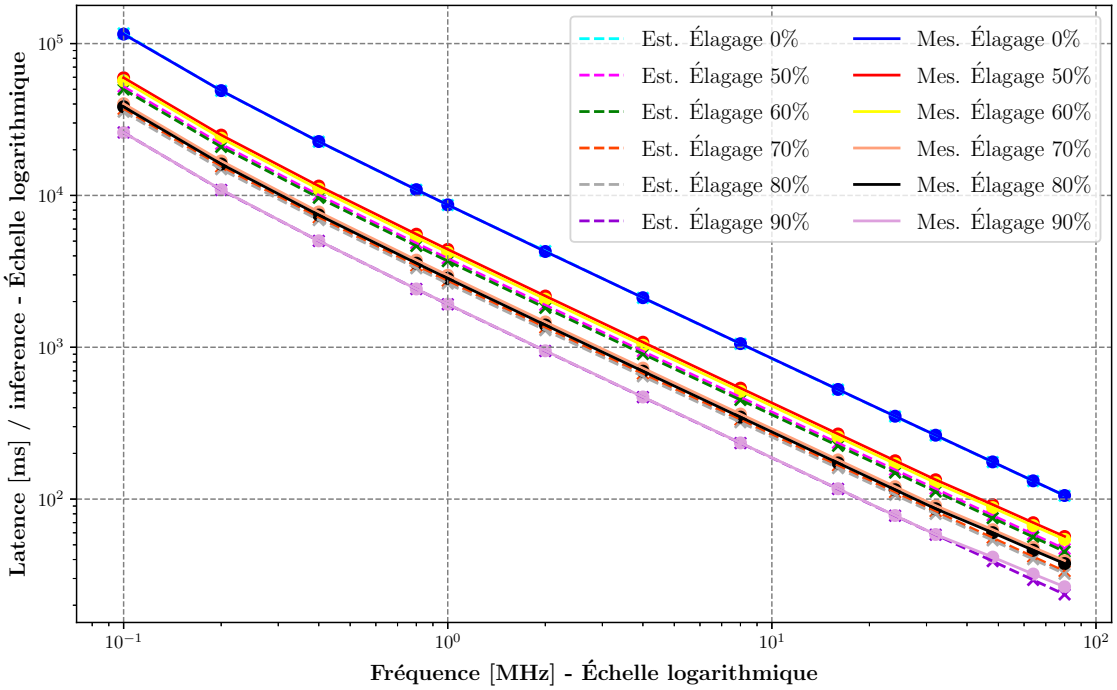


FIGURE 4.16 – Estimations *vs.* mesures de la latence de LeNet5 en FP32 au sein du STM32L496ZG, échelle logarithmique.

Sur la Figure 4.15 la réduction de la latence en fonction du taux d'élagage par rapport au réseau non élagué est clairement visible. L'élagage à 50% et 60% offre une réduction en latence du même ordre de grandeur par rapport au réseau non élagué. Cela est cohérent par rapport au nombre de primitives référencées dans le Tableau 4.12. La réduction en latence est plus prononcée sur le second palier majeur d'élagage pour LeNet5, à 70% et 80%. Finalement, 90% d'élagage offre la réduction en latence la plus prononcée.

Le Tableau 4.14 référence, en moyenne sur les 14 fréquences des MCU, le gain en % entre LeNet5 non élagué et LeNet5 élagué aux différents taux d'élagage. La représentation graphique de ces résultats est présentée sur la Figure 4.17. La réduction en latence obtenue est de l'ordre de 55-60% pour un élagage de 50% et 60% des filtres de convolution. Avec un taux d'élagage de 60-70% la réduction en latence obtenue est de l'ordre de 65-70%. Finalement, avec un taux d'élagage de 90%, la réduction est d'environ 80%. Il est remarquable que la réduction en latence grâce à l'élagage soit estimée avec précision en utilisant le modèle d'estimation. La réduction en latence estimée est en effet proche de la réduction en latence mesurée, avec une différence moyenne de 2,33%.

TABLE 4.14 – Réduction en latence en fonction du taux d'élagage appliqué à LeNet5 par rapport à LeNet5 non élagué. La comparaison est faite entre la réduction de latence estimée (*Est.*) et mesurée (*Mes.*) et est répertoriée dans la colonne différence (*Diff.*).

Taux d'élagage comparé à 0%	**Réduction sur le STM32L496ZG [%]**			**Réduction sur le STM32F446ZE [%]**			**Réduction sur le STM32F746ZG [%]**		
	Mes.	Est.	Diff.	Mes.	Est.	Diff.	Mes.	Est.	Diff.
50%	48,6	55,5	+6,9	57,2	55,8	-1,4	58,2	57,4	-0,8
60%	50,9	57,3	+6,4	59,1	57,7	-1,4	60,1	59,2	-0,8
70%	65,1	68,4	+3,2	70,7	68,9	-1,8	71,9	70,5	-1,4
80%	66,7	69,5	+2,8	71,9	70,0	-1,9	73,1	71,7	-1,4
90%	77,3	77,8	+0,4	80,6	78,4	-2,2	81,9	80,0	-1,9

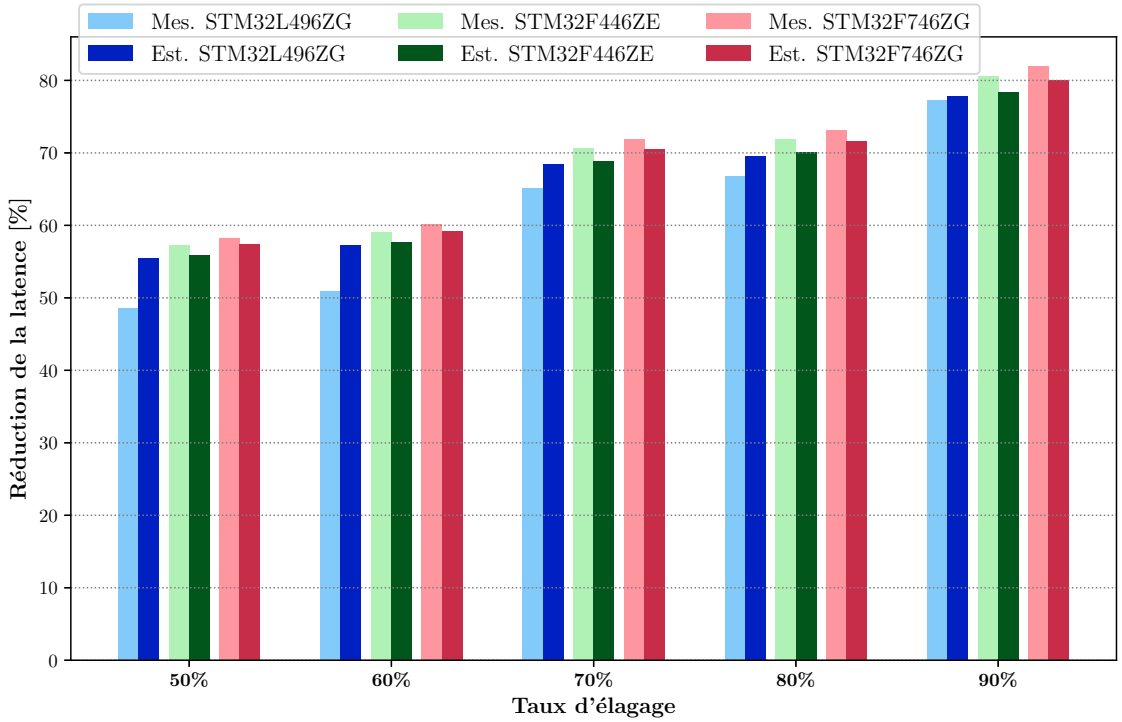


FIGURE 4.17 – Réduction en latence estimée et mesurée en fonction du taux d'élagage appliqué à LeNet5 par rapport à LeNet5 non élagué. Le modèle d'estimation caractérise avec précision la réduction possible liée à l'application des différents taux d'élagage. La différence moyenne avec la mesure est de 2,33%.

Estimation de la consommation d'énergie :

La comparaison entre l'estimation et la mesure de la consommation d'énergie pour une inférence de LeNet5 en FP32 et aux différents pourcentages d'élagage est présentée sur la Figure 4.18 et 4.19 pour le STM32L496ZG. Les Figure B.5 et B.6 pour le STM32F446ZE et les Figure B.7 et B.8 pour le STM32F746ZG sont présentées en Annexe B.1.

Sur le STM32L496ZG, LeNet5 élagué à 90% offre la réduction énergétique la plus importante par rapport au réseau non élagué, en moyenne sur les 14 fréquences de fonctionnement, une réduction de 77% de joule par inférence, d'après l'estimation et la mesure. Sur la Figure 4.18, est visible la réduction énergétique apportée par

les différents taux d'élagage au sein du STM32L496ZG. Comme la première estimation sur LeNet5 en FP32 non élagué l'a montré, les fréquences de fonctionnement offrant la plus faible consommation énergétique ne sont pas les basses fréquences. Concernant le STM32L496ZG, conçu spécifiquement pour la faible consommation, les fréquences de fonctionnement de fin de plage de fréquence, 24MHz à 80MHz, offrent la consommation énergétique la plus faible sur tous les taux d'élagage. La différence entre l'estimation et la mesure de la consommation varie entre 1,17% et 22,74% sur toutes les configurations d'élagage et toutes les fréquences. Cette forte disparité est due majoritairement à la multiplication de la puissance estimée par la latence estimée pour obtenir l'énergie consommée estimée. L'erreur amenée par l'estimation de la latence se combine et l'erreur sur la consommation d'énergie augmente en conséquence pour certains points de mesures.

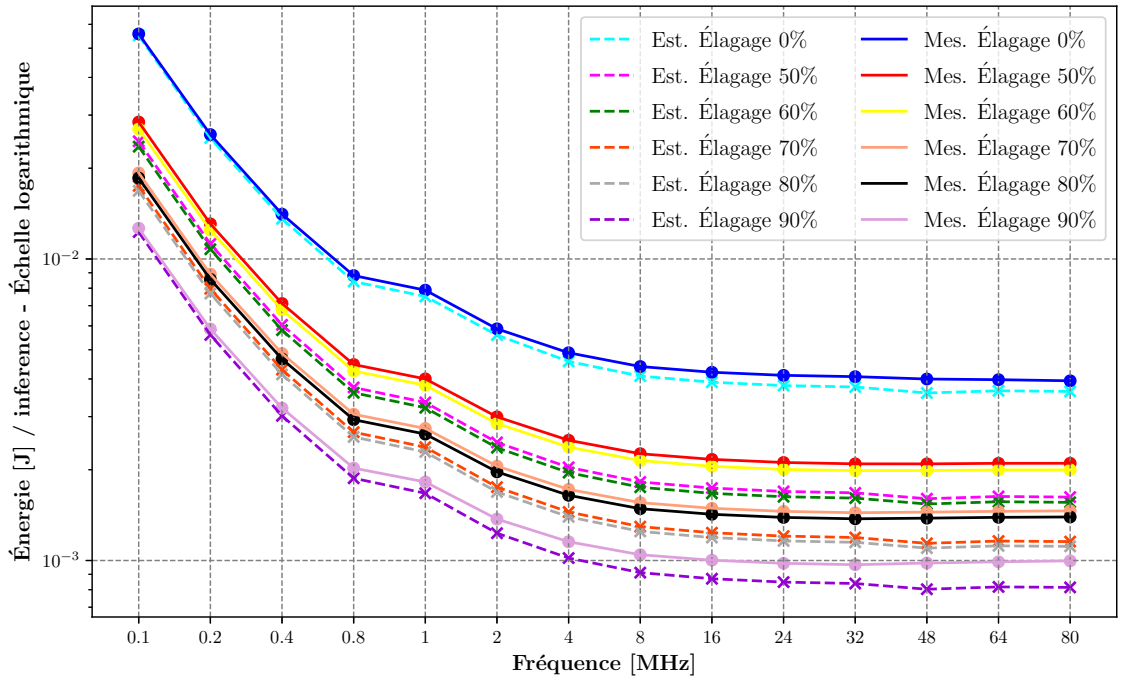


FIGURE 4.18 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32L496ZG. Élagage à : 0% (cyan *vs.* bleu), 50% (rose *vs.* rouge), 60% (vert *vs.* jaune), 70% (orange *vs.* saumon), 80% (gris *vs.* noir) et 90% (violet foncé vs violet clair). Échelle logarithmique sur l'axe Y.

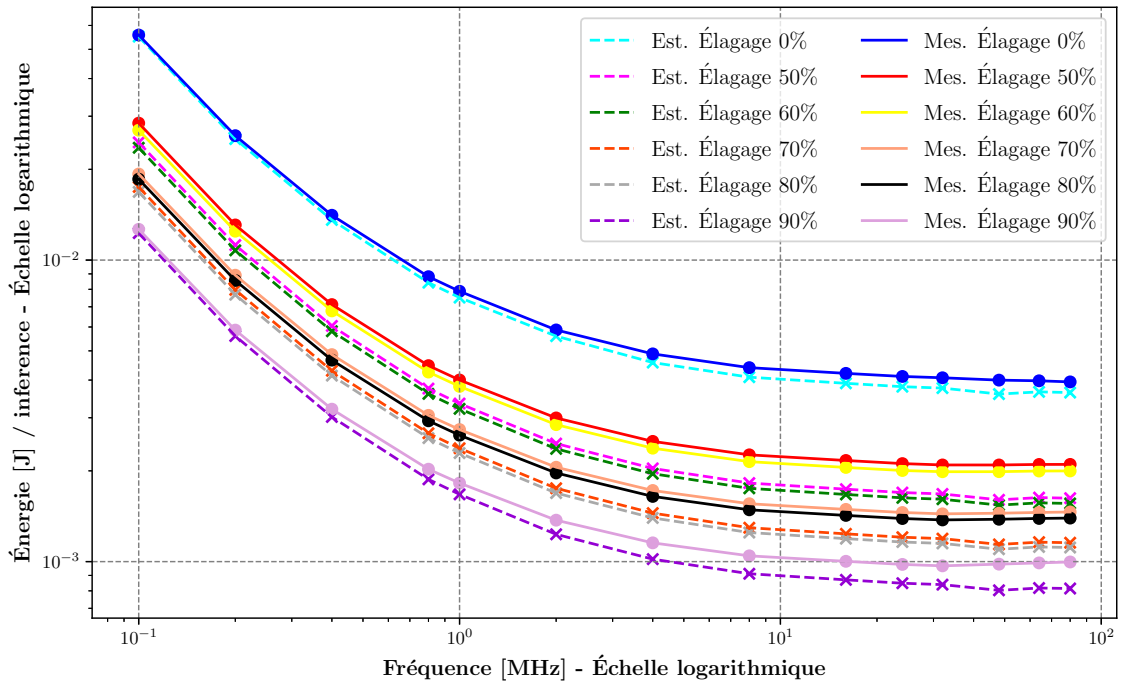


FIGURE 4.19 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32L496ZG, échelle logarithmique.

Pour mettre en avant l'influence de l'erreur de la latence dans l'estimation de la consommation d'énergie sur le STM32L496ZG, le Tableau 4.15 indique la différence entre l'estimation et la mesure de la puissance, de l'énergie et de la latence. L'erreur d'estimation de la consommation énergétique d'une inférence est comprise entre 11% et 18% pour les réseaux élagués. Concernant l'erreur d'estimation de la puissance, elle est comprise entre 5,72% et 9,42%. La différence en consommation énergétique est la plus importante là où la différence en latence est la plus importante. Avec un taux de 50% d'élagage, la différence en latence est de 13,58%. La différence en consommation d'énergie augmente en conséquence, passant à 18,51%, alors que la différence en puissance est de 5,72%.

TABLE 4.15 – Comparaison entre l'estimation de la puissance et de l'énergie consommée pour une inférence de LeNet5 à taux d'élagage varié au sein du STM32L496ZG. La comparaison est faite entre la différence des estimations et des mesures de puissance et de consommation d'énergie sur une inférence. La différence de l'estimation de la latence est également référencée, l'énergie étant directement liée à ce paramètre.

Taux d'élagage	**STM32L496ZG**		
	Diff. Puissance [%]	Diff. Énergie [%]	Diff.Latence [%]
0%	5,77%	5,92%	0,41%
50%	5,72%	18,51%	13,58%
60%	5,28%	17,66%	13,08%
70%	6,90%	15,65%	9,41%
80%	7,07%	15,02%	8,57%
90%	9,42%	11,23%	2,04%

La réduction de la consommation énergétique au sein des trois MCU, apportée par l'élagage en comparaison avec le réseau non élagué, est indiquée dans le Ta-

bleau 4.16 et illustrée sur la Figure 4.20. Il est remarquable que la réduction de consommation d'énergie soit du même ordre de grandeur entre les MCU aux différents taux d'élagage. Cela correspond au comportement observé pour la réduction en latence, la latence d'une inférence impactant directement sa consommation d'énergie. Le modèle d'estimation est de nouveau capable de caractériser avec précision la réduction en consommation énergétique apportée par l'élagage. La différence entre la réduction estimée et mesurée est en moyenne de 2,82%.

TABLE 4.16 – Réduction en consommation énergétique en fonction du taux d'élagage appliqué à LeNet5 par rapport à LeNet5 non élagué. La comparaison est faite entre la réduction en consommation énergétique estimée (*Est.*) et mesurée (*Mes.*) et est répertoriée dans la colonne différence (*Diff.*).

Taux d'élagage comparé à 0%	**Réduction sur le STM32L496ZG [%]**			**Réduction sur le STM32F446ZE [%]**			**Réduction sur le STM32F746ZG [%]**		
	Mes.	Est.	Diff.	Mes.	Est.	Diff.	Mes.	Est.	Diff.
50%	48,6	55,5	+6,9	55,8	55,8	-0,1	59,1	56,2	-2,9
60%	51,1	57,2	+6,1	60,7	57,6	-3,1	61,5	57,9	-3,5
70%	64,7	68,4	+3,7	69,3	68,8	-0,4	72,9	69,2	-3,7
80%	66,2	69,5	+3,3	70,2	70,0	-0,2	73,6	70,4	-3,3
90%	76,4	77,7	+1,4	78,1	78,4	+0,3	82,3	78,8	-3,5

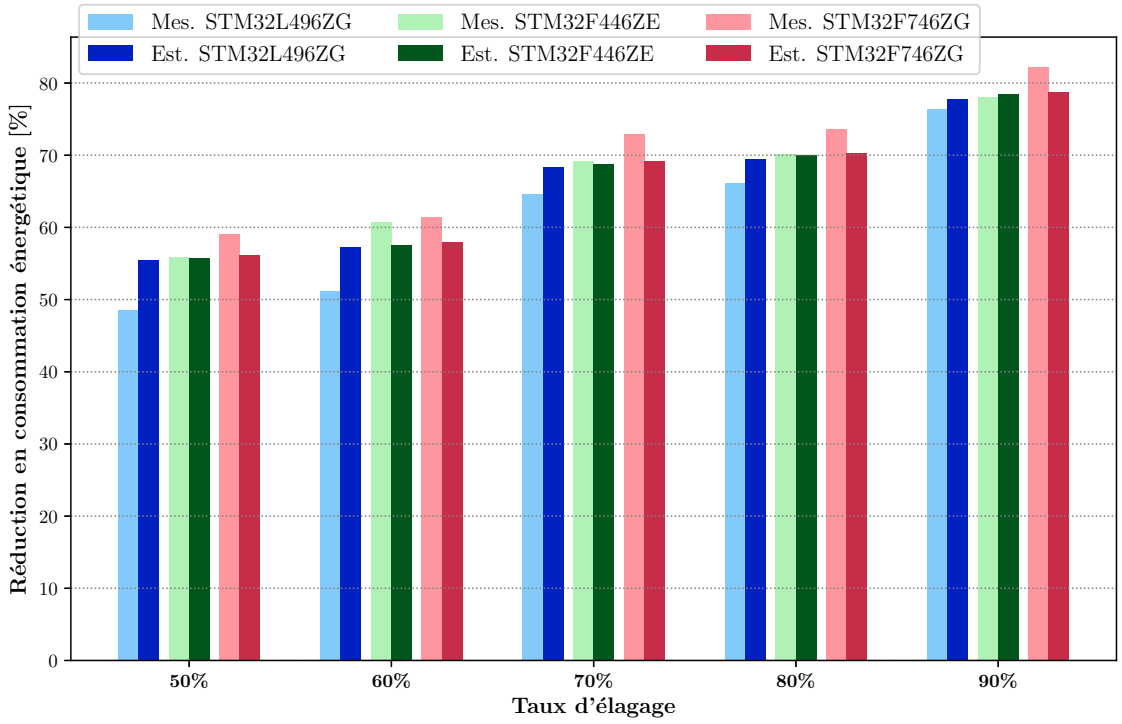


FIGURE 4.20 – Réduction en consommation énergétique estimée et mesurée en fonction du taux d'élagage. Le modèle d'estimation caractérise avec précision la réduction en consommation énergétique liée à l'application des différents taux d'élagage. La différence moyenne avec la mesure est de 2,82%.

Estimation de l'espace mémoire :

L'estimation de l'espace mémoire est effectuée pour l'espace mémoire Flash permettant de stocker les poids et le code de LeNet5 ainsi que l'espace mémoire RAM pour stocker les calculs intermédiaires. La majorité des poids de LeNet5 sont dans les dernières couches entièrement connectées. La technique d'élagage appliquée dans ces travaux est effective sur les filtres de convolution. Ainsi, la réduction en espace mémoire Flash n'est pas significative. Cependant le modèle d'estimation caractérise correctement cette réduction d'espace mémoire Flash, comme illustré sur le Tableau 4.17. L'espace mémoire Flash des réseaux LeNet5 élagué est mis en relief par rapport à l'espace mémoire Flash nécessaire au réseau non élagué. Une dizaine de kilooctets sépare les besoins en mémoire Flash de LeNet5 non élagué et de LeNet5 élagué à 90%. Sur l'ensemble des réseaux élagués, la différence entre l'estimation et la mesure est inférieure à 1%. L'estimation met ainsi en avant que cette technique d'élagage réduit très peu l'espace mémoire Flash si elle est appliquée sur le réseau LeNet5.

TABLE 4.17 – Estimation de l'espace mémoire Flash en fonction du taux d'élagage

MCU	Taux d'élagage	Mémoire Flash estimée [ko]	Mémoire Flash mesurée [ko]	Différence [%]
STM32L496ZG	0%	248,31	254,02	2,25%
	50%	238,86	239,85	0,41%
	60%	238,51	239,55	0,43%
	70%	237,12	238,35	0,52%
	80%	236,88	238,15	0,53%
	90%	236,08	237,45	0,58%
STM32F446ZE	0%	248,31	253,24	1,95%
	50%	238,86	236,82	0,40%
	60%	238,51	239,52	0,42%
	70%	237,12	238,24	0,50%
	80%	236,88	238,13	0,52%
	90%	236,07	237,42	0,57%
STM32F746ZG	0%	248,31	258,44	3,92%
	50%	238,86	239,90	0,44%
	60%	238,51	239,60	0,46%
	70%	237,12	238,41	0,54%
	80%	236,89	238,21	0,55%
	90%	236,08	237,51	0,60%

L'estimation de l'espace mémoire RAM est illustrée sur le Tableau 4.18. L'analyse de l'espace mémoire RAM nécessaire à l'exécution de chaque couche est faite en fonction du taux d'élagage. La comparaison est également faite en fonction de LeNet5 non élagué. Il est remarquable que la couche $\mathcal{L}\mathbf{2}$ reste celle ayant le plus fort besoin en espace mémoire. En effet, le nombre de primitives mises en jeu pour LeNet5 élagué à 50% et 60% est de 2 940. Multiplié par le nombre d'octets utilisés dans le format binaire FP32, à savoir 4, cela résulte en espace mémoire RAM nécessaire de $2940 \times 4 = 11,760ko$. Concernant un élagage à 70% et 80%, 7,84ko de mémoire RAM sont nécessaires. Finalement, 3,92ko de mémoire RAM sont nécessaires à l'inférence de LeNet5 dans le cas ou un élagage de 90% est appliqué.

TABLE 4.18 – Estimation de l'espace mémoire RAM nécessaire au stockage des calculs intermédiaires.

Taux d'élagage		**Couche**						
		$\mathcal{L}$1	**$\mathcal{L}$2**	**$\mathcal{L}$3**	**$\mathcal{L}$4**	**$\mathcal{L}$5**	**$\mathcal{L}$6**	**$\mathcal{L}$7**
Élagage 0%	Primitives en entrée	NA	4 704	1 176	1 600	400	120	84
	Primitives en sortie	4 704	1 176	1 600	400	120	84	10
	Total Primitives	4 704	5 880	2 776	2 000	520	204	94
	Total Mémoire [ko]	18,82	23,52	11,10	8,00	2,08	0,82	0,38
Élagage 50%	Primitives en entrée	NA	2 352	588	800	400	120	84
	Primitives en sortie	2 352	588	800	225	120	84	10
	Total Primitives	2 352	2 940	1 388	1 025	520	204	94
	Total Mémoire [ko]	9,41	11,76	5,55	4,10	2,08	0,82	0,38
Élagage 60%	Primitives en entrée	NA	2 352	588	700	400	120	84
	Primitives en sortie	2 352	588	700	175	120	84	10
	Total Primitives	2 352	2 940	1 288	875	520	204	94
	Total Mémoire [ko]	9,41	11,76	5,15	3,50	2,08	0,82	0,38
Élagage 70%	Primitives en entrée	NA	1 568	392	500	400	120	84
	Primitives en sortie	1 568	392	500	125	120	84	10
	Total Primitives	1 568	1 960	892	625	520	204	94
	Total Mémoire [ko]	6,27	7,84	3,57	2,50	2,08	0,82	0,38
Élagage 80%	Primitives en entrée	NA	1 568	392	400	400	120	84
	Primitives en sortie	1 568	392	400	100	120	84	10
	Total Primitives	1 568	1 960	792	500	520	204	94
	Total Mémoire [ko]	6,27	7,84	3,17	2,00	2,08	0,82	0,38
Élagage 90%	Primitives en entrée	NA	784	196	200	400	120	84
	Primitives en sortie	784	196	200	50	120	84	10
	Total Primitives	784	980	396	250	520	204	90
	Total Mémoire [ko]	3,14	3,92	1,58	1	2,08	0,82	0,38

L'élagage des filtres de convolution a peu d'impact sur l'espace mémoire Flash nécessaire à l'inférence de LeNet5. Cependant l'impact sur l'espace mémoire RAM est significatif, comme le montre le Tableau 4.19. Pour le taux d'élagage le plus important, à savoir 90% des filtres de convolution, l'espace RAM sauvegardé est de 83,33% par rapport à LeNet5 non élagué.

TABLE 4.19 – Réduction en espace mémoire RAM nécessaire à une inférence en fonction du taux d'élagage appliqué à LeNet5 par rapport à LeNet5 non élagué.

Taux d'élagage comparé à 0%	**Réduction [%]**
50%	50%
60%	50%
70%	66,67%
80%	66,67%
90%	93,33%

Synthèse de l'estimation du coût d'inférence d'un CNN élagué :

L'élagage a été appliqué à LeNet5, à 5 taux différents : 50%, 60%, 70%, 80% et 90% des filtres de convolution. Sur les 3 MCU et les 14 fréquences de fonctionnement, les estimations du coût d'inférence ont été comparées à des mesures. Cela représente un total de 210 comparaisons estimation/mesure. Le modèle d'estimation présente une précision moyenne de 88,2% pour estimer la latence, 85,2% pour estimer la consommation énergétique et 99,5% pour estimer l'espace mémoire nécessaire à une inférence d'un CNN. La précision du modèle d'estimation concernant la consommation énergétique est influencée par la précision du modèle d'estimation en latence. En effet, la précision du modèle d'estimation passe de 85,2% pour estimer la consommation énergétique à 91,25% pour estimer la puissance lors d'une inférence. L'ensemble de ces estimations a été réalisé avec le même modèle, les mêmes codes et le même processus de mesure. Finalement, la réduction du coût d'inférence apportée par la technique d'élagage est fidèlement caractérisée par le modèle d'estimation. La réduction annoncée par l'estimation est toujours proche de la réduction mesurée avec une différence de l'ordre de 2%. Cette technique d'élagage est effective sur les filtres de convolution. La majorité de l'espace mémoire Flash requis par LeNet5 est dans les couches entièrement connectées, la réduction en espace mémoire Flash est donc faible. Cependant, la réduction en espace mémoire RAM nécessaire à une inférence est de 83,33% dans le cas d'un élagage à 90%. Concernant la latence et la consommation d'énergie, la réduction suite à l'élagage est de l'ordre de 80% si l'élagage appliqué est de 90% des filtres de convolution.

4.5 Synthèse

Ce chapitre a présenté la mise en œuvre du modèle d'estimation basé sur les primitives. Grâce à ce modèle, le coût d'inférence du CNN LeNet5 a été caractérisé. La latence, la consommation énergétique et l'espace mémoire ont été quantitativement estimés avec le même modèle. Le coût d'inférence de LeNet5 non réduit, et réduit avec des techniques de quantification et d'élagage, a été estimé et comparé a des mesures. Chaque configuration de LeNet5 a été implémentée sur les 3 MCU, puis exécutée à 14 fréquences différentes. Sur l'ensemble des mesures du coût d'inférence effectuées, à savoir 336, le modèle d'estimation est précis à 90,19% concernant la latence et à 87,42% concernant la consommation énergétique. La latence estimée impacte directement la consommation énergétique estimée. L'estimation de la puissance consommée lors d'une inférence est dès lors plus précise, passant à 92,41% de précision. L'estimation du coût d'inférence par rapport aux fréquences de fonctionnement des MCU permet de mettre en évidence la fréquence la plus efficace, équilibrant au mieux la latence et la consommation énergétique. Le nombre de mesures concernant l'espace mémoire est de 24 et le modèle d'estimation a une précision de 98,82%. Ces informations quantitatives sont essentielles pour la mise en œuvre de la méthodologie ZIP-CNN. En comparaison avec l'état-de-l'art, le modèle présenté durant ce chapitre présente plusieurs avantages. Le modèle Roofline [119] et les estimateurs utilisés dans les approches NAS [50], [52]-[54] se focalisent sur l'estimation du temps d'exécution du CNN et/ou de l'espace mémoire associé. Aucune approche ne fournit avec le même modèle une estimation de la latence, de la consommation énergétique et de l'espace mémoire Flash et RAM nécessaires à une inférence. De plus, aucun modèle ne fournit une estimation de la réduction apportée par les techniques de réduction telles que l'élagage ou la quantification. Ces techniques sont centrales pour embarquer des CNN. Une implémentation du modèle réduit suivi d'une phase de mesure est nécessaire pour valider l'efficacité de la réduction. Ce processus est chronophage et n'offre aucune garantie. En comparaison, le modèle d'estimation introduit dans ce chapitre, assure une estimation de la latence, de la consommation énergétique et de l'espace mémoire Flash et RAM. Ces estimations sont fiables dans le cas où le CNN est décrit dans le format par défaut des outils d'apprentissage profond, c'est à dire en FP32, mais également dans le cas d'application de technique de quantification et d'élagage. Finalement, une approche d'estimation basée sur la mesure de métriques perceptibles est perçue comme un processus contraignant. Notre approche assure une estimation basée sur des métriques perceptibles et non sur des *proxys*. Ce processus est en grande partie automatisé et offre une caractérisation efficace de nouveaux MCU. Finalement, ce modèle d'estimation permet d'estimer efficacement si un CNN peut être embarqué sans réduction, et, sinon, quelle est la réduction du coût d'inférence apportée par les techniques de réduction de l'état-de-l'art. Au sein de la méthodologie ZIP-CNN, le modèle d'estimation permet une exploration rapide et efficace de l'espace des solutions. Une illustration de cette exploration est présentée dans le chapitre suivant, basé sur un CNN de l'état-de-l'art, ResNet.

Chapitre 5

Cas d'études de la méthodologie ZIP-CNN

Sommaire

5.1 Introduction

Ce chapitre a pour objectif d'illustrer les possibilités fournies par la méthodologie ZIP-CNN à un concepteur de système embarqué. La méthodologie est mise en œuvre pour explorer l'espace des solutions concernant l'intégration d'un réseau de neurones de l'état-de-l'art : ResNet. Avec cent fois plus de primitives de convolutions, ce réseau est bien plus complexe que le réseau LeNet5 utilisé au cours du chapitre précédent. L'étude de l'implémentation de ResNet est réalisée sur les contraintes de latence, de consommation énergétique et d'espace mémoire. Les contraintes de latence et de consommation énergétique sont propres à chaque application. La contrainte d'espace mémoire est universelle à toutes les applications embarquées. Sans un espace mémoire suffisant, l'implémentation du CNN ne peut se faire. L'intérêt d'utiliser la méthodologie ZIP-CNN pour un concepteur de système embarqué est d'être renseigné sur les contraintes d'espace mémoire, de latence et de consommation énergétique pour une inférence, sans aucun besoin d'implémentation ou d'application de techniques de réduction. Il pourra ainsi identifier, le cas échéant, la ou les solutions pour respecter les contraintes de son application.

5.2 Le réseau ResNet

La topologie des CNN de type ResNet est présentée dans l'état-de-l'art (*c.f. Section 2.2.1*). La représentation fonctionnelle du CNN ResNet8 est illustrée sur la Figure 5.1. Cette topologie correspond à ResNet avec 8 couches. Comme il est visible sur ce schéma, la topologie ResNet est plus complexe que le réseau LeNet5. LeNet5 comporte seulement deux couches de convolution et trois couches entièrement connectées. En comparaison, ResNet comporte huit couches de convolutions, des noyaux de convolution de taille $k = 3 \times 3$ et $k = 1 \times 1$, des opérations de *Batch Normalization* ou encore des connexions résiduelles.

Pour le cas d'études présenté dans ce chapitre, deux topologies de ResNet sont utilisées : ResNet8 et ResNet26. La topologie de ResNet26 comporte 26 couches de convolution. Le nombre de paramètres et la précision en classification obtenue avec un entrainement des réseaux ResNet8 et ResNet26 sont répertoriés dans le Tableau 5.1. La base de données CIFAR-10 a été utilisée pour l'entrainement de ces réseaux. Cette base de données est composée de 60 000 images couleur (RGB), de taille 32x32, réparties en 10 classes. Chaque classe comporte 6 000 images. Le jeu d'entrainement comporte 50 000 images, le jeu de test comporte 10 000 images.

Table 5.1 – Nombre de paramètres et précision en classification sur la base CIFAR-10 pour ResNet8 et ResNet26.

CNN	Nombre de paramètres	Précision en classification
ResNet26	372 330	90,54%
ResNet8	78 666	70,95%

ResNet est en partie composé de primitives déjà caractérisées précédemment comme la convolution ou la fonction d'activation ReLU. De nouvelles primitives ont été caractérisées, comme la connexion résiduelle ou la normalisation par lot - *Batch Normalization*. La décomposition en primitives des ResNet utilisés est référencée dans le Tableau 5.2

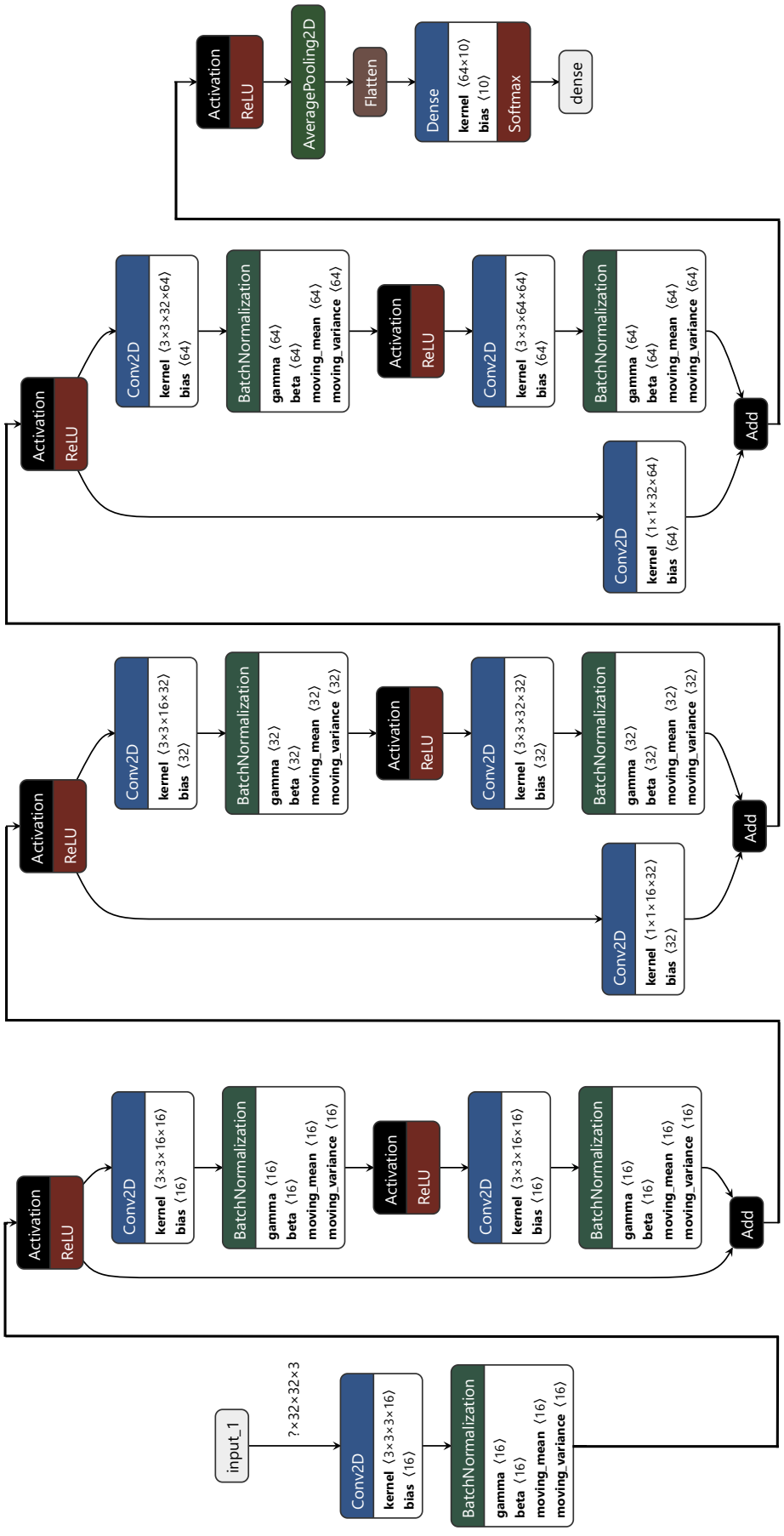


FIGURE 5.1 – Représentation fonctionnelle de ResNet8.

TABLE 5.2 – Décomposition en primitives de ResNet8 et ResNet26.

Famille de primitives	**Primitive**	**Nombre d'exécutions**	
		ResNet8	**ResNet26**
Extraction de caractéristiques	Convolution RGB $k = 3 \times 3$	49 152	49 152
Extraction de caractéristiques	Convolution 2D $k = 3 \times 3$	1 310 720	6 029 312
Extraction de caractéristiques	Convolution 2D $k = 1 \times 1$	262 144	262 144
Sous-échantillonnage	Regroupement par valeur moyenne $k = 8 \times 8$	64	64
Combinaison linéaire	Entièrement connectée 64/10	1	1
Normalisation	ReLU	73 728	245 760
Normalisation	Batch Normalization	73 728	245 760
Normalisation	Connexion résiduelle	28 672	114 688
Normalisation	Softmax 10/10	1	1

Le nombre de primitives augmente en conséquence par rapport à la profondeur du réseau ResNet. Les nouvelles primitives caractérisées pour estimer le coût d'inférence de la topologie ResNet sont :

- la convolution RGB
- la connexion résiduelle
- la batch normalization

Les algorithmes de ces nouvelles primitives sont détaillés en Annexe A.1. Les autres primitives utilisées au sein de ResNet ont déjà été caractérisées dans les MCU, pour les estimations du coût d'inférence de LeNet5. Il n'est donc pas nécessaire de réitérer l'étape de caractérisation, l'estimation à l'aide de ces primitives peut-être directement appliquée.

La suite de ce chapitre illustre l'utilisation de la méthodologie ZIP-CNN dans le cas où un concepteur de système embarqué souhaite implémenter une topologie ResNet dans les MCU utilisés précédemment.

5.3 Déploiement de ResNet

Ce cas d'étude décrit l'utilisation de la méthodologie pour guider un concepteur de système embarqué souhaitant implémenter la topologie ResNet. La contrainte fonctionnelle d'espace mémoire est la première qui doit être vérifiée. Elle est universelle à toutes les applications embarquées. Si cette contrainte n'est pas respectée, aucune implémentation du CNN n'est envisageable. Les contraintes de latence et de consommation énergétique peuvent ensuite être estimées. Ces contraintes sont directement liées à l'application embarquée cible. Elles sont à l'appréciation du concepteur du système. La méthodologie a pour rôle de fournir des détails sur l'impact des choix d'implémentation du CNN sur ces contraintes.

5.3.1 Estimation de l'espace mémoire de ResNet26

Basé sur le nombre de primitives référencées sur le Tableau 5.2, l'espace mémoire Flash estimé pour stocker le code et les poids de ResNet26 en format binaire FP32 est de 2,267Mo. Pour rappel, les MCU STM32L496ZG et STM32F746ZG ont 1Mo de mémoire Flash disponible. Concernant le STM32F446ZE, 512ko de mémoire Flash est disponible. Il est donc impossible d'embarquer ResNet26 sur ces 3 MCU à cause du manque d'espace mémoire Flash.

Il est nécessaire d'analyser l'espace mémoire requis par ResNet26 lorsqu'une technique de réduction lui est appliquée. La ou les techniques de réduction permettant de réduire suffisamment le besoin d'espace mémoire Flash de ResNet26 seront identifiées. Avant l'application d'une quelconque technique de réduction, le Tableau 5.3 détaille l'espace mémoire Flash nécessaire à ResNet26 dans le cas d'une quantification en INT8 et de l'élagage des filtres de convolution à différents pourcentages. Il est également envisageable d'appliquer la technique de distillation de connaissances en prenant ResNet26 comme enseignant, et une topologie de ResNet plus petite, par exemple ResNet8. Avant d'utiliser la distillation de connaissances, il est nécessaire d'estimer l'espace Flash nécessaire au CNN étudiant. Si le besoin d'espace mémoire Flash du CNN étudiant est trop important, la technique de distillation de connaissances seule ne pourra pas satisfaire la contrainte mémoire.

TABLE 5.3 – Espace mémoire Flash estimé nécessaire aux stockages des poids et du code des différentes configurations de ResNet26 réduit par la quantification en INT8, par l'élagage des filtres de convolution et par la distillation de connaissances.

Réduction	**Mémoire Flash estimée [ko]**
Quantification INT8	1 146,15
Élagage 50% FP32	642,56
Élagage 60% FP32	405,51
Élagage 70% FP32	245,85
Élagage 80% FP32	119,14
Élagage 90% FP32	41,16
Distillation vers ResNet8 FP32	494,74

Au regard de la contrainte d'espace mémoire Flash, la méthodologie donne les choix possibles au concepteur de système pour embarquer un ResNet, en partant de ResNet26. La quantification en INT8 et l'élagage de 50% des filtres de convolution ne suffiront pas à réduire suffisamment le besoin d'espace mémoire Flash pour passer en dessous des 512ko du STM32F446ZE. La quantification INT8 ne suffira pas à implémenter ResNet26 au sein du STM32L496ZG et du STM32F746ZG, mais les réductions liées à l'élagage et à la distillation seraient suffisantes. La Figure 5.2 résume les options qui s'offrent au concepteur, suite à l'utilisation de la première étape de la méthodologie. Pour la suite nous choisirons d'appliquer la distillation de connaissance de ResNet26 en tant qu'enseignant vers ResNet8 en tant qu'élève. Il s'agit d'un choix arbitraire et le concepteur de système embarqué pourrait tout aussi bien orienter son choix vers d'autres techniques de réduction, en fonction de ses contraintes ou de ses préférences.

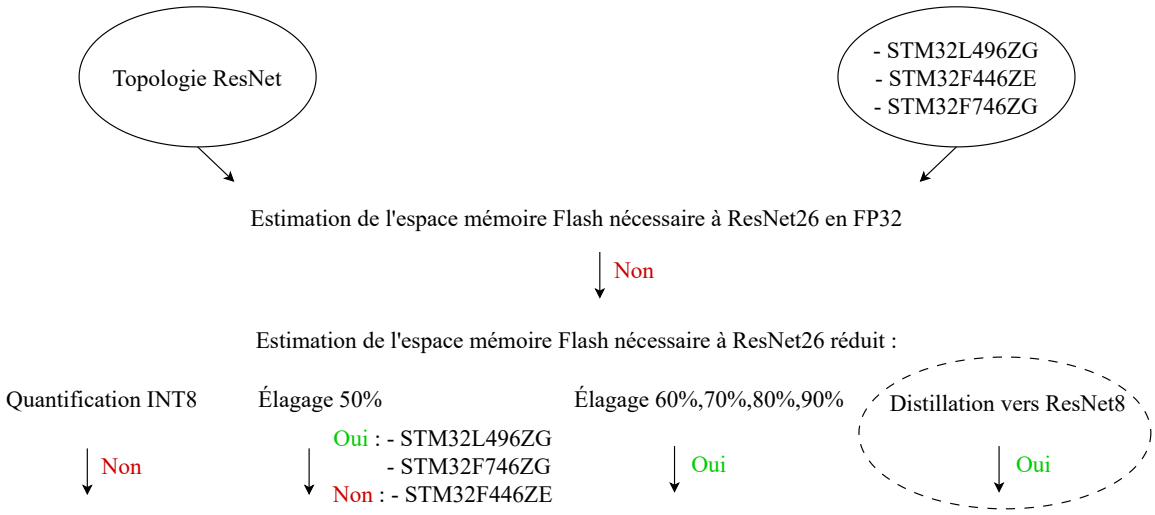


FIGURE 5.2 – Solutions possibles mises en avant par la méthodologie pour guider le choix des techniques de réduction pertinentes pour intégrer la topologie ResNet.

Avant d'appliquer la distillation de connaissances, il est nécessaire d'estimer la contrainte d'espace mémoire RAM de ResNet8. Une fois cette contrainte validée, la technique de réduction peut-être appliquée.

5.3.2 Estimation du coût d'inférence de ResNet8

Estimation de l'espace mémoire RAM :

Basé sur le nombre de primitives de ResNet8, l'espace mémoire Flash nécessaire au stockage du code et des poids est de 494,74ko. Cela signifie que ResNet8 peut être stocké dans les 3 MCU. L'estimation de l'espace mémoire RAM nécessaire à l'inférence de ResNet8 est faite sur le Tableau 5.4.

TABLE 5.4 – Estimation de l'espace mémoire RAM maximal nécessaire au stockage des calculs intermédiaires de ResNet8 en format binaire FP32.

Couche	Primitives en entrée	Primitives en sortie	Total primitives	Total mémoire [ko]
$\mathcal{L}1$	NA	16 384	16 384	65,536
$\mathcal{L}2$	16 384	16 384	32 768	131,072
$\mathcal{L}3$	16 384	16 384	32 768	131,072
$\mathcal{L}4$	32 768	16 384	49 152	196,608
$\mathcal{L}5$	16 384	8 192	24 576	98,304
$\mathcal{L}6$	8 192	8 192	16 384	65,536
$\mathcal{L}7$	16 384	8 192	24 576	98,304
$\mathcal{L}8$	16 384	8 192	24 576	98,304
$\mathcal{L}9$	8 192	4 096	12 288	49,152
$\mathcal{L}10$	4 096	4 096	8 192	32,768
$\mathcal{L}11$	8 192	4 096	12 288	49,152
$\mathcal{L}12$	8 192	4 096	12 288	49,152

La couche $\mathcal{L}4$ nécessite le plus d'espace RAM lors de l'inférence de ResNet8 en format binaire FP32. Cette couche correspond à une connexion résiduelle, prend en entrée la sortie de deux couches, correspondant à 32 768 primitives et génère en sortie

16 384 primitives. Le total de primitives mises en jeu dans cette couche est de 49 152, pour un espace mémoire RAM nécessaire en FP32 égale à $49152 \times 4 = 196,61ko$. L'espace mémoire RAM disponible sur le STM32L496ZG et STM32F746ZG est de 320ko. Une inférence de ResNet8 sur ces deux MCU est donc possible. Cependant, l'espace RAM disponible sur le STM32F446ZE est de 132ko, l'inférence de ResNet8 est donc impossible sur ce MCU.

L'estimation de l'espace mémoire Flash et RAM nécessaire à l'inférence de ResNet8 a montré que l'exécution de ce CNN est possible au sein du STM32L496ZG et du STM32F746ZG. L'utilisateur peut donc appliquer la distillation de connaissances du ResNet26 vers le ResNet8 s'il souhaite améliorer la précision en classification de ResNet8. Une augmentation de la base d'entraînement, à travers des rotations et des retournements horizontaux a été effectuée dans l'objectif de réduire la différence de précision entre ResNet8 et ResNet26. La différence de précision entre l'étudiant et l'élève est un paramètre important pour améliorer l'efficacité de la distillation de connaissances [95]. La précision en classification de ResNet8 est ainsi passée de 70,95% à 79,39%. La distillation de connaissances a permis une augmentation de 2% de précision avec les paramètres de distillation $\alpha = 0,1$, $\beta = 0,9$ et $T = 5$.

La contrainte fonctionnelle d'espace mémoire Flash et RAM est validée sur deux matériels cibles. L'estimation du coût d'inférence concernant la latence et la consommation énergétique de ResNet8 est faite sur le STM32L496ZG et le STM32F746ZG. Le concepteur peut ainsi vérifier si les contraintes de son application sont respectées où non. Ces estimations sont directement validées expérimentalement pour illustrer la précision de l'approche.

Estimation de la latence de ResNet8 sur les MCUs STM32L496ZG et STM32F746ZG :

L'estimation de la latence pour une inférence de ResNet8 en FP32 est représentée sur la Figure 5.3. Cette estimation est un abaque pour le concepteur de systèmes embarqués. Sur ce graphique, une ligne noire est un exemple d'une contrainte de latence au quelle un concepteur peut faire face. Ces estimations permettent d'identifier si la solution explorée permet le respect de la contrainte de latence. Il est également possible d'identifier les fréquences de fonctionnement permettant de respecter cette contrainte. Ici, la gamme de fréquences [100 kHz ;4 MHz] est à éviter. Les fréquences [8 MHz ;80 MHz] et [8 MHz ;216 MHz] permettent de respecter la contrainte de latence, dans le cas où ResNet8 est implémenté respectivement dans le STM32L496ZG et le STM32F746ZG.

En comparaison avec l'abaque, la validation expérimentale de la latence pour une inférence de ResNet8 en FP32 est représentée sur la Figure 5.4. ResNet8 a été implémenté au format FP32 dans le STM32L496ZG et STM32F746ZG. La mesure illustre la précision du modèle d'estimation sur un CNN avec une topologie complexe comme ResNet. Il est observable que la mesure valide l'estimation fournie précédemment, par rapport aux plages de fréquences [8MHz ;80MHz] et [8MHz ;216MHz] du STM32L496ZG et du STM32F746ZG qui permettent le respect de la contrainte de latence. Le détail quantitatif de ces résultats est disponible en Annexe C.1. Des contraintes de consommation énergétique peuvent également être à respecter. Il est donc nécessaire de coupler l'estimation de la latence avec une estimation de la consommation énergétique liée à l'inférence de ResNet8.

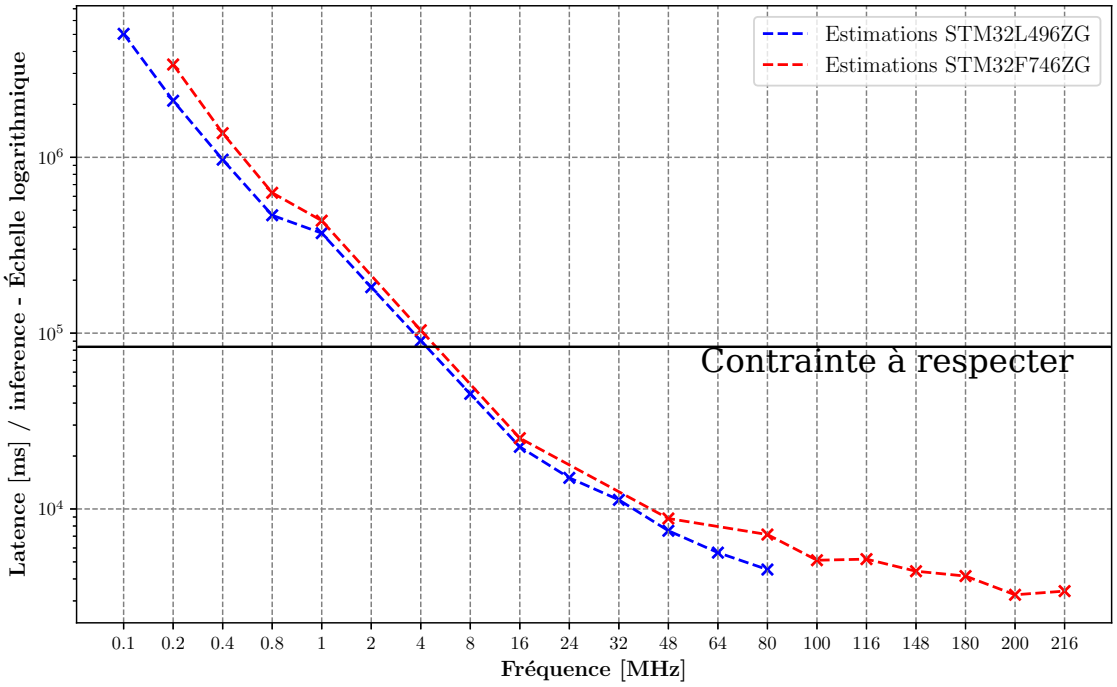


FIGURE 5.3 – Estimations de la latence pour une inférence de ResNet8 au format binaire FP32 au sein des STM32L496ZG (bleu), et STM32F746ZG (rouge). Échelle logarithmique sur l'axe Y. La ligne noire représente une contrainte de latence à respecter au quelle un concepteur peut faire face. Il s'agit d'un exemple pour illustrer l'utilisation de la méthodologie.

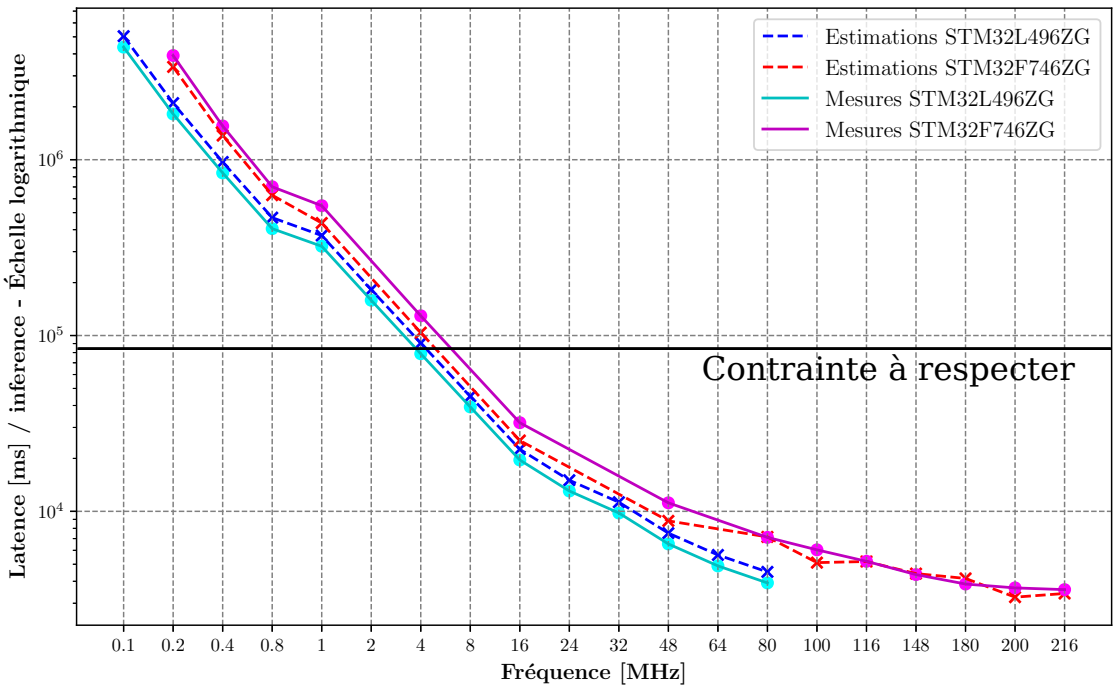


FIGURE 5.4 – Estimations de la latence *vs.* mesures de la latence pour une inférence de ResNet8 au format binaire FP32 au sein des STM32L496ZG (bleu *vs.* cyan), et STM32F746ZG (rouge *vs.* magenta). Échelle logarithmique sur l'axe Y.

Estimation de la consommation énergétique de ResNet8 sur les MCUs STM32L496ZG et STM32F746ZG :

L'estimation de la consommation énergétique est représentée sur la Figure 5.5. Cette estimation fait également office d'abaque pour le concepteur de système embarqué. De nouveau, une ligne noire est un exemple d'une contrainte de consommation énergétique à respecter. L'estimation fournit par la méthodologie permet de rapidement identifier que ResNet8 implémenté au sein du STM32F746ZG ne pourra pas répondre aux contraintes de l'application. En effet, quelle que soit la fréquence de fonctionnement du MCU, la consommation énergétique est toujours supérieure à la contrainte. Il est donc nécessaire d'orienter le choix d'implémentation vers le STM32L496ZG. Les fréquences permettant de respecter la contrainte énergétique avec ce MCU sont comprises entre [800 kHz ;80 MHz]. Cependant, en croisant l'estimation de latence et de consommation énergétique, les fréquences permettant de respecter les contraintes de l'application sont [8 MHz ;80 MHz] avec le STM32L496ZG.

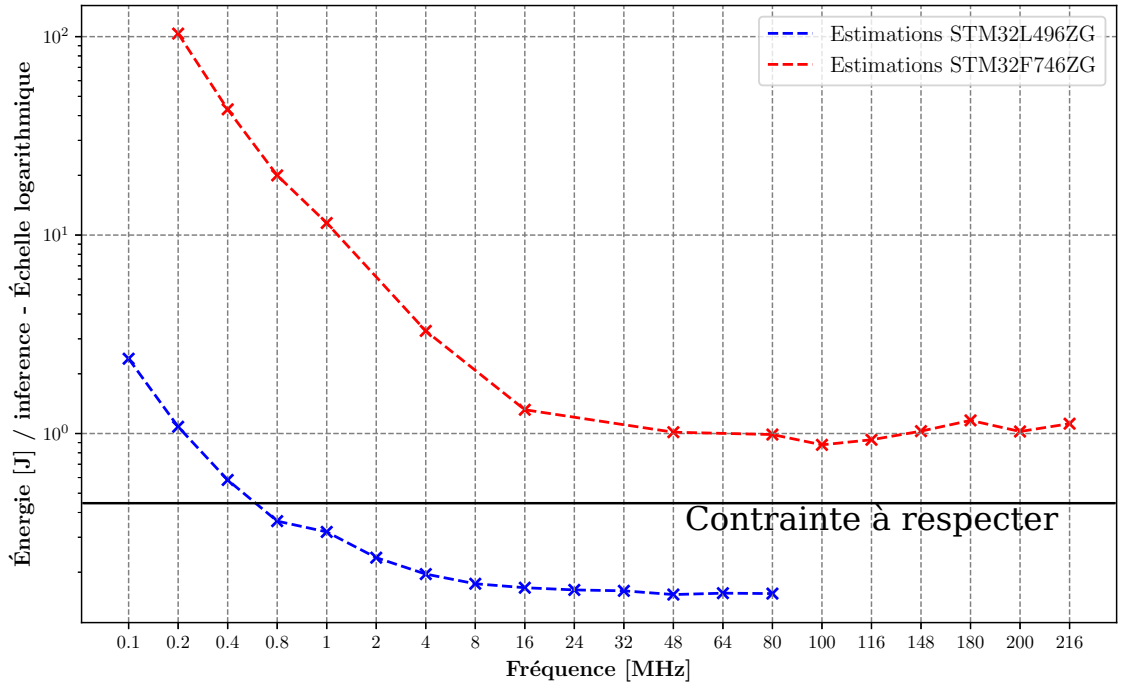


FIGURE 5.5 – Estimations de la consommation énergétique pour une inférence de ResNet8 au format binaire FP32 au sein des STM32L496ZG (bleu) et STM32F746ZG (rouge). Échelle logarithmique sur l'axe Y. De nouveau, la ligne noire représente une contrainte de consommation énergétique à respecter pour illustrer l'utilisation de la méthodologie.

En comparaison avec l'abaque, la validation expérimentale de la consommation énergétique pour une inférence de ResNet8 en FP32 est représentée sur la Figure 5.6. La mesure valide la précision du modèle d'estimation pour la consommation énergétique de ResNet. De plus, l'exploration des solutions possibles fournie précédemment est vérifiée. Une implémentation dans le STM32F746ZG ne permettra pas le respect de la contrainte en consommation énergétique. L'exécution de ResNet8 aux fréquences de fonctionnement comprises entre [800kHz ;80MHz] sur le STM32L496ZG est une solution possible. Le détail quantitatif de ces résultats est disponible en

Annexe C.1.

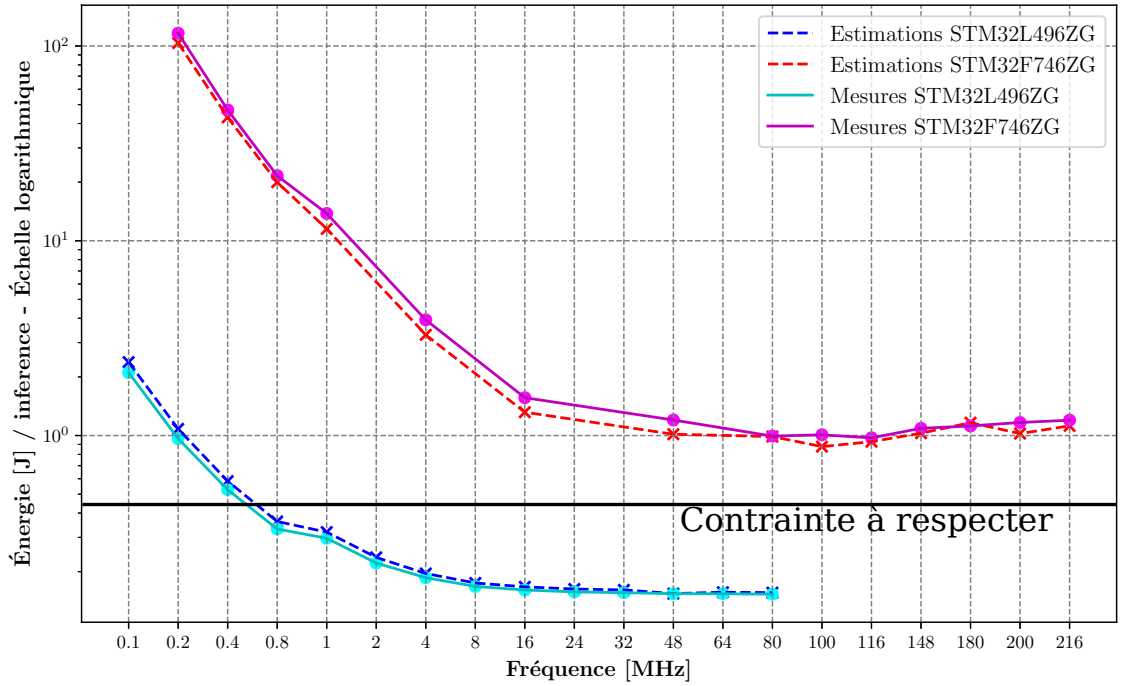


FIGURE 5.6 – Estimations de la consommation énergétique *vs.* mesures de la consommation énergétique pour une inférence de ResNet8 au format binaire FP32 au sein des STM32L496ZG (bleu *vs.* cyan), et STM32F746ZG (rouge *vs.* magenta). Échelle logarithmique sur l'axe Y.

Synthèse de la première implémentation de la topologie ResNet :

La méthodologie a permis d'identifier la technique de réduction de distillation de connaissances pour implémenter un ResNet8. L'estimation de la mémoire Flash de ResNet8 a permis d'assurer que le réseau pouvait être implémenté dans les 3 MCU. Cependant, l'estimation de l'espace mémoire RAM a identifié une impossibilité d'implémentation de ResNet8 au sein du STM32F446ZE. Par la suite, l'estimation a permis de valider la contrainte de latence sous certaines fréquences pour le STM32L496ZG et le STM32F746ZG. L'estimation de la consommation énergétique a finalement mis en avant un non-respect de la contrainte de consommation énergétique au sein du STM32F746ZG. La solution retenue est donc une implémentation de ResNet8 au sein du STM32L496ZG et une exécution du code à une fréquence comprise entre 8MHz et 80MHz. La Figure 5.7 résume l'exploration de l'espace des solutions pour l'implémentation de ResNet au sein des MCU.

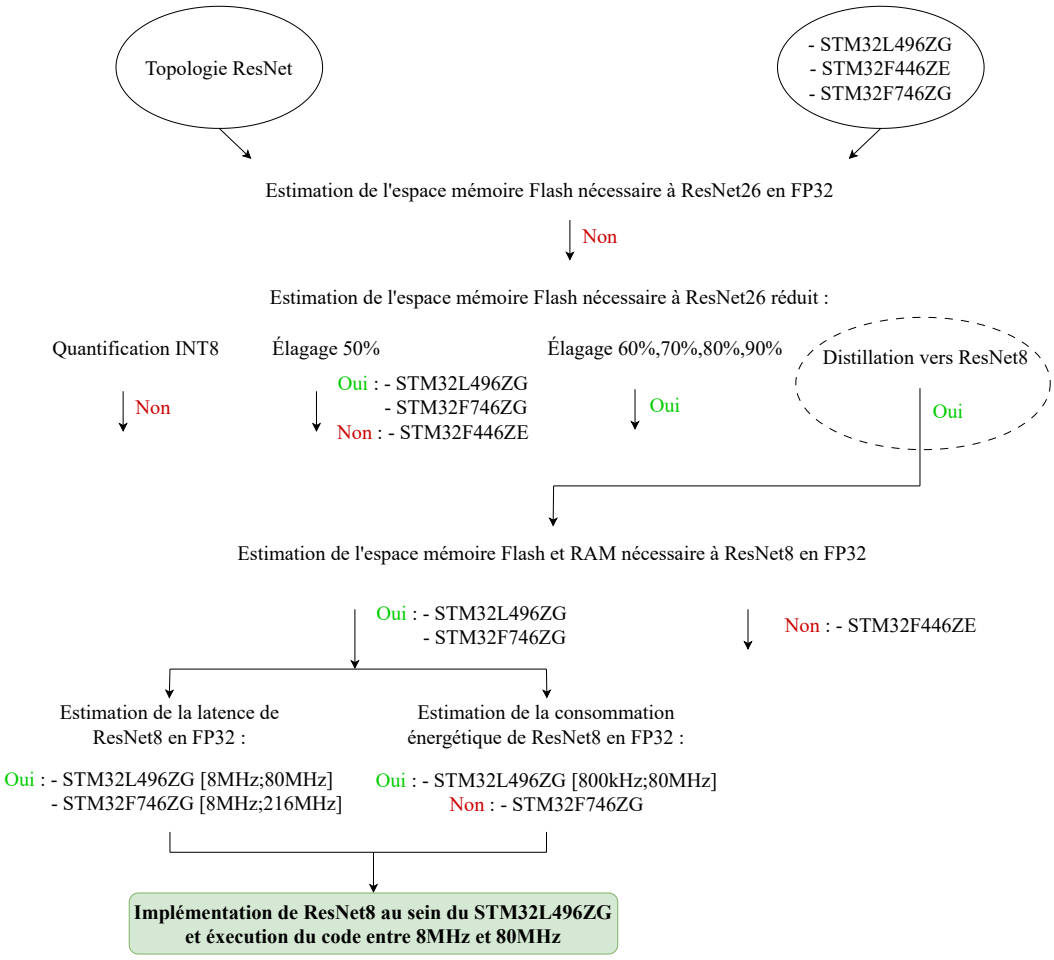


FIGURE 5.7 – Solutions possibles mises en avant par la méthodologie pour guider le choix des techniques de réduction pertinentes pour intégrer la topologie ResNet.

Concernant l'implémentation de ResNet8 au sein du STM32F446ZE, il est maintenant nécessaire d'explorer l'espace des solutions pour réduire suffisamment le réseau. La contrainte d'espace mémoire RAM est non respectée, il est donc nécessaire d'estimer cette contrainte dans le cas où des techniques de réduction seraient appliquées à ResNet8.

5.3.3 Estimation du coût d'inférence de ResNet8 réduit au sein du STM32F446ZE

ResNet8 en format binaire FP32 ne peut pas être implémenté au sein du MCU STM32F446ZE, l'espace mémoire RAM étant insuffisant pour assurer une inférence. Les possibilités de réduction pour ResNet8 sont la quantification et l'élagage. Les estimations suivantes peuvent être immédiatement générées pour proposer à l'utilisateur l'ensemble des possibilités d'implémentations au sein de l'espace des solutions. Le détail quantitatif des résultats présentés ci-dessous est disponible en Annexe C.2. La distillation de connaissances est une solution non envisagée, car pouvant être difficile à appliquer avec ResNet8 en tant qu'enseignant. La difficulté réside dans le choix d'un CNN élève pertinent. Une configuration de distillation de connaissances efficace avec ResNet8 en tant qu'enseignant n'a pas été répertoriée au sein de l'état-de-l'art.

Estimation de l'espace mémoire :

L'espace mémoire Flash disponible dans le STM32F446ZE est suffisant pour ResNet8 en FP32. Cependant, l'estimation de l'espace mémoire Flash nécessaire à ResNet8 réduit est une information importante pour un concepteur. Cela indique l'espace mémoire restant pour les autres éléments de l'application embarquée. L'estimation est référencée sur le Tableau 5.5.

TABLE 5.5 – Espace mémoire Flash estimé nécessaire au stockage des poids et du code des différentes configurations de ResNet8 réduit par la quantification en format binaire INT8 et par l'élagage des filtres de convolution.

Réduction	**Mémoire Flash nécessaire [ko]**
Quantification INT8	259,11
Élagage 50% FP32	128,11
Élagage 60% FP32	87,36
Élagage 70% FP32	52,14
Élagage 80% FP32	25,63
Élagage 90% FP32	8,56

La technique d'élagage réduit significativement l'espace mémoire Flash nécessaire à ResNet8. Cela n'était pas le cas pour l'élagage appliqué à LeNet5, majoritairement composé de poids liés aux opérations entièrement connectées. ResNet8 est majoritairement composé d'opérations de convolution et la technique d'élagage affecte directement les filtres de convolution. De plus, l'élagage supprime de nombreuses lignes de code.

Concernant l'espace mémoire RAM, la partie du réseau nécessitant le plus de primitives est la $\mathcal{L}4$, comme mis en avant dans le Tableau 5.4. Par souci de lisibilité, uniquement l'espace RAM nécessaire à cette partie de ResNet8 est référencé dans le Tableau 5.6, pour les différentes topologies de ResNet8 réduit.

TABLE 5.6 – Estimation de l'espace mémoire RAM maximal nécessaire au stockage des calculs intermédiaires de ResNet8 en format binaire FP32.

Réduction	**Primitives en entrée**	**Primitives en sortie**	**Total primitives**	**Total mémoire [ko]**
Quantification INT8	32 768	16 384	49 152	49,15
Élagage 50% FP32	16 384	16 384	32 768	131,07
Élagage 60% FP32	14 336	12 902	27 238	108,95
Élagage 70% FP32	10 240	8 192	18 432	73,73
Élagage 80% FP32	8 192	5734	13 926	55,71
Élagage 90% FP32	4 096	2458	6 553	26,21

La technique d'élagage à 50% des filtres de convolution offre un espace mémoire RAM nécessaire à une inférence de ResNet8 de $131,07ko$. Même si cette valeur est inférieure à l'espace mémoire RAM fournie par le STM32F446ZE, à savoir $132ko$, un élagage à 50% n'est pas envisageable. En effet, cette estimation prend en compte l'espace mémoire RAM pour le CNN uniquement. Des variables nécessaires au fonctionnement du MCU, ou bien le code supplémentaire permettant d'exécuter l'ensemble de l'application embarquée ne pourraient pas s'exécuter. L'utilisateur a donc la possibilité d'appliquer la quantification INT8 ou l'élagage à 60%, 70%, 80% ou 90% pour respecter la contrainte de l'espace mémoire. Il est maintenant nécessaire d'analyser l'estimation de la latence et de la consommation énergétique pour une inférence de ResNet8 réduit.

Estimation de la latence de ResNet8 sur le MCU STM32F446ZE :

L'estimation de la latence pour une inférence de ResNet8 réduit est représentée sur la Figure 5.8. La quantification en INT8 résulte en la latence la plus élevée, l'élagage à 90% des filtres de convolution offre la latence la plus faible. La ligne noire représente un exemple d'une contrainte à respecter. Au vu de l'estimation fournie par la méthodologie, la technique d'élagage à 70%, 80% et 90% permet de respecter la contrainte en latence, quelle que soit la fréquence de fonctionnement du MCU. L'élagage à 60% est possible, si la fréquence de fonctionnement est comprise dans la gamme [32 MHz ;180 MHz]. L'estimation confirme également la fréquence de 24 MHz, mais la contrainte est très proche de la latence estimée. Concernant l'élagage à 50% les fréquences fiables sont comprises dans la gamme [48 MHz ;180 MHz]. Finalement la quantification INT8 est exploitable uniquement entre les fréquences 116 MHz et 180 MHz. De nouveau, coupler l'estimation de la latence avec l'estimation en consommation énergétique permet de raffiner la recherche de solutions.

Estimation de la consommation énergétique de ResNet8 sur le MCU STM32F446ZE :

L'estimation de la consommation d'énergie pour une inférence de ResNet8 réduit est représentée sur la Figure 5.9. La quantification INT8 résulte en une consommation énergétique la plus élevée et l'élagage à 90% des filtres de convolution offre la consommation énergétique la plus faible. La quantification en INT8, l'élagage à 50% et 60% ne permettront pas de respecter la contrainte de consommation en énergie, quelle que soit la fréquence de fonctionnement du MCU. Concernant l'élagage

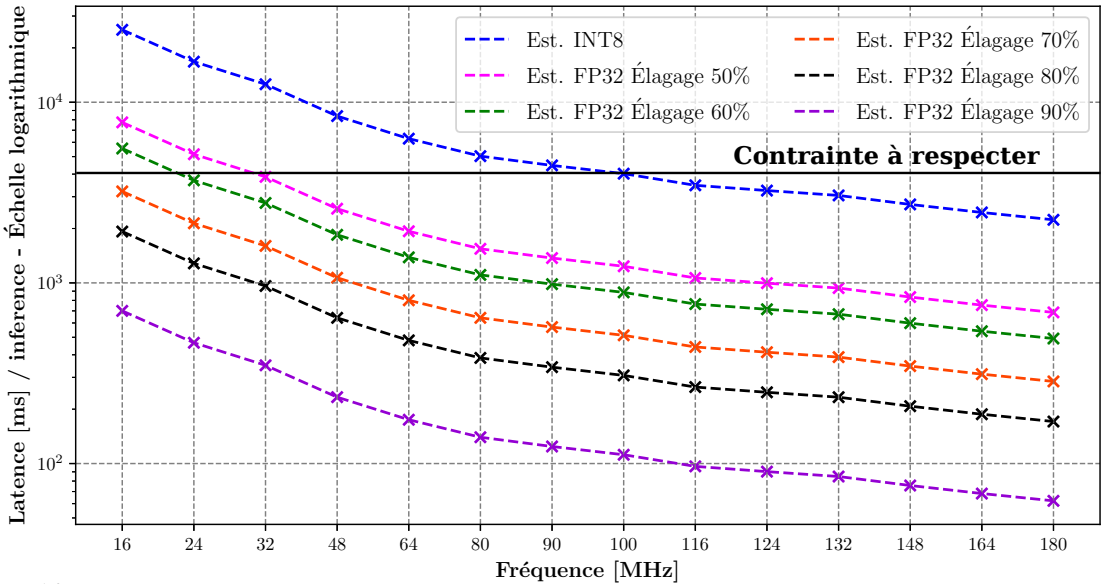


FIGURE 5.8 – Estimations de la latence pour une inférence de ResNet8 réduit par la quantification INT8 (bleu) et l'élagage à 50% (rose), 60% (vert), 70% (orange), 80% (noir), 90% (violet) au sein du STM32F446ZE. Échelle logarithmique sur l'axe Y.

à 70%, les fréquences dans la gamme [32 MHz ;180 MHz] permettraient de respecter la contrainte. L'élagage à 80% ou 90% assure le respect de la consommation énergétique.

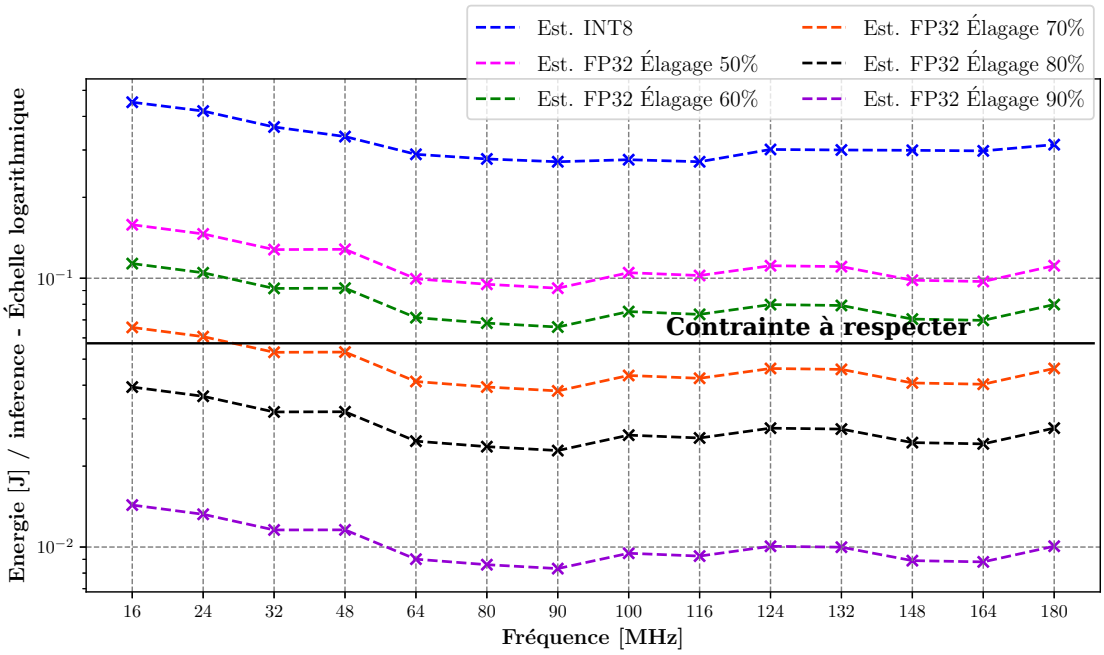


FIGURE 5.9 – Estimations de la consommation énergétique pour une inférence de ResNet8 réduit par la quantification INT8 (bleu) et l'élagage à 50% (rose), 60% (vert), 70% (orange), 80% (noir), 90% (violet) au sein du STM32F446ZE. Échelle logarithmique sur l'axe Y.

Synthèse de l'exploration de l'espace des solutions pour implémenter ResNet8 au sein du STM32F446ZE :

La méthodologie ZIP-CNN a permis d'identifier l'impossibilité d'implémenter ResNet8 au sein du STM32F446ZE, due à un manque d'espace mémoire RAM. L'exploration de l'espace des solutions a d'abord mis en avant que la quantification INT8 et l'élagage des filtres de convolution à 60%, 70%, 80% et 90% permettraient de réduire suffisamment l'espace RAM. Ensuite, l'étude des solutions pour respecter les contraintes de latence et de consommation énergétique a permis de raffiner le choix de la technique de réduction à appliquer. L'élagage à 70% des filtres de convolution permettrait ainsi de respecter les contraintes d'espace mémoire Flash, RAM, de latence et de consommation énergétique. De plus, l'exploration de l'espace des solutions a mis en avant que les fréquences de fonctionnement comprises entre 32 MHz et 180 MHz assureraient le respect de ces contraintes. Le choix de la technique de réduction pour l'exemple d'application de la méthodologie ZIP-CNN s'est donc porté sur un élagage à 70% de ResNet8. La Figure 5.10 résume l'exploration de l'espace des solutions pour implémenter ResNet8 au sein du STM32F446ZE. Dans la suite, la technique de réduction identifiée par la méthodologie ZIP-CNN est appliquée, le réseau ResNet8 réduit est implémenté au sein du STM32F446ZE et des mesures viennent confirmer expérimentalement l'exploration de l'espace des solutions effectuées.

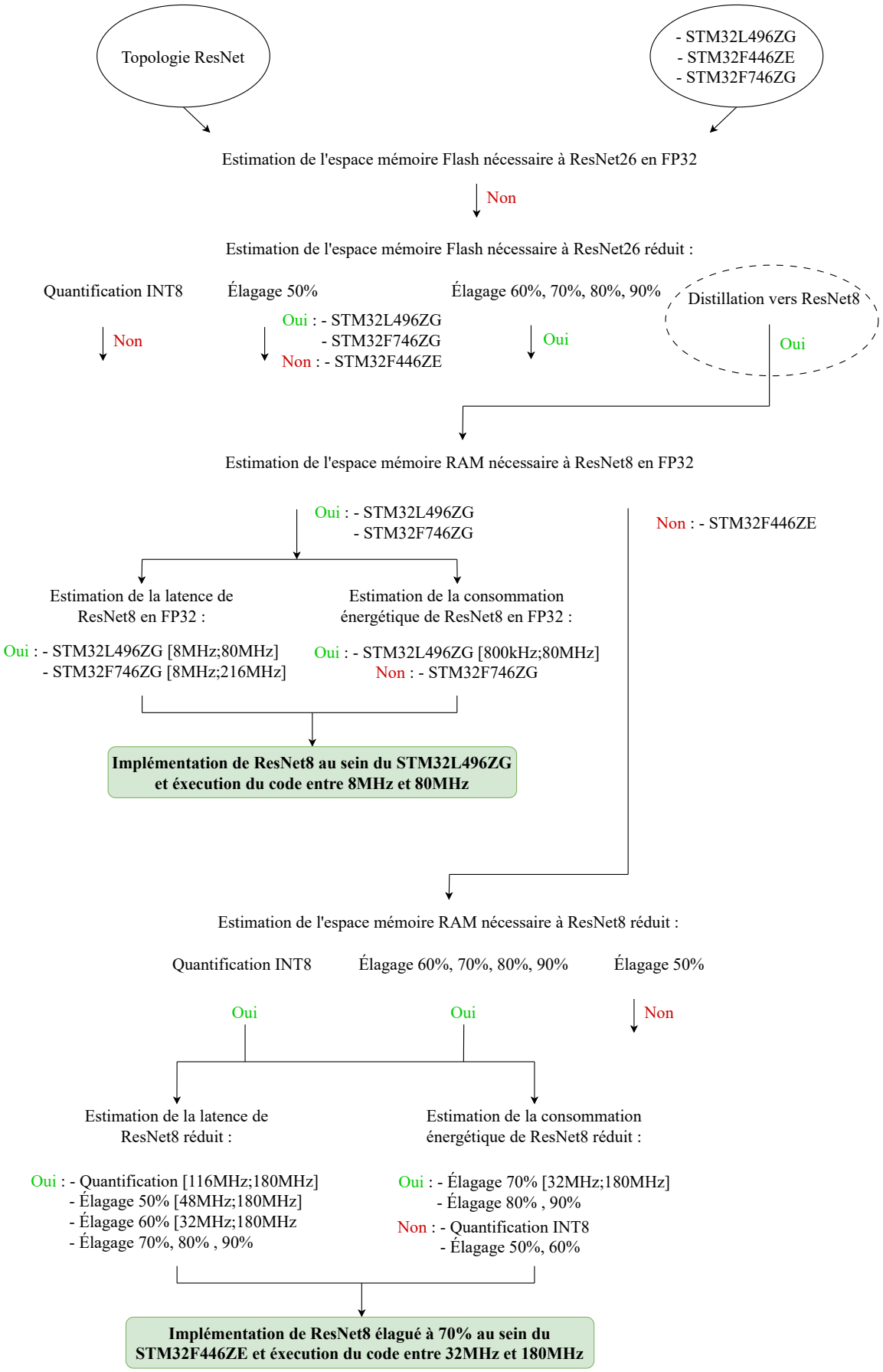


FIGURE 5.10 – Solutions possibles mises en avant par la méthodologie pour guider le choix des techniques de réduction pertinentes pour intégrer la topologie ResNet.

5.3.4 Implémentation de ResNet8 au sein du STM32F446ZE

L'élagage à 70% des filtres de convolution a été appliqué à ResNet8. La précision en classification résultante est de 71,47%. Ce résultat est cohérent par rapport à la précision en classification de ResNet8 non réduit de 70,95%. Il est notable que le réseau élagué à 70% offre une meilleure précision en classification, de +0,52%, que le réseau non élagué. Malgré le nombre de paramètres de ResNet8 fortement réduit par l'élagage, l'entraînement du réseau doit assurer une capacité de généralisation pertinente pour le CNN. L'entraînement d'un CNN avec moins de paramètres peut offrir une fonction de classification qui offre une meilleure généralisation.

Validation expérimentale de la contrainte de latence :

La validation expérimentale de l'estimation de la latence pour une inférence de ResNet8 élagué à 70% de ces filtres de convolution est représentée sur la Figure 5.11. La contrainte en latence n'apparaît pas sur la Figure 5.11, cette contrainte étant bien au-dessus de l'estimation et des mesures de latence effectuées. Les choix effectués suite à l'exploration de l'espace des solutions sont ainsi validés concernant la contrainte de latence.

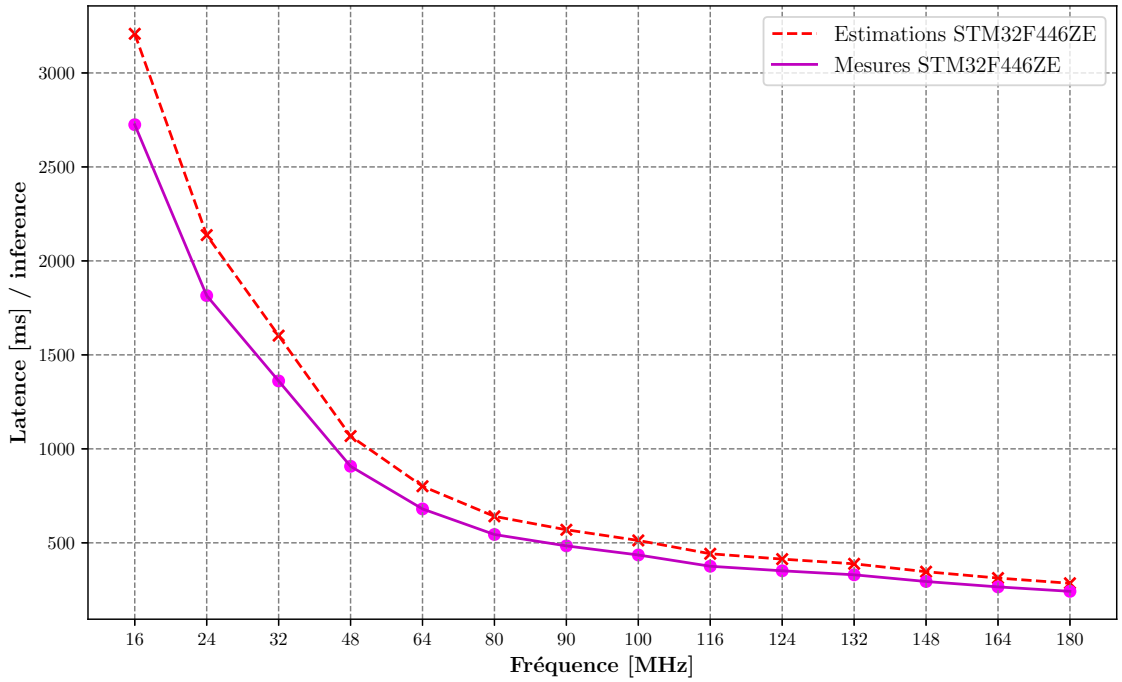


FIGURE 5.11 – Estimations (rouge) de la latence *vs.* mesures (magenta) de la latence pour une inférence de ResNet8 implémenté au sein du STM32F446ZE, au format binaire FP32 et élagué à 70% des filtres de convolution.

Validation expérimentale de la contrainte de consommation d'énergie :

La comparaison entre l'estimation de la consommation d'énergie et l'expérimentation, pour une inférence de ResNet8 élagué à 70% de ces filtres de convolution, est représentée sur la Figure 5.12. La contrainte en consommation énergétique est représentée par la ligne noire. Comme annoncée par l'exploration de l'espace des solutions, la mesure expérimentale confirme que l'élagage à 70% des filtres de convolution permet de respecter la contrainte de consommation en énergie si la fréquence du MCU se situe dans la gamme [32MHz ;180MHz].

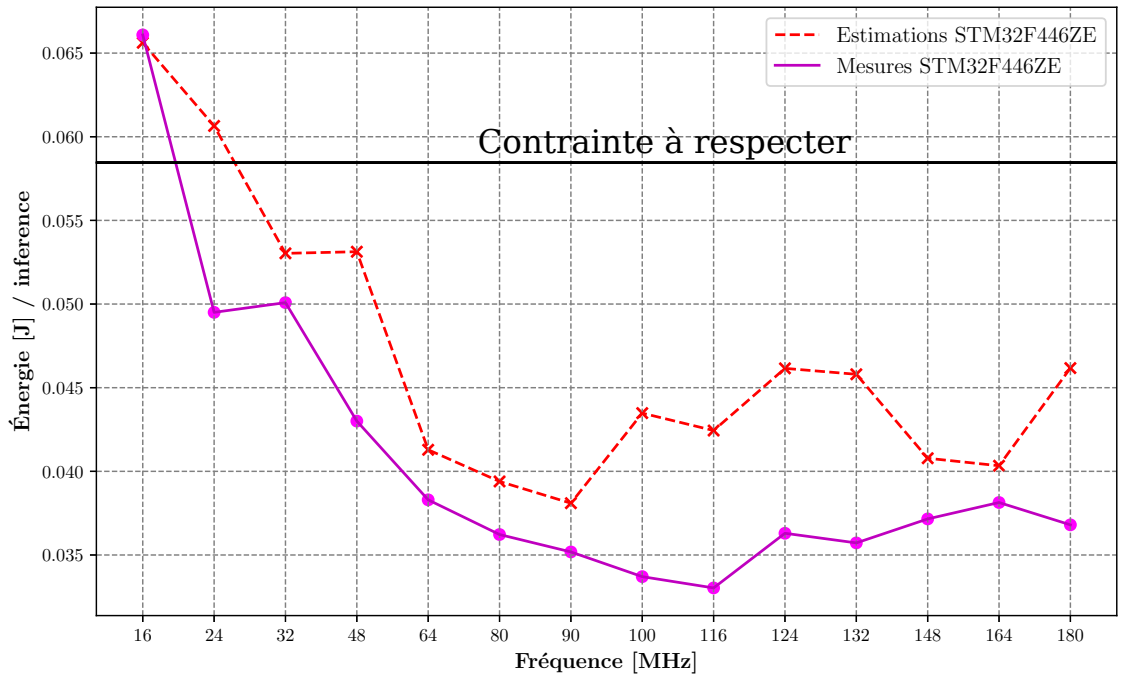


FIGURE 5.12 – Estimations (rouge) de la consommation énergétique *vs.* mesures (magenta) de la consommation énergétique pour une inférence de ResNet8 implémenté au sein du STM32F446ZE, au format binaire FP32 et élagué à 70% des filtres de convolution.

5.4 Synthèse

Ce chapitre illustre un cas d'études de la méthodologie ZIP-CNN dans le cas de l'implémentation de la topologie ResNet au sein des MCU. Ce cas d'études met en avant des avantages de la méthodologie ZIP-CNN sur les solutions de l'état-de-l'art. Avant toute tentative d'implémentation du CNN, la méthodologie fournit l'information au concepteur sur la faisabilité de cette dernière. L'implémentation du CNN par défaut, ou réduit par des techniques d'élagage, de quantification et de distillation de connaissances est évaluée. Là où les propositions de l'état-de-l'art [114]-[116] nécessitent une phase de déploiement pour évaluer l'efficacité de l'implémentation, la méthodologie ZIP-CNN fournit en amont des indications au concepteur sur les contraintes en latence, en consommation énergétique et en espace mémoire. La méthodologie est applicable simplement sur tous les CNN, réduit ou non, grâce au modèle d'estimation s'adaptant efficacement aux différentes topologies de réseaux de neurones. L'exploration de l'espace des solutions grâce à la méthodologie a permis d'implémenter la topologie ResNet avec succès au sein des trois MCU. L'espace mémoire Flash a rapidement était identifié comme limite pour l'implémentation de ResNet26. La méthodologie a ensuite orienté le choix de l'implémentation vers un CNN de taille plus modeste. La distillation de connaissances a été appliquée pour augmenter la précision en classification du réseau ResNet8, l'élève, grâce au réseau ResNet26, l'enseignant. L'estimation de l'espace mémoire nécessaire à une inférence de ResNet8 a permis de mettre en avant que son implémentation était faisable dans le STM32L496ZG et le STM32F746ZG. ResNet8 a donc été implémenté en format binaire FP32. Une validation expérimentale des contraintes de latence et de consommation énergétique a permis de valider l'exploration de l'espace des solutions effectuée. Concernant l'implémentation au sein du STM32F446ZE, le manque d'espace mémoire RAM était une limite. La méthodologie a identifié la quantification en format binaire INT8 et l'élagage des filtres de convolution à 60%, 70%, 80% et 90%, comme des solutions envisageables. En suivant, l'estimation de la latence et de la consommation énergétique de ResNet8 résultant de ces possibles réductions a fourni des indications quantitatives sur le respect de ces contraintes. La solution retenue a été l'élagage à 70% des filtres de convolution de ResNet8 et l'exécution du code à des fréquences comprises entre 32MHz et 180MHz. Finalement, l'implémentation de ResNet8 élagué à 70% a été faite au sein du STM32F446ZE, et une validation expérimentale a confirmé les choix effectués pour respecter les contraintes de mémoire, de latence et de consommation énergétique.

Chapitre 6

Conclusion et perspectives

Sommaire

6.1 Conclusion

Les systèmes embarqués basés sur des microcontrôleurs sont utilisés dans des environnements variés, de par leur capacité à s'interfacer avec de nombreux capteurs. Mais la consommation énergétique, la puissance de calcul et l'espace mémoire sont limités sur ces architectures. Les réseaux de neurones convolutifs ont un rôle majeur dans le futur des applications embarquées. Cependant, le coût d'inférence des CNN est important. L'implémentation de ces derniers au sein de systèmes contraints, tels que les microcontrôleurs, est une étape complexe. La diversité des MCU et des topologies de CNN rend difficile la normalisation d'étapes de conception. Une étude de l'état de l'art a mis en avant les limites des solutions d'implémentation existantes. La nécessité d'implémenter le CNN pour vérifier le respect des contraintes d'une application embarquée est un préjudice à une exploration efficace de l'espace des solutions. Cela induit de nombreux tests pouvant être infructueux. De plus, peu de détails sont fournis au concepteur de l'application embarquée concernant l'impact de l'algorithme sur le matériel cible en termes de latence, d'espace mémoire et de consommation énergétique.

La méthodologie ZIP-CNN développée dans ces travaux de thèse permet d'explorer l'espace des solutions pour l'implémentation d'un CNN au sein d'un MCU. Dans un premier temps, cette méthodologie fournit des informations quantitatives sur la latence, la consommation énergétique et l'espace mémoire nécessaire à l'inférence du CNN. Il est possible de déterminer rapidement si le CNN peut-être implémenté ou non. Cette étape est basée sur une décomposition des CNN en primitives, à une granularité des motifs de calcul. Cette décomposition offre un bon équilibre entre l'effort de caractérisation nécessaire à l'estimation, et l'adaptation aux différentes topologies de CNN. Un protocole de mesure automatisé a été mis en place. Il permet d'employer ces primitives sur un nouveau matériel cible, et ainsi caractériser rapidement de nouvelles configurations de MCU. Dans un second temps, la méthodologie ZIP-CNN permet de guider le choix des éventuelles réductions algorithmiques à appliquer au CNN. Dans ces travaux, la distillation de connaissances, la quantification vers un format binaire en entier sur 8-bit et l'élagage des filtres de convolution du CNN ont été modélisés. Les informations fournies par la méthodologie ZIP-CNN ne nécessitent aucune implémentation du CNN sur la cible matérielle et aucune application des techniques de réduction. Un concepteur de système embarqué peut ainsi rapidement identifier les tâches à effectuer pour implémenter un CNN dans le matériel, tout en respectant les contraintes de l'application cible.

Le modèle d'estimation a d'abord était testé sur le CNN LeNet5, au sein de 3 MCU : le STM32L496ZG, le STM32F446ZE et le STM32F746ZG. Le coût d'inférence de LeNet5 au format binaire FP32, INT8 et élagué à 50%, 60%, 70%, 80% et 90% de ses filtres de convolution a été estimé au sein des 3 MCU. Pour valider la précision des estimations fournies en latence, consommation énergétique et espace mémoire, chacune des configurations de LeNet5 a été implémentée dans les cibles matérielles. Une étape de mesures expérimentales a permis d'effectuer une comparaison avec les valeurs estimées. Toutes les estimations et mesures ont été faites sur 14 fréquences de fonctionnement différentes, propres à chaque MCU. Sur plus de 300 mesures, le modèle d'estimation a fourni des résultats cohérents, permettant d'estimer si le modèle peut être embarqué sans réductions, et le cas échéant, de quantifier la diminution du coût d'inférence apportée par une technique de réduction

de l'état-de-l'art.

Une utilisation de la méthodologie ZIP-CNN appliquée à l'implémentation de la topologie ResNet, a permis d'illustrer l'exploration de l'espace des solutions. L'implémentation de ResNet26 au sein des 3 MCU a été étudiée. Ce réseau n'a pas pu être implémenté directement de par la contrainte d'espace mémoire Flash. L'estimation du coût d'inférence, dans le cadre de la distillation de connaissances, a identifié ResNet8 comme étant un candidat adéquat pour une implémentation au sein du STM32L496ZG et du STM32F746ZG. Concernant le STM32F446ZE, le modèle d'estimation a mis en avant un non respect de la contrainte de l'espace mémoire RAM. Après avoir estimé le coût d'inférence de ResNet8, réduit par quantification et élagage, la technique d'élagage à 70% des filtres de convolution a été retenue. Une validation expérimentale a permis de valider les choix effectués suite à l'exploration de l'espace des solutions. La méthodologie ZIP-CNN a ainsi permis de guider les choix d'implémentation en identifiant les contraintes non respectées, et les solutions adéquates.

Ces travaux de thèse s'inscrivent dans un contexte où les systèmes embarqués basés sur des MCU sont en forte croissance. Les acteurs du domaine prévoient plusieurs milliards de MCU déployés de plus par an, s'ajoutant aux 250 milliards déployés actuellement. Ces systèmes étant déployés dans de nombreux domaines industriels et domestiques, l'implémentation de CNN permettrait la résolution de problématiques variées. La méthodologie ZIP-CNN démocratise l'implémentation de CNN au sein de MCU, en guidant les concepteurs de systèmes embarqués dans leur choix d'implémentation. Les résultats obtenus ouvrent également une voie d'exploration pour appliquer les modèles développés à d'autres plateformes matérielles, comme les architectures multicœurs ou les FPGA. De plus, les résultats d'estimations sont également exploitables dans l'utilisation d'algorithmes de recherche de réseaux de neurones. Cela permettrait potentiellement de découvrir de nouvelles topologies de réseaux de neurones, adaptées à l'inférence à faible coût.

6.2 Perspectives

Ces travaux de thèse ont permis de valider une première approche facilitant l'exploration de l'espace des solutions pour l'implémentation d'un CNN au sein d'un MCU. Plusieurs perspectives permettraient d'améliorer la méthodologie décrite dans ce manuscrit.

- Un système embarqué ne consiste pas seulement en un MCU. De nombreux capteurs ou éléments communicants sont également utilisés. Une étude de l'impact du coût d'inférence du CNN sur le coût d'exécution de l'ensemble du système permettra de mettre en avant son influence. L'efficacité des techniques de réduction doit également être mise en perspective par rapport à l'ensemble du système embarqué. Une réduction de 50% de la consommation énergétique du CNN peut correspondre à seulement quelques pour-cent de réduction pour l'ensemble du système. De plus, la durée de vie de la batterie est un paramètre essentiel. Il s'agit d'un objectif de la collaboration avec la société Wisebatt, maintenant implémentée chez STMicroelectronics. Le modèle de consommation énergétique des CNN est implémenté dans leur outil. Il est donc possible de le combiner avec leur modèle réel de comportement de batterie, pour effectuer une estimation précise de la durée de vie de la batterie d'un système embarquant un CNN. L'impact des techniques de réduction sur la durée de vie de la batterie pourra également être mis en avant.

- La NAS est une solution très développée dans la recherche pour trouver des topologies de CNN efficaces dans les cibles matérielles contraintes. Comme explicité dans l'état-de-l'art, les méthodes NAS ont besoin d'une stratégie d'estimation des performances, pour évaluer une itération de recherche de topologie. Le modèle d'estimation développé durant cette thèse pourrait être employé dans ce cadre. La consommation énergétique pourrait être un critère de recherche pour trouver des topologies de CNN à faible coût d'exécution énergétique. Il s'agirait d'une étude inédite dans l'état-de-l'art. D'autres hypothèses de recherche sont la prise en compte de la consommation énergétique de l'ensemble du système dans l'algorithme NAS, ou bien de la durée de vie de la batterie. Ainsi, la topologie du CNN trouvée pourrait tenir compte de la consommation des autres éléments, ou bien du comportement réel de la batterie.

- Les MCU sont des cibles matérielles très utilisées dans le domaine des systèmes embarqués. Cependant, d'autres cibles comme les architectures multi-cœurs ou les FPGA sont régulièrement employées. De plus, ces travaux se sont concentrés sur l'implémentation de CNN au sein de cibles matérielles de type bare métal. Dans le paysage des systèmes embarqués, certains fonctionnent avec un OS comme les plateformes Raspberry Pi. Le modèle développé dans cette thèse pourrait être appliqué sur de telles cibles. Une adaptation de la caractérisation des primitives sera nécessaire pour prendre en compte la spécificité de ces architectures numériques. Cela résulterait en une diversité importante de solutions possibles pour un concepteur.

- Le modèle d'estimation développé durant ces travaux offre une précision cohérente par rapport aux mesures, environ 90% de précision. Des investigations supplémentaires sur des comportements matériels spécifiques aux architectures numériques cibles amélioreraient la précision de ce modèle. Une meilleure précision permettrait ainsi de raffiner l'exploration de l'espace des solutions.

- Les primitives utilisées dans le modèle d'estimation suivent une exécution de base sur le matériel cible, c'est-à-dire non optimisée par rapport à l'architecture matérielle. Des exécutions spécialisées sur certains matériels permettent de fortement améliorer le coût d'inférence du CNN. L'optimisation d'opérations *Single Instruction Multiple Data* - SIMD au sein des CPU, l'utilisation des librairies industrielles comme CMSIS-NN sur MCU ou encore les outils comme STM32CubeAI permettent d'améliorer le coût d'inférence sur l'architecture cible. Appliquer le modèle d'estimation avec des primitives reflétant l'exécution d'un code optimisé permettrait une exploration de l'espace des solutions au plus proche des performances optimisées réelles.

6.3 Publications

Ces travaux de thèses ont donné lieu à plusieurs publications, référencées ci-après :

- Conférences internationales :

  [146] T. Garbay, K. Hachicha, P. Dobias et al., "Accurate Estimation of the CNN Inference Cost for TinyML Devices", in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, 2022

  [147] P. Dobiáš, T. Garbay, B. Granado et al., "Comparative Study of Scheduling a Convolutional Neural Network on Multicore MCU", in *Design and Architecture for Signal and Image Processing*, 2022

  [148] T. Garbay, P. Dobias, W. Dron et al., "CNN Inference Costs Estimation on Microcontrollers : the EST Primitive-based Model", in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021

  [149] T. Garbay, O. Chuquimia, A. Pinna et al., "Distilling the knowledge in CNN for WCE screening tool", in *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, IEEE, 2019

- Communication internationale :

  [150] T. Garbay, K. Hachicha, P. Dobiáš et al., *Accurate Estimation of the CNN Inference Cost within Microcontrollers*, Poster, Published : TinyML EMEA 2022, 2022

- Communications nationales :

  [151] T. Garbay, P. Dobiáš, W. Dron et al., "Estimation du coût d'inférence des réseaux de neurones convolutifs dans un microcontrôleur", in *16ème Colloque du GDR SoC2*, Strasbourg, France, 2022

  [152] O. Chuquimia, T. Garbay, W. Xu et al., "Study to integrate CNN inside a WCE to realize a screening tool", in *Journées d'Etude sur la TéléSanté*, 2019

Annexe A

Caractérisation des primitives

A.1 Algorithmes des primitives utilisées

Data: Input I[n][m] ; Kernel k[i][j] ; Bias b ; Stride s
Result: Feature map $F_{out}[x][y]$ of size $((n-i)/s)+1$

initialization;
forall *n and m in I* **do**
  forall *i and j in k* **do**
    $F[x][y] = \sum I[n][m] \times k[i][j]$;
    shifting kernel of stride s on I;
  end
  $F[x][y] = F[x][y] + b$;
end
Return F;

Algorithm 1: Algorithme de la primitive de Convolution 2D.

Data: Input RGB[n][m] ; Kernel $k[i \times j]$; Vector $v[i \times j]$; Bias b ; Stride s
Result: Feature map $F_{out}[x][y]$ of size $((n-i)/s)+1$

forall *i+n and j+m in I* **do**
  Vectorisation of RGB components in v;
end
forall $i \times j$ *in k* **do**
  $F[x][y] = \sum v[i \times j] \times k[i \times j]$;
end
$F[x][y] = F[x][y] + b$;
Return F;

Algorithm 2: Algorithme de la primitive de Convolution 2D sur entrée RGB.

Data: Input I[n][m] ; Kernel k[i][j] ; Stride s
Result: Feature map $F_out[x][y]$ of size $((n-i)/s)+1$

forall *n and m in I* **do**
  forall *i and j in k* **do**
    $sum = \sum I[i+(s\times n)][j+(s\times m)]$;
  end
end
$F[x][y] = \frac{sum}{i\times j}$;
Return F;

Algorithm 3: Algorithme de la primitive de Regroupement par Valeur Moyenne.

Data: Input I[n][m] ; LUT of 1024 Sigmoïd values L[d]
Result: Feature map F[n][m]

forall *n and m in I* **do**
  if $I[n][m] < -5$ **then**
    $I[n][m] = -5$;
  else
    if $I[n][m] > 5$ **then**
      $I[n][m] = 5$;
    else
      calcul d;
      if $d > 512$ **then**
        $d = 512$;
        $F[n][m] = L[d]$;
      else
        if $d < -512$ **then**
          $d = -512$;
          $F[n][m] = L[d]$;
        else
          $F[n][m] = L[d]$;
        end
      end
    end
  end
end
Return F;

Algorithm 4: Algorithme de la primitive Sigmoïde.

Data: Input Vector $v_{in}[d]$
Result: Normalized output vector $v_{out}[d]$

forall *d in* v_{in} **do**
  $sum = \sum exp(v_{in}[d])$
end
forall *d in* v_{out} **do**
  $v_{out} = \frac{exp(v_{in}[d])}{sum}$
end
Return v_{out};

Algorithm 5: Algorithme de la primitive Softmax.

Data: Input feature map $F_{in}[x][y]$
Result: Output feature map $F_{out}[x][y]$

forall *x and y in* F_{in} **do**
  if $F_{in}[x][y] \leq 0$ **then**
    $F_{out}[x][y] = 0$;
  else
    if $F_{in}[x][y] > 0$ **then**
      $F_{out}[x][y] = F_{in}[x][y]$;
    end
  end
end
Return F_{out};

Algorithm 6: Algorithme de la primitive ReLU.

Data: Batch Normalization parameters : β, γ, moving variance, moving mean ; Input feature map $F_{in}[x][y]$
Result: Output feature map $F_{out}[x][y]$

forall *x and y in* F_{in} **do**
  $F_{out}[x][y] = \gamma \times \frac{(F_{in}[x][y] - \text{moving mean})}{\text{moving variance}} + \beta$;
end
Return F_{out};

Algorithm 7: Algorithme de la primitive *Batch Normalization*.

Data: Input 1 $I_1[n][m]$; Input 2 $I_2[n][m]$
Result: Output matrix $O[n][m]$

forall *n and m in* I_1 *and* I_2 **do**
    $O[n][m] = I_1[n][m] + I_2[n][m]$;
    $I_1[n][m] = 0$;
    $I_2[n][m] = 0$;
end
Return $O[n][m]$;

Algorithm 8: Algorithme de la primitive de connexion résiduelle.

Data: Input vector $v_{in}[d]$; Weights $v_{weights}[i][d]$; Biais b
Result: Output vector $v_{out}[i]$

forall *i in* v_{out} **do**
    forall *d in* v_{in} **do**
        $v_{out}[p] = \sum v_{in}[d] \times v_{weights}[i][d]$;
    end
    $v_{out}[p] = v_{out}[p] + b$;
end
Return v_{out};

Algorithm 9: Algorithme de la primitive de couche entièrement connectée.

A.2 Mesures de caractérisation des primitives

Les Tableaux A.1 et A.2 référencent les mesures de latence des opérations au sein du STM32L496ZG. Les Tableaux A.3 et A.4 référencent les mesures de courant consommé par les opérations au sein du STM32L496ZG.

TABLE A.1 – Mesures de latence des opérations au format binaire FP32 au sein du STM32L496ZG

	Latence mesurée [ms]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,1	0,239600	0,239600	0,239600	0,397600	71,783600
0,2	0,103200	0,103200	0,103200	0,166800	29,997600
0,4	0,045900	0,045900	0,045900	0,080600	13,855700
0,8	0,021600	0,021600	0,021600	0,038900	6,652050
1	0,017280	0,017280	0,017280	0,031280	5,279520
2	0,008640	0,008640	0,008640	0,015480	2,598840
4	0,004320	0,004320	0,004320	0,007570	1,289090
8	0,002160	0,002160	0,002160	0,003785	0,642135
16	0,001080	0,001080	0,001080	0,001892	0,320448
24	0,000720	0,000720	0,000720	0,001262	0,213472
32	0,000540	0,000540	0,000540	0,000946	0,160069
48	0,000360	0,000360	0,000360	0,000631	0,114982
64	0,000270	0,000270	0,000270	0,000473	0,090359
80	0,000216	0,000216	0,000216	0,000378	0,075887

TABLE A.2 – Mesures de latence des opérations au format binaire INT8 au sein du STM32L496ZG

	Latence mesurée [ms]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,1	0,169612	0,169484	0,169484	0,213620	70,517952
0,2	0,070952	0,070952	0,070952	0,089034	29,418168
0,4	0,032819	0,032817	0,032817	0,041136	13,590309
0,8	0,015782	0,015783	0,015783	0,019789	6,522852
1	0,012546	0,012546	0,012546	0,015735	5,176050
2	0,006172	0,006172	0,006172	0,007758	2,546638
4	0,003072	0,003072	0,003072	0,003847	1,263196
8	0,001530	0,001530	0,001530	0,001916	0,629109
16	0,000764	0,000764	0,000764	0,000957	0,313937
24	0,000509	0,000509	0,000509	0,000637	0,209154
32	0,000382	0,000382	0,000382	0,000478	0,156813
48	0,000254	0,000254	0,000254	0,000319	0,110882
64	0,000191	0,000191	0,000191	0,000239	0,088225
80	0,000153	0,000153	0,000153	0,000191	0,074086

TABLE A.3 – Mesures de courant consommé par les opérations au format binaire FP32 au sein du STM32L496ZG

	Courant mesuré [mA]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,1	0,142840	0,143090	0,143138	0,142867	0,146277
0,2	0,155810	0,156198	0,156096	0,155002	0,163893
0,4	0,181801	0,182173	0,182238	0,179318	0,200150
0,8	0,233456	0,234058	0,234300	0,227582	0,271466
1	0,262277	0,262889	0,263244	0,254605	0,309651
2	0,391849	0,392812	0,393615	0,375967	0,485643
4	0,651785	0,653631	0,655736	0,619081	0,838791
8	1,169131	1,172686	1,177132	1,103118	1,544838
16	2,235296	2,241607	2,250087	2,101176	2,989871
24	3,269684	3,278716	3,291052	3,070086	4,372775
32	4,314893	4,327093	4,342739	4,048144	5,762115
48	6,404236	6,421215	6,445247	6,000236	7,281297
64	8,376642	8,400356	8,435193	7,870320	9,090969
80	10,407792	10,436765	10,485482	9,778059	10,679613

TABLE A.4 – Mesures de courant consommé par les opérations au format binaire INT8 au sein du STM32L496ZG

	Courant mesuré [mA]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,1	0,144636	0,145094	0,145075	0,144307	0,149144
0,2	0,157480	0,158771	0,158389	0,156065	0,167004
0,4	0,183113	0,186089	0,185108	0,179537	0,203069
0,8	0,234172	0,240502	0,238082	0,226233	0,273829
1	0,262737	0,270829	0,267656	0,252558	0,312447
2	0,390880	0,407160	0,400652	0,369670	0,489389
4	0,647854	0,680766	0,667493	0,604498	0,844833
8	1,160991	1,226963	1,200167	1,073337	1,556265
16	2,218021	2,350790	2,296019	2,042245	3,012577
24	3,246542	3,445596	3,362952	2,980710	4,407136
32	4,284260	4,548496	4,438773	3,929920	5,807175
48	6,346611	6,743229	6,577336	5,814973	7,383977
64	8,371090	8,778425	8,563627	7,623494	9,273818
80	10,348390	10,862084	10,591816	9,430646	10,915469

Les Tableaux A.5 et A.6 référencent les mesures de latence des opérations au sein du STM32F446ZE. Les Tableaux A.7 et A.8 référencent les mesures de courant consommé par les opérations au sein du STM32F446ZE.

TABLE A.5 – Mesures de latence des opérations au format binaire FP32 au sein du STM32F446ZE

	Latence mesurée [ms]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
16	0,001090	0,001090	0,001090	0,001902	0,308842
24	0,000727	0,000727	0,000727	0,001268	0,205745
32	0,000545	0,000545	0,000545	0,000951	0,155864
48	0,000367	0,000367	0,000367	0,000637	0,103916
64	0,000276	0,000276	0,000276	0,000478	0,078702
80	0,000220	0,000220	0,000220	0,000382	0,062946
90	0,000196	0,000196	0,000196	0,000340	0,055952
100	0,000182	0,000182	0,000182	0,000311	0,050951
116	0,000157	0,000157	0,000157	0,000268	0,043910
124	0,000146	0,000146	0,000146	0,000251	0,041642
132	0,000138	0,000138	0,000138	0,000236	0,039119
148	0,000123	0,000123	0,000123	0,000210	0,034889
164	0,000111	0,000111	0,000111	0,000190	0,032025
180	0,000101	0,000101	0,000101	0,000173	0,029178

TABLE A.6 – Mesures de latence des opérations au format binaire INT8 au sein du STM32F446ZE

	Latence mesurée [ms]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
16	0,000775	0,000775	0,000775	0,000970	0,308645
24	0,000517	0,000517	0,000517	0,000647	0,205663
32	0,000387	0,000387	0,000387	0,000485	0,155643
48	0,000262	0,000262	0,000262	0,000327	0,103717
64	0,000197	0,000197	0,000197	0,000246	0,078659
80	0,000158	0,000158	0,000158	0,000197	0,062909
90	0,000140	0,000140	0,000140	0,000175	0,056046
100	0,000130	0,000130	0,000130	0,000162	0,050788
116	0,000112	0,000112	0,000112	0,000140	0,043780
124	0,000105	0,000105	0,000105	0,000131	0,041339
132	0,000099	0,000099	0,000099	0,000123	0,038843
148	0,000088	0,000088	0,000088	0,000110	0,034639
164	0,000080	0,000080	0,000080	0,000099	0,031716
180	0,000072	0,000072	0,000072	0,000090	0,028898

TABLE A.7 – Mesures de courant consommé par les opérations au format binaire FP32 au sein du STM32F446ZE

	Courant mesuré [mA]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
16	6,414874	7,154878	5,919171	6,368279	9,194328
24	8,820721	9,465005	8,312230	8,669705	12,374192
32	10,264234	10,839382	9,724247	10,020481	12,676232
48	15,336103	15,778759	14,794673	12,745616	16,741383
64	15,845778	16,290550	15,327540	15,070140	20,236252
80	18,829308	19,237612	18,364482	17,805676	24,262934
90	20,461568	20,851981	20,005690	19,527229	26,747463
100	25,963434	26,240811	25,465008	20,708074	28,372388
116	29,277575	29,649488	28,973213	23,342578	32,137700
124	33,928936	34,354331	33,805238	27,127814	37,168146
132	35,817681	36,180071	35,718971	28,617391	39,217921
148	35,657798	36,154354	35,644410	33,114500	45,062044
164	39,063444	39,626626	39,080855	36,329532	47,355390
180	49,091618	49,751471	49,253390	39,431779	52,600124

TABLE A.8 – Mesures de courant consommé par les opérations au format binaire INT8 au sein du STM32F446ZE

	Courant mesuré [mA]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
16	5,446945	5,746444	5,403839	5,018181	9,925802
24	7,412039	8,011296	7,706801	7,090797	13,402715
32	8,562254	9,380514	9,034409	8,204850	13,765228
48	12,061129	12,752513	12,249379	10,947144	17,605702
64	13,456946	15,076113	14,412360	12,733501	21,166258
80	16,180083	18,140705	17,296972	15,168419	25,016731
90	17,889423	19,844607	18,881107	16,481843	27,604846
100	20,625996	21,960244	20,919794	18,157687	29,500233
116	23,511561	25,047567	23,850763	20,628549	33,022285
124	27,871797	29,901050	28,379357	24,446895	38,339421
132	29,512394	31,713318	30,081552	25,864963	40,451270
148	32,524440	36,196686	34,150876	29,347047	45,574307
164	35,911009	39,850812	37,601069	32,269221	47,835775
180	42,320855	45,229819	42,764321	36,670812	53,191006

Les Tableaux A.9 et A.10 référencent les mesures de latence des opérations au sein du STM32F746ZG. Les Tableaux A.11 et A.12 référencent les mesures de courant consommé par les opérations au sein du STM32F746ZG.

TABLE A.9 – Mesures de latence des opérations au format binaire FP32 au sein du STM32F746ZG

	Latence mesurée [ms]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,2	0,140772	0,140772	0,140772	0,183380	17,750984
0,4	0,058953	0,058953	0,058953	0,078797	7,574324
0,8	0,027124	0,027124	0,027124	0,036305	3,537912
1	0,022123	0,022123	0,022123	0,030142	2,767805
4	0,005297	0,005297	0,005297	0,007173	0,665446
16	0,001312	0,001312	0,001312	0,001781	0,164903
48	0,000527	0,000527	0,000527	0,000619	0,061003
80	0,000316	0,000316	0,000316	0,000373	0,040376
100	0,000307	0,000307	0,000307	0,000309	0,035468
116	0,000232	0,000232	0,000232	0,000271	0,030144
148	0,000198	0,000198	0,000198	0,000203	0,025628
180	0,000183	0,000183	0,000183	0,000185	0,023275
200	0,000196	0,000196	0,000196	0,000196	0,022608
216	0,000193	0,000193	0,000193	0,000194	0,022478

TABLE A.10 – Mesures de latence des opérations au format binaire INT8 au sein du STM32F746ZG

	Latence mesurée [ms]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,2	0,084592	0,085739	0,088885	0,116713	19,482372
0,4	0,033948	0,030593	0,037117	0,048900	7,915241
0,8	0,015632	0,015966	0,016792	0,022171	3,637699
1	0,020322	0,020255	0,020316	0,020774	2,851708
4	0,004825	0,004824	0,004824	0,004896	0,679631
16	0,001268	0,001266	0,001266	0,001266	0,168993
48	0,000439	0,000442	0,000442	0,000450	0,062170
80	0,000159	0,000166	0,000173	0,000221	0,040979
100	0,000260	0,000263	0,000263	0,000266	0,035559
116	0,000122	0,000232	0,000232	0,000261	0,030598
148	0,000100	0,000197	0,000197	0,000204	0,025867
180	0,000083	0,000088	0,000090	0,000111	0,022954
200	0,000168	0,000171	0,000171	0,000171	0,022238
216	0,000167	0,000169	0,000169	0,000169	0,022022

TABLE A.11 – Mesures de courant consommé par les opérations au format binaire FP32 au sein du STM32F746ZG

	Courant mesuré [mA]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,2	9,287934	9,356515	9,294716	9,455247	9,503155
0,4	9,411572	9,538681	9,553593	9,425359	9,555749
0,8	9,557830	9,773965	9,684352	9,674752	9,774305
1	7,884157	8,063682	8,077381	8,047310	7,998697
4	9,601533	9,759508	9,556045	9,548885	9,666959
16	15,669964	15,760905	15,886912	15,193979	15,826101
48	34,845741	34,852393	34,782531	33,885790	34,458329
80	41,714924	41,765817	41,862122	41,851534	44,716004
100	51,960874	52,923331	51,883312	52,350026	51,637540
116	54,239167	54,161674	54,358743	54,489065	57,150498
148	70,224246	70,242442	70,322241	70,478427	73,521324
180	84,690869	84,665094	84,969223	83,847208	88,763251
200	95,194437	95,575874	95,382133	94,884482	94,948755
216	99,493996	99,622092	99,382830	99,042781	98,867373

TABLE A.12 – Mesures de courant consommé par les opérations au format binaire INT8 au sein du STM32F746ZG

	Courant mesuré [mA]				
Fréquence [MHz]	**Addition**	**Soustraction**	**Multiplication**	**Division**	**Exponentielle**
0,2	9,537926	9,126720	9,376151	9,195970	9,207830
0,4	9,575535	9,224291	9,449468	9,536192	9,056054
0,8	9,748395	9,482935	9,607149	9,543734	9,189218
1	8,108052	7,888482	7,932161	8,028033	7,398818
4	9,590174	9,332141	9,508106	9,449013	9,007828
16	14,009474	14,918387	15,121773	14,964245	15,005099
48	33,003197	32,784269	32,981952	32,865762	33,466995
80	41,420930	42,107991	40,479976	35,812226	42,733018
100	49,051732	49,080487	49,102398	49,102465	50,825774
116	55,180041	56,451995	54,133485	47,959662	55,908170
148	73,851230	76,000545	72,723110	64,188278	72,692361
180	89,744011	92,357314	88,320170	78,685805	87,285484
200	90,947209	91,736261	91,436894	91,180570	95,267912
216	94,970515	95,846001	95,531745	95,187408	99,319962

Annexe B

Mesures en latence et en consommation énergétique, LeNet5

B.1 Estimation du coût d'inférence de LeNet5 élagué sur le STM32F446ZE et STM32F746ZG

Estimation de la latence :

La comparaison entre l'estimation et la mesure de la latence de LeNet5 en FP32 aux différents pourcentages d'élagage est présentée sur la Figure B.1 pour le STM32F446ZE et Figure B.3 pour le STM32F746ZG.

Estimation de la consommation d'énergie :

La comparaison entre l'estimation et la mesure de la consommation d'énergie pour une inférence de LeNet5 en FP32 et aux différents pourcentages d'élagage est présentée sur la Figure B.5 pour le STM32F446ZE, Figure B.7 pour le STM32F746ZG.

Sur la Figure B.5 LeNet5 élagué à 90% et implémenté dans ce MCU offre la réduction énergétique la plus importante par rapport au réseau non élagué, en moyenne sur les 14 fréquences de fonctionnement, une réduction de 78% de joule par inférence, d'après l'estimation et la mesure. Quels que soient les taux d'élagage, l'estimation montre en corrélation avec les mesures que les fréquences de fonctionnement offrant la consommation énergétique la plus faible sont celles de milieu de plage de fréquence pour le STM32F446ZE, à savoir de 100MHz à 132MHz.

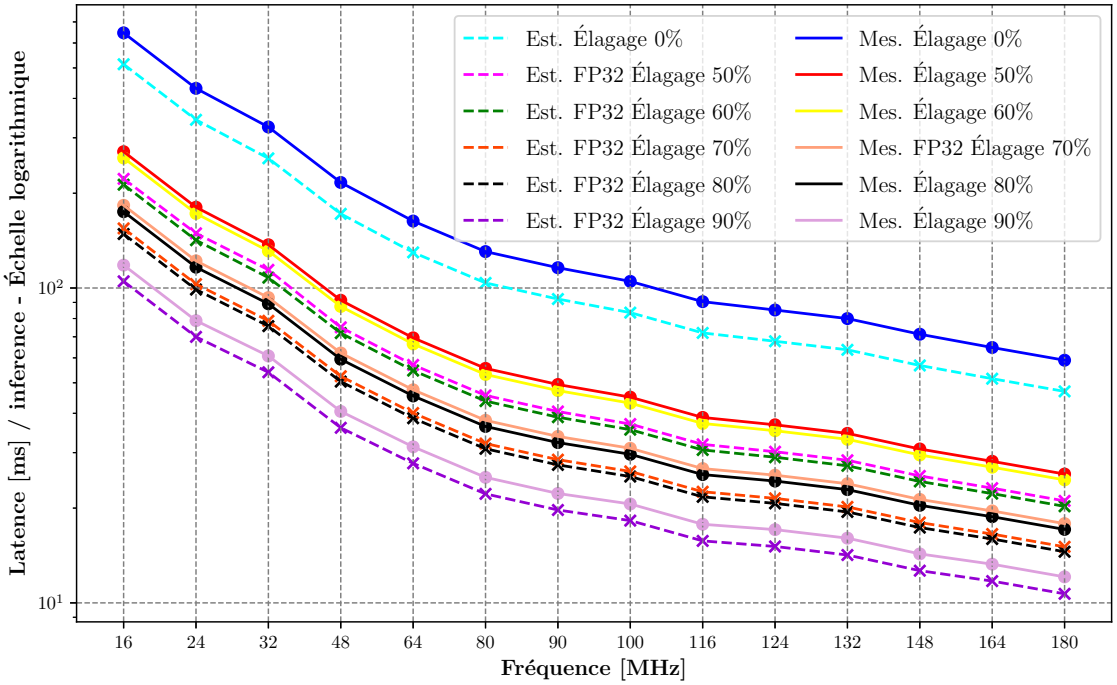


FIGURE B.1 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F446ZE. Élagage à : 0% (cyan *vs.* bleu),50% (rose *vs.* rouge), 60% (vert *vs.* jaune), 70% (orange *vs.* saumon), 80% (gris *vs.* noir) et 90% (violet foncé vs violet clair). Échelle logarithmique sur l'axe Y.

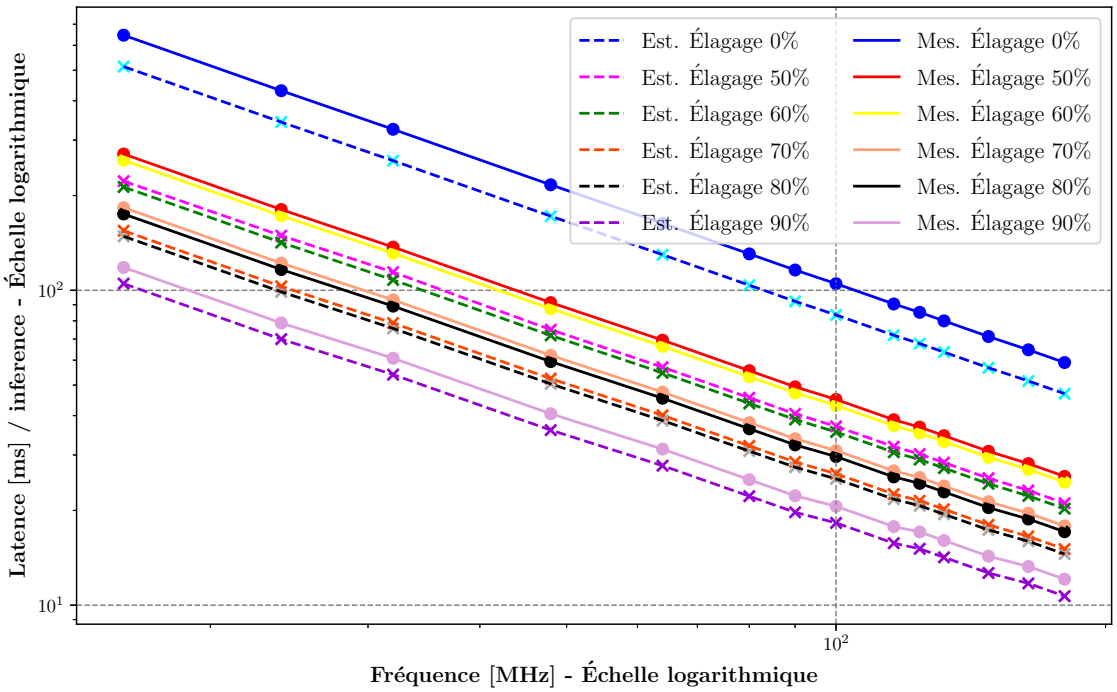


FIGURE B.2 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F446ZE, échelle logarithmique.

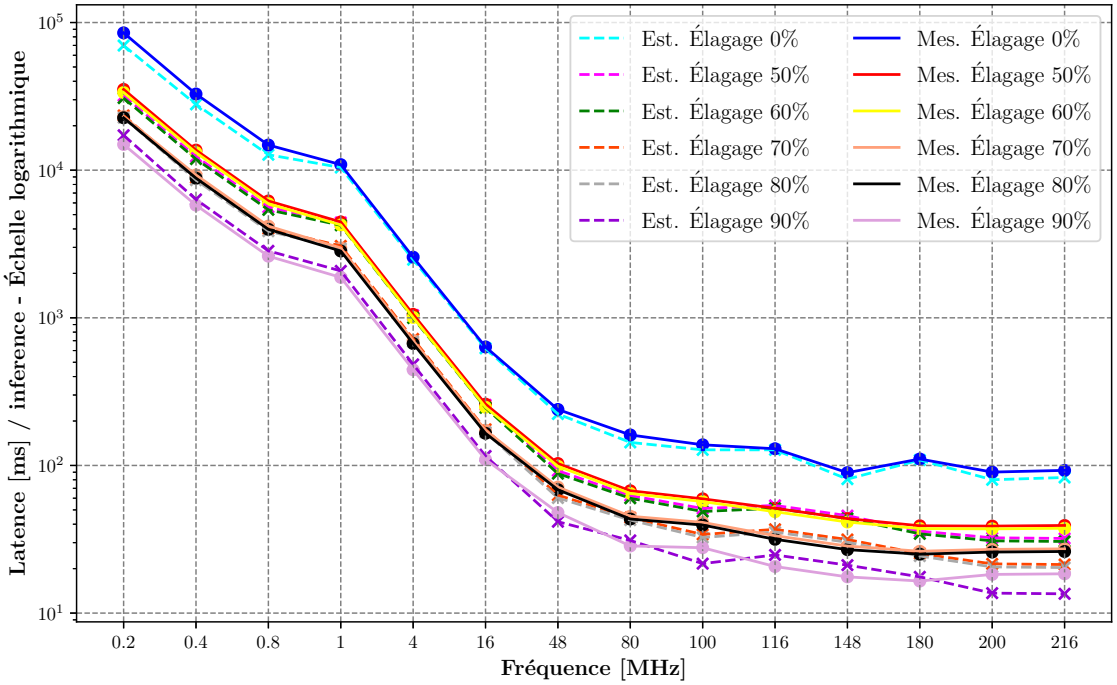


FIGURE B.3 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F746ZG. Élagage 0% (cyan *vs.* bleu), 50% (rose *vs.* rouge), 60% (vert *vs.* jaune), 70% (orange *vs.* saumon), 80% (gris *vs.* noir) et 90% (violet foncé vs violet clair). Échelle logarithmique sur l'axe Y.

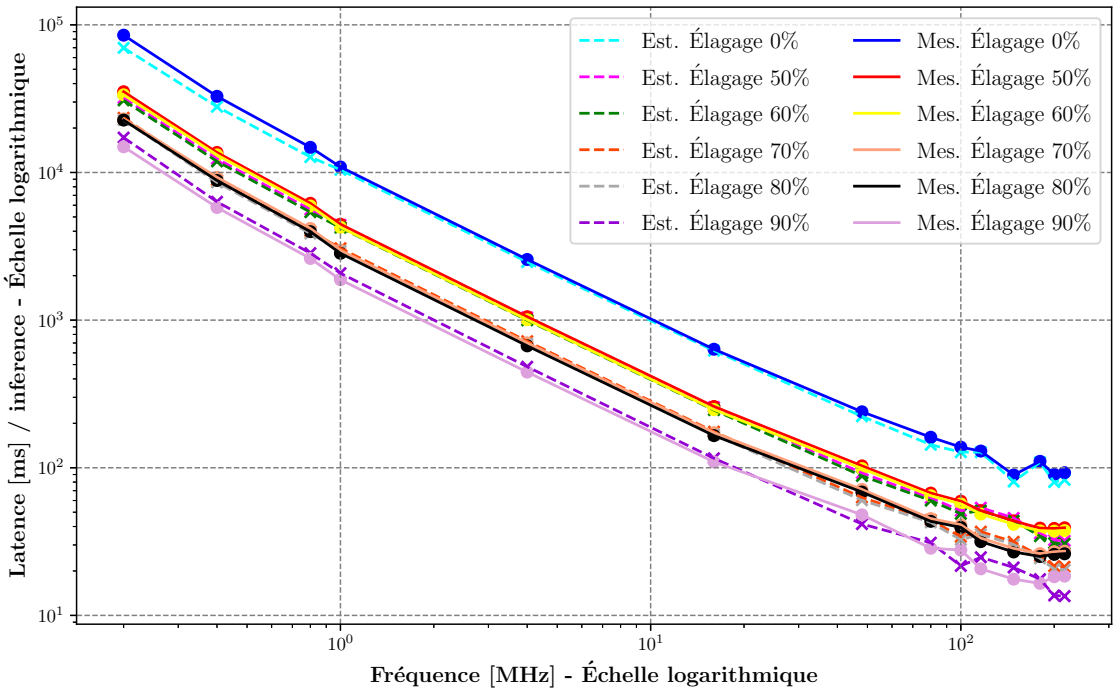


FIGURE B.4 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F746ZG, échelle logarithmique.

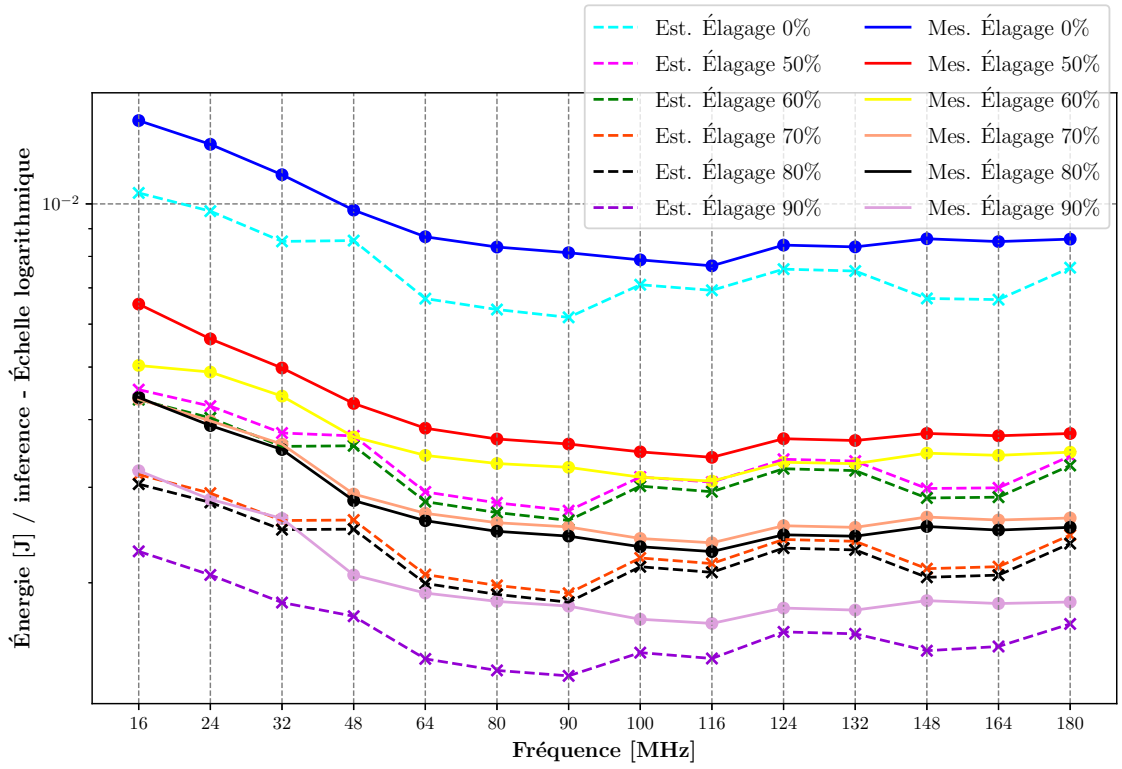


FIGURE B.5 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F446ZE. Élagage à : 0% (cyan *vs.* bleu),50% (rose *vs.* rouge), 60% (vert *vs.* jaune), 70% (orange *vs.* saumon), 80% (gris *vs.* noir) et 90% (violet foncé vs violet clair). Échelle logarithmique sur l'axe Y.

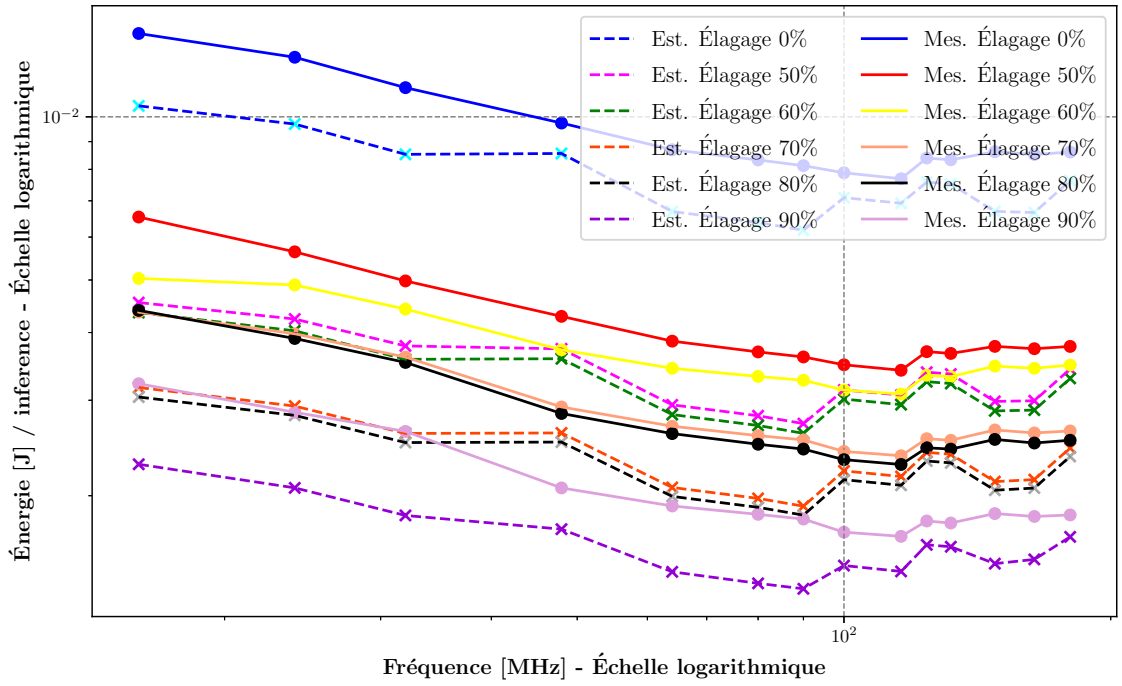


FIGURE B.6 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F446ZE, échelle logarithmique.

Pour mettre en avant l'influence de l'erreur de la latence dans l'estimation de l'énergie, le Tableau B.1 indique la différence entre l'estimation et la mesure de la puissance, de l'énergie et de la latence.

TABLE B.1 – Comparaison entre l'estimation de la puissance et de l'énergie consommée pour une inférence de LeNet5 à taux d'élagage varié au sein du STM32F446ZE. La comparaison est faite entre la différence des estimations et des mesures de puissance et de consommation d'énergie sur une inférence. La différence de l'estimation de la latence est également référencée, l'énergie étant directement liée à ce paramètre.

Taux d'élagage	**STM32F446ZE**		
	Diff. Puissance [%]	Diff. Énergie [%]	Diff.Latence [%]
0%	7,76%	18,10%	20,45%
50%	8,60%	17,93%	17,77%
60%	8,15%	11,70%	17,66%
70%	9,62%	16,81%	15,59%
80%	10,07%	17,25%	15,05%
90%	9,87%	19,21%	11,42%

Sur la Figure B.7, LeNet5 élagué à 90% offre de nouveau au sein de ce MCU la réduction énergétique la plus importante par rapport au réseau non élagué, en moyenne sur les 14 fréquences de fonctionnement, une réduction de 80% de joule par inférence, d'après l'estimation et la mesure. Comme le STM32F446ZE, quel que soit le taux d'élagage, les fréquences de fonctionnement du milieu de la plage de fonctionnement du STM32F746ZG offrent les plus faibles consommations énergétiques, à savoir de 48MHz à 148MHz.

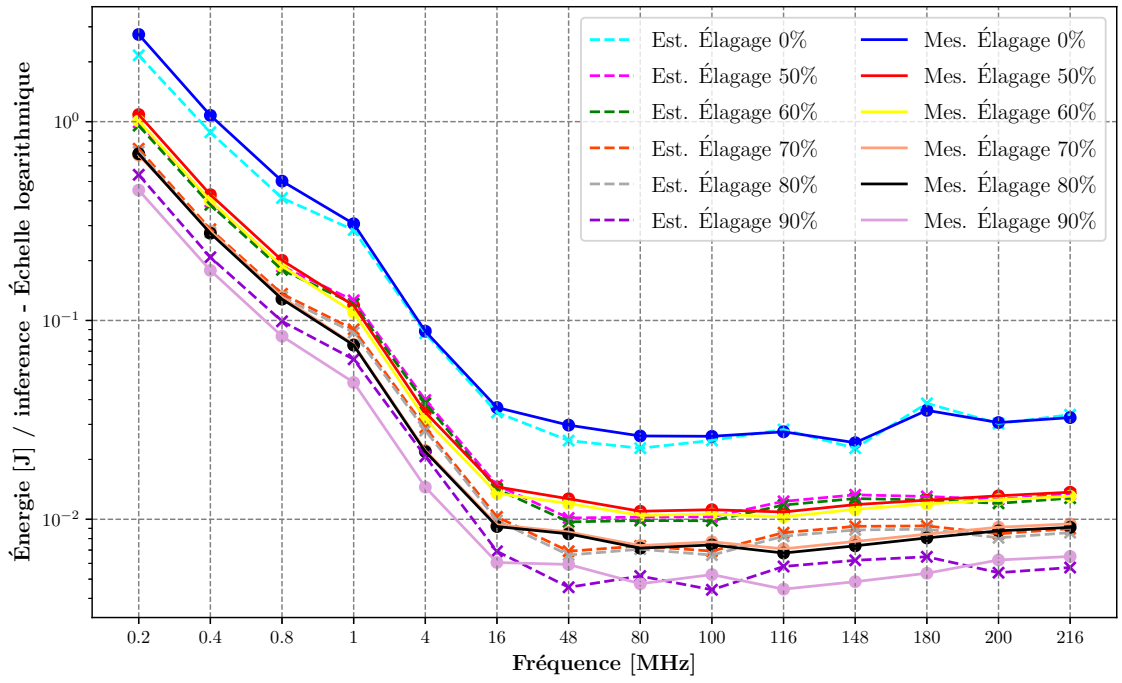


FIGURE B.7 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F746ZG. Élagage 0% (cyan *vs.* bleu), 50% (rose *vs.* rouge), 60% (vert *vs.* jaune), 70% (orange *vs.* saumon), 80% (gris *vs.* noir) et 90% (violet foncé vs violet clair). Échelle logarithmique sur l'axe Y.

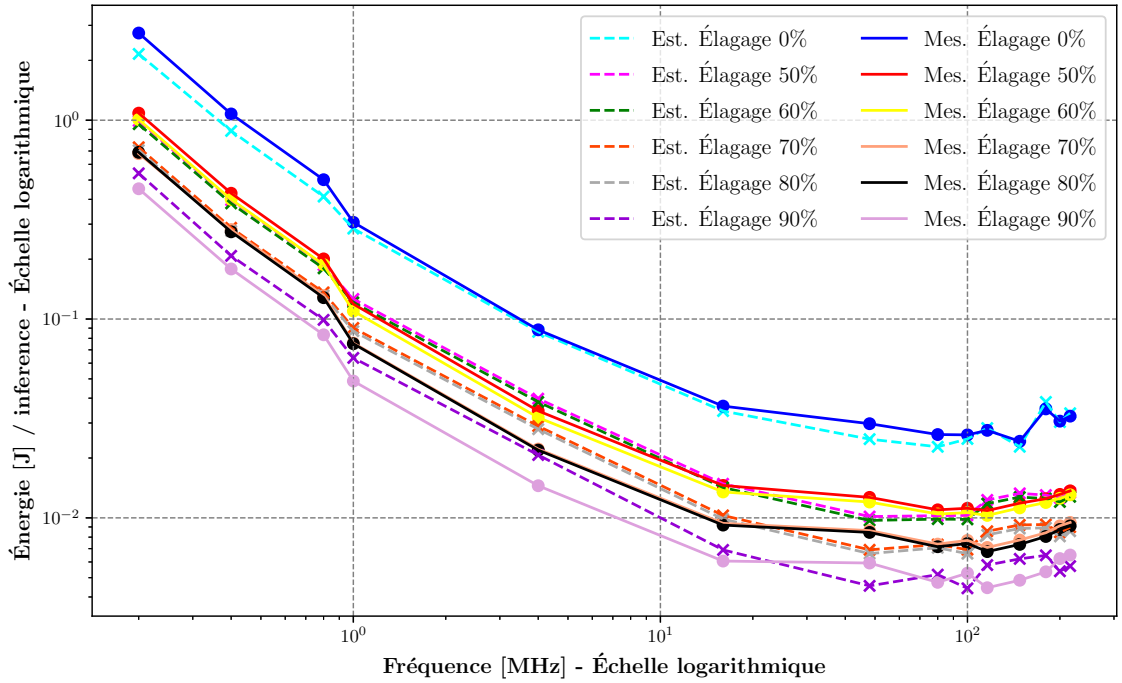


FIGURE B.8 – Estimations *vs.* mesures. LeNet5 en FP32 au sein du STM32F746ZG, échelle logarithmique.

Pour mettre en avant l'influence de l'erreur de la latence dans l'estimation de l'énergie, le Tableau B.2 réfère la différence entre l'estimation et la mesure de la puissance, de l'énergie et de la latence.

TABLE B.2 – Comparaison entre l'estimation de la puissance et de l'énergie consommée pour une inférence de LeNet5 à taux d'élagage varié au sein du STM32F746ZG. La comparaison est faite entre la différence des estimations et des mesures de puissance et de consommation d'énergie sur une inférence. La différence de l'estimation de la latence est également référencée, l'énergie étant directement liée à ce paramètre.

Taux d'élagage	**STM32F746ZG**		
	Diff. Puissance [%]	Diff. Énergie [%]	Diff.Latence [%]
0%	5,79%	9,13%	8,38%
50%	8,13%	8,31%	8,23%
60%	9,74%	8,64%	8,29%
70%	11,33%	11,83%	8,21%
80%	9,91%	11,82%	8,26%
90%	11,46%	21,13%	14,38%

B.2 Estimations et mesures du coût d'inférence de LeNet5 sur les MCU

TABLE B.3 – Mesures de la latence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - LATENCE [ms/inférence]

	LeNet FP32 Sig.			LeNet FP32		LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	212562,9300	228876,1738	7,6745	115444,4100	116589,7246	0,9921	59525,4368	51941,7209	12,7403	56818,7543	49887,4389	12,1990
0,2	89079,6650	95699,5061	7,4314	49006,1100	48872,7399	0,2722	24959,7350	21744,7267	12,8808	23826,6651	20882,7214	12,3557
0,4	41176,9025	44206,8283	7,3225	22645,5650	22585,1579	0,2668	11526,8273	10050,0646	12,8115	11002,7750	9651,7521	12,2789
0,8	19838,7138	21291,3960	7,3225	10916,1088	10881,9825	0,3126	5554,1939	4842,4412	12,8147	5301,6070	4650,5301	12,2807
1,0	15759,6940	16914,0194	7,3245	8670,3270	8640,1307	0,3483	4411,8066	3843,5458	12,8805	4211,2303	3691,1297	12,3503
2	7767,3480	8333,7694	7,2923	4272,4540	4305,8780	0,7823	2174,6184	1893,7869	12,9141	2075,6843	1818,7008	12,3807
4	3856,4628	4138,2648	7,3073	2121,0738	2113,7549	0,3451	1079,6522	940,2865	12,9084	1030,5465	902,9985	12,3767
8	1921,5760	2061,7970	7,2972	1056,8754	1053,2060	0,3472	537,9374	468,4973	12,9086	513,4679	449,9176	12,3767
16	959,1167	1029,0590	7,2924	527,5145	525,6492	0,3536	268,5001	233,8262	12,9139	256,2877	224,5533	12,3823
24	639,0351	685,6538	7,2952	351,4650	350,2326	0,3507	178,8949	155,7927	12,9138	170,7581	149,6142	12,3824
32	479,1386	514,0839	7,2934	263,5241	262,5854	0,3562	134,1318	116,8064	12,9167	128,0311	112,1741	12,3852
48	335,2523	360,4951	7,5295	175,8370	175,2102	0,3564	91,9372	78,0436	15,1121	87,8593	74,9560	14,6862
64	259,7158	281,3524	8,3309	131,9541	131,4744	0,3635	69,9324	58,6132	16,1860	66,8686	56,2979	15,8081
80	214,5681	233,5361	8,8401	105,6425	105,2548	0,3670	56,7331	46,9662	17,2156	54,2765	45,1141	16,8810
		Moyenne	7,5395		**Moyenne**	0,4153		**Moyenne**	13,5798		**Moyenne**	13,0803

TABLE B.4 – Mesures de la latence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - LATENCE [ms/inférence]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	40185,7736	36966,8753	8,0100	38360,9348	35597,2980	7,2043	25947,5958	25986,7154	0,1508	110383,4318	106785,4780	3,2595
0,2	16862,5147	15444,0970	8,4117	16097,2450	14886,3501	7,5224	10901,1619	10853,6912	0,4355	46031,7446	44521,9836	3,2798
0,4	7781,1112	7138,6723	8,2564	7427,3986	6880,9596	7,3571	5024,1728	5017,5317	0,1322	21283,9399	20584,9794	3,2840
0,8	3749,3409	3439,7075	8,2583	3578,9070	3315,5373	7,3589	2421,0041	2417,7238	0,1355	10256,0324	9918,6985	3,2891
1	2978,3851	2729,4929	8,3566	2843,0843	2630,8776	7,4640	1923,3993	1917,8303	0,2895	8145,6902	7877,8651	3,2879
2	1468,0449	1344,9581	8,3844	1401,3300	1296,3769	7,4895	948,0237	945,0995	0,3084	4015,0272	3931,8951	2,0705
4	728,8523	667,7367	8,3852	695,7338	643,6111	7,4918	470,6635	469,1660	0,3182	1993,3575	1927,7825	3,2897
8	363,1463	332,6927	8,3860	346,6453	320,6715	7,4929	234,5037	233,7499	0,3214	993,1801	960,4758	3,2929
16	181,2596	166,0472	8,3926	173,0231	160,0476	7,4993	117,0510	116,6657	0,3292	495,7194	479,3766	3,2968
24	120,7685	110,6322	8,3932	115,2808	106,6346	7,5001	77,9874	77,7297	0,3305	330,2864	319,4227	3,2892
32	90,5498	82,9477	8,3955	86,4353	79,9506	7,5023	58,4734	58,2794	0,3317	247,6425	239,4947	3,2901
48	62,8760	55,4756	11,7698	60,1226	53,4780	11,0518	41,4958	39,0334	5,9341	167,6022	162,5001	3,0442
64	48,1338	41,6903	13,3866	46,0603	40,1923	12,7399	32,1015	29,3608	8,5376	126,6746	122,8576	3,0132
80	39,2903	33,4284	14,9197	37,6265	32,2300	14,3425	26,4657	23,5652	10,9596	102,1278	99,1281	2,9372
		Moyenne	9,4076		**Moyenne**	8,5726		**Moyenne**	2,0367		**Moyenne**	3,1374

TABLE B.5 – Mesures de la consommation d'énergie pour une inférence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - ÉNERGIE [J/inférence]

	LeNet FP32 Sig.			**LeNet FP32**		**LeNet FP32 50%**			**LeNet FP32 60%**		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	0,1040	0,1104	6,1416	0,0558	0,0551	1,1702	0,0285	0,0246	13,7033	0,0270	0,0236	12,5246
0,2	0,0482	0,0512	6,2430	0,0259	0,0252	2,5524	0,0131	0,0112	14,2391	0,0125	0,0108	13,6143
0,4	0,0264	0,0283	7,4945	0,0141	0,0136	3,4952	0,0071	0,0060	15,0595	0,0068	0,0058	14,3952
0,8	0,0164	0,0181	9,8300	0,0088	0,0084	4,4502	0,0045	0,0037	16,0745	0,0043	0,0036	15,3334
1,0	0,0147	0,0163	11,0473	0,0079	0,0075	4,7002	0,0040	0,0033	16,4461	0,0038	0,0032	15,7266
2	0,0109	0,0124	13,0214	0,0059	0,0056	4,6136	0,0030	0,0025	17,7619	0,0028	0,0024	16,8679
4	0,0091	0,0105	15,5916	0,0049	0,0046	6,3973	0,0025	0,0020	18,6605	0,0024	0,0020	17,7162
8	0,0082	0,0096	16,7940	0,0044	0,0041	7,0226	0,0023	0,0018	19,3846	0,0021	0,0017	18,4423
16	0,0078	0,0092	17,5178	0,0042	0,0039	7,3161	0,0022	0,0017	19,6834	0,0021	0,0017	18,6473
24	0,0076	0,0090	17,1828	0,0041	0,0038	7,5196	0,0021	0,0017	19,8436	0,0020	0,0016	18,8392
32	0,0076	0,0088	16,9940	0,0041	0,0038	7,5758	0,0021	0,0017	19,8676	0,0020	0,0016	18,9492
48	0,0078	0,0081	3,8929	0,0040	0,0036	10,1082	0,0021	0,0016	23,3716	0,0020	0,0015	22,5407
64	0,0080	0,0082	1,5032	0,0040	0,0037	8,1020	0,0021	0,0016	22,3599	0,0020	0,0016	21,5833
80	0,0083	0,0081	1,5013	0,0039	0,0036	7,8143	0,0021	0,0016	22,7488	0,0020	0,0016	21,9989
		Moyenne	10,3397		**Moyenne**	5,9170		**Moyenne**	18,5146		**Moyenne**	17,6556

TABLE B.6 – Mesures de la consommation d'énergie pour une inférence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - ÉNERGIE [J/inférence]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	0,0193	0,0175	9,4115	0,0185	0,0168	9,2665	0,0126	0,0123	2,8530	0,0532	0,0511	3,9230
0,2	0,0089	0,0080	10,5762	0,0086	0,0077	10,2723	0,0059	0,0056	4,4047	0,0243	0,0232	4,1626
0,4	0,0049	0,0043	11,5962	0,0047	0,0041	11,0990	0,0032	0,0030	5,7411	0,0131	0,0125	4,2443
0,8	0,0031	0,0027	12,8004	0,0029	0,0026	12,3691	0,0020	0,0019	7,4469	0,0081	0,0077	4,4593
1	0,0027	0,0024	13,3166	0,0026	0,0023	12,7567	0,0018	0,0017	8,2641	0,0072	0,0069	4,4147
2	0,0021	0,0018	14,8278	0,0020	0,0017	14,2304	0,0014	0,0012	10,2277	0,0053	0,0051	3,5399
4	0,0017	0,0014	15,8798	0,0016	0,0014	15,1489	0,0012	0,0010	11,6971	0,0044	0,0042	4,9570
8	0,0016	0,0013	16,7374	0,0015	0,0012	15,9360	0,0010	0,0009	12,7704	0,0040	0,0038	5,1806
16	0,0015	0,0012	17,0393	0,0014	0,0012	16,2644	0,0010	0,0009	13,1938	0,0038	0,0036	5,2090
24	0,0015	0,0012	17,2125	0,0014	0,0012	16,4286	0,0010	0,0008	13,3818	0,0037	0,0035	5,2552
32	0,0014	0,0012	17,2403	0,0014	0,0011	16,4296	0,0010	0,0008	13,3606	0,0036	0,0035	5,2115
48	0,0014	0,0011	21,2120	0,0014	0,0011	20,4560	0,0010	0,0008	18,1257	0,0036	0,0035	4,2303
64	0,0015	0,0012	20,3036	0,0014	0,0011	19,5546	0,0010	0,0008	17,4454	0,0036	0,0034	5,1596
80	0,0015	0,0012	20,9079	0,0014	0,0011	20,1221	0,0010	0,0008	18,3408	0,0036	0,0034	5,1948
		Moyenne	15,6473		**Moyenne**	15,0239		**Moyenne**	11,2324		**Moyenne**	4,6530

TABLE B.7 – Mesures de la puissance pour une inférence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - PUISSANCE [W]

	LeNet FP32 Sig.			LeNet FP32			LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	0,0005	0,0005	1,4237	0,0005	0,0005	2,1411	0,0005	0,0005	1,1037	0,0005	0,0005	0,3709
0,2	0,0005	0,0005	1,1062	0,0005	0,0005	2,2865	0,0005	0,0005	1,5591	0,0005	0,0005	1,4361
0,4	0,0006	0,0006	0,1268	0,0006	0,0006	3,2370	0,0006	0,0006	2,5782	0,0006	0,0006	2,4125
0,8	0,0008	0,0008	2,3365	0,0008	0,0008	4,1505	0,0008	0,0008	3,7389	0,0008	0,0008	3,4800
1,0	0,0009	0,0010	3,4687	0,0009	0,0009	4,3671	0,0009	0,0009	4,0928	0,0009	0,0009	3,8520
2	0,0014	0,0015	5,3397	0,0014	0,0013	5,3540	0,0014	0,0013	5,5668	0,0014	0,0013	5,1212
4	0,0024	0,0025	7,7202	0,0023	0,0022	6,0732	0,0023	0,0022	6,6047	0,0023	0,0022	6,0936
8	0,0043	0,0046	8,8509	0,0042	0,0039	6,6987	0,0042	0,0039	7,4359	0,0042	0,0039	6,9224
16	0,0082	0,0089	9,5305	0,0080	0,0074	6,9872	0,0081	0,0074	7,7733	0,0080	0,0074	7,1503
24	0,0120	0,0131	9,2154	0,0117	0,0109	7,1941	0,0118	0,0109	7,9574	0,0117	0,0109	7,3693
32	0,0158	0,0172	9,0412	0,0154	0,0143	7,2454	0,0156	0,0144	7,9819	0,0155	0,0144	7,4919
48	0,0232	0,0225	3,3819	0,0227	0,0205	9,7867	0,0228	0,0205	9,7299	0,0226	0,0205	9,2065
64	0,0310	0,0290	6,3026	0,0301	0,0278	7,7667	0,0300	0,0278	7,3662	0,0299	0,0278	6,8595
80	0,0385	0,0348	9,5015	0,0373	0,0345	7,4748	0,0370	0,0346	6,6839	0,0368	0,0346	6,1573
		Moyenne	5,5247		**Moyenne**	5,7688		**Moyenne**	5,7266		**Moyenne**	5,2803

TABLE B.8 – Mesures de la puissance pour une inférence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - PUISSANCE [W]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	0,0005	0,0005	1,5235	0,0005	0,0005	2,2223	0,0005	0,0005	2,9993	0,0005	0,0005	0,6859
0,2	0,0005	0,0005	2,3634	0,0005	0,0005	2,9736	0,0005	0,0005	3,9866	0,0005	0,0005	0,9128
0,4	0,0006	0,0006	3,6404	0,0006	0,0006	4,0391	0,0006	0,0006	5,6164	0,0006	0,0006	0,9929
0,8	0,0008	0,0008	4,9509	0,0008	0,0008	5,4081	0,0008	0,0008	7,3213	0,0008	0,0008	1,2099
1	0,0009	0,0009	5,4123	0,0009	0,0009	5,7196	0,0009	0,0009	7,9977	0,0009	0,0009	1,1651
2	0,0014	0,0013	7,0331	0,0014	0,0013	7,2866	0,0014	0,0013	9,9499	0,0013	0,0013	1,5004
4	0,0024	0,0022	8,1806	0,0024	0,0022	8,2772	0,0025	0,0022	11,4153	0,0022	0,0022	1,7240
8	0,0043	0,0039	9,1158	0,0043	0,0039	9,1270	0,0045	0,0039	12,4891	0,0040	0,0039	1,9520
16	0,0082	0,0074	9,4389	0,0082	0,0074	9,4757	0,0086	0,0075	12,9071	0,0076	0,0075	1,9774
24	0,0120	0,0109	9,6274	0,0121	0,0109	9,6524	0,0126	0,0109	13,0946	0,0112	0,0109	2,0328
32	0,0159	0,0144	9,6554	0,0159	0,0144	9,6514	0,0166	0,0144	13,0723	0,0147	0,0144	1,9867
48	0,0230	0,0206	10,7019	0,0230	0,0206	10,5727	0,0236	0,0206	12,9608	0,0216	0,0214	1,2234
64	0,0302	0,0278	7,9861	0,0302	0,0278	7,8097	0,0308	0,0278	9,7392	0,0286	0,0280	2,2131
80	0,0372	0,0346	7,0384	0,0371	0,0346	6,7474	0,0377	0,0346	8,2898	0,0354	0,0346	2,3259
		Moyenne	6,9049		**Moyenne**	7,0688		**Moyenne**	9,4171		**Moyenne**	1,5644

TABLE B.9 – Mesures du courant pour une inférence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - COURANT [mA]

	LeNet FP32 Sig.			**LeNet FP32**			**LeNet FP32 50%**			**LeNet FP32 60%**		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	0,1480	0,1459	1,4237	0,1462	0,1430	2,1411	0,1446	0,1430	1,1037	0,1436	0,1430	0,3709
0,2	0,1637	0,1619	1,1062	0,1597	0,1560	2,2865	0,1585	0,1561	1,5591	0,1583	0,1561	1,4361
0,4	0,1938	0,1940	0,1268	0,1881	0,1821	3,2370	0,1869	0,1821	2,5782	0,1866	0,1821	2,4125
0,8	0,2508	0,2567	2,3365	0,2442	0,2341	4,1505	0,2433	0,2342	3,7389	0,2427	0,2343	3,4800
1,0	0,2827	0,2925	3,4687	0,2750	0,2630	4,3671	0,2745	0,2632	4,0928	0,2738	0,2633	3,8520
2	0,4257	0,4485	5,3397	0,4154	0,3932	5,3540	0,4168	0,3936	5,5668	0,4149	0,3936	5,1212
4	0,7137	0,7687	7,7202	0,6970	0,6547	6,0732	0,7019	0,6556	43,9261	0,6982	0,6556	6,0936
8	1,2888	1,4028	8,8509	1,2592	1,1749	6,6987	1,2712	1,1767	7,4359	1,2644	1,1768	6,9224
16	2,4681	2,7033	9,5305	2,4150	2,2462	6,9872	2,4395	2,2499	7,7733	2,4234	2,2501	7,1503
24	3,6186	3,9521	9,2154	3,5403	3,2856	7,1941	3,5754	3,2909	7,9574	3,5532	3,2913	7,3693
32	4,7762	5,2080	9,0412	4,6743	4,3357	7,2454	4,7194	4,3427	7,9819	4,6949	4,3432	7,4919
48	7,0319	6,7941	3,3819	6,8805	6,2071	9,7867	6,8840	6,2142	9,7299	6,8449	6,2147	9,2065
64	9,3711	8,7804	6,3026	9,1219	8,4134	7,7667	9,0872	8,4178	7,3662	9,0381	8,4181	6,8595
80	11,6429	10,5367	9,5015	11,2981	10,4536	7,4748	11,2046	10,4557	6,6839	11,1419	10,4559	6,1573
		Moyenne	5,5247		**Moyenne**	5,7688		**Moyenne**	8,3924		**Moyenne**	5,2803

TABLE B.10 – Mesures du courant pour une inférence de LeNet5 sur le STM32L496ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32L496ZG - COURANT [mA]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,1	0,1453	0,1431	1,5235	0,1463	0,1431	2,2223	0,1475	0,1431	2,9993	0,1459	0,1449	0,6859
0,2	0,1599	0,1561	2,3634	0,1609	0,1561	2,9736	0,1626	0,1561	3,9866	0,1594	0,1580	0,9128
0,4	0,1889	0,1821	3,6404	0,1897	0,1821	4,0391	0,1929	0,1821	5,6164	0,1861	0,1842	0,9929
0,8	0,2466	0,2344	4,9509	0,2478	0,2344	5,4081	0,2531	0,2346	7,3213	0,2392	0,2363	1,2099
1	0,2785	0,2634	5,4123	0,2794	0,2634	5,7196	0,2866	0,2637	7,9977	0,2686	0,2655	1,1651
2	0,4237	0,3939	7,0331	0,4249	0,3940	7,2866	0,4380	0,3944	9,9499	0,4023	0,3963	1,5004
4	0,7147	0,6562	8,1806	0,7156	0,6563	8,2772	0,7419	0,6572	11,4153	0,6703	0,6587	1,7240
8	1,2962	1,1780	9,1158	1,2965	1,1782	9,1270	1,3484	1,1800	12,4891	1,2063	1,1827	1,9520
16	2,4873	2,2525	9,4389	2,4887	2,2529	9,4757	2,5909	2,2565	12,9071	2,3069	2,2613	1,9774
24	3,6458	3,2948	9,6274	3,6474	3,2953	9,6524	3,7979	3,3006	13,0946	3,3797	3,3110	2,0328
32	4,8124	4,3478	9,6554	4,8130	4,3485	9,6514	5,0103	4,3553	13,0723	4,4583	4,3698	1,9867
48	6,9640	6,2187	10,7019	6,9547	6,2194	10,5727	7,1518	6,2248	12,9608	6,5485	6,4684	1,2234
64	9,1512	8,4204	7,9861	9,1341	8,4207	7,8097	9,3323	8,4234	9,7392	8,6663	8,4745	2,2131
80	11,2481	10,4565	7,0384	11,2131	10,4565	6,7474	11,4016	10,4564	8,2898	10,7261	10,4766	2,3259
		Moyenne	6,9049		**Moyenne**	7,0688		**Moyenne**	9,4171		**Moyenne**	1,5644

TABLE B.11 – Mesures de la latence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - LATENCE [ms/inférence]

	LeNet FP32 Sig.			**LeNet FP32**		**LeNet FP32 50%**			**LeNet FP32 60%**		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	960,6476	1025,7559	6,7775	646,1610	513,5568	20,5219	270,9652	222,1771	18,0053	258,8639	212,9066	17,7534
24	639,7541	683,4384	6,8283	430,5216	342,1849	20,5185	180,5398	149,1638	17,3790	172,4747	141,8593	17,7506
32	482,3763	513,9580	6,5471	324,6242	258,0765	20,4999	137,1694	114,2154	16,7340	131,1177	107,8929	17,7129
48	321,4602	342,5451	6,5591	216,3540	172,0225	20,4902	91,4190	74,9974	17,9629	87,3890	71,9106	17,7120
64	241,9785	257,6258	6,4664	163,1523	129,7501	20,4730	69,4530	56,9955	17,9366	66,4280	54,6810	17,6839
80	193,5658	206,0977	6,4742	130,5096	103,7985	20,4668	55,5580	45,5965	17,9299	53,1377	43,7448	17,6766
90	171,9055	183,1835	6,5606	116,0036	92,2537	20,4734	49,3826	40,5265	17,9336	47,2319	38,8808	17,6811
100	155,4141	165,4310	6,4453	104,9853	83,5343	20,4324	45,0209	36,9756	17,8701	43,0866	35,4943	17,6210
116	133,9705	142,5533	6,4065	90,5005	72,0066	20,4352	38,8095	31,8741	17,8703	37,1426	30,5972	17,6223
124	125,6554	133,7888	6,4728	85,1342	67,7457	20,4248	36,7736	30,2034	17,8666	35,2136	29,0090	17,6198
132	118,0379	125,6881	6,4811	79,9734	63,6409	20,4224	34,5440	28,3738	17,8617	33,0788	27,2517	17,6157
148	105,2726	112,0941	6,4799	71,3259	56,7631	20,4172	30,8088	25,3066	17,8591	29,4950	24,3058	17,5935
164	95,3482	101,4439	6,3930	64,7252	51,5275	20,3904	28,1580	23,1408	17,8179	26,9749	22,2378	17,5612
180	86,8688	92,4161	6,3858	58,9709	46,9447	20,3934	25,6545	21,0813	17,8261	24,5807	20,2587	17,5828
		Moyenne	6,5198		**Moyenne**	20,4543		**Moyenne**	17,7752		**Moyenne**	17,6562

TABLE B.12 – Mesures de la latence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - LATENCE [ms/inférence]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	183,0761	154,6078	15,5500	174,7570	148,4281	15,0660	118,2180	105,0042	11,1775	503,4884	486,1511	3,4435
24	121,9797	102,8958	15,6451	116,4365	98,8970	15,0635	78,7657	69,9639	11,1746	335,4657	323,9297	3,4388
32	93,2547	78,6805	15,6284	89,0955	75,6825	15,0547	60,8505	53,9886	11,2766	253,3313	244,7168	3,4005
48	62,1535	52,4393	15,6294	59,3839	50,4410	15,0596	40,5591	35,9826	11,2835	169,0348	163,2442	3,4257
64	47,4992	40,0814	15,6167	45,4220	38,5829	15,0568	31,3054	27,7428	11,3802	127,5785	123,1945	3,4363
80	37,9956	32,0646	15,6096	36,3350	30,8658	15,0520	25,0421	22,1926	11,3786	102,0577	98,6344	3,3543
90	33,7747	28,4995	15,6186	32,2955	27,4341	15,0529	22,2585	19,7251	11,3817	90,7304	87,6316	3,4154
100	30,9755	26,1504	15,5771	29,6451	25,1914	15,0233	20,6115	18,2528	11,4437	82,1677	79,5279	3,2128
116	26,7020	22,5426	15,5770	25,5552	21,7160	15,0233	17,7680	15,7344	11,4453	70,8315	68,5710	3,1914
124	25,4447	21,4748	15,6022	24,3730	20,7016	15,0633	17,0870	15,1071	11,5877	66,7229	64,4892	3,3476
132	23,9022	20,1737	15,5990	22,8957	19,4473	15,0614	16,0512	14,1909	11,5895	62,6780	60,5767	3,3526
148	21,3106	17,9928	15,5687	20,4127	17,3449	15,0288	14,3078	12,6570	11,5376	55,8963	54,0470	3,3085
164	19,5907	16,5415	15,5644	18,7791	15,9569	15,0285	13,2710	11,7275	11,6303	50,8015	49,1003	3,3487
180	17,8529	15,0697	15,5894	17,1137	14,5371	15,0559	12,0950	10,6851	11,6571	46,2854	44,7395	3,3399
		Moyenne	15,5983		**Moyenne**	15,0493		**Moyenne**	11,4246		**Moyenne**	3,3583

TABLE B.13 – Mesures de la consommation d'énergie pour une inférence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - ÉNERGIE [J/inférence]

	LeNet FP32 Sig.			LeNet FP32			LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	0,0236	0,0223	5,2060	0,0142	0,0105	26,4635	0,0065	0,0045	30,4590	0,0050	0,0044	13,4959
24	0,0213	0,0202	5,1333	0,0129	0,0097	24,7120	0,0056	0,0042	24,8381	0,0049	0,0040	17,6961
32	0,0183	0,0175	4,3790	0,0113	0,0085	24,6696	0,0050	0,0038	24,1313	0,0044	0,0036	19,2222
48	0,0159	0,0165	3,3139	0,0097	0,0086	12,1361	0,0043	0,0037	12,8841	0,0037	0,0036	3,7204
64	0,0142	0,0134	5,4536	0,0087	0,0067	23,0905	0,0039	0,0029	23,7051	0,0034	0,0028	17,8439
80	0,0135	0,0128	5,6318	0,0083	0,0064	23,3151	0,0037	0,0028	23,7519	0,0033	0,0027	18,8064
90	0,0133	0,0125	5,7796	0,0081	0,0062	24,0113	0,0036	0,0027	24,6573	0,0033	0,0026	20,1823
100	0,0127	0,0134	6,0212	0,0079	0,0071	10,0153	0,0035	0,0031	10,1534	0,0031	0,0030	3,7325
116	0,0123	0,0131	6,0332	0,0077	0,0069	9,8935	0,0034	0,0031	9,9954	0,0031	0,0029	4,3998
124	0,0132	0,0143	8,5178	0,0084	0,0076	9,7128	0,0037	0,0034	8,3588	0,0033	0,0032	2,7144
132	0,0131	0,0142	8,7092	0,0083	0,0075	9,7379	0,0037	0,0034	8,3730	0,0033	0,0032	2,8915
148	0,0135	0,0133	1,8336	0,0086	0,0067	22,3957	0,0038	0,0030	20,8510	0,0035	0,0029	17,3297
164	0,0132	0,0132	0,4416	0,0085	0,0067	21,8469	0,0037	0,0030	19,8345	0,0034	0,0029	16,2786
180	0,0134	0,0145	8,1536	0,0086	0,0076	11,4296	0,0038	0,0034	9,1609	0,0035	0,0033	5,5074
		Moyenne	5,3291		**Moyenne**	18,1021		**Moyenne**	17,9396		**Moyenne**	11,7015

TABLE B.14 – Mesures de la consommation d'énergie pour une inférence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - ÉNERGIE [J/inférence]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	0,0044	0,0032	27,4077	0,0044	0,0030	30,7864	0,0032	0,0023	28,9890	0,0123	0,0088	28,9132
24	0,0040	0,0029	26,4379	0,0039	0,0028	27,8650	0,0029	0,0021	27,4678	0,0106	0,0081	23,2745
32	0,0036	0,0026	27,6945	0,0035	0,0025	28,8453	0,0026	0,0018	30,0248	0,0090	0,0071	20,5964
48	0,0029	0,0026	10,4469	0,0028	0,0025	11,4424	0,0021	0,0017	16,0715	0,0075	0,0066	12,6705
64	0,0027	0,0021	22,9066	0,0026	0,0020	23,4150	0,0019	0,0014	24,4012	0,0068	0,0057	16,1256
80	0,0026	0,0020	23,3984	0,0025	0,0019	23,5049	0,0018	0,0014	25,4192	0,0064	0,0055	15,0204
90	0,0025	0,0019	24,5580	0,0024	0,0018	24,4726	0,0018	0,0013	25,6385	0,0063	0,0053	14,9660
100	0,0024	0,0022	7,9455	0,0023	0,0021	8,1542	0,0017	0,0015	13,2173	0,0059	0,0055	7,6338
116	0,0024	0,0022	8,4402	0,0023	0,0021	8,4524	0,0017	0,0014	13,8438	0,0058	0,0054	6,5609
124	0,0025	0,0024	5,6595	0,0025	0,0023	5,4840	0,0018	0,0016	9,6350	0,0062	0,0060	3,2518
132	0,0025	0,0024	5,7545	0,0024	0,0023	5,6674	0,0018	0,0016	9,5867	0,0062	0,0060	2,8514
148	0,0026	0,0021	19,7002	0,0025	0,0020	19,4084	0,0019	0,0015	19,1585	0,0064	0,0060	6,7474
164	0,0026	0,0021	17,9806	0,0025	0,0021	17,4199	0,0018	0,0015	16,6814	0,0063	0,0060	5,7728
180	0,0026	0,0024	6,9937	0,0025	0,0024	6,6578	0,0018	0,0017	8,8824	0,0063	0,0063	0,0236
		Moyenne	16,8089		**Moyenne**	17,2554		**Moyenne**	19,2155		**Moyenne**	11,7434

TABLE B.15 – Mesures de la puissance pour une inférence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - PUISSANCE [W]

	LeNet FP32 Sig.			LeNet FP32			LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	0,0245	0,0218	11,2229	0,0220	0,0204	7,4759	0,0241	0,0204	15,1884	0,0194	0,0205	5,1765
24	0,0332	0,0295	11,1970	0,0299	0,0283	5,2760	0,0312	0,0284	9,0282	0,0284	0,0284	0,0664
32	0,0379	0,0341	10,2548	0,0349	0,0330	5,2449	0,0363	0,0331	8,8839	0,0337	0,0331	1,8342
48	0,0495	0,0480	3,0455	0,0450	0,0497	10,5070	0,0469	0,0498	6,1909	0,0425	0,0498	17,0032
64	0,0586	0,0521	11,1960	0,0533	0,0515	3,2913	0,0555	0,0516	7,0293	0,0517	0,0516	0,1944
80	0,0699	0,0620	11,3699	0,0638	0,0615	3,5812	0,0663	0,0616	7,0940	0,0625	0,0616	1,3724
90	0,0771	0,0682	11,5804	0,0701	0,0669	4,4487	0,0730	0,0670	8,1930	0,0691	0,0670	3,0385
100	0,0814	0,0811	0,3984	0,0751	0,0849	13,0921	0,0775	0,0847	9,3958	0,0727	0,0850	16,8594
116	0,0920	0,0917	0,3508	0,0849	0,0962	13,2492	0,0878	0,0962	9,5884	0,0829	0,0962	16,0511
124	0,1050	0,1070	1,9207	0,0986	0,1119	13,4616	0,1003	0,1119	11,5761	0,0948	0,1119	18,0935
132	0,1107	0,1130	2,0924	0,1042	0,1182	13,4266	0,1060	0,1182	11,5521	0,1003	0,1182	17,8725
148	0,1283	0,1183	7,8076	0,1209	0,1179	2,4861	0,1224	0,1180	3,6424	0,1176	0,1180	0,3201
164	0,1388	0,1299	6,4239	0,1317	0,1292	1,8296	0,1326	0,1294	2,4539	0,1274	0,1294	1,5558
180	0,1541	0,1566	1,6617	0,1460	0,1625	11,2602	0,1470	0,1625	10,5449	0,1417	0,1625	14,6516
		Moyenne	6,4659		**Moyenne**	7,7593		**Moyenne**	8,5972		**Moyenne**	8,1493

TABLE B.16 – Mesures de la puissance pour une inférence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - PUISSANCE [W]

	LeNet FP32 70%			**LeNet FP32 80%**			**LeNet FP32 90%**			**LeNet INT8**		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	0,0238	0,0205	14,0411	0,0251	0,0205	18,5089	0,0272	0,0218	20,0529	0,0245	0,0180	26,3780
24	0,0326	0,0284	12,7945	0,0335	0,0284	15,0718	0,0362	0,0296	18,3429	0,0316	0,0251	20,5421
32	0,0386	0,0331	14,3012	0,0395	0,0331	16,2346	0,0432	0,0341	21,1311	0,0355	0,0292	17,8013
48	0,0469	0,0498	6,1426	0,0478	0,0498	4,2585	0,0510	0,0482	5,3969	0,0446	0,0403	9,5727
64	0,0566	0,0517	8,6390	0,0573	0,0517	9,8398	0,0612	0,0522	14,6931	0,0532	0,0462	13,1408
80	0,0679	0,0617	9,2295	0,0685	0,0617	9,9506	0,0738	0,0621	15,8434	0,0632	0,0555	12,0710
90	0,0751	0,0671	10,5940	0,0755	0,0671	11,0889	0,0814	0,0683	16,0878	0,0693	0,0610	11,9591
100	0,0779	0,0850	9,0398	0,0786	0,0850	8,0836	0,0831	0,0814	2,0028	0,0721	0,0688	4,5678
116	0,0888	0,0963	8,4536	0,0894	0,0963	7,7326	0,0947	0,0922	2,7085	0,0813	0,0785	3,4806
124	0,1001	0,1119	11,7807	0,1006	0,1119	11,2782	0,1051	0,1075	2,2085	0,0931	0,0932	0,0991
132	0,1059	0,1182	11,6640	0,1065	0,1182	11,0597	0,1110	0,1135	2,2653	0,0982	0,0987	0,5186
148	0,1241	0,1180	4,8933	0,1244	0,1180	5,1542	0,1296	0,1184	8,6149	0,1146	0,1105	3,5565
164	0,1333	0,1295	2,8616	0,1332	0,1295	2,8144	0,1380	0,1301	5,7159	0,1249	0,1218	2,5080
180	0,1475	0,1625	10,1832	0,1479	0,1625	9,8866	0,1524	0,1572	3,1408	0,1362	0,1408	3,4309
		Moyenne	9,6156		**Moyenne**	10,0688		**Moyenne**	9,8718		**Moyenne**	9,2590

TABLE B.17 – Mesures du courant pour une inférence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - COURANT [mA]

	LeNet FP32 Sig.			LeNet FP32			LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	7,4204	6,5876	11,2229	6,6717	6,1729	7,4759	7,2954	6,1874	15,1884	5,8839	6,1884	5,1765
24	10,0597	8,9333	11,1970	9,0520	8,5744	5,2760	9,4450	8,5923	9,0282	8,5879	8,5936	0,0664
32	11,4805	10,3032	10,2548	10,5508	9,9974	5,2449	10,9862	10,0102	8,8839	10,1982	10,0112	1,8342
48	14,9906	14,5341	3,0455	13,6213	15,0525	10,5070	14,1804	15,0583	6,1909	12,8704	15,0587	17,0032
64	17,7410	15,7547	11,1960	16,1282	15,5974	3,2913	16,8001	15,6191	7,0293	15,6512	15,6208	0,1944
80	21,1535	18,7484	11,3699	19,3035	18,6122	3,5812	20,0618	18,6386	7,0940	18,9000	18,6406	1,3724
90	23,3288	20,6272	11,5804	21,1966	20,2536	4,4487	22,0942	20,2840	8,1930	20,9221	20,2864	3,0385
100	24,6287	24,5305	0,3984	22,7204	25,6950	13,0921	23,4401	25,6425	9,3958	21,9953	25,7036	16,8594
116	27,8514	27,7537	0,3508	25,7012	29,1064	13,2492	26,5677	29,1151	9,5884	25,0888	29,1158	2,4504
124	31,7612	32,3712	1,9207	29,8328	33,8488	13,4616	30,3455	33,8583	11,5761	28,6714	33,8590	18,0935
132	33,4845	34,1852	2,0924	31,5197	35,7517	13,4266	32,0585	35,7620	11,5521	30,3402	35,7627	17,8725
148	38,8156	35,7850	7,8076	36,5791	35,6697	2,4861	37,0413	35,6921	3,6424	35,5799	35,6938	0,3201
164	42,0032	39,3050	6,4239	39,8339	39,1051	1,8296	40,1283	39,1436	2,4539	38,5468	39,1465	1,5558
180	46,6150	47,3896	1,6617	44,1784	49,1529	11,2602	44,4723	49,1619	10,5449	42,8799	49,1626	14,6516
		Moyenne	6,4659		**Moyenne**	7,7593		**Moyenne**	8,5972		**Moyenne**	7,1778

TABLE B.18 – Mesures du courant pour une inférence de LeNet5 sur le STM32F446ZE. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F446ZE - COURANT [mA]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
16	7,2109	6,1984	14,0411	7,6081	6,1999	18,5089	8,2378	6,5859	20,0529	7,4069	5,4531	26,3780
24	9,8686	8,6060	12,7945	10,1354	8,6079	15,0718	10,9513	8,9425	18,3429	9,5624	7,5981	20,5421
32	11,6924	10,0202	14,3012	11,9639	10,0216	16,2346	13,0682	10,3067	21,1311	10,7485	8,8351	17,8013
48	14,1934	15,0652	6,1426	14,4508	15,0662	4,2585	15,4215	14,5893	5,3969	13,4808	12,1903	9,5727
64	17,1144	15,6359	8,6390	17,3449	15,6382	9,8398	18,5053	15,7863	14,6931	16,1079	13,9911	13,1408
80	20,5560	18,6588	9,2295	20,7237	18,6616	9,9506	22,3279	18,7904	15,8434	19,1109	16,8040	12,0710
90	22,7133	20,3071	10,5940	22,8433	20,3102	11,0889	24,6202	20,6594	16,0878	20,9642	18,4570	11,9591
100	23,5817	25,7134	9,0398	23,7916	25,7148	8,0836	25,1462	24,6426	2,0028	21,8243	20,8274	4,5678
116	26,8563	29,1266	8,4536	27,0375	29,1282	7,7326	28,6585	27,8822	2,7085	24,5964	23,7403	3,4806
124	30,3011	33,8708	11,7807	30,4395	33,8725	11,2782	31,8089	32,5115	2,2085	28,1629	28,1908	0,0991
132	32,0379	35,7748	11,6640	32,2138	35,7766	11,0597	33,5726	34,3331	2,2653	29,7137	29,8678	0,5186
148	37,5461	35,7089	4,8933	37,6518	35,7112	5,1542	39,2041	35,8267	8,6149	34,6687	33,4357	3,5565
164	40,3263	39,1723	2,8616	40,3108	39,1763	2,8144	41,7515	39,3650	5,7159	37,7975	36,8496	2,5080
180	44,6300	49,1747	10,1832	44,7521	49,1765	9,8866	46,1040	47,5521	3,1408	41,1993	42,6128	3,4309
		Moyenne	9,6156		**Moyenne**	10,0688		**Moyenne**	9,8718		**Moyenne**	9,2590

TABLE B.19 – Mesures de la latence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - LATENCE [ms/inférence]

	LeNet FP32 Sig.			LeNet FP32		LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	88217,4272	87354,6348	0,9780	85125,8016	69856,1001	17,9378	35239,3799	31975,6975	9,2615	33561,4060	30798,2982	8,2330
0,4	35363,7233	34906,0296	1,2942	32802,3426	27906,4087	14,9256	13667,7864	12377,2310	9,4423	13155,0427	11894,5517	9,5818
0,8	16009,5262	15905,4303	0,6502	14801,4147	12722,9402	14,0424	6184,3748	5610,4526	9,2802	5929,0448	5389,3455	9,1026
1	12776,2386	12836,2477	0,4697	10903,8061	10428,1315	4,3625	4462,8822	4421,1826	0,9344	4257,9473	4234,0806	0,5605
4	3024,3189	3040,5997	0,5383	2575,4620	2476,1674	3,8554	1055,1263	1042,5760	1,1895	1006,0464	997,9195	0,8078
16	765,3008	759,6993	0,7319	634,9816	621,1179	2,1833	260,1859	257,3783	1,0791	247,9915	246,0866	0,7681
48	275,3221	275,0074	0,1143	239,4481	223,1191	6,8194	103,0308	92,1147	10,5950	98,7380	88,0801	10,7942
80	179,0045	178,8173	0,1046	160,9733	143,2511	11,0094	67,3869	62,4874	7,2708	64,6329	59,9770	7,2036
100	151,9776	157,7889	3,8238	137,9922	127,5462	7,5700	59,4101	51,1349	13,9291	56,9131	48,7791	14,2920
116	138,7316	154,3903	11,2870	129,7704	127,5838	1,6850	51,2771	53,5921	4,5146	48,6063	51,2922	5,5258
148	96,0996	100,1649	4,2303	89,3575	80,7646	1,5927	43,6642	45,7188	4,7055	41,3856	43,7564	5,7287
180	114,0281	131,6911	15,4901	110,6098	108,8480	10,4965	39,0206	35,8954	8,0089	37,3654	34,4342	7,8446
200	95,3813	99,3556	4,1668	90,1760	80,0204	9,6163	38,8357	32,3495	16,7015	37,2052	30,8566	17,0637
216	100,9471	103,2806	2,3116	92,6121	82,8911	11,2620	39,2226	32,0367	18,3209	37,5354	30,5572	18,5910
		Moyenne	3,2993		**Moyenne**	8,3827		**Moyenne**	8,2309		**Moyenne**	8,2927

TABLE B.20 – Mesures de la latence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - LATENCE [ms/inférence]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	23064,4163	23453,1267	1,6853	22637,3991	22674,1860	0,1625	14933,8895	17231,4249	15,3847	69593,9849	66011,7752	5,1473
0,4	9280,1261	8883,3184	4,2759	8825,7190	8563,9822	2,9656	5792,3592	6332,7139	9,3288	28142,6063	26366,5038	6,3111
0,8	4180,7275	4010,0179	4,0833	3973,7979	3863,7437	2,7695	2615,4955	2841,6231	8,6457	12816,4410	12000,4975	6,3664
1	2973,9134	3066,9528	3,1285	2835,3210	2943,1829	3,8042	1878,9705	2078,2441	10,6055	10266,2443	10464,5410	1,9315
4	701,7317	719,3597	2,5121	669,4968	689,8195	3,0355	444,4980	483,3793	8,7472	2440,7321	2478,8693	1,5625
16	173,0485	175,5334	1,4359	165,0625	168,0495	1,8096	109,6466	115,8763	5,6816	568,4775	611,0863	7,4952
48	71,3594	62,9079	11,8436	68,4678	60,2384	12,0194	48,0009	41,5882	13,3596	219,1308	187,4932	14,4378
80	45,3004	44,3176	2,1695	43,3682	42,6570	1,6399	28,4984	31,0521	8,9607	141,7607	125,9945	11,1217
100	41,1509	34,0807	17,1812	39,4794	32,5219	17,6231	27,7212	21,6325	21,9640	122,5846	102,1277	16,6880
116	33,4242	36,9178	10,4524	31,6062	35,3930	11,9811	20,6765	24,7675	19,7860	106,2913	117,2023	10,2652
148	28,4470	31,4913	10,7016	26,9036	30,1902	12,2163	17,5946	21,1240	20,0597	90,0235	100,0879	11,1797
180	26,2390	25,3186	3,5077	25,0657	24,3519	2,8479	16,4884	17,5972	6,7244	80,8967	66,4465	17,8625
200	27,0146	21,5413	20,2605	25,9060	20,5534	20,6618	18,2774	13,6526	25,3034	76,2662	60,6779	20,4394
216	27,2221	21,3253	21,6617	26,1357	20,3463	22,1512	18,4590	13,5075	26,8245	75,4303	59,2440	21,4586
		Moyenne	8,2071		**Moyenne**	8,2634		**Moyenne**	14,3840		**Moyenne**	10,8762

TABLE B.21 – Mesures de la consommation d'énergie pour une inférence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - ÉNERGIE [J/inférence]

	LeNet FP32 Sig.			LeNet FP32			LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	2,6981	2,6914	0,2481	2,7444	2,1544	21,4959	1,0853	0,9919	8,6126	1,0077	0,9558	5,1518
0,4	1,0966	1,0961	0,0394	1,0752	0,8833	17,8479	0,4292	0,3968	7,5309	0,3996	0,3817	4,4803
0,8	0,4993	0,5071	1,5650	0,5021	0,4131	17,7347	0,2007	0,1869	6,8644	0,1879	0,1799	4,2525
1	0,3263	0,3394	4,0121	0,3065	0,2851	6,9880	0,1189	0,1262	6,0807	0,1097	0,1212	10,4933
4	0,0938	0,0966	2,9132	0,0883	0,0862	2,3068	0,0345	0,0398	15,2325	0,0319	0,0383	19,8785
16	0,0391	0,0397	1,5822	0,0365	0,0345	5,5404	0,0146	0,0148	1,9151	0,0135	0,0142	5,2730
48	0,0306	0,0317	3,3995	0,0298	0,0249	16,3820	0,0127	0,0102	19,9119	0,0120	0,0097	19,1873
80	0,0258	0,0250	3,2248	0,0262	0,0228	13,2236	0,0110	0,0103	6,5708	0,0105	0,0099	5,7719
100	0,0261	0,0271	4,1428	0,0261	0,0249	4,6348	0,0112	0,0103	7,9211	0,0106	0,0098	7,4724
116	0,0266	0,0280	5,2140	0,0275	0,0283	2,6526	0,0109	0,0123	13,1324	0,0103	0,0118	14,6556
148	0,0239	0,0235	1,5209	0,0243	0,0228	6,1388	0,0119	0,0133	11,9698	0,0112	0,0127	13,3125
180	0,0333	0,0372	11,6661	0,0353	0,0383	8,4017	0,0125	0,0130	4,3134	0,0119	0,0125	4,6789
200	0,0305	0,0314	2,9803	0,0307	0,0303	1,0711	0,0131	0,0126	4,0168	0,0125	0,0120	4,1038
216	0,0336	0,0340	1,2466	0,0325	0,0336	3,3893	0,0137	0,0134	2,2375	0,0131	0,0128	2,1917
		Moyenne	3,1254		**Moyenne**	9,1291		**Moyenne**	8,3079		**Moyenne**	8,6359

TABLE B.22 – Mesures de la consommation d'énergie pour une inférence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - ÉNERGIE [J/inférence]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	0,6789	0,7307	7,6334	0,6892	0,7069	2,5663	0,4521	0,5405	19,5622	2,2063	2,2671	2,7563
0,4	0,2818	0,2876	2,0591	0,2746	0,2776	1,0977	0,1786	0,2080	16,4727	0,8964	0,9203	2,6678
0,8	0,1310	0,1361	3,8665	0,1281	0,1314	2,6310	0,0833	0,0990	18,8745	0,4060	0,4262	4,9880
1	0,0757	0,0902	19,0940	0,0751	0,0869	15,6712	0,0488	0,0638	30,7861	0,2776	0,3209	15,5888
4	0,0222	0,0289	30,4175	0,0219	0,0279	27,4562	0,0145	0,0207	42,7514	0,0821	0,0873	6,3227
16	0,0094	0,0103	9,5670	0,0092	0,0099	7,2450	0,0061	0,0069	13,7715	0,0318	0,0305	4,1451
48	0,0086	0,0069	19,9526	0,0084	0,0066	21,7434	0,0059	0,0046	23,2350	0,0252	0,0204	18,8676
80	0,0074	0,0073	0,4938	0,0072	0,0071	1,1853	0,0047	0,0052	9,3788	0,0190	0,0176	7,3234
100	0,0077	0,0069	10,2099	0,0074	0,0066	11,0743	0,0053	0,0044	16,1345	0,0214	0,0171	19,9169
116	0,0071	0,0086	20,6357	0,0068	0,0082	21,4604	0,0045	0,0058	30,0201	0,0198	0,0220	10,7866
148	0,0077	0,0092	19,0902	0,0074	0,0088	20,2121	0,0049	0,0062	28,0992	0,0218	0,0249	14,4810
180	0,0084	0,0093	10,0245	0,0081	0,0089	10,5871	0,0054	0,0065	21,0116	0,0225	0,0202	10,5287
200	0,0091	0,0085	7,3213	0,0087	0,0081	7,7047	0,0062	0,0054	13,6655	0,0243	0,0190	21,6890
216	0,0095	0,0090	5,3643	0,0091	0,0086	5,9248	0,0065	0,0057	12,0131	0,0249	0,0195	21,9947
		Moyenne	11,8379		**Moyenne**	11,1828		**Moyenne**	21,1269		**Moyenne**	11,5755

TABLE B.23 – Mesures de la puissance pour une inférence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - PUISSANCE [W]

	LeNet FP32 Sig.			LeNet FP32			LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	0,0306	0,0308	0,7371	0,0322	0,0308	4,3358	0,0308	0,0310	0,7151	0,0300	0,0310	3,3577
0,4	0,0310	0,0314	1,2713	0,0328	0,0317	3,4350	0,0314	0,0321	2,1107	0,0304	0,0321	5,6421
0,8	0,0312	0,0319	2,2297	0,0339	0,0325	4,2955	0,0324	0,0333	2,6629	0,0317	0,0334	5,3359
1	0,0255	0,0264	3,5258	0,0281	0,0273	2,7453	0,0266	0,0285	7,0813	0,0258	0,0286	11,1161
4	0,0310	0,0318	2,3622	0,0343	0,0348	1,6107	0,0327	0,0381	16,6197	0,0318	0,0384	20,8548
16	0,0511	0,0523	2,3312	0,0575	0,0555	3,4320	0,0559	0,0576	3,0268	0,0544	0,0577	6,0879
48	0,1113	0,1152	3,5178	0,1243	0,1115	10,2624	0,1231	0,1102	10,4210	0,1216	0,1102	9,4087
80	0,1443	0,1398	3,1234	0,1631	0,1590	2,4881	0,1629	0,1641	0,7549	0,1618	0,1643	1,5429
100	0,1714	0,1719	0,3073	0,1895	0,1955	3,1756	0,1881	0,2012	6,9803	0,1866	0,2014	7,9568
116	0,1920	0,1815	5,4570	0,2123	0,2217	4,4120	0,2121	0,2296	8,2456	0,2116	0,2299	8,6517
148	0,2484	0,2347	5,5177	0,2717	0,2821	3,8475	0,2715	0,2903	6,9379	0,2712	0,2906	7,1728
180	0,2924	0,2828	3,3111	0,3193	0,3517	10,1562	0,3198	0,3627	13,3951	0,3197	0,3631	13,5895
200	0,3193	0,3157	1,1390	0,3399	0,3790	11,4843	0,3380	0,3894	15,2281	0,3371	0,3898	15,6263
216	0,3328	0,3293	1,0409	0,3506	0,4050	15,5142	0,3487	0,4174	19,6911	0,3478	0,4178	20,1444
		Moyenne	2,5623		**Moyenne**	5,7996		**Moyenne**	8,1336		**Moyenne**	9,7491

TABLE B.24 – Mesures de la puissance pour une inférence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - PUISSANCE [W]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	0,0294	0,0312	5,8495	0,0304	0,0312	2,3999	0,0303	0,0314	3,6205	0,0317	0,0343	8,3325
0,4	0,0304	0,0324	6,6180	0,0311	0,0324	4,1875	0,0308	0,0328	6,5344	0,0319	0,0349	9,5837
0,8	0,0313	0,0339	8,2882	0,0322	0,0340	5,5544	0,0319	0,0349	9,4149	0,0317	0,0355	12,1264
1	0,0255	0,0294	15,4812	0,0265	0,0295	11,4321	0,0260	0,0307	18,2456	0,0270	0,0307	13,3984
4	0,0316	0,0402	27,2216	0,0327	0,0405	23,7012	0,0326	0,0428	31,2690	0,0337	0,0352	4,6869
16	0,0542	0,0586	8,0160	0,0557	0,0587	5,3388	0,0553	0,0595	7,6550	0,0560	0,0499	10,8287
48	0,1209	0,1098	9,1985	0,1234	0,1097	11,0525	0,1235	0,1094	11,3981	0,1149	0,1090	5,1773
80	0,1629	0,1657	1,7128	0,1650	0,1658	0,4622	0,1663	0,1669	0,3838	0,1337	0,1394	4,2736
100	0,1872	0,2030	8,4175	0,1882	0,2031	7,9498	0,1901	0,2044	7,4703	0,1744	0,1676	3,8757
116	0,2123	0,2319	9,2197	0,2140	0,2321	8,4651	0,2153	0,2337	8,5436	0,1865	0,1874	0,4729
148	0,2720	0,2926	7,5777	0,2734	0,2928	7,1254	0,2759	0,2944	6,6963	0,2417	0,2489	2,9693
180	0,3207	0,3657	14,0242	0,3215	0,3660	13,8288	0,3246	0,3681	13,3870	0,2787	0,3036	8,9286
200	0,3376	0,3924	16,2268	0,3375	0,3926	16,3314	0,3414	0,3946	15,5803	0,3181	0,3131	1,5707
216	0,3483	0,4208	20,8038	0,3485	0,4211	20,8435	0,3522	0,4235	20,2408	0,3307	0,3284	0,6825
		Moyenne	11,3325		**Moyenne**	9,9052		**Moyenne**	11,4600		**Moyenne**	6,2077

TABLE B.25 – Mesures du courant pour une inférence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - COURANT [mA]

	LeNet FP32 Sig.			LeNet FP32		LeNet FP32 50%			LeNet FP32 60%		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	9,2540	9,3222	0,7371	9,7545	9,3316	4,3358	9,3189	9,3855	0,7151	9,0846	9,3897	3,3577
0,4	9,3822	9,5015	1,2713	9,9181	9,5774	3,4350	9,5005	9,7010	2,1107	9,1918	9,7104	5,6421
0,8	9,4367	9,6471	2,2297	10,2645	9,8236	4,2955	9,8179	10,0793	2,6629	9,5869	10,0984	5,3359
1	7,7270	7,9994	3,5258	8,5048	8,2713	2,7453	8,0630	8,6340	7,0813	7,7944	8,6609	11,1161
4	9,3873	9,6091	2,3622	10,3684	10,5354	1,6107	9,8976	11,5426	16,6197	9,6068	11,6102	20,8548
16	15,4534	15,8137	2,3312	17,3843	16,7877	3,4320	16,9254	17,4377	3,0268	16,4696	17,4722	6,0879
48	33,6653	34,8496	3,5178	37,6014	33,7426	10,2624	37,2328	33,3528	10,4210	36,7980	33,3357	9,4087
80	43,6499	42,2865	3,1234	49,3344	48,1069	2,4881	49,2799	49,6519	0,7549	48,9580	49,7133	1,5429
100	51,8649	52,0243	0,3073	57,3308	59,1514	3,1756	56,9102	60,8827	6,9803	56,4588	60,9512	7,9568
116	58,0884	54,9185	5,4570	64,2338	67,0678	4,4120	64,1874	69,4800	8,2456	64,0316	69,5714	8,6517
148	75,1724	71,0246	5,5177	82,1982	85,3608	3,8475	82,1455	87,8447	6,9379	82,0514	87,9369	7,1728
180	88,4863	85,5565	3,3111	96,6110	106,4229	10,1562	96,7764	109,7397	13,3951	96,7180	109,8615	13,5895
200	96,6156	95,5151	1,1390	102,8529	114,6649	11,4843	102,2594	117,8316	15,2281	102,0088	117,9490	15,6263
216	100,6911	99,6430	1,0409	106,0949	122,5547	15,5142	105,5077	126,2833	19,6911	105,2244	126,4212	20,1444
		Moyenne	2,5623		**Moyenne**	5,7996		**Moyenne**	8,1336		**Moyenne**	9,7491

TABLE B.26 – Mesures de la puissance pour une inférence de LeNet5 sur le STM32F746ZG. "Sig." signifie "Sigmoïde". Les autres configurations de LeNet5 ont été déployées avec la fonction d'activation ReLU.

STM32F746ZG - COURANT [mA]

	LeNet FP32 70%			LeNet FP32 80%			LeNet FP32 90%			LeNet INT8		
Fréq. [MHz]	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**	**Mes.**	**Est.**	**Diff. [%]**
0,2	8,9061	9,4270	5,8495	9,2116	9,4327	2,3999	9,1595	9,4911	3,6205	9,5923	10,3916	8,3325
0,4	9,1869	9,7949	6,6180	9,4135	9,8077	4,1875	9,3281	9,9376	6,5344	9,6376	10,5613	9,5837
0,8	9,4820	10,2679	8,2882	9,7516	10,2932	5,5544	9,6386	10,5460	9,4149	9,5845	10,7468	12,1264
1	7,7056	8,8986	15,4812	8,0173	8,9339	11,4321	7,8518	9,2844	18,2456	8,1826	9,2789	13,3984
4	9,5672	12,1715	27,2216	9,9027	12,2497	23,7012	9,8752	12,9631	31,2690	10,1820	10,6592	4,6869
16	16,4117	17,7272	8,0160	16,8588	17,7589	5,3388	16,7314	18,0122	7,6550	16,9318	15,0983	10,8287
48	36,5839	33,2187	9,1985	37,3312	33,2052	11,0525	37,3626	33,1040	11,3981	34,7704	32,9703	5,1773
80	49,2780	50,1220	1,7128	49,9371	50,1679	0,4622	50,3095	50,5026	0,3838	40,4580	42,1870	4,2736
100	56,6395	61,4071	8,4175	56,9322	61,4582	7,9498	57,5334	61,8313	7,4703	52,7553	50,7107	3,8757
116	64,2483	70,1718	9,2197	64,7566	70,2382	8,4651	65,1521	70,7184	8,5436	56,4373	56,7041	0,4729
148	82,3018	88,5384	7,5777	82,7111	88,6046	7,1254	83,4900	89,0807	6,6963	73,1270	75,2984	2,9693
180	97,0449	110,6547	14,0242	97,2880	110,7418	13,8288	98,2182	111,3667	13,3870	84,3325	91,8622	8,9286
200	102,1413	118,7155	16,2268	102,1219	118,7999	16,3314	103,3108	119,4070	15,5803	96,2400	94,7284	1,5707
216	105,3954	127,3216	20,8038	105,4427	127,4207	20,8435	106,5638	128,1332	20,2408	100,0563	99,3734	0,6825
		Moyenne	11,3325		**Moyenne**	9,9052		**Moyenne**	10,7845		**Moyenne**	6,2077

Annexe C

Mesures de l'inférence de ResNet8

C.1 Estimations et mesures de ResNet8 sur le STM32L496ZG et le STM32F746ZG

Estimations et mesures sur le STM32L496ZG :

Table C.1 – Mesures de la latence d'une inférence de ResNet8, au format binaire FP32, sur le STM32L496ZG

	Latence [ms/inférence]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,1	4367133,61	5041285,43	15,44
0,2	1819775,60	2098631,66	15,32
0,4	841283,91	969895,42	15,29
0,8	405319,94	468263,19	15,53
1	321876,88	370469,26	15,10
2	158678,17	182501,62	15,01
4	78778,57	90605,01	15,01
8	39251,29	45142,82	15,01
16	19591,20	22553,42	15,12
24	13053,23	15027,83	15,13
32	9787,07	11269,74	15,15
48	6525,36	7516,73	15,19
64	4894,40	5646,56	15,37
80	3916,07	4522,96	15,50

TABLE C.2 – Mesures de l'énergie consommée pour une inférence de ResNet8, au format binaire FP32, sur le STM32L496ZG

	Énergie [J/inférence]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,1	2,1049	2,3824	13,1855
0,2	0,9641	1,0817	12,1989
0,4	0,5271	0,5834	10,6929
0,8	0,3312	0,3619	9,2736
1	0,2969	0,3193	7,5513
2	0,2215	0,2368	6,9222
4	0,1863	0,1957	5,0563
8	0,1681	0,1750	4,1082
16	0,1610	0,1671	3,7927
24	0,1575	0,1629	3,4235
32	0,1560	0,1612	3,3226
48	0,1542	0,1544	0,1027
64	0,1540	0,1568	1,8087
80	0,1533	0,1561	1,8634

TABLE C.3 – Mesures de puissance consommée pour une inférence de ResNet8, au format binaire FP32, sur le STM32L496ZG

	Puissance [W]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,1	0,000482	0,000473	1,9504
0,2	0,000530	0,000515	2,7095
0,4	0,000627	0,000602	3,9854
0,8	0,000817	0,000773	5,4148
1	0,000922	0,000862	6,5556
2	0,001396	0,001298	7,0352
4	0,002365	0,002160	8,6565
8	0,004282	0,003876	9,4788
16	0,008218	0,007410	9,8397
24	0,012064	0,010838	10,1660
32	0,015939	0,014302	10,2708
48	0,023630	0,020535	13,0997
64	0,031472	0,027773	11,7530
80	0,039135	0,034516	11,8046

TABLE C.4 – Mesures du courant consommé pour une inférence de ResNet8, au format binaire FP32, sur le STM32L496ZG

	Courant [mA]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,1	0,1458	0,1430	1,9504
0,2	0,1603	0,1560	2,7095
0,4	0,1896	0,1820	3,9854
0,8	0,2472	0,2338	5,4148
1	0,2790	0,2608	6,5556
2	0,4224	0,3926	7,0352
4	0,7155	0,6536	8,6565
8	1,2955	1,1727	9,4788
16	2,4866	2,2419	9,8397
24	3,6504	3,2793	10,1660
32	4,8227	4,3274	10,2708
48	7,1498	6,2132	13,0997
64	9,5227	8,4035	11,7530
80	11,8413	10,4434	11,8046

Estimations et mesures sur le STM32F746ZG :

TABLE C.5 – Mesures de la latence d'une inférence de ResNet8, au format binaire FP32, sur le STM32F746ZG

	Latence [ms/inférence]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,2	3912070,45	3373335,99	13,77
0,4	1555313,67	1372814,30	11,73
0,8	701435,59	628713,06	10,37
1	547523,59	436360,07	20,30
4	129452,38	103878,75	19,76
16	31927,93	25281,64	20,82
48	11177,69	8826,24	21,04
80	7110,65	7154,56	0,62
100	6040,08	5110,24	15,39
116	5198,33	5183,48	0,29
148	4367,24	4425,60	1,34
180	3859,81	4153,50	7,61
200	3669,16	3250,83	11,40
216	3586,86	3410,76	4,91

TABLE C.6 – Mesures de l'énergie consommée pour une inférence de ResNet8, au format binaire FP32, sur le STM32F746ZG

	Énergie [J/inférence]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,2	116,304763	103,592899	10,9298
0,4	46,955766	43,005187	8,4134
0,8	21,581843	19,986107	7,3939
1,0	13,804069	11,502869	16,6704
4	3,924902	3,288920	16,2038
16	1,564295	1,317612	15,7696
48	1,201554	1,015495	15,4849
80	0,993320	0,988110	0,5245
100	1,008158	0,877052	13,0045
116	0,973075	0,930236	4,4025
148	1,088828	1,027916	5,5942
180	1,119826	1,164418	3,9820
200	1,166755	1,023766	12,2553
216	1,198475	1,120978	6,4663

TABLE C.7 – Mesures de puissance consommée pour une inférence de ResNet8, au format binaire FP32, sur le STM32F746ZG

	Puissance [W]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,2	0,029730	0,030709	3,2951
0,4	0,030191	0,031326	3,7619
0,8	0,030768	0,031789	3,3178
1,0	0,025212	0,026361	4,5579
4	0,030319	0,031661	4,4258
16	0,048995	0,052117	6,3737
48	0,107496	0,115054	7,0313
80	0,139695	0,138109	1,1351
100	0,166911	0,171627	2,8249
116	0,187190	0,179462	4,1286
148	0,249317	0,232266	6,8392
180	0,290125	0,280346	3,3704
200	0,317990	0,314924	0,9639
216	0,334130	0,328660	1,6371

TABLE C.8 – Mesures du courant consommé pour une inférence de ResNet8, au format binaire FP32, sur le STM32F746ZG

	Courant [mA]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
0,2	8,995377	9,291780	3,2951
0,4	9,134809	9,478456	3,7619
0,8	9,309562	9,618431	3,3178
1,0	7,628389	7,976084	4,5579
4	9,173759	9,579770	4,4258
16	14,824373	15,769239	6,3737
48	32,525172	34,812104	7,0313
80	42,267686	41,787904	1,1351
100	50,502694	51,929361	2,8249
116	56,638405	54,300049	4,1286
148	75,436394	70,277126	6,8392
180	87,783636	84,824957	3,3704
200	96,214694	95,287252	0,9639
216	101,098237	99,443127	1,6371

De nouveau, la précision de l'estimation en consommation énergétique est à interpréter en tenant compte de la précision en estimation de la latence. Pour mettre en avant l'influence de ce paramètre, le Tableau C.9 réfère la précision de l'estimation en énergie, puissance et latence sur le STM32L496ZG et STM32F746ZG pour une inférence de ResNet8. Ici, la précision de la consommation énergétique est améliorée par rapport à l'estimation de la puissance concernant le STM32L496ZG, passant de 8,05% de différence entre la mesure et l'estimation de la puissance à 5,95% pour la consommation énergétique. Pour le STM32F746ZG, l'erreur d'estimation se dégrade de 3,83% concernant la puissance à 9,79% concernant la consommation énergétique.

TABLE C.9 – Comparaison entre l'estimation de la puissance et de l'énergie consommée pour une inférence de ResNet8 au sein du STM32L496ZG et du STM32F746ZG. La différence de l'estimation de la latence est également référencée, l'énergie étant directement liée à ce paramètre.

MCU	Diff. Puissance [%]	Diff. Énergie [%]	Diff.Latence [%]
STM32L496ZG	8,05%	5,95%	15,23%
STM32F746ZG	3,83%	9,79%	11,38%

C.2 Estimations et mesures de ResNet8 réduit sur le STM32F446ZE

Estimations sur ResNet8 réduit sur le STM32F446ZE

TABLE C.10 – Estimations de la latence d'une inférence de ResNet8 réduit par techniques de quantification INT8 et élagage des filtres de convolution, sur le STM32F446ZE

	Latence estimée [ms/inférence]					
		Élagage				
Fréquence [MHz]	**INT8**	**50%**	**60%**	**70%**	**80%**	**90%**
16	25175,71	7734,25	5545,11	3208,24	1924,59	700,11
24	16772,38	5152,00	3693,81	2137,16	1282,09	466,39
32	12576,52	3862,95	2769,65	1602,49	961,38	349,76
48	8382,86	2574,52	1845,84	1067,98	640,70	233,09
64	6285,72	1930,63	1384,22	800,91	480,49	174,82
80	5029,07	1544,42	1107,30	640,68	384,36	139,84
90	4469,07	1372,60	984,13	569,41	341,61	124,29
100	4023,14	1235,67	885,92	512,58	307,51	111,89
116	3467,67	1065,05	763,60	441,82	265,06	96,45
124	3243,98	996,26	714,29	413,29	247,95	90,23
132	3048,32	936,08	671,13	388,32	232,96	84,77
148	2718,23	834,73	598,47	346,28	207,75	75,60
164	2453,50	753,23	540,06	312,49	187,49	68,24
180	2234,77	686,44	492,15	284,76	170,84	62,17

TABLE C.11 – Estimations de l'énergie consommée pour une inférence de ResNet8 réduit par techniques de quantification INT8 et élagage des filtres de convolution, sur le STM32F446ZE

	Énergie estimée [J/inférence]					
		Élagage				
Fréquence [MHz]	**INT8**	**50%**	**60%**	**70%**	**80%**	**90%**
16	0,4517	0,1581	0,1134	0,0656	0,0394	0,0143
24	0,4188	0,1462	0,1048	0,0606	0,0364	0,0132
32	0,3654	0,1278	0,0916	0,0530	0,0318	0,0116
48	0,3366	0,1281	0,0919	0,0531	0,0319	0,0116
64	0,2891	0,0995	0,0714	0,0413	0,0248	0,0090
80	0,2779	0,0950	0,0681	0,0394	0,0236	0,0086
90	0,2713	0,0918	0,0658	0,0381	0,0229	0,0083
100	0,2761	0,1049	0,0752	0,0435	0,0261	0,0095
116	0,2713	0,1024	0,0734	0,0424	0,0255	0,0093
124	0,3014	0,1113	0,0798	0,0462	0,0277	0,0101
132	0,3000	0,1105	0,0792	0,0458	0,0275	0,0100
148	0,2992	0,0983	0,0705	0,0408	0,0245	0,0089
164	0,2977	0,0972	0,0697	0,0403	0,0242	0,0088
180	0,3141	0,1114	0,0798	0,0462	0,0277	0,0101

TABLE C.12 – Estimations de la puissance consommée pour une inférence de ResNet8 réduit par techniques de quantification INT8 et élagage des filtres de convolution, sur le STM32F446ZE

	Puissance estimée [W]					
		Élagage				
Fréquence [MHz]	**INT8**	**50%**	**60%**	**70%**	**80%**	**90%**
16	0,0179	0,0204	0,0204	0,0205	0,0205	0,0205
24	0,0250	0,0284	0,0284	0,0284	0,0284	0,0284
32	0,0291	0,0331	0,0331	0,0331	0,0331	0,0331
48	0,0402	0,0498	0,0498	0,0497	0,0497	0,0497
64	0,0460	0,0516	0,0516	0,0516	0,0516	0,0516
80	0,0552	0,0615	0,0615	0,0615	0,0615	0,0615
90	0,0607	0,0669	0,0669	0,0669	0,0669	0,0669
100	0,0686	0,0849	0,0849	0,0848	0,0848	0,0847
116	0,0782	0,0961	0,0961	0,0961	0,0960	0,0960
124	0,0929	0,1117	0,1117	0,1117	0,1116	0,1116
132	0,0984	0,1180	0,1180	0,1179	0,1179	0,1178
148	0,1101	0,1178	0,1178	0,1178	0,1178	0,1178
164	0,1214	0,1291	0,1291	0,1291	0,1291	0,1290
180	0,1405	0,1622	0,1622	0,1621	0,1621	0,1620

TABLE C.13 – Estimations du courant consommé pour une inférence de ResNet8 réduit par techniques de quantification INT8 et élagage des filtres de convolution, sur le STM32F446ZE

	Courant estimé [mA]					
		Élagage				
Fréquence [MHz]	**INT8**	**50%**	**60%**	**70%**	**80%**	**90%**
16	5,4283	6,1866	6,1871	6,1886	6,1896	6,1923
24	7,5551	8,5847	8,5851	8,5862	8,5869	8,5890
32	8,7900	10,0118	10,0121	10,0128	10,0133	10,0147
48	12,1508	15,0584	15,0566	15,0519	15,0487	15,0397
64	13,9163	15,5991	15,5990	15,5988	15,5986	15,5981
80	16,7168	18,6060	18,6057	18,6051	18,6046	18,6033
90	18,3662	20,2439	20,2438	20,2434	20,2431	20,2424
100	20,7639	25,6802	25,6763	25,6659	25,6589	25,6389
116	23,6704	29,0802	29,0757	29,0639	29,0561	29,0334
124	28,1101	33,8095	33,8043	33,7909	33,7819	33,7561
132	29,7806	35,7072	35,7019	35,6878	35,6784	35,6514
148	33,3032	35,6391	35,6380	35,6352	35,6334	35,6280
164	36,7190	39,0581	39,0569	39,0540	39,0519	39,0462
180	42,5243	49,0853	49,0783	49,0600	49,0477	49,0125

Estimations et mesures sur ResNet8 en FP32 élagué à 70% sur le STM32F446ZE

TABLE C.14 – Mesures de la latence d'une inférence de ResNet8 élagué à 70%, au format binaire FP32, sur le STM32F446ZE

	Latence [ms/inférence]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
16	2724,63	3208,24	17,70
24	1815,23	2137,16	17,72
32	1361,17	1602,49	17,72
48	907,15	1067,98	17,73
64	680,33	800,91	17,73
80	544,21	640,68	17,75
90	483,72	569,41	17,73
100	435,39	512,58	17,73
116	375,32	441,82	17,73
124	351,15	413,29	17,72
132	329,86	388,32	17,71
148	294,19	346,28	17,71
164	265,52	312,49	17,72
180	241,92	284,76	17,69

TABLE C.15 – Mesures de l'énergie consommée pour une inférence de ResNet8 élagué à 70%, au format binaire FP32, sur le STM32F446ZE

	Énergie [J/inférence]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
16	0,0661	0,0656	0,7083
24	0,0495	0,0606	22,5107
32	0,0501	0,0530	5,8823
48	0,0430	0,0531	23,5410
64	0,0383	0,0413	7,8158
80	0,0362	0,0394	8,7565
90	0,0352	0,0381	8,2813
100	0,0337	0,0435	28,9996
116	0,0330	0,0424	28,4857
124	0,0363	0,0462	27,1566
132	0,0357	0,0458	28,2150
148	0,0372	0,0408	9,7628
164	0,0381	0,0403	5,7604
180	0,0368	0,0462	25,4580

TABLE C.16 – Mesures de puissance consommée pour une inférence de ResNet8 élagué à 70%, au format binaire FP32, sur le STM32F446ZE

	Puissance [W]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
16	0,0243	0,0205	15,6951
24	0,0273	0,0284	4,0432
32	0,0368	0,0331	10,0741
48	0,0474	0,0497	4,9191
64	0,0563	0,0516	8,4604
80	0,0666	0,0615	7,6547
90	0,0728	0,0669	8,1513
100	0,0775	0,0848	9,4746
116	0,0881	0,0961	9,0537
124	0,1034	0,1117	7,9921
132	0,1086	0,1179	8,6299
148	0,1264	0,1178	6,8084
164	0,1439	0,1291	10,3133
180	0,1527	0,1621	6,1772

TABLE C.17 – Mesures du courant consommé pour une inférence de ResNet8 élagué à 70%, au format binaire FP32, sur le STM32F446ZE

	Courant [mA]		
Fréquence [MHz]	**Mesures**	**Estimations**	**Différence [%]**
16	7,3407	6,1886	15,6951
24	8,2525	8,5862	4,0432
32	11,1345	10,0128	10,0741
48	14,3462	15,0519	4,9191
64	17,0405	15,5988	8,4604
80	20,1473	18,6051	7,6547
90	22,0399	20,2434	8,1513
100	23,4446	25,6659	9,4746
116	26,6510	29,0639	9,0537
124	31,2902	33,7909	7,9921
132	32,8527	35,6878	8,6299
148	38,2387	35,6352	6,8084
164	43,5449	39,0540	10,3133
180	46,2058	49,0600	6,1772

Bibliographie

[1] J. L. Hennessy et D. A. Patterson, *Computer Architecture, Sixth Edition : A Quantitative Approach*, 6th. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2017, isbn : 978-0-12-811905-1.

[2] G. E. Moore, "Cramming more components onto integrated circuits", en, t. 38, n° 8, p. 4, 1965.

[3] K. Rupp, M. Horovitz, F. Labonte et al., "Years of microprocessor trend data", *Figure available on webpage http ://www. karlrupp. net/wp-content/uploads/2015*, t. 6, p. 40,

[4] R. David, J. Duke, A. Jain et al., "TensorFlow Lite Micro : Embedded Machine Learning on TinyML Systems", en, p. 12,

[5] A. Ltd, *The Arm ecosystem ships a record 6.7 billion Arm-based chips in a single quarter*, en. adresse : `https://www.arm.com/company/news/2021/02/arm-ecosystem-ships-record-6-billion-arm-based-chips-in-a-single-quarter` (visité le 28/07/2022).

[6] S. Prakash, M. Stewart, C. Banbury et al., *Is TinyML Sustainable ? Assessing the Environmental Impacts of Machine Learning on Microcontrollers*, arXiv :2301.11899 [cs], jan. 2023. adresse : `http://arxiv.org/abs/2301.11899` (visité le 02/02/2023).

[7] *TMPM4KHFWAUG | Microcontrollers | Toshiba Electronic Devices & Storage Corporation | Europe(EMEA)*, en-GB. adresse : `https://toshiba.semicon-storage.com/eu/semiconductor/product/microcontrollers/detail.TMPM4KHFWAUG.html` (visité le 29/08/2022).

[8] *PIC16F877A | Microchip Technology*. adresse : `https://www.microchip.com/en-us/product/PIC16F877A#document-table` (visité le 29/08/2022).

[9] *STM32L496ZG - Ultra-low-power with FPU Arm Cortex-M4 MCU 80 MHz with 1 Mbyte of Flash memory, USB OTG, LCD, DFSDM - STMicroelectronics*, en. adresse : `https://www.st.com/en/microcontrollers-microprocessors/stm32l496zg.html` (visité le 29/07/2022).

[10] Y. Zhang, N. Suda, L. Lai et V. Chandra, "Hello Edge : Keyword Spotting on Microcontrollers", *arXiv :1711.07128 [cs, eess]*, fév. 2018, arXiv : 1711.07128. adresse : `http://arxiv.org/abs/1711.07128` (visité le 01/07/2021).

[11] M. Siekkinen, M. Hiienkari, J. K. Nurminen et J. Nieminen, "How low energy is bluetooth low energy ? Comparative measurements with ZigBee/802.15.4", en, in *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, Paris, France : IEEE, avr. 2012, p. 232-237, isbn : 978-1-4673-0682-9 978-1-4673-0681-2 978-1-4673-0680-5. doi : 10.1109/WCNCW.2012.6215496. adresse : http://ieeexplore.ieee.org/document/6215496/ (visité le 01/08/2022).

[12] G. Crocioni, G. Gruosso, D. Pau, D. Denaro, L. Zambrano et G. di Giore, "Characterization of Neural Networks Automatically Mapped on Automotive-grade Microcontrollers", arXiv, rapp. tech. arXiv :2103.00201, fév. 2021, arXiv :2103.00201 [cs, eess] type : article. adresse : http://arxiv.org/abs/2103.00201 (visité le 25/05/2022).

[13] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone et A. Beghi, "Machine Learning for Predictive Maintenance : A Multiple Classifier Approach", en, *IEEE Transactions on Industrial Informatics*, t. 11, n° 3, p. 812-820, juin 2015, issn : 1551-3203, 1941-0050. doi : 10.1109/TII.2014.2349359. adresse : http://ieeexplore.ieee.org/document/6879441/ (visité le 01/07/2021).

[14] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", en, p. 60,

[15] A. Chowdhery, P. Warden, J. Shlens, A. Howard et R. Rhodes, "Visual Wake Words Dataset", *arXiv :1906.05721 [cs, eess]*, juin 2019, arXiv : 1906.05721. adresse : http://arxiv.org/abs/1906.05721 (visité le 01/07/2021).

[16] P. Warden, *Speech Commands : A Dataset for Limited-Vocabulary Speech Recognition*, arXiv :1804.03209 [cs], avr. 2018. adresse : http://arxiv.org/abs/1804.03209 (visité le 01/08/2022).

[17] Y. LeCun, *Collège de France - Recherches sur l'intelligence artificielle*, FR. adresse : https://www.college-de-france.fr/site/yann-lecun/Recherches-sur-l-intelligence-artificielle.htm (visité le 29/08/2022).

[18] W. S. McCulloch et W. Pitts, "A logical calculus of the ideas immanent in nervous activity", en, *The bulletin of mathematical biophysics*, t. 5, n° 4, p. 115-133, déc. 1943, issn : 1522-9602. doi : 10.1007/BF02478259. adresse : https://doi.org/10.1007/BF02478259 (visité le 14/07/2022).

[19] F. Rosenblatt, "The perceptron : A probabilistic model for information storage and organization in the brain", *Psychological Review*, t. 65, n° 6, p. 386-408, 1958, Place : US Publisher : American Psychological Association, issn : 1939-1471. doi : 10.1037/h0042519.

[20] B. Widrow et M. E. Hoff, "Adaptive switching circuits", *IRE WESCON Convention Record*, n° 46,48,49, p. 96-104, 1960.

[21] D. E. Rumelhart, G. E. Hinton et R. J. Williams, "Learning representations by back-propagating errors", en, *Nature*, t. 323, n° 6088, p. 533-536, oct. 1986, Number : 6088 Publisher : Nature Publishing Group, issn : 1476-4687. doi : 10.1038/323533a0. adresse : https://www.nature.com/articles/323533a0 (visité le 14/07/2022).

[22] Y. LECUN, L. BOTTOU, Y. BENGIO et P. HAFFNER, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, t. 86, nº 11, p. 2278-2324, nov. 1998, Conference Name : Proceedings of the IEEE, ISSN : 1558-2256. DOI : 10.1109/5.726791.

[23] O. RUSSAKOVSKY, J. DENG, H. SU et al., "ImageNet Large Scale Visual Recognition Challenge", en, *International Journal of Computer Vision*, t. 115, nº 3, p. 211-252, déc. 2015, ISSN : 1573-1405. DOI : 10.1007/s11263-015-0816-y. adresse : https://doi.org/10.1007/s11263-015-0816-y (visité le 14/07/2022).

[24] A. KRIZHEVSKY, I. SUTSKEVER et G. E. HINTON, "ImageNet Classification with Deep Convolutional Neural Networks", in *Advances in Neural Information Processing Systems 25*, F. PEREIRA, C. J. C. BURGES, L. BOTTOU et K. Q. WEINBERGER, éd., Curran Associates, Inc., 2012, p. 1097-1105. adresse : http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[25] D. H. HUBEL et T. N. WIESEL, "Receptive fields and functional architecture of monkey striate cortex", en, *The Journal of Physiology*, t. 195, nº 1, p. 215-243, mars 1968, ISSN : 1469-7793. DOI : 10.1113/jphysiol.1968.sp008455. adresse : https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1968.sp008455 (visité le 29/01/2019).

[26] A. CANZIANI, A. PASZKE et E. CULURCIELLO, *An Analysis of Deep Neural Network Models for Practical Applications*, arXiv :1605.07678 [cs], avr. 2017. DOI : 10.48550/arXiv.1605.07678. adresse : http://arxiv.org/abs/1605.07678 (visité le 18/07/2022).

[27] K. SIMONYAN et A. ZISSERMAN, "Very Deep Convolutional Networks for Large-Scale Image Recognition", en, *arXiv :1409.1556 [cs]*, sept. 2014, arXiv : 1409.1556. adresse : http://arxiv.org/abs/1409.1556 (visité le 13/05/2019).

[28] K. HE, X. ZHANG, S. REN et J. SUN, "Deep Residual Learning for Image Recognition", en, *arXiv :1512.03385 [cs]*, déc. 2015, arXiv : 1512.03385. adresse : http://arxiv.org/abs/1512.03385 (visité le 18/02/2020).

[29] M. TAN et Q. V. LE, *EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks*, arXiv :1905.11946 [cs, stat], sept. 2020. adresse : http://arxiv.org/abs/1905.11946 (visité le 19/07/2022).

[30] H. PHAM, Z. DAI, Q. XIE, M.-T. LUONG et Q. V. LE, *Meta Pseudo Labels*, arXiv :2003.10580 [cs, stat] version : 4, mars 2021. adresse : http://arxiv.org/abs/2003.10580 (visité le 19/07/2022).

[31] *TinyML*, en. adresse : https://www.tinyml.org/ (visité le 28/02/2023).

[32] S. J. PAN et Q. YANG, *A Survey on Transfer Learning*.

[33] *TensorFlow*. adresse : https://www.tensorflow.org/?hl=fr (visité le 07/09/2022).

[34] *PyTorch*, en. adresse : https://www.pytorch.org (visité le 07/09/2022).

[35] T. BURD et R. BRODERSEN, *Energy efficient CMOS microprocessor design*. fév. 1995, Pages : 297 vol.1, ISBN : 978-0-8186-6930-9. DOI : 10.1109/HICSS.1995.375385.

[36] *STM32F779II - High-performance and DSP with FPU, Arm Cortex-M7 MCU with 2 Mbytes of Flash memory, 216 MHz CPU, Art Accelerator, L1 cache, HW crypto, SDRAM, TFT, MIPI-DSI, JPEG codec, DFSDM - STMicroelectronics*, en. adresse : https://www.st.com/en/microcontrollers-microprocessors/stm32f779ii.html (visité le 09/08/2022).

[37] *K80-150 MHz Advanced Security and QuadSPI MCU*. adresse : https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/k-series-cortex-m4/k8x-secure/kinetis-k80-150-mhz-advanced-security-and-quadspi-microcontrollers-mcus-based-on-arm-cortex-m4-core:K80_150 (visité le 09/08/2022).

[38] *PIC32MM Family of Microcontrollers | Microchip Technology*. adresse : https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/pic32-32-bit-mcus/pic32mm (visité le 09/08/2022).

[39] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato et N. de Freitas, "Predicting Parameters in Deep Learning", in *Advances in Neural Information Processing Systems*, t. 26, Curran Associates, Inc., 2013. adresse : https://proceedings.neurips.cc/paper/2013/hash/7fec306d1e665bc9c748b5d2b99a6e97-Abstract.html (visité le 18/09/2022).

[40] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally et K. Keutzer, "SqueezeNet : AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", en, *arXiv :1602.07360 [cs]*, fév. 2016, arXiv : 1602.07360. adresse : http://arxiv.org/abs/1602.07360 (visité le 17/01/2019).

[41] A. G. Howard, M. Zhu, B. Chen et al., "MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications", *arXiv :1704.04861 [cs]*, avr. 2017, arXiv : 1704.04861. adresse : http://arxiv.org/abs/1704.04861 (visité le 07/02/2020).

[42] K. He et J. Sun, "Convolutional Neural Networks at Constrained Time Cost", en, p. 8,

[43] L. Sifre, "Rigid-Motion Scattering for Image Classification", Thèse Ph.D. Ecole Polytechnique, Palaiseau, oct. 2014.

[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov et L.-C. Chen, "MobileNetV2 : Inverted Residuals and Linear Bottlenecks", en, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT : IEEE, juin 2018, p. 4510-4520, isbn : 978-1-5386-6420-9. doi : 10.1109/CVPR.2018.00474. adresse : https://ieeexplore.ieee.org/document/8578572/ (visité le 25/04/2020).

[45] A. Howard, M. Sandler, B. Chen et al., "Searching for MobileNetV3", en, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South) : IEEE, oct. 2019, p. 1314-1324, isbn : 978-1-72814-803-8. doi : 10.1109/ICCV.2019.00140. adresse : https://ieeexplore.ieee.org/document/9008835/ (visité le 17/09/2022).

[46] R. Pascanu, T. Mikolov et Y. Bengio, "On the difficulty of training recurrent neural networks", en, p. 9,

[47] T. Elsken, J. H. Metzen et F. Hutter, "Neural Architecture Search : A Survey", en, p. 21,

[48] H. Liu, K. Simonyan et Y. Yang, *DARTS : Differentiable Architecture Search*, arXiv :1806.09055 [cs, stat], avr. 2019. DOI : 10.48550/arXiv.1806.09055. adresse : http://arxiv.org/abs/1806.09055 (visité le 15/09/2022).

[49] H. Wang, Z. Wu, Z. Liu et al., *HAT : Hardware-Aware Transformers for Efficient Natural Language Processing*, arXiv :2005.14187 [cs], mai 2020. DOI : 10.48550/arXiv.2005.14187. adresse : http://arxiv.org/abs/2005.14187 (visité le 15/09/2022).

[50] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan et S. Han, *MCUNet : Tiny Deep Learning on IoT Devices*, arXiv :2007.10319 [cs], nov. 2020. adresse : http://arxiv.org/abs/2007.10319 (visité le 15/09/2022).

[51] J. Lin, W.-M. Chen, H. Cai, C. Gan et S. Han, *MCUNetV2 : Memory-Efficient Patch-based Inference for Tiny Deep Learning*, arXiv :2110.15352 [cs], oct. 2021. DOI : 10.48550/arXiv.2110.15352. adresse : http://arxiv.org/abs/2110.15352 (visité le 16/09/2022).

[52] C. Banbury, C. Zhou, I. Fedorov et al., *MicroNets : Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers*, arXiv :2010.11267 [cs], avr. 2021. adresse : http://arxiv.org/abs/2010.11267 (visité le 15/09/2022).

[53] I. Fedorov, R. P. Adams, M. Mattina et P. Whatmough, "SpArSe : Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers", in *Advances in Neural Information Processing Systems*, t. 32, Curran Associates, Inc., 2019. adresse : https://proceedings.neurips.cc/paper/2019/hash/044a23cadb567653eb51d4eb40acaa88-Abstract.html (visité le 16/09/2022).

[54] E. Liberis, L. Dudziak et N. D. Lane, "uNAS : Constrained Neural Architecture Search for Microcontrollers", en, in *Proceedings of the 1st Workshop on Machine Learning and Systems*, Online United Kingdom : ACM, avr. 2021, p. 70-79, ISBN : 978-1-4503-8298-4. DOI : 10.1145/3437984.3458836. adresse : https://dl.acm.org/doi/10.1145/3437984.3458836 (visité le 16/09/2022).

[55] M. Tan, B. Chen, R. Pang et al., *MnasNet : Platform-Aware Neural Architecture Search for Mobile*, arXiv :1807.11626 [cs], mai 2019. DOI : 10.48550/arXiv.1807.11626. adresse : http://arxiv.org/abs/1807.11626 (visité le 21/09/2022).

[56] X. Dai, Y. Jia, P. Vajda et al., "ChamNet : Towards Efficient Network Design Through Platform-Aware Model Adaptation", en, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA : IEEE, juin 2019, p. 11 390-11 399, ISBN : 978-1-72813-293-8. DOI : 10.1109/CVPR.2019.01166. adresse : https://ieeexplore.ieee.org/document/8953371/ (visité le 21/09/2022).

[57] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun et R. Fergus, "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation", in *Advances in Neural Information Processing Systems*, t. 27, Curran Associates, Inc., 2014. adresse : https://proceedings.neurips.cc/paper/2014/hash/2afe4567e1bf64d32a5527244d104cea-Abstract.html (visité le 16/09/2022).

[58] C. Tai, T. Xiao, Y. Zhang, X. Wang et W. E, *Convolutional neural networks with low-rank regularization*, arXiv :1511.06067 [cs, stat], fév. 2016. adresse : http://arxiv.org/abs/1511.06067 (visité le 16/09/2022).

[59] K. Hayashi, T. Yamaguchi, Y. Sugawara et S.-i. Maeda, "Exploring Unexplored Tensor Network Decompositions for Convolutional Neural Networks", in *Advances in Neural Information Processing Systems*, t. 32, Curran Associates, Inc., 2019. adresse : https://proceedings.neurips.cc/paper/2019/hash/2bd2e3373dce441c6c3bfadd1daa953e-Abstract.html (visité le 17/09/2022).

[60] T. G. Kolda et B. W. Bader, "Tensor Decompositions and Applications", en, *SIAM Review*, t. 51, n° 3, p. 455-500, août 2009, issn : 0036-1445, 1095-7200. doi : 10.1137/07070111X. adresse : http://epubs.siam.org/doi/10.1137/07070111X (visité le 17/09/2022).

[61] A. Goel, C. Tung, Y.-H. Lu et G. K. Thiruvathukal, "A Survey of Methods for Low-Power Deep Learning and Computer Vision", *arXiv :2003.11066 [cs]*, mars 2020, arXiv : 2003.11066. adresse : http://arxiv.org/abs/2003.11066 (visité le 09/09/2020).

[62] L. Deng, G. Li, S. Han, L. Shi et Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks : A Comprehensive Survey", *Proceedings of the IEEE*, t. 108, n° 4, p. 485-532, avr. 2020, Conference Name : Proceedings of the IEEE, issn : 1558-2256. doi : 10.1109/JPROC.2020.2976475.

[63] R. M. Gray et D. L. Neuhoff, "Quantization", *IEEE Transactions on Information Theory*, t. 44, n° 6, p. 2325-2383, sept. 2006, issn : 0018-9448. doi : 10.1109/18.720541. adresse : https://doi.org/10.1109/18.720541 (visité le 17/09/2022).

[64] B. Jacob, S. Kligys, B. Chen et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference", en, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT : IEEE, juin 2018, p. 2704-2713, isbn : 978-1-5386-6420-9. doi : 10.1109/CVPR.2018.00286. adresse : https://ieeexplore.ieee.org/document/8578384/ (visité le 18/09/2022).

[65] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney et K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference", *arXiv :2103.13630 [cs]*, juin 2021, arXiv : 2103.13630. adresse : http://arxiv.org/abs/2103.13630 (visité le 03/03/2022).

[66] A. Zhou, A. Yao, Y. Guo, L. Xu et Y. Chen, *Incremental Network Quantization : Towards Lossless CNNs with Low-Precision Weights*, arXiv :1702.03044 [cs], août 2017. adresse : http://arxiv.org/abs/1702.03044 (visité le 18/09/2022).

[67] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger et Y. Chen, "Compressing Neural Networks with the Hashing Trick", en, p. 10,

[68] E. Park et S. Yoo, *PROFIT : A Novel Training Method for sub-4-bit MobileNet Models*, arXiv :2008.04693 [cs], août 2020. adresse : http://arxiv.org/abs/2008.04693 (visité le 18/09/2022).

[69] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy et D. S. Modha, "LEARNED STEP SIZE QUANTIZATION", en, p. 12, 2020.

[70] S. Zhuo, H. Chen, R. K. Ramakrishnan et al., *An Empirical Study of Low Precision Quantization for TinyML*, arXiv :2203.05492 [cs], mars 2022. adresse : http://arxiv.org/abs/2203.05492 (visité le 18/09/2022).

[71] M. Courbariaux, Y. Bengio et J.-P. David, "BinaryConnect : Training Deep Neural Networks with binary weights during propagations", in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama et R. Garnett, éd., Curran Associates, Inc., 2015, p. 3123-3131. adresse : http://papers.nips.cc/paper/5647-binaryconnect-training-deep-neural-networks-with-binary-weights-during-propagations.pdf (visité le 21/09/2020).

[72] M. Rastegari, V. Ordonez, J. Redmon et A. Farhadi, "XNOR-Net : ImageNet Classification Using Binary Convolutional Neural Networks", en, in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe et M. Welling, éd., t. 9908, Series Title : Lecture Notes in Computer Science, Cham : Springer International Publishing, 2016, p. 525-542, isbn : 978-3-319-46492-3 978-3-319-46493-0. doi : 10.1007/978-3-319-46493-0_32. adresse : http://link.springer.com/10.1007/978-3-319-46493-0_32 (visité le 18/09/2022).

[73] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv et Y. Bengio, *Binarized Neural Networks : Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*, arXiv :1602.02830 [cs], mars 2016. adresse : http://arxiv.org/abs/1602.02830 (visité le 18/09/2022).

[74] D. Przewlocka-Rus, S. S. Sarwar, H. E. Sumbul, Y. Li et B. De Salvo, *Power-of-Two Quantization for Low Bitwidth and Hardware Compliant Neural Networks*, en, Number : arXiv :2203.05025 arXiv :2203.05025 [cs], mars 2022. adresse : http://arxiv.org/abs/2203.05025 (visité le 17/06/2022).

[75] F. Li, B. Zhang et B. Liu, *Ternary Weight Networks*, arXiv :1605.04711 [cs], nov. 2016. adresse : http://arxiv.org/abs/1605.04711 (visité le 18/09/2022).

[76] Y. LeCun, J. Denker et S. Solla, "Optimal Brain Damage", in *Advances in Neural Information Processing Systems*, t. 2, Morgan-Kaufmann, 1989. adresse : https://proceedings.neurips.cc/paper/1989/hash/6c9882bbac1c7093bd250Abstract.html (visité le 18/09/2022).

[77] B. Hassibi et D. Stork, "Second order derivatives for network pruning : Optimal Brain Surgeon", in *Advances in Neural Information Processing Systems*, t. 5, Morgan-Kaufmann, 1992. adresse : https://proceedings.neurips.cc/paper/1992/hash/303ed4c69846ab36c2904d3ba8573050-Abstract.html (visité le 18/09/2022).

[78] S. Han, J. Pool, J. Tran et W. Dally, "Learning both Weights and Connections for Efficient Neural Network", in *Advances in Neural Information Processing Systems*, t. 28, Curran Associates, Inc., 2015. adresse : https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html (visité le 19/09/2022).

[79] Y. Guo, A. Yao et Y. Chen, "Dynamic Network Surgery for Efficient DNNs", in *Advances in Neural Information Processing Systems*, t. 29, Curran Associates, Inc., 2016. adresse : https://proceedings.neurips.cc/paper/2016/hash/2823f4797102ce1a1aec05359cc16dd9-Abstract.html (visité le 19/09/2022).

[80] M. A. Carreira-Perpinan et Y. Idelbayev, ""Learning-Compression" Algorithms for Neural Net Pruning", en, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT : IEEE, juin 2018, p. 8532-8541, isbn : 978-1-5386-6420-9. doi : 10.1109/CVPR.2018.00890. adresse : https://ieeexplore.ieee.org/document/8578988/ (visité le 19/09/2022).

[81] T. Gale, E. Elsen et S. Hooker, *The State of Sparsity in Deep Neural Networks*, arXiv :1902.09574 [cs, stat], fév. 2019. adresse : http://arxiv.org/abs/1902.09574 (visité le 18/09/2022).

[82] H. Li, A. Kadav, I. Durdanovic, H. Samet et H. P. Graf, *Pruning Filters for Efficient ConvNets*, arXiv :1608.08710 [cs], mars 2017. adresse : http://arxiv.org/abs/1608.08710 (visité le 19/09/2022).

[83] Y. He, G. Kang, X. Dong, Y. Fu et Y. Yang, *Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks*, en, arXiv :1808.06866 [cs], août 2018. adresse : http://arxiv.org/abs/1808.06866 (visité le 19/09/2022).

[84] J. Ye, X. Lu, Z. Lin et J. Z. Wang, *Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers*, arXiv :1802.00124 [cs], fév. 2018. adresse : http://arxiv.org/abs/1802.00124 (visité le 19/09/2022).

[85] Y. He, P. Liu, Z. Wang, Z. Hu et Y. Yang, *Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration*, arXiv :1811.00250 [cs], juill. 2019. adresse : http://arxiv.org/abs/1811.00250 (visité le 19/09/2022).

[86] Y. Bengio, A. Courville et P. Vincent, *Representation Learning : A Review and New Perspectives*, en, arXiv :1206.5538 [cs], avr. 2014. adresse : http://arxiv.org/abs/1206.5538 (visité le 20/09/2022).

[87] J. Gou, B. Yu, S. J. Maybank et D. Tao, "Knowledge Distillation : A Survey", *International Journal of Computer Vision*, t. 129, nº 6, p. 1789-1819, juin 2021, arXiv :2006.05525 [cs, stat], issn : 0920-5691, 1573-1405. doi : 10.1007/s11263-021-01453-z. adresse : http://arxiv.org/abs/2006.05525 (visité le 20/09/2022).

[88] C. Buciluǎ, R. Caruana et A. Niculescu-Mizil, "Model compression", in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, sér. KDD '06, New York, NY, USA : Association for Computing Machinery, août 2006, p. 535-541, ISBN : 978-1-59593-339-3. DOI : 10.1145/1150402.1150464. adresse : https://doi.org/10.1145/1150402.1150464 (visité le 20/09/2022).

[89] G. Hinton, O. Vinyals et J. Dean, *Distilling the Knowledge in a Neural Network*, arXiv :1503.02531 [cs, stat], mars 2015. adresse : http://arxiv.org/abs/1503.02531 (visité le 20/09/2022).

[90] J. Ba et R. Caruana, "Do Deep Nets Really Need to be Deep ?", in *Advances in Neural Information Processing Systems*, t. 27, Curran Associates, Inc., 2014. adresse : https://proceedings.neurips.cc/paper/2014/hash/ea8fcd92d59581717e06eb187f10666d-Abstract.html (visité le 20/09/2022).

[91] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta et Y. Bengio, *FitNets : Hints for Thin Deep Nets*, arXiv :1412.6550 [cs], mars 2015. DOI : 10.48550/arXiv.1412.6550. adresse : http://arxiv.org/abs/1412.6550 (visité le 20/09/2022).

[92] J. Yim, D. Joo, J. Bae et J. Kim, "A Gift from Knowledge Distillation : Fast Optimization, Network Minimization and Transfer Learning", en, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI : IEEE, juill. 2017, p. 7130-7138, ISBN : 978-1-5386-0457-1. DOI : 10.1109/CVPR.2017.754. adresse : http://ieeexplore.ieee.org/document/8100237/ (visité le 20/09/2022).

[93] Y. Zhang, T. Xiang, T. M. Hospedales et H. Lu, "Deep Mutual Learning", 2018, p. 4320-4328. adresse : https://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_Deep_Mutual_Learning_CVPR_2018_paper.html (visité le 20/09/2022).

[94] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao et K. Ma, *Be Your Own Teacher : Improve the Performance of Convolutional Neural Networks via Self Distillation*, arXiv :1905.08094 [cs, stat], mai 2019. adresse : http://arxiv.org/abs/1905.08094 (visité le 20/09/2022).

[95] J. H. Cho et B. Hariharan, *On the Efficacy of Knowledge Distillation*, arXiv :1910.01348 [cs], oct. 2019. adresse : http://arxiv.org/abs/1910.01348 (visité le 20/09/2022).

[96] A. Mishra et D. Marr, *Apprentice : Using Knowledge Distillation Techniques To Improve Low-Precision Network Accuracy*, arXiv :1711.05852 [cs], nov. 2017. adresse : http://arxiv.org/abs/1711.05852 (visité le 21/09/2022).

[97] A. Polino, R. Pascanu et D. Alistarh, *Model compression via distillation and quantization*, arXiv :1802.05668 [cs], fév. 2018. adresse : http://arxiv.org/abs/1802.05668 (visité le 21/09/2022).

[98] H. Alemdar, V. Leroy, A. Prost-Boucle et F. Pétrot, *Ternary Neural Networks for Resource-Efficient AI Applications*, arXiv :1609.00222 [cs], fév. 2017. adresse : http://arxiv.org/abs/1609.00222 (visité le 21/09/2022).

[99] Y. Li, S. Gu, C. Mayer, L. Van Gool et R. Timofte, "Group Sparsity : The Hinge Between Filter Pruning and Decomposition for Network Compression", en, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA : IEEE, juin 2020, p. 8015-8024, ISBN : 978-1-72817-168-5. DOI : 10.1109/CVPR42600.2020.00804. adresse : https://ieeexplore.ieee.org/document/9157445/ (visité le 21/09/2022).

[100] N. Aghli et E. Ribeiro, "Combining Weight Pruning and Knowledge Distillation For CNN Compression", en, in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Nashville, TN, USA : IEEE, juin 2021, p. 3185-3192, ISBN : 978-1-66544-899-4. DOI : 10.1109/CVPRW53098.2021.00356. adresse : https://ieeexplore.ieee.org/document/9523139/ (visité le 21/09/2022).

[101] B. Hawks, J. Duarte, N. J. Fraser, A. Pappalardo, N. Tran et Y. Umuroglu, "Ps and Qs : Quantization-aware pruning for efficient low latency neural network inference", *Frontiers in Artificial Intelligence*, t. 4, p. 676 564, juill. 2021, arXiv :2102.11289 [hep-ex, physics :physics], ISSN : 2624-8212. DOI : 10.3389/frai.2021.676564. adresse : http://arxiv.org/abs/2102.11289 (visité le 21/09/2022).

[102] S. Han, H. Mao et W. J. Dally, *Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*, en, arXiv :1510.00149 [cs], fév. 2016. adresse : http://arxiv.org/abs/1510.00149 (visité le 21/09/2022).

[103] *Keil Product Overview*. adresse : https://www.keil.com/product/ (visité le 07/09/2022).

[104] *STM32Cube Development Software - STM32 Open Development Environment - STMicroelectronics*, en. adresse : https://www.st.com/en/ecosystems/stm32cube.html (visité le 07/09/2022).

[105] *MCUXpresso Integrated Development Environment (IDE)*, en. adresse : https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE (visité le 07/09/2022).

[106] A. Ltd, *CMSIS*, en. adresse : https://developer.arm.com/tools-and-software/embedded/cmsis (visité le 07/09/2022).

[107] *Keras : the Python deep learning API*. adresse : https://keras.io/ (visité le 07/09/2022).

[108] *Caffe2*. adresse : http://caffe2.ai/ (visité le 07/09/2022).

[109] *Apache MXNet*, en. adresse : https://mxnet.apache.org/versions/1.9.1/ (visité le 07/09/2022).

[110] *ONNX | Home*. adresse : https://onnx.ai/ (visité le 07/09/2022).

[111] L. Lai, N. Suda et V. Chandra, "CMSIS-NN : Efficient Neural Network Kernels for Arm Cortex-M CPUs", *arXiv :1801.06601 [cs]*, jan. 2018, arXiv : 1801.06601. adresse : http://arxiv.org/abs/1801.06601 (visité le 02/01/2021).

[112] *X-CUBE-AI - AI expansion pack for STM32CubeMX - STMicroelectronics*, en. adresse : https://www.st.com/en/embedded-software/x-cube-ai.html (visité le 07/09/2022).

[113] R. David, J. Duke, A. Jain et al., "TensorFlow Lite Micro : Embedded Machine Learning on TinyML Systems", *arXiv :2010.08678 [cs]*, oct. 2020, arXiv : 2010.08678. adresse : http://arxiv.org/abs/2010.08678 (visité le 26/10/2020).

[114] A. Sankaran, O. Mastropietro, E. Saboori et al., "Deeplite Neutrino : An End-to-End Framework for Constrained Deep Learning Model Optimization", arXiv, rapp. tech. arXiv :2101.04073, jan. 2021, arXiv :2101.04073 [cs] type : article. adresse : http://arxiv.org/abs/2101.04073 (visité le 30/05/2022).

[115] M. Deutel, P. Woller, C. Mutschler et J. Teich, "Deployment of Energy-Efficient Deep Learning Models on Cortex-M based Microcontrollers using Deep Compression", arXiv, rapp. tech. arXiv :2205.10369, mai 2022, arXiv :2205.10369 [cs] type : article. adresse : http://arxiv.org/abs/2205.10369 (visité le 30/05/2022).

[116] L. Heim, A. Biri, Z. Qu et L. Thiele, "Measuring what Really Matters : Optimizing Neural Networks for TinyML", *arXiv :2104.10645 [cs]*, avr. 2021, arXiv : 2104.10645. adresse : http://arxiv.org/abs/2104.10645 (visité le 25/01/2022).

[117] A. Wong, *NetScore : Towards Universal Metrics for Large-scale Performance Analysis of Deep Neural Networks for Practical On-Device Edge Usage*, arXiv :1806.05512 [cs, stat], août 2018. adresse : http://arxiv.org/abs/1806.05512 (visité le 23/09/2022).

[118] D. Velasco-Montero, J. Fernandez-Berni, R. Carmona-Galan et A. Rodriguez-Vazquez, "PreVIous : A Methodology for Prediction of Visual Inference Performance on IoT Devices", *arXiv :1912.06442 [cs]*, mars 2020, arXiv : 1912.06442. doi : 10.1109/JIOT.2020.2981684. adresse : http://arxiv.org/abs/1912.06442 (visité le 06/10/2021).

[119] S. Williams, A. Waterman et D. Patterson, "Roofline : An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures", en, rapp. tech. 1407078, sept. 2009, p. 1 407 078. doi : 10.2172/1407078. adresse : http://www.osti.gov/servlets/purl/1407078/ (visité le 23/09/2022).

[120] *Mark I Perceptron at the Cornell Aeronautical Laboratory*, en. adresse : https://digital.library.cornell.edu/catalog/ss:550351 (visité le 23/10/2022).

[121] E. Säckinger, B. E. Boser, J. M. Bromley, Y. LeCun et L. D. Jackel, "Application of the ANNA neural network chip to high-speed character recognition", en, *IEEE Transactions on Neural Networks*, t. 3, n° 3, p. 498-505, 1992, Number : 3 Publisher : IEEE, issn : 1045-9227. adresse : https://oro.open.ac.uk/35661/ (visité le 23/10/2022).

[122] N. P. Jouppi, C. Young, N. Patil et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, sér. ISCA '17, New York, NY, USA : Association for Computing Machinery, juin 2017, p. 1-12, isbn : 978-1-4503-4892-8. doi : 10.1145/3079856.3080246. adresse : https://doi.org/10.1145/3079856.3080246 (visité le 02/09/2022).

[123] A. Putnam, A. M. Caulfield, E. S. Chung et al., "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services", en, p. 12,

[124] Y.-H. Chen, T.-J. Yang, J. Emer et V. Sze, *Eyeriss v2 : A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices*, arXiv :1807.07928 [cs], mai 2019. adresse : http://arxiv.org/abs/1807.07928 (visité le 02/09/2022).

[125] F. Conti, D. Rossi, A. Pullini, I. Loi et L. Benini, "PULP : A Ultra-Low Power Parallel Accelerator for Energy-Efficient and Flexible Embedded Vision", *Journal of Signal Processing Systems*, t. 84, sept. 2016. doi : 10.1007/s11265-015-1070-9.

[126] A. Garofalo, M. Rusci, F. Conti, D. Rossi et L. Benini, "PULP-NN : accelerating quantized neural networks on parallel ultra-low-power RISC-V processors", en, p. 21,

[127] *PULP Implementation*. adresse : https://pulp-platform.org/implementation.html (visité le 06/09/2022).

[128] *Store*, en-US. adresse : https://greenwaves-technologies.com/store/ (visité le 06/09/2022).

[129] A. Ltd, *Ethos-U55 Machine Learning Processor*, en. adresse : https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55 (visité le 23/10/2022).

[130] *alwaysAI - Computer Vision Platform*. adresse : https://alwaysai.co/ (visité le 20/07/2021).

[131] *Qeexo AutoML*, en-US. adresse : https://qeexo.com/ (visité le 20/07/2021).

[132] *SensiML Analytics Toolkit : AI Tools for IoT Developers*, en-US. adresse : https://sensiml.com/ (visité le 24/09/2022).

[133] *Edge Impulse*, en. adresse : https://www.edgeimpulse.com (visité le 20/07/2021).

[134] T.-J. Yang, A. Howard, B. Chen et al., "NetAdapt : Platform-Aware Neural Network Adaptation for Mobile Applications", en, in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu et Y. Weiss, éd., t. 11214, Series Title : Lecture Notes in Computer Science, Cham : Springer International Publishing, 2018, p. 289-304, isbn : 978-3-030-01248-9 978-3-030-01249-6. doi : 10.1007/978-3-030-01249-6_18. adresse : http://link.springer.com/10.1007/978-3-030-01249-6_18 (visité le 30/09/2022).

[135] F. Paissan, A. Ancilotto et E. Farella, *PhiNets : a scalable backbone for low-power AI at the edge*, arXiv :2110.00337 [cs], oct. 2021. adresse : http://arxiv.org/abs/2110.00337 (visité le 30/09/2022).

[136] *ULINKplus Debug Adapter*. adresse : https://www2.keil.com/mdk5/ulink/ulinkplus (visité le 30/09/2022).

[137] *Wisebatt | Where do you want to start ?* Adresse : https://wisebatt.com/ (visité le 30/09/2022).

[138] W. Dron, K. Hachicha et P. Garda, "A fixed frequency sampling method for wireless sensors power consumption estimation", in *2013 IEEE 11th International New Circuits and Systems Conference (NEWCAS)*, juin 2013, p. 1-4. doi : 10.1109/NEWCAS.2013.6573653.

[139] W. Dron, S. Duquennoy, T. Voigt, K. Hachicha et P. Garda, "An Emulation-Based Method for Lifetime Estimation of Wireless Sensor Networks", in *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, ISSN : 2325-2944, mai 2014, p. 241-248. doi : 10.1109/DCOSS.2014.10.

[140] W. Dron, K. Hachicha et P. Garda, "Simulation method of the functionality of an electronic circuit and program", Française, brev. WO2017005883A1, Brevet français : FR3038755A1, Brevet Européen n° EP3320461B1, Extension aux USA US20180203957A1, juill. 2018.

[141] *The HDF5® Library & File Format*, en-US. adresse : https://www.hdfgroup.org/solutions/hdf5/ (visité le 16/02/2023).

[142] Y. LeCun et C. Cortes, *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. adresse : http://yann.lecun.com/exdb/mnist/ (visité le 28/10/2022).

[143] *NUCLEO-L496ZG - STM32 Nucleo-144 development board with STM32L496ZG MCU, supports Arduino, ST Zio and morpho connectivity - STMicroelectronics*, en. adresse : https://www.st.com/en/evaluation-tools/nucleo-l496zg.html (visité le 28/10/2022).

[144] *NUCLEO-F446ZE - STM32 Nucleo-144 development board with STM32F446ZE MCU, supports Arduino, ST Zio and morpho connectivity - STMicroelectronics*, en. adresse : https://www.st.com/en/evaluation-tools/nucleo-f446ze.html (visité le 28/10/2022).

[145] *NUCLEO-F746ZG - STM32 Nucleo-144 development board with STM32F746ZG MCU, supports Arduino, ST Zio and morpho connectivity - STMicroelectronics*, en. adresse : https://www.st.com/en/evaluation-tools/nucleo-f746zg.html (visité le 28/10/2022).

[146] T. Garbay, K. Hachicha, P. Dobias et al., "Accurate Estimation of the CNN Inference Cost for TinyML Devices", in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, 2022.

[147] P. Dobiáš, T. Garbay, B. Granado, K. Hachicha et A. Pinna, "Comparative Study of Scheduling a Convolutional Neural Network on Multicore MCU", in *Design and Architecture for Signal and Image Processing*, 2022.

[148] T. Garbay, P. Dobias, W. Dron et al., "CNN Inference Costs Estimation on Microcontrollers : the EST Primitive-based Model", in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021.

[149] T. Garbay, O. Chuquimia, A. Pinna, H. Sahbi, X. Dray et B. Granado, "Distilling the knowledge in CNN for WCE screening tool", in *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, IEEE, 2019.

[150] T. Garbay, K. Hachicha, P. Dobiáš et al., *Accurate Estimation of the CNN Inference Cost within Microcontrollers*, Poster, Published : TinyML EMEA 2022, 2022.

[151] T. Garbay, P. Dobiáš, W. Dron et al., "Estimation du coût d'inférence des réseaux de neurones convolutifs dans un microcontrôleur", in *16ème Colloque du GDR SoC2*, Strasbourg, France, 2022.

[152] O. Chuquimia, T. Garbay, W. Xu et al., "Study to integrate CNN inside a WCE to realize a screening tool", in *Journées d'Etude sur la TéléSanté*, 2019.