



SORBONNE UNIVERSITE

MASTER 2 INGÉNIÈRIE DES SYSTÈMES INTELLIGENTS

RAPPORT DE STAGE DE FIN D'ÉTUDES

Estimation de l'impact des techniques de réduction des réseaux de neurones sur Microcontrôleur

Réalisé par
Karim HOCINE

Encadrants
Thomas GARBAY
Bertrand GRANADO
Khalil HACHICHA

16 septembre 2022

Remerciements

Je tiens à remercier toute l'équipe SYEL dans laquelle j'ai eu l'occasion d'évoluer pendant ces six mois de stage et plus particulièrement mes encadrants Thomas GARBAY, Bertrand GRANADO et Khalil HACHICHA, avec qui j'ai beaucoup appris. Ce fut pour moi une expérience humaine et professionnelle extraordinaire durant laquelle j'ai rencontré des personnes fortement sympathiques avec qui j'ai tissé des liens d'amitiés.

Résumé

Certains travaux de recherche se sont intéressés à l'intégration d'une intelligence artificielle à des systèmes embarqués dans l'optique de développer de nouveaux systèmes intelligents qui pourraient permettre l'accélération de la transformation numérique de certains secteurs, le développement de nouveaux systèmes autonomes, ou créer des écosystèmes numériques intelligents. C'est de là qu'est né le paradigme du *Tiny Machine Learning*.

Les systèmes embarqués étant très souvent limités en ressources matérielles (puissance de calcul, mémoire) l'intégration d'algorithmes d'intelligence artificielle peut s'avérer être une tâche très difficile, dû à l'inadaptation de ces algorithmes aux contraintes des systèmes embarqués, surtout dans le cas des réseaux de neurones profonds où les architectures peuvent vite devenir très conséquentes en matière de ressources de calcul nécessaires à leur fonctionnement, l'espace mémoire permettant leur intégration et la consommation énergétique essentielle à leur déploiement. C'est de là qu'apparaît un besoin de techniques et de meilleures pratiques qui permettent la création de modèles efficaces plus adaptés aux contraintes matérielles des systèmes embarqués.

Il existe trois grandes catégories de techniques de réduction de réseaux de neurones : celles basées sur la quantification, celles basées sur la réduction des poids et celles basées sur la distillation de connaissances. Le but de ce rapport est de faire une étude comparative approfondie sur les différentes approches utilisées dans l'état de l'art pour la réduction des réseaux de neurones en implémentant un algorithme correspondant à chaque technique et en faisant une estimation de l'impact de son utilisation sur microcontrôleur.

Abstract

Some research works have focused on the integration of artificial intelligence into embedded systems with the perspective of developing new intelligent systems that could accelerate the digital transformation of certain sectors, develop new autonomous systems, or create intelligent ecosystems. This is where the paradigm of Tiny Machine Learning was born.

Embedded systems are often limited in hardware resources (computing power, memory) and the integration of artificial intelligence algorithms can be a very difficult task, due to the unsuitability of these algorithms to the constraints of embedded systems, especially in the case of deep neural networks, where the architectures can quickly become very consequent in terms of the computing resources necessary for their functioning, the memory space allowing their integration and the energy consumption essential for their deployment. This is where the need for techniques and best practices that allow the creation of efficient models more adapted to the hardware constraints of embedded systems arises.

There are three main categories of neural network reduction techniques: those based on quantization, those based on weight reduction and those based on knowledge distillation. The aim of this report is to make an in-depth comparative study of the different approaches used in the state of the art for neural network reduction by implementing an algorithm corresponding to each technique and by estimating the impact of its use on microcontrollers.

Organisme d'accueil

Fort de plus de 500 membres, dont 200 permanents, le Laboratoire d'informatique de Paris 6 (LIP6) est l'un des plus grands laboratoires de recherche en informatique logicielle et matérielle de France. Situé en plein cœur de Paris, le LIP6 est une Unité Mixte de Recherche (UMR 7606) de Sorbonne Université (SU) et du Centre National de la Recherche Scientifique (CNRS).

Les 21 équipes du laboratoire couvrent un champ large des sciences informatiques : de l'électronique jusqu'à l'intelligence artificielle. Les collaborations du LIP6 relèvent tant de la recherche fondamentale (modélisation et résolution de problèmes fondamentaux) que de la recherche appliquée (mise en œuvre et validation de solutions en conditions réelles).

Les activités du laboratoire s'articulent autour de quatre axes transverses :

- Intelligence artificielle et sciences des données (AID)
- Architecture, systèmes et réseaux (ASN)
- Sécurité, sûreté et fiabilité (SSR)
- Théorie et outils mathématiques pour l'informatique (TMC).

La production scientifique au sein du LIP6 est très dynamique, durant les années 2017-2022, les membres du LIP6 ont publié 32 livres, 699 articles en revues et 1 385 communications dans des conférences avec actes. Cela permet au laboratoire de disséminer ses travaux au sein des différentes communautés scientifiques et industrielles ainsi que de maintenir et de développer de nouveaux partenariats.

Les partenariats industriels du LIP6 prennent des formes très diverses, toujours adaptées à chaque projet : une soixantaine de thèses CIFRE actuellement en cours, des laboratoires communs, des chaires d'excellence, des prestations de conseil et/ou de services.

La coopération internationale est une constante pour les activités du laboratoire. Il entretient des relations suivies avec des universités de nombreux pays comme le Brésil, le Canada, les États-Unis, le Japon, la Chine et de nombreux pays européens. En complément de la recherche académique, le LIP6 a une longue tradition de coopération avec des partenaires industriels dans de nombreux projets nationaux, européens ou internationaux.

À l'échelle locale, le laboratoire est également très actif au sein de Sorbonne Université où il favorise la collaboration transdisciplinaire entre laboratoires. Le LIP6 est très fortement impliqué dans deux initiatives majeures de Sorbonne Université : SCAI (*Sorbonne Center for Artificial Intelligence*) lancée en 2019, et QICS (*Quantum Information Center Sorbonne*), lancé en 2020. Le laboratoire est aussi grandement impliqué dans des enseignements liés à la recherche en master. L'école doctorale EDITE de Paris (Ecole Doctorale d'Informatique, Télécommunication et Electronique de Paris) accueille les 181 doctorants du laboratoire.

Sorbonne Université est une université pluridisciplinaire de recherche créée au 1er janvier 2018 par regroupement des universités Paris-Sorbonne et l'Université Pierre et Marie Curie (UPMC). Grâce aux 55 300 étudiants, 6 400 enseignants-chercheurs et chercheurs et 3 600 personnels administratifs et techniques qui la font vivre au quotidien, Sorbonne Université promeut la diversité, la créativité, l'innovation et l'ouverture sur le monde.

Le Centre national de la recherche scientifique (CNRS) est un organisme public de recherche pluridisciplinaire placé sous la tutelle du ministère de l'Enseignement supérieur, de la Recherche et de l'Innovation. Le CNRS est une institution de recherche parmi les plus importantes au monde. Pour relever les grands défis présents et à venir, ses scientifiques explorent le vivant, la matière, l'Univers et le fonctionnement des sociétés humaines. Internationalement reconnu pour l'excellence de ses travaux scientifiques, il est une référence aussi bien dans l'univers de la recherche et développement que pour le grand public.

Table des matières

1	Introduction	7
1.1	Tiny Machine Learning	7
1.1.1	Utilisation des microcontrôleurs	7
1.1.2	Outils d'intégration	7
1.1.3	Challenges	7
1.2	Techniques de réduction de réseaux de neurones	8
1.2.1	Les méthodes de réduction de réseaux de neurones	8
1.2.2	Évaluation des performances	8
1.3	Les réseaux de neurones	9
1.3.1	Les réseaux de neurones convolutifs	10
2	Distillation de connaissances	12
2.1	Modélisation et transfert des connaissances	12
2.1.1	Connaissance basée sur la réponse	13
2.1.2	Connaissance basée sur les caractéristiques	13
2.1.3	Connaissance basée sur la relation	14
2.2	Stratégies de distillation des connaissances	14
2.2.1	Distillation hors ligne	14
2.2.2	Distillation en ligne	14
2.2.3	Auto Distillation	15
2.3	Architecture Enseignant-Elève	15
2.4	Partie expérimentale	15
2.4.1	Méthodologie	15
2.4.2	Résultats et discussion	17
2.4.2.1	Paramètres α et β	17
2.4.2.2	Paramètre de température T	18
2.4.3	Conclusion	19
3	Quantification des réseaux de neurones	20
3.1	Processus de quantification	20
3.2	Quantification et entraînement des réseaux de neurones	21
3.2.1	Quantification pendant l'entraînement	21
3.2.2	Quantification post entraînement	22
3.3	Quantification et inférence	22
3.4	Partie expérimentale	23
3.4.1	Méthodologie	23
3.4.2	Résultats et discussion	25
3.4.3	Conclusion	26
4	Élagage	28
4.1	Les approches d'élagage	28
4.2	Partie expérimentale	29
4.2.1	Méthodologie	29
4.2.2	Résultats	30
4.2.3	Conclusion	32
5	Conclusion Générale	33
5.1	Comparaison des méthodes de réduction de réseaux de neurones implémentées	33
5.2	Perspectives d'amélioration de nos travaux	33
5.3	Difficultés rencontrées	33

Liste des abréviations

- **AdaIn** : Normalisation d'instance adaptative (*Adaptive Instance Normalization*).
- **CNN** : Réseau de neurones convolutifs (*Convolutional Neural Network*).
- **CPU** : Unité centrale de traitement (*Central Processing Unit*).
- **DNN** : Réseau de neurones profond (*Deep Neural Network*).
- **FC** : Entièrement connecté (*Fully Connected*).
- **FLOPs** : Opérations flottantes par seconde (*floating-point operations per second*).
- **FP32** : Représentation des nombres en point flottant 32 bits.
- **GPU** : Processeur graphique (*Graphics Processing Unit*).
- **IA** : Intelligence artificielle.
- **INT8** : Représentation entière des nombres en virgule fixe 8 bits.
- **KD** : Distillation de connaissances (*Knowledge Distillation*).
- **ML** : Apprentissage machine (*Machine Learning*).
- **NN** : Réseaux de neurones (*Neural Network*).
- **PQT** : Post quantification (*Post Quantization*).
- **QAT** : Quantification pendant l'entraînement (*Quantization Aware Training*).
- **µC** : Microcontrôleur.

Table des figures

1	Réseau de neurones avec des couches entièrement connectées. [6]	9
2	Réseau de neurones convolutifs.[7]	10
3	Produit de convolution.[7]	11
4	Opération de sous-échantillonnage : par valeur maximale (gauche), par valeur moyenne (droite) [8].	11
5	Principe de la distillation de connaissances.	12
6	Modélisation de la connaissance dans un réseau de neurones [10].	12
7	Modélisation de la connaissance basée sur la réponse [10].	13
8	Modélisation de la connaissance basée sur les caractéristiques [10].	13
9	Modélisation de la connaissance basée sur la relation [10].	14
10	Les différents schémas de distillation de connaissances [10].	14
11	Schéma blocs d'une distillation de connaissances hors ligne.	15
12	Schéma représentant l'intégration du processus de pré traitement des données au réseau Resnet8.	17
13	Évolution des courbes de précision du réseau élève en fonction du paramètre α . .	18
14	Évolution des courbes de perte du réseau élève en fonction du paramètre α	18
15	Évolution des courbes de précision du réseau élève en fonction du paramètre T . .	19
16	Évolution des courbes de perte du réseau élève en fonction du paramètre T	19
17	Intervalle de représentation : Symétrique (gauche), Asymétrique(Droite) [18]. . .	21
18	Comparatif des approches d'ajustement des paramètres d'un modèle lors de la quantification : QAT (gauche), PQT (Droite) [18].	21
19	Les différents schémas d'inférence : Standard (gauche), Quantification simulée (centre), Quantification en entiers uniquement (droite) [18].	22
20	(Gauche) Comparaison entre le débit maximum pour différentes précisions en bit sur GPU Titan RTX et A100. (Droite) Comparaison du coût énergétique correspondant et du coût relatif de la surface pour différentes précisions pour des technologie 45 nm [18].	23
21	Schéma d'apprentissage par quantification simulée [5].	24
22	Schéma d'inférence utilisant l'arithmétique entière uniquement [5].	25
23	Courbe d'entraînement d'une réseau Lenet5 avec une schéma de quantification simulée.	26
24	Architecture d'un réseau de neurones avant et après élagage.[21]	28
25	Conditions nécessaires pour l'application du critère d'élagage basé sur la norme[23].	29
26	Évolution de la précision maximale sur la base de validation en fonction du taux d'élagage.	31
27	Évolution de la précision maximale sur la base de validation en fonction du nombre de paramètres du réseau.	32

1 Introduction

1.1 Tiny Machine Learning

Le terme de "*Tiny Machine Learning*" (*TinyML*) désigne le paradigme proposant l'intégration des algorithmes d'apprentissage machine (*machine learning*) à des systèmes embarqués basés sur du matériel à faible puissance [1]. Ce concept a pour objectif le développement d'applications et de services totalement autonomes, à faible consommation d'énergie et indépendants en matière de traitement de l'information, qui peuvent ensuite être intégrés dans des systèmes plus complexes ou dans un écosystème numérique (maison connectée, smart grids ...). De plus, cela permettrait l'amélioration de la sécurité des systèmes embarqués en traitant les données "*On-chip*". Ainsi, les données ne seraient plus transmises à l'extérieur du système vers des serveurs. Cela permettrait d'éviter de détériorer la qualité des données lors du processus de communication, d'améliorer la sécurité du système en évitant toute fuite de données et de réduire la consommation énergétique supplémentaire liée au transport des données.

La transformation numérique de divers domaines et secteurs a démocratisé l'utilisation quotidienne d'une pléthore d'objets connectés, ex. smartphones, des collecteurs de données environnementales, actionneurs, etc. L'intégration d'une intelligence artificielle dans ces petits appareils apporte une série d'avantages et de perspectives. Parmi ces perspectives on citera :

- Accélération de la transformation numérique de certains secteurs tels que la santé, l'agriculture (ex. Smart monitoring, Contrôle de qualité, sécurité...)
- Optimisation des chaînes de production (*Industry 4.0*)
- Développement des véhicules autonomes.
- Développement d'espace intelligents (ex. maisons intelligentes...)
- Amélioration de la flexibilité et des capacités de calculs des systèmes embarqués.
- Généralisation de l'utilisation de l'intelligence artificielle.

1.1.1 Utilisation des microcontrôleurs

Grâce à leur faible coût de production et faible consommation d'énergie, les microcontrôleurs permettraient un déploiement à grande échelle de systèmes intégrant une intelligence artificielle. Cependant les μC sont soumis à des contraintes matérielles fortes : Peu de mémoire, puissance de calcul limitée, budget énergétique limité.... Ces contraintes doivent être prises en considération lors de l'intégration des algorithmes d'apprentissage machine qui ne sont pas toujours adaptés à une intégration directe. De plus, dans une problématique de *TinyML* le problème posé par l'utilisation des μC est l'hétérogénéité du matériel ce qui rend presque impossible la généralisation des *framework* [1].

1.1.2 Outils d'intégration

Ils existent trois approches possibles lors de l'intégration d'algorithmes d'apprentissage machine sur μC . La première approche consiste à utiliser des *frameworks* qui peuvent importer directement des modèles pré-entraînés et les adapter en un langage compréhensible par le μC (ex. Tensorflow Lite [2]). La deuxième se base sur l'intégration des *frameworks* d'apprentissage machine directement sur les μC (ex. AIfES [3]) en créant des plateformes de développement dédiées. Cela permet de générer des modèles directement à partir des données collectées par l'appareil ou aussi de faire de l'apprentissage non supervisé. La dernière solution, est l'utilisation d'unité de calculs dédiées à l'apprentissage machine, par exemple : Un *Tensor Processing Unit* (TPU), ce qui est la solution la moins commune vu le prix des plateformes.

1.1.3 Challenges

En analysant les diverses contraintes matérielles et logicielles, divers challenges peuvent être identifiés :

- Pour une meilleure comparaison des méthodes d'intégration une généralisation des *frameworks* est nécessaire. Cela permettra un meilleur *benchmarking* performances.
- Dans le but de développer des systèmes totalement autonomes, la question de la consommation énergétique reste cruciale.
- Les avancées scientifiques sur des sujets tels que les DNN montrent que performance rime avec puissance de calcul et souvent avec grande consommation d'énergie. De ce fait, prévoir les utilisations de demain en développant de nouvelles technologies matérielles plus puissantes prenant en considération les contraintes énergétiques est impératif.

1.2 Techniques de réduction de réseaux de neurones

Au cours de la décennie passée, les réseaux de neurones ont réalisé des avancées spectaculaires dans divers domaines d'application ex. la classification d'images, la détection d'objets, la conduite autonome ... Parmi les différents algorithmes existants, les réseaux de neurones convolutifs (CNN) sont particulièrement populaires, car ils excellent dans l'apprentissage des caractéristiques spatiales.

À mesure que les tâches augmentent en complexité, les architectures des réseaux de neurones deviennent plus profondes, plus coûteuses en ressources de calcul et en espace mémoire occupé. De ce fait, il devient plus complexes d'intégrer des réseaux de neurones profonds (DNN) sur des μC qui sont soumis à des contraintes matérielles. C'est pour cela que le développement d'algorithmes pour réduire les coûts de calcul et de stockage lors de l'inférence sont essentiels.

1.2.1 Les méthodes de réduction de réseaux de neurones

Des travaux récents ont montré qu'avec l'utilisation de techniques d'approximation, le déploiement des réseaux de neurones devient possible sur des cibles matérielles fortement contraintes grâce à la réduction des besoins en mémoire et en ressources de calcul. Cela grâce à la réduction de l'espace mémoire utilisé et de la complexité de calcul [4].

Les algorithmes de réduction de réseaux de neurones peuvent être classés en trois grandes catégories [4] :

1. **La Distillations de connaissances** : consiste à transférer les connaissances d'un réseau complexe, très profond vers un réseau plus simple et relativement moins profond.
2. **La Quantification** : les méthodes de quantification consistent en la réduction de la précision des poids et/ou activations d'un réseau de neurones. Par exemple passage du FP32 au INT8 [5].
3. **Réduction de poids** : ces méthodes consistent à supprimer les paramètres et poids redondants d'un réseau ex. Élagage.

1.2.2 Évaluation des performances

A fin de comparer l'efficacité des différentes méthodes de réduction de réseaux de neurones, il existe dans l'état de l'art divers critères qui sont pris en considération [4] :

1. **Précision (*Accuracy*)** : permet d'évaluer la proportion des classifications correctes sur la base d'entraînement/de test.
2. **Taux de compression** : rapport entre l'espace mémoire occupé par le réseau compressé et celui occupé par le réseau d'origine (*Baseline*).
3. **Débit** : caractérise le nombre d'opérations de classifications effectuées par unité de temps (cl/s).
4. **Latence** : temps nécessaire pour effectuer une opération de classification exprimé en second (s).
5. **Consommation énergétique** : estimée en opération de classification par unité d'énergie (cl/J).

1.3 Les réseaux de neurones

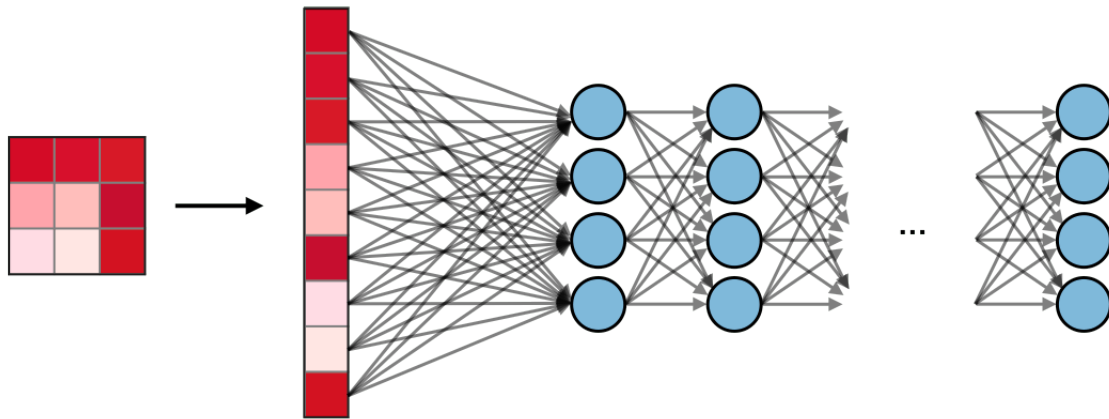


FIGURE 1 – Réseau de neurones avec des couches entièrement connectées. [6]

Un réseau de neurones artificiel est un algorithme utilisé dans une branche de l'apprentissage machine appelée apprentissage profond (*Deep Learning*). Ce type d'algorithme peut être utilisé dans divers applications, par exemple : la classification d'images ou la régression.

Le fonctionnement d'un réseau de neurones est schématiquement basé sur celui des neurones biologiques. Il est constitué de différentes couches interconnectées : une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque couche est constituée à son tour de nœuds appelés neurones, et chaque neurone est associé à des poids w_i et un biais b . Les poids et biais d'une couche donnée sont associés à la sortie de la couche précédente pour produire une nouvelle sortie. Si la valeur de la sortie d'un nœud est supérieur à un certain seuil spécifié, elle est activée puis transmise à la couche suivante.

$$y_j(x) = \sigma(f_j(x)) = \sigma\left(\sum_{i=1}^N (w_{ij} \cdot x_i) + b_j\right). \quad (1)$$

Avec :

- $f_j(x)$: sortie du neurone j .
- $y_j(x)$: sortie du neurone j après activation.
- $\sigma(\dots)$: fonction d'activation (voir Tab. 1).
- x : vecteur des entrées.
- w_{ij}, b_j : poids et biais du neurone j .

Le phénomène de transmission de l'information des couches primaires vers les couches les plus profondes s'appelle la propagation directe ou vers l'avant. Cela permet de produire des sorties dépendantes des entrées fournies au réseau et de la valeurs des poids/biais de chaque couche. Ces sorties sont alors exploitées pour effectuer différentes tâches, par exemple de la classification.

Les paramètres d'un réseau (poids/biais) peuvent être optimisés lors d'un processus itératif appelé apprentissage. Cet apprentissage a pour but de trouver les valeurs optimales des paramètres du réseau pour obtenir les sorties voulues. Cette procédure d'optimisation est basée sur la minimisation d'une fonction, appelée fonction de perte, qui associe les sorties produites par le réseau, ses paramètres et les sorties voulues. Cette recherche de solution optimale utilise le principe de rétropropagation du gradient (eq. 2) pour la mise à jour itérative des paramètres à partir de la couche de sortie vers la couche d'entrée.

$$w_{t+1} = w_t - \alpha \cdot \nabla_t L(w) \quad (2)$$

Avec :

- w_{t+1} : valeur d'un paramètre à l'itération $t + 1$.
- w_t : valeur d'un paramètre à l'itération t .
- α : taux d'apprentissage (< 1)
- ∇ : opérateur de gradient.
- $L(w)$: fonction de perte.

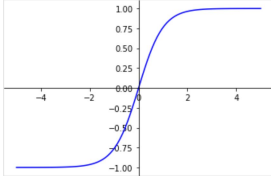
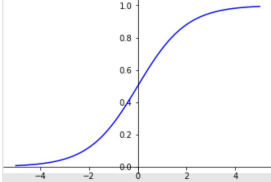
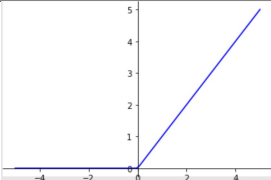
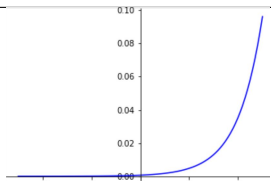
NOM	Graphe	Équation
Tanh		$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3)$
Sigmoïde		$f(x) = (1 + e^{-x})^{-1} \quad (4)$
ReLU		$f(x) = \max(0, x) \quad (5)$
Softmax		$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad for \ i = 1, 2, \dots, K \quad (6)$

TABLE 1 – Les fonctions d'activation récurrentes.

1.3.1 Les réseaux de neurones convolutifs

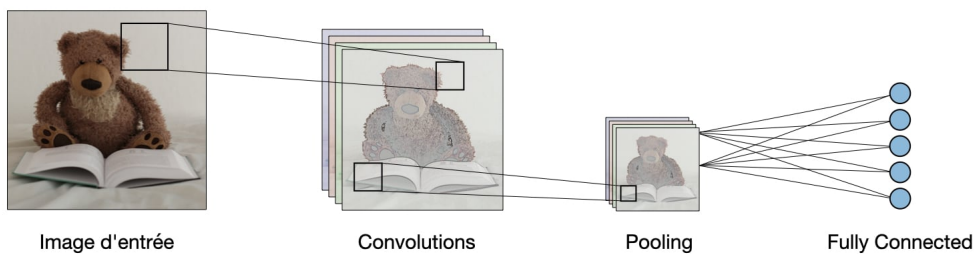


FIGURE 2 – Réseau de neurones convolutifs.[7]

Un réseau de neurones convolutifs ou *Convolutional Neural Network* (CNN) en anglais, est un type de réseau de neurones qui est généralement utilisé pour des problématique de vision assisté par ordinateur. Les réseaux de neurones convolutifs sont généralement constitués de deux parties : une partie extraction des caractéristiques, représentée par des opérations de convolution, et une partie classification représentée par la dernière couche du réseau qui est une couche

entièrement connectée.

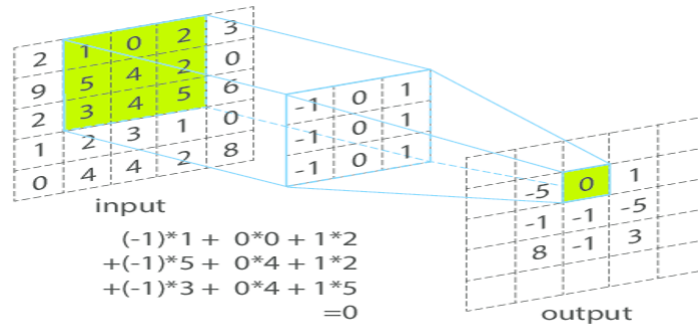


FIGURE 3 – Produit de convolution.[7]

Une opération de convolution implique une entrée à deux dimensions et un noyau (voir fig.3). Le noyau est appliqué plusieurs fois sur différentes parties de l'entrée pour produire une sortie (eq.7).

$$\forall (x, y) \in \mathbb{Z}^2, (I * k)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} k(i, j).I(x - i, y - j) \quad (7)$$

Avec :

- I : entrée à deux dimensions.
- k : noyau de convolution.

En plus des opérations de convolution nous pouvons retrouver dans les réseaux de neurones convolutifs d'autres types d'opération. Parmi ces opérations on peut retrouver les opérations de sous-échantillonnage (*Subsampling*).

Un sous-échantillonnage consiste à résumer l'information extraite d'une image par une réduction de la dimensionnalité des cartes de caractéristiques dans le but de garder que les caractéristiques les plus importantes des images en appliquant une matrice (matrice de pooling) de taille $N \times N$ pixels, qui va parcourir l'image en effectuant un pas de s pixels afin d'obtenir une carte de caractéristiques en sortie. Cette opération est généralement placée entre deux couches de convolution.

Les deux types de sous-échantillonnage les plus utilisés sont :

- Sous-échantillonnage par valeur moyenne (*Average pooling*) : cette opération consiste à faire la valeur moyenne des pixels sélectionnés par la matrice de pooling.
- Sous-échantillonnage par valeur maximum (*Max pooling*) : cette opération consiste à prendre la valeur maximale sélectionnée par la matrice du pooling.

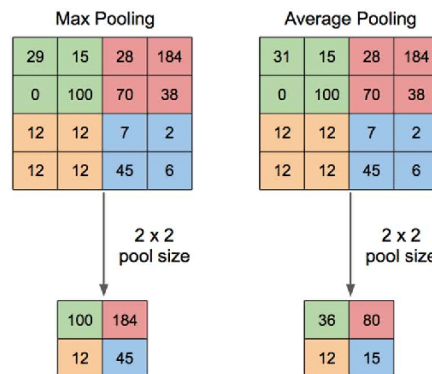


FIGURE 4 – Opération de sous-échantillonnage : par valeur maximale (gauche), par valeur moyenne (droite) [8].

2 Distillation de connaissances

La distillation de connaissance, ou plus communément appelée "*Knowledge Distillation*" (KD) est une technique de réduction de réseaux de neurones qui consiste à transférer les connaissances d'un réseau vers un autre. Cela implique le plus souvent un réseau très profond et complexe appelé "enseignant" (*Teacher*) utilisé pour entraîner un autre réseau moins complexe appelé "élève" (*Student*). Cette approche a été introduite pour la première fois par Geoffrey E. Hinton [9] (2015).

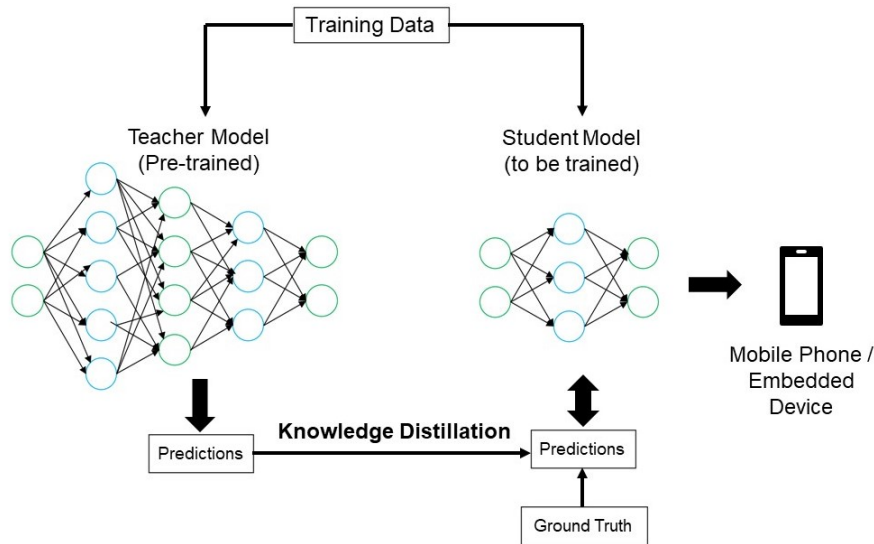


FIGURE 5 – Principe de la distillation de connaissances.

Source : <https://www.analyticsvidhya.com/blog/2022/01/knowledge-distillation-theory-and-end-to-end-case-study/>

2.1 Modélisation et transfert des connaissances

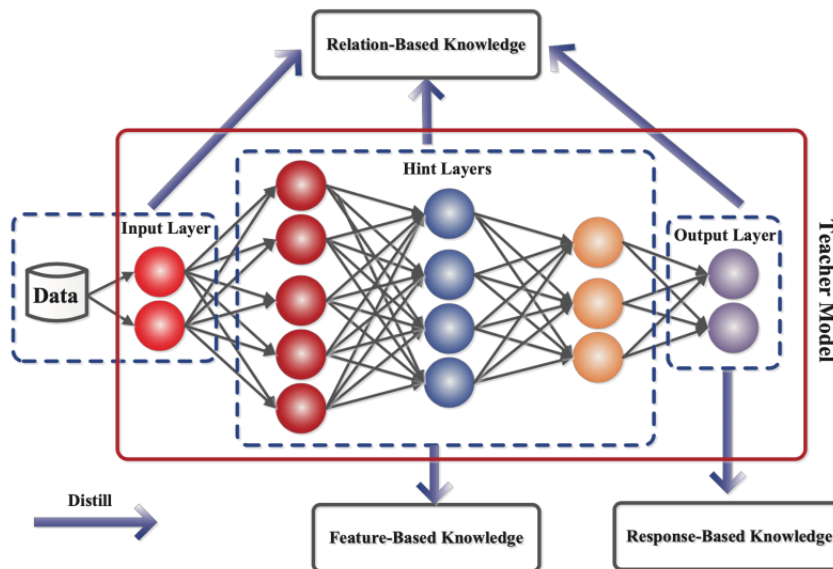


FIGURE 6 – Modélisation de la connaissance dans un réseau de neurones [10].

Dans le cas des réseaux de neurones, la connaissance peut être modélisée de différentes façons. Cette modélisation permet de définir une stratégie de distillation. Le plus souvent, les connais-

sances d'un réseau de neurones sont modélisées comme étant ses prédictions (sorties). Les activations des neurones, les cartes caractéristiques des couches intermédiaires ou la relation entre les paires neurones/activations peuvent être aussi un moyen de représenter les connaissances acquises par le réseau lors de son apprentissage sur une base de données [10] et porteuses d'informations très utiles lors de la distillation.

2.1.1 Connaissance basée sur la réponse

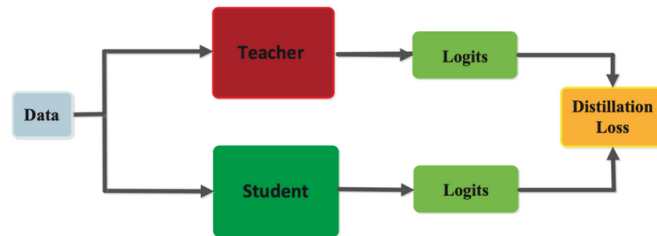


FIGURE 7 – Modélisation de la connaissance basée sur la réponse [10].

Ce type de transfert de connaissances se concentre sur la couche de sortie du réseau enseignant. Le but est de transférer la connaissance en permettant au réseau élève d'imiter les prédictions du réseau enseignant [9]. Cela est réalisé grâce à une fonction de perte (*Loss Function*) introduite, appelée "*Distillation Loss*" qui est le plus souvent la fonction "*Kullback-Leibler divergence loss*" (KLDivergence). Cette fonction permet de modéliser la différence entre les prédictions faites par l'enseignant et celles produites par l'élève. Minimiser cette fonction de perte lors de l'entraînement permet de faire converger les prédictions du réseau élève vers celles de l'enseignant [9] et de ce fait guider l'entraînement du réseau élève pour obtenir de meilleures performances comparé à un apprentissage standard.

2.1.2 Connaissance basée sur les caractéristiques

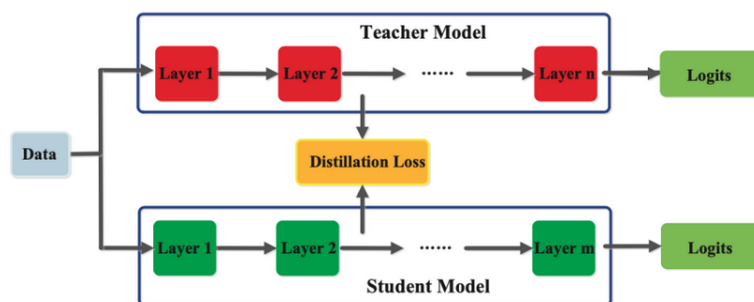


FIGURE 8 – Modélisation de la connaissance basée sur les caractéristiques [10].

Un réseau enseignant pré-entraîné modélise la connaissance propre aux données utilisées pour son entraînement à travers les couches intermédiaires. Les couches intermédiaires apprennent à discriminer des caractéristiques spécifiques et ces connaissances peuvent être utilisées pour entraîner un réseau élève. Ces connaissances peuvent être modélisées à travers des mécanismes d'attention, la distribution des paramètres du réseau ou des statistiques liées aux cartes caractéristiques [10]. L'objectif du transfert de connaissance basé sur les caractéristiques est d'entraîner l'élève à apprendre les mêmes activations que le réseau enseignant. La fonction de perte (*Distillation Loss*) y parvient en minimisant la différence entre les activations des deux réseaux enseignant et élève.

2.1.3 Connaissance basée sur la relation

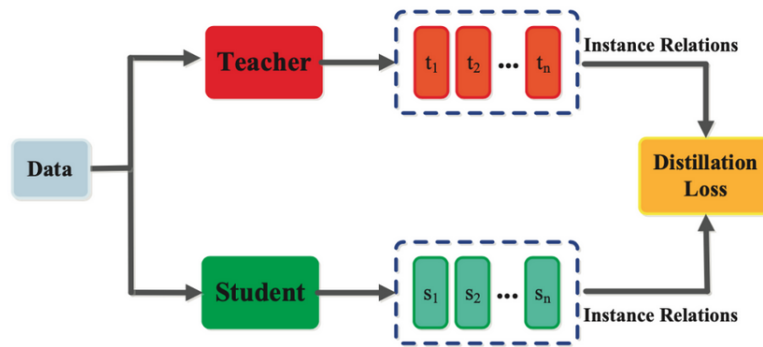


FIGURE 9 – Modélisation de la connaissance basée sur la relation [10].

En plus des connaissances représentées dans les couches de sortie et les couches intermédiaires d'un réseau de neurones, les connaissances qui capturent la relation entre les cartes caractéristiques et couches du réseau peuvent également être utilisées pour entraîner un réseau élève. Cette relation peut être modélisée par une corrélation entre les cartes caractéristiques, une matrice de similarité, ou les distributions probabilistes de la représentation des cartes caractéristiques [10].

2.2 Stratégies de distillation des connaissances

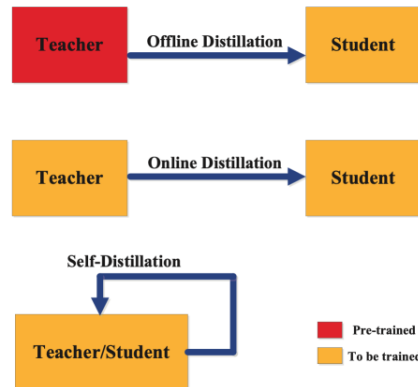


FIGURE 10 – Les différents schémas de distillation de connaissances [10].

2.2.1 Distillation hors ligne

La distillation hors ligne (*Offline Distillation*) est la méthode la plus courante, où un réseau enseignant pré entraîné est utilisé pour guider l'entraînement d'un réseau élève. Dans ce schéma, le réseau enseignant est d'abord préentraîné sur un ensemble de données de formation, puis ses connaissances sont distillées pour entraîner le réseau élève [10].

2.2.2 Distillation en ligne

Dans certains cas d'utilisation, un modèle pré entraîné peut ne pas être disponible pour la distillation hors ligne. Pour remédier à cette limitation, la distillation en ligne (*Online Distillation*) peut être utilisée. Ce schéma de distillation consiste à former les modèles enseignant et élève simultanément dans un seul processus d'entraînement. La distillation en ligne peut être mise en place en utilisant le calcul parallèle, ce qui en fait une méthode très efficace [10].

2.2.3 Auto Distillation

En Auto Distillation (*Self-distillation*), le même modèle est utilisé comme modèle enseignant et élève. Par exemple, les connaissances des couches profondes d'un réseau de neurones peuvent être utilisées pour entraîner ses couches primaires (les premières couches). Une autre approche consiste à utiliser les connaissances des epochs antérieures du modèle enseignant et les transférer à ses epochs ultérieures pour entraîner le modèle élève [10]. Ce schéma de distillation peut être considéré comme un cas particulier de distillation en ligne.

2.3 Architecture Enseignant-Elève

Le choix de l'architecture du réseau enseignant et celle de l'élève est essentiel pour une acquisition et une distillation efficace des connaissances. En règle générale, il existe un écart de capacité entre le modèle enseignant (plus complexe) et le modèle élève (moins profonds). Cet écart structurel peut être réduit en optimisant le transfert de connaissances via des architectures élèves-enseignants efficaces [10].

Le transfert de connaissances à partir de réseaux de neurones profonds n'est pas simple en raison de leur profondeur et de leur étendue. Les architectures les plus courantes pour le transfert de connaissances incluent un réseau élève qui est :

- Une version moins profonde du modèle enseignant avec moins de couches et moins de neurones par couche.
- Une version quantifiée du modèle de l'enseignant.
- Un réseau plus petit avec des opérations de base efficaces.
- Un réseau plus petit avec une architecture globale optimisée.
- Le même réseau que l'enseignant (Auto Distillation).

2.4 Partie expérimentale

Pour pouvoir analyser les avantages et inconvénients de l'utilisation d'algorithmes de distillation de connaissances nous avons décidé d'implémenter un algorithme de distillation hors ligne pour la simplicité de mise en place et d'utilisation qu'offre ce dernier. Nos travaux sont basés sur ceux de G.Hinton et al. [9].

2.4.1 Méthodologie

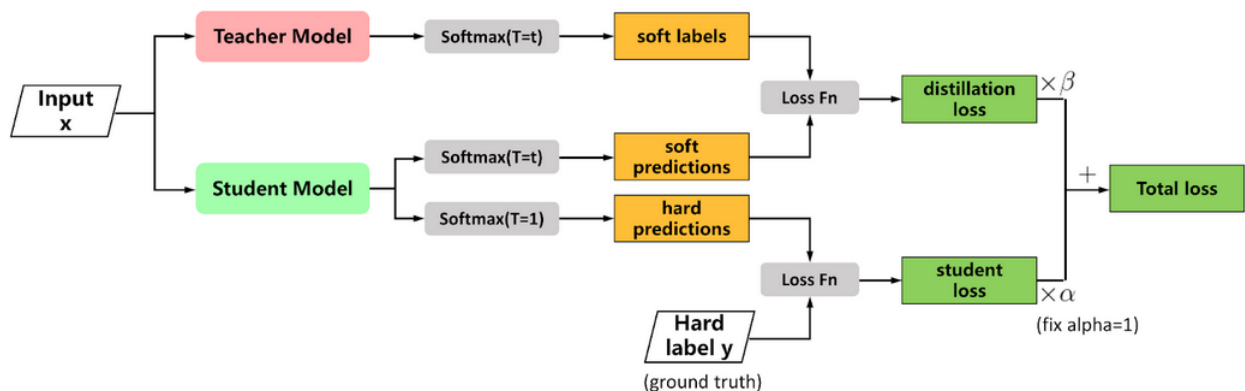


FIGURE 11 – Schéma blocs d'une distillation de connaissances hors ligne.

Le schéma de distillation de connaissances utilisé est illustré par la figure 11. Dans ce schéma deux réseaux de neurones sont utilisés : un réseau enseignant qui est un réseau pré entraîné, et un autre réseau élève, non entraîné, relativement plus petit que le premier. Le choix des architectures de ces deux réseaux est important et doit respecter certaines conditions (voir section 2.3) pour

assurer le bon fonctionnement du processus de distillation. Dans notre cas, nous avons utilisé deux réseaux à connexions résiduelles [11] : un Resnet26 comme enseignant et un Resnet8 comme élève.

Le réseau élève est entraîné en exploitant ses prédictions à travers la fonction *Softmax* pour calculer les probabilités d'appartenance des données aux différentes classes (*Soft predictions*). Ces probabilités et les étiquettes des données (*Ground Truth*) sont associées pour calculer une fonction de perte (*Student loss*). La minimisation de cette fonction assure la convergence des prédictions du réseau élève vers les bonnes étiquettes correspondantes aux données. L'équation 8 illustre la fonction de perte utilisée correspondant à l'entropie croisée.

$$L_S(p_s, gt) = -\frac{1}{N} \cdot \sum_{gt} gt \cdot \log p_s \quad (8)$$

- L_S : fonction de perte élève.
- p_s : *soft predictions* du réseaux élève.
- gt : étiquettes correspondantes des données prédites (*Ground Truth*).

D'un autre côté, les mêmes probabilités sont associées aux *Soft Labels* (Eq.10) issus des prédictions du réseau enseignant pour calculer une deuxième fonction de perte propre au processus de distillation appelée perte de distillation (*Distillation Loss*). Minimiser cette fonction permet de faire converger les prédictions du réseau élève vers celles du réseau enseignant, ce qui permet d'orienter l'entraînement du réseau élève vers de meilleures performances. L'équation 9 illustre la fonction de perte de distillation utilisée correspondant à l'entropie relative (KLD : Divergence de Kullback-Leibler).

$$L_D(p_t, p_s) = \sum p_t \cdot \log \frac{p_t}{p_s} \quad (9)$$

Avec :

$$p_t = \text{Softmax}\left(\frac{x}{T}\right) = \frac{e^{(-\frac{x}{T})}}{\sum_N e^{(-\frac{x}{T})}} \quad (10)$$

- L_D : fonction de perte de distillation.
- p_s : *soft predictions* du réseaux élève.
- p_t : *soft Labels* du réseau enseignant.
- x : sortie du réseau enseignant.
- T : paramètre de la distillation appelé Température.

Les deux fonctions de perte (Eq.8, Eq.9) sont alors associées pour former la fonction de perte totale décrite par l'équation suivante :

$$Loss = \alpha \cdot L_S + \beta \cdot L_D \quad (11)$$

Avec :

$$\alpha = 1 - \beta \quad (12)$$

- α, β : coefficients de pondération permettant de contrôler la contribution de chacune des deux fonctions de perte précédemment définies (Eq.8, Eq.9) dans le calcul de la valeur de la fonction de perte totale.

A partir des équations 8, 9, 11 il apparaît qu'en plus des hyperparamètres liés à l'apprentissage du réseau élève, le processus de distillation de connaissances dépend de trois paramètres clés : la température T et les deux coefficients de pondération α et β . L'effet de ces paramètres n'étant pas clairement défini dans l'état de l'art [13, 12, 10, 9, 14, 4, 15], nous avons décidé de faire une étude paramétrique pour déterminer l'effet de ces différents paramètres sur la distillation de connaissances et l'apprentissage du réseau élève.

L'étude paramétrique consiste à faire varier individuellement l'un des paramètres α, β, T pour observer l'effet de ce dernier sur la distillation de connaissances à travers l'analyse des courbes d'apprentissage et l'évolution de la fonction de perte du réseau élève (Resnet8).

L'étude paramétrique a été faite avec les mêmes conditions d'entraînement : le réseau élève (Resnet8) a été entraîné sur 100 **epochs** avec une taille de **batch** de 32 sur **Tensorflow** en utilisant l'optimiseur SGD et un pas d'apprentissage variant de $1e - 1$ à $1e - 6$ grâce à une fonction **Callback** qui permet d'éviter le phénomène de dispersion de gradient.

La base de données utilisée est Cifar10 [16]. Elle est constituée de 60000 images en couleurs de taille 32×32 représentant 10 classes différentes, chacune contenant 6000 images. Cette base a été découpée en une base d'apprentissage de 50000 images et une base de test contenant 10000 images. Une étape de pré traitement des images a été ajoutée : il s'agit d'un rognage aléatoire et d'une rotation horizontale aléatoire. Le pré traitement effectué à pour but d'améliorer la précision en test du réseau élève et réduire l'écart de performances avec le réseau enseignant. La figure 12 illustre les traitements de données utilisés.

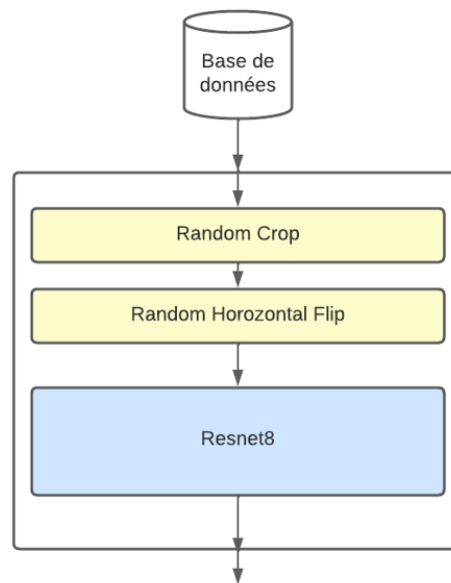


FIGURE 12 – Schéma représentant l'intégration du processus de pré traitement des données au réseau Resnet8.

2.4.2 Résultats et discussion

2.4.2.1 Paramètres α et β

Les paramètres α et β étant liés par la relation illustrée par l'équation 12 une seule étude a été faite. Nous avons fait varier le paramètre α entre 0.1 et 0.9 avec un pas de 0.1 pour une valeur de $T = 5$, et nous avons observé les effets de cette variation sur l'évolution de la fonction de perte totale et la précision du réseau élève tout en comparant cela à une courbe de référence (*Baseline*) correspondant à l'entraînement du même réseau sans mécanisme de distillation de connaissances.

En analysant les courbes de précision illustrées par la figure 13 on remarque que la meilleure précision obtenue correspond aux valeurs $\alpha = 0.1, \beta = 0.9$ ce qui signifie une contribution de 10% de la fonction de perte élève et 90% de la fonction de perte de distillation dans la valeur de fonction de perte totale. Cette configuration des paramètres permet d'améliorer la précision obtenue lors d'un apprentissage standard (*Baseline*) de +2%. On peut aussi constater que d'autres configurations permettent d'obtenir de meilleures précisions que la référence (ex. $\alpha = 0.3, 0.6, 0.8...$). D'un autre côté, on peut constater que la distillation de connaissances permet de converger plus rapidement. Cela peut être observé en prenant l'epoch 15 comme point de référence : on remarque très nettement que la courbe de référence progresse plus lentement que celles correspondant à la distillation de connaissances.

En observant les courbes illustrées par la figure 14, on remarque que toutes les courbes d'apprentissage utilisant la distillation de connaissances arrivent à mieux optimiser la fonction de perte en obtenant un minimum plus faible comparées à la courbe de référence (*Baseline*). On remarque aussi une relation linéaire entre la valeur de α et la valeur de la fonction de perte totale tel que : la valeur de la fonction de perte est inversement proportionnelle au paramètre α . Cela reste vrai sauf pour les deux valeurs $\alpha = 0.8, 0.9$ où la tendance s'inverse autour de l'époch 70.

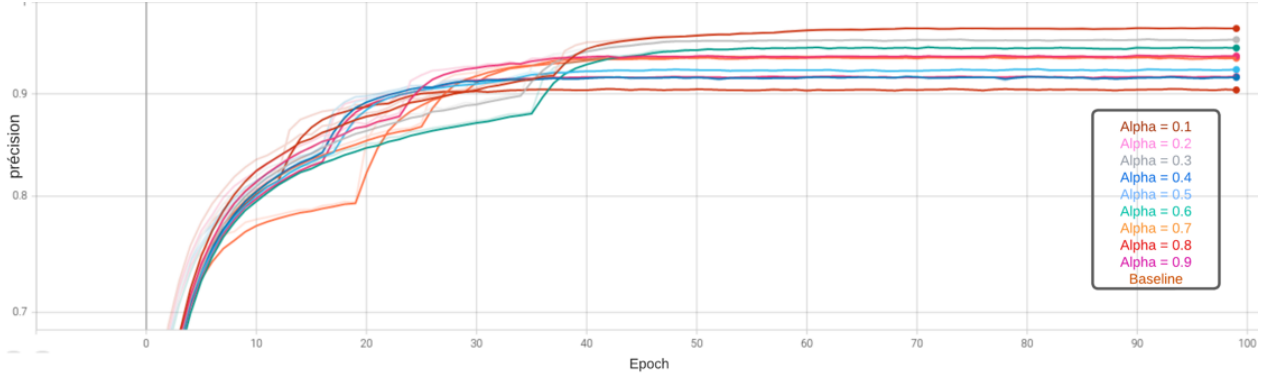


FIGURE 13 – Évolution des courbes de précision du réseau élève en fonction du paramètre α .

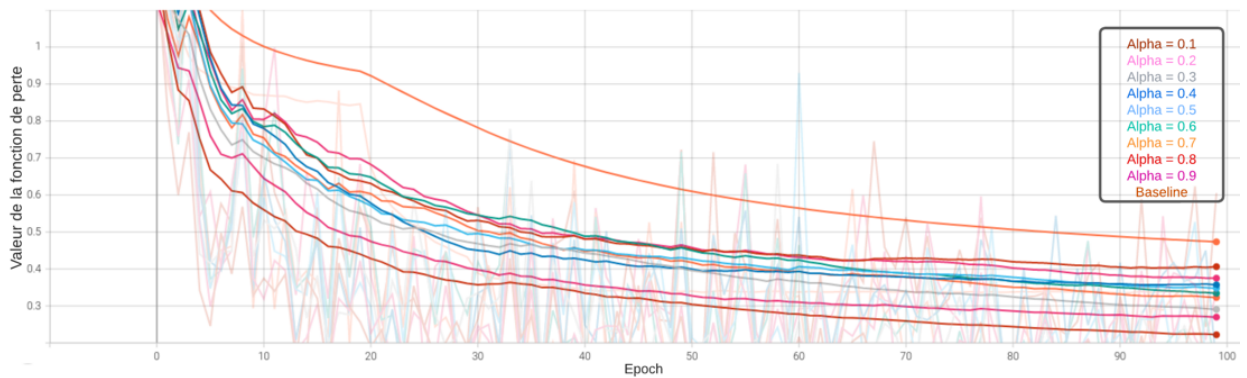


FIGURE 14 – Évolution des courbes de perte du réseau élève en fonction du paramètre α .

2.4.2.2 Paramètre de température T

Pour étudier l'effet du paramètre de température T . Nous avons fait varier ce dernier entre 5 et 40 avec un pas de 5 tout en fixant la valeur du paramètre α à 0.2.

A travers l'analyse de l'évolution des courbes de précision illustrées par la figure 15, on remarque que les performances de la distillation de connaissances démontre une certaine sensibilité aux valeurs intermédiaires du paramètre T se situant entre 15 et 30. Ces valeurs arrivent à donner de meilleures précisions comparées à la courbe de référence (*Baseline*). La meilleure précision obtenue étant 0.958 qui correspond à une valeur de $T = 20$.

Concernant l'effet de la variation du paramètre T sur l'évolution de la fonction de perte, on remarque en observant les différentes courbes illustrées par la figure 16 que le paramètre T introduit un effet d'amortissement qui permet de réduire les oscillations des valeurs de la fonction de perte lors de l'apprentissage. Cet effet d'amortissement est proportionnel à la valeur du paramètre de température T .

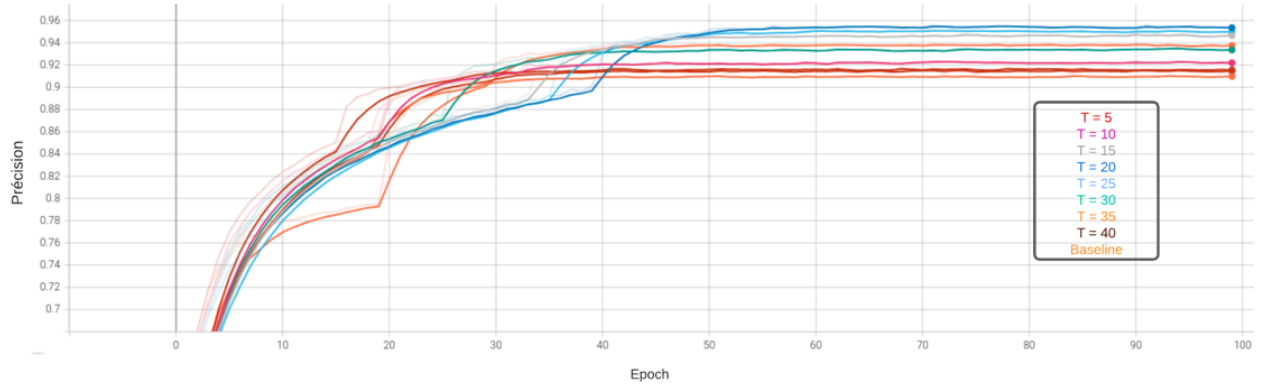


FIGURE 15 – Évolution des courbes de précision du réseau élève en fonction du paramètre T .

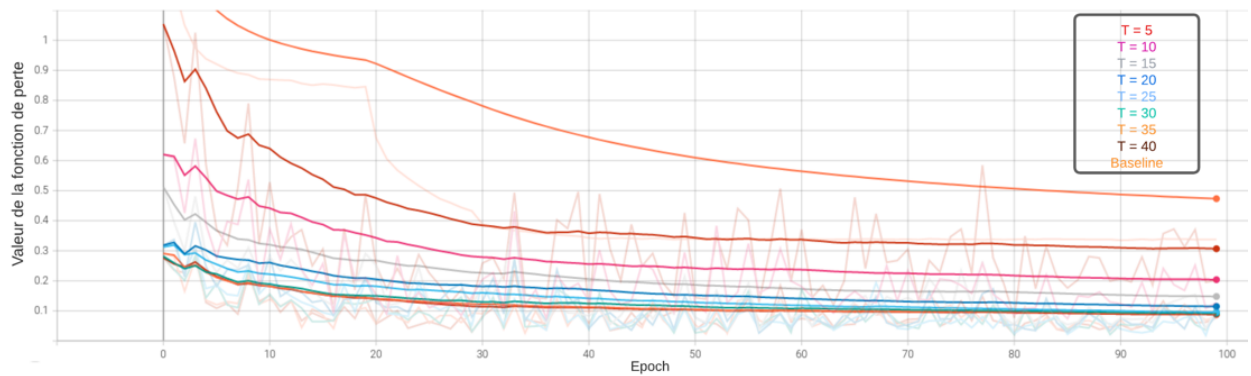


FIGURE 16 – Évolution des courbes de perte du réseau élève en fonction du paramètre T .

2.4.3 Conclusion

Un apprentissage utilisant de la distillation de connaissances permet d'obtenir une meilleure précision en classification comparé à un entraînement standard. Cependant, cela dépend fortement du choix des paramètres utilisés. D'après les résultats obtenus lors de nos expérimentations, nous avons observé qu'il était très difficile de déterminer les paramètres optimaux d'une distillation de connaissances car les performances du réseau élève pouvait être plus ou moins sensibles à certaines valeurs de paramètres et non à d'autres. De plus, la distillation de connaissances ne se concentre que sur l'optimisation des performances d'un réseau lors de son apprentissage et néglige complètement l'aspect matériel et l'optimisation du fonctionnement du réseau lors de son intégration dans une cible embarquée. Pour y remédier, il faudra combiner la distillation de connaissances à une autre technique de réduction telle que la quantification ou l'élagage. A. Polino et al. [17] ont prouvé que cela pouvait être très efficace en combinant la distillation de connaissances à la quantification. Ils ont démontré que la distillation de connaissances permettait au réseau d'atténuer la dégradation de la précision causée par la quantification, surtout lors d'une quantification extrême (< 8 bit).

3 Quantification des réseaux de neurones

La quantification est une méthode de réduction de réseaux de neurones qui consiste en la réduction de la précision des poids et/ou des activations d'un réseau avec une dégradation des performances négligeable. Cette méthode de compression a démontré un succès notable dans l'entraînement et l'inférence des réseaux de neurones [18].

L'objectif de la quantification est de réduire l'espace mémoire occupé par les paramètres d'un réseau de neurones et accélérer les opérations arithmétiques, ce qui peut permettre d'intégrer ce réseau à des applications ou des cibles embarquées où la mémoire et les ressources de calculs sont limitées.

Il existe deux types de quantification : uniforme et non uniformes. La différence entre ces deux types de quantification réside dans la manière de représenter les valeurs quantifiées (Niveaux de quantification). Lors d'une quantification uniforme les différentes valeurs possibles sont équidistantes les unes des autres, et elles ne le sont pas lors d'une quantification non uniforme. De ce fait, la méthode non uniforme permet de se concentrer sur des sous ensembles de valeurs plus sensibles que d'autres lors de la quantification, où l'approche uniforme offre un schéma plus général indépendant des valeurs à quantifier.

La quantification non uniforme a démontré dans certains cas de meilleurs résultats que la quantification uniforme, mais reste moins utilisée car plus difficile à implémenter. De plus, La quantification uniforme donne des résultats satisfaisants dans la plupart des cas et reste l'approche la plus utilisée dans le domaine de la quantification des réseaux de neurones car plus facile à implémenter [18]

Dans ce qui suivra nous ne parlerons que de quantification uniforme.

3.1 Processus de quantification

La quantification uniforme est définie par la formule suivante :

$$Q(r) = \text{Int}\left(\frac{r}{S}\right) - Z \quad (13)$$

- $Q(r)$: valeur quantifiée.
- r : réel à quantifier.
- $\text{Int}(\dots)$: fonction d'arrondi.
- S : facteur d'échelle.
- Z : décalage (*offset*).

Il est possible d'effectuer l'opération inverse qui consiste à récupérer les valeurs réelles à partir des valeurs quantifiées à travers une opération appelée déquantification définie par la formule suivante :

$$\tilde{r} = S.(Q(r) + Z) \quad (14)$$

Remarque : \tilde{r} est une valeur approchée et non exacte du réel r et cela à cause de la fonction d'arrondi $\text{Int}(\dots)$.

Avant de pouvoir utiliser la formule de quantification, il faudra passer obligatoirement par les étapes suivantes : définir l'intervalle de représentation $[\alpha, \beta]$, puis calculer le facteur d'échelle S . Le processus consistant à choisir l'espace de représentation des valeurs à quantifier est souvent appelé le processus de **calibration** [18]. Première étape de la quantification, le choix de l'intervalle de représentation servira par la suite à déterminer le facteur d'échelle.

Il existe différentes approches pour choisir l'intervalle $[\alpha, \beta]$, la plus communément utilisée étant le min/max de l'ensemble contenant les valeurs à quantifier ex. $\alpha = r_{\min}, \beta = r_{\max}$. Cette approche est une quantification asymétrique ($-\alpha \neq \beta$). Pour appliquer un schéma de quantification symétrique il suffit de choisir un intervalle de représentation symétrique ($\alpha = \beta$). Pour cela , il existe différentes approches ex $-\alpha = \beta = \max(|r_{\max}|, |r_{\min}|)$.

Une quantification asymétrique peut résulter souvent en des espaces de représentation de valeurs plus restreints. Cela peut être très utile lorsque les valeurs à quantifier sont déséquilibrées, par exemple la sortie de la fonction d'activation Relu qui ne présente pas de valeurs négatives.

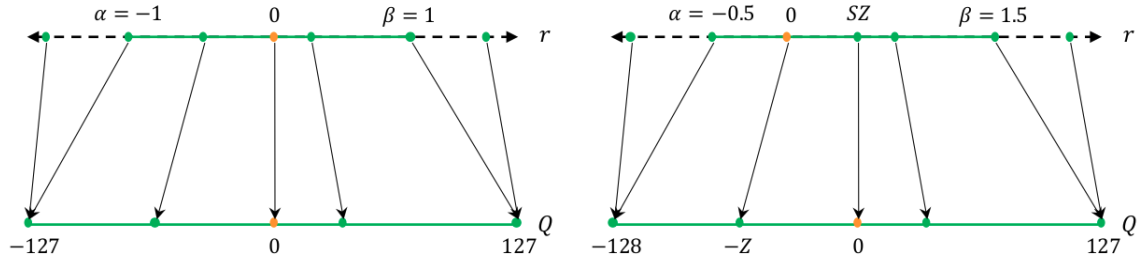


FIGURE 17 – Intervalle de représentation : Symétrique (gauche), Asymétrique(Droite) [18].

Remarque : L'intervalle de représentation $[\alpha, \beta]$ peut être déterminé de manière statique, où les valeurs α, β sont des constantes définies au début du processus de quantification, ou de manière dynamique où ces valeurs sont mises à jour en temps réel lors d'un processus de quantification itératif.

Le facteur d'échelle S est défini par la formule suivante :

$$S = \frac{\beta - \alpha}{2^b - 1} \quad (15)$$

b étant la précision (16, 8, 4, 2 bits).

Après avoir défini la précision, l'espace de représentation $[\alpha, \beta]$ et calculer le facteur d'échelle S , il ne reste plus qu'à appliquer la formule décrite par l'équation 13.

3.2 Quantification et entraînement des réseaux de neurones

Il est très commun d'ajuster les paramètres du réseau de neurones après une opération de quantification. Cet ajustement peut s'effectuer en réentraînant le modèle. Cette méthode est appelée **Quantization Aware Training (QAT)**. Une autre manière de procéder existe qui ne consiste pas à réentraîner le modèle. Cette approche est appelée Post Quantification (**Post-Quantization (PQT)**) [18]. Un schéma comparatif des deux approches est illustré par la figure 18.

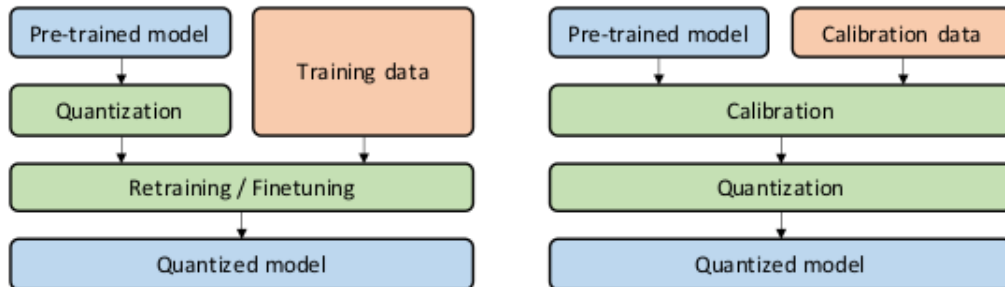


FIGURE 18 – Comparatif des approches d'ajustement des paramètres d'un modèle lors de la quantification : QAT (gauche), PQT (Droite) [18].

3.2.1 Quantification pendant l'entraînement

La quantification d'un modèle pré entraîné peut introduire des perturbations qui peuvent mener à la dégradation des performances de ce dernier. Cela est causé par l'éloignement du modèle du

point de convergence optimale atteint lors de l'entraînement avec des paramètres en flottants 32 bit (précision standard). Il est possible de résoudre ce problème en réentraînant le modèle avec les paramètres quantifiés dans le but de se rapprocher du point de convergence initial (avant quantification). Pour ce faire, l'approche la plus populaire est de quantifier le réseau de neurones pendant l'entraînement (*Quantization Aware Training*). Cette méthode consiste à effectuer la propagation direct et la rétropropagation sur le modèle quantifié en point flottant et de quantifier les paramètres de ce dernier après le calcul du gradient. À noter qu'il est important d'effectuer la rétropropagation en point flottant, car le faire avec la précision de quantification peut mener à des valeurs de gradient nulles, ou des valeurs avec un degré d'erreur très élevé surtout pour les précisions basses. Ce phénomène est causé par l'opération d'arrondi effectuée lors de la quantification, qui est une fonction non différentiable [18].

Cette approche a démontré son efficacité, mais a l'inconvénient d'être très coûteuse en ressources de calcul du fait de réentraîner le réseau de neurones, ce qui peut prendre une centaines d'itérations pour pouvoir restaurer ses performances initiales.

3.2.2 Quantification post entraînement

Comme alternative moins coûteuse qu'une (*Quantization Aware Training*) la post quantification consiste à effectuer une quantification directement sur le modèle pré entraîné. En plus d'être indépendante des données d'entraînement, cette approche est plus rapide et plus facile à appliquer, mais ces avantages sont accompagnés d'une dégradation des performances plus prononcée comparée à la première approche [18].

3.3 Quantification et inférence

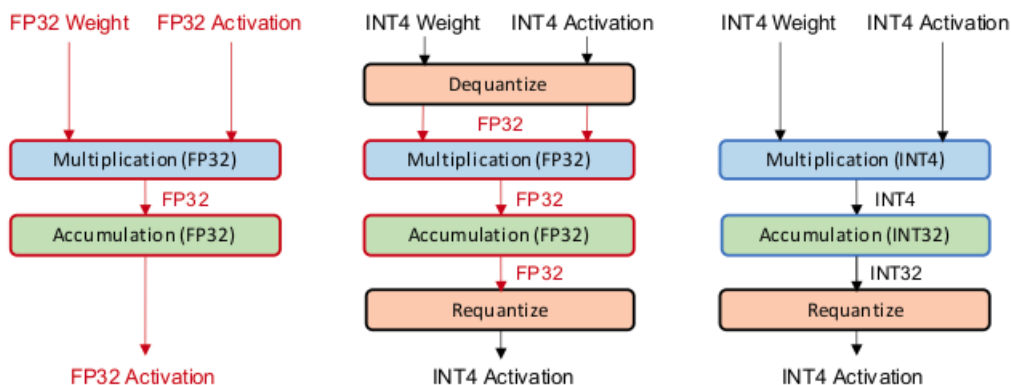


FIGURE 19 – Les différents schémas d'inférence : Standard (gauche), Quantification simulée (centre), Quantification en entiers uniquement (droite) [18].

Il existe deux schémas d'inférence lors du déploiement d'un réseau de neurones quantifié : la quantification simulée (*Simulated Quantization*), et la quantification entière (*Integer-Only Quantization*) (voir Fig.19).

Lors de la quantification simulée les paramètres du modèle quantifié sont stockés en basse précision mais les opérations mathématiques (e.x. multiplication matricielle, convolution ...) sont effectuées en précision flottante (32 bits). De ce fait, les paramètres ont besoin d'être déquantifiés avant d'effectuer les opérations arithmétiques. Cependant, dans un schéma d'inférence en entiers uniquement, toutes les opérations sont effectuées en basse précision, ce qui permet une logique arithmétique complètement en entiers. De ce fait aucune étape de déquantification des paramètres n'est nécessaire [18].

En général, effectuer l'inférence sur une cible embarquée avec une quantification simulée permet de conserver les performances du modèle entraîné, mais cela au prix de ne pas bénéficier des avantages qu'offre une inférence en basse précision en termes de latence et de consommation

énergétique comme illustré sur la figure 20. Comme on peut le voir, une précision inférieure offre une meilleure efficacité énergétique et des débits plus hauts. Mais cela ne veut pas dire qu'un schéma de quantification simulée n'est pas utile. Elle peut être bénéfique pour des limitations liés à la bande passante plutôt que liés au calcul [18].

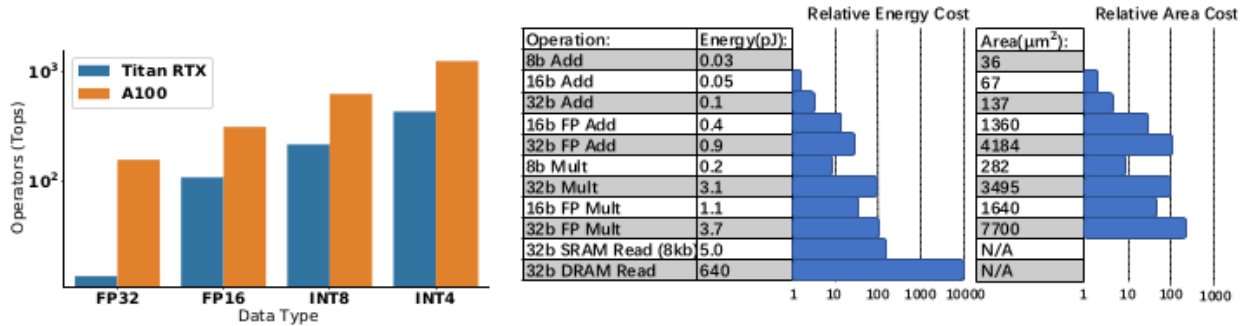


FIGURE 20 – (Gauche) Comparaison entre le débit maximum pour différentes précisions en bit sur GPU Titan RTX et A100. (Droite) Comparaison du coût énergétique correspondant et du coût relatif de la surface pour différentes précisions pour des technologie 45 nm [18].

3.4 Partie expérimentale

La plupart des microcontrôleurs basiques effectuent les opérations mathématiques avec des unités de calcul en entiers seulement. N'ayant pas d'unité de calcul dédiée aux opérations flottantes, ces microcontrôleurs doivent passer par des étapes intermédiaires d'adaptation pour effectuer des calculs sur des valeurs flottantes, ce qui peut mener à une perte de performances notable. Dans ce contexte, nous avons choisi pour l'étude de l'estimation de l'impact de la quantification dans l'intégration d'un réseau de neurones sur une cible embarquée d'implémenter un algorithme qui prend en compte la contrainte citée ci-dessus.

3.4.1 Méthodologie

Sur **Tensorflow** il existe une bibliothèque appelée **Tensorflow Lite** qui permet d'obtenir à partir d'un modèle pré entraîné un modèle quantifié prêt à l'emploi. Nos travaux étant des travaux de recherches, donc par soucis d'explicabilité l'utilisation de cette méthode directement n'est pas permise car les différentes étapes de la quantification ne sont pas explicitement décrites lors de l'utilisation de cet outil. De ce fait, nous avons décider d'implémenter l'algorithme de quantification sur lequel est basé l'outil **Tensorflow Lite** décrit dans l'article [5]. L'algorithme développé se compose de deux partie, une partie apprentissage et une partie inférence.

L'approche la plus commune pour quantifier un réseau de neurones, est d'entraîner ce dernier avec des paramètres flottants puis procéder à une quantification au terme de l'apprentissage. Cette méthode a démontré son efficacité pour des modèles très profonds avec des capacités de représentation considérables, mais peut mener à une dégradation considérable de la précision pour des réseaux relativement plus petits [5]. Donc, pour de plus petites architectures on privilégiera l'utilisation de la quantification simulée qui consiste à simuler l'effet de la quantification pendant l'entraînement du réseau pour lui permettre d'optimiser ses poids en prenant en considération l'opération de quantification, ce qui permettra d'éviter la chute de la précision après la quantification du modèle entraîné.

La figure 21 représente l'intégration du processus de quantification simulée dans une couche de convolution lors d'un apprentissage comme décrit dans [5]. La propagation directe simule une

inférence quantifiée tel que :

- Les poids sont quantifiés avant l'opération de convolution. S'il y a une opération de *batch normalization*, les paramètres de cette dernière devront être fusionnés aux paramètres de la couche qui la précède comme décrit dans [5].
- Les activations sont quantifiées après l'application de la fonction d'activation en sortie de la couche de convolution, de la couche entièrement connectée ou de la concaténation des sorties de deux couches comme dans les Resnet [11].

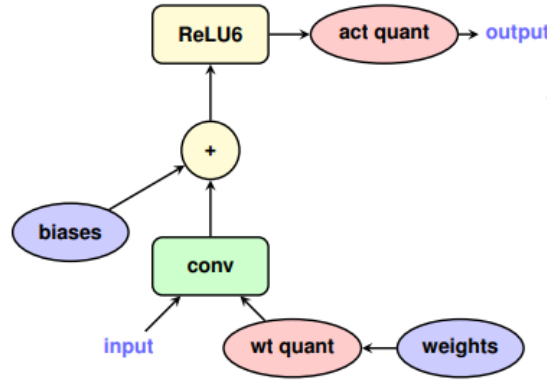


FIGURE 21 – Schéma d'apprentissage par quantification simulée [5].

La quantification est effectuée couche par couche et dépend de deux paramètres : le nombre de niveaux de quantification, et l'intervalle de représentation des valeurs à quantifier. Les filtres et poids des différentes couches sont quantifiés point par point avec la fonction q définie ci-dessous.

$$\text{clamp}(r, a, b) = \min(\max(r, a), b) \quad (16)$$

$$s(a, b, n) = \frac{b - a}{n - 1} \quad (17)$$

$$q(r, a, b, n) = \lfloor \frac{\text{clamp}(r, a, b) - a}{s(a, b, n)} \rfloor \cdot s(a, b, n) + a \quad (18)$$

Où :

- r : nombre réel à quantifier.
- $[a; b]$: l'intervalle de représentation des valeurs réelles.
- n : nombre de niveaux de quantification qui est fixé pour toutes les couches e.x $n = 2^8 = 256$ pour une quantification 8 bit.
- $\lfloor \dots \rfloor$ fonction d'arrondi au plus proche.

Les intervalles de représentation sont traités de deux manières différentes lors de la quantification des poids et des activations.

- Pour les **poids** : l'intervalle de représentation est défini simplement pour $a = \min(w)$; $b = \max(w)$. Une légère modification est apportée pour déplacer les valeurs une fois quantifiées de l'intervalle $[-128, 127]$ vers l'intervalle en $[-127, 127]$ dans le but d'obtenir un intervalle symétrique pour des questions d'optimisation [5].
- Pour les **activations** : L'intervalle de représentation des activations dépend des entrées du réseau. De ce fait, pour mieux estimer l'intervalle $[a, b]$, les valeurs a et b sont stockées pendant l'apprentissage du réseau, puis sont exploitées à travers la fonction des moyennes exponentielles mobiles (EMA : Exponential Moving Averages) qui servira par la suite à mettre à jours l'intervalle $[a, b]$. Il a été prouvé dans les travaux de J.Benoit et al. [5] que

le fait d'accorder un certain délais à l'EMA permettait de mieux estimer l'intervalle de représentation. De ce fait il est préférable de désactiver la quantification des activations pendant un certain nombre d'epochs pour que les variations rapides de ces dernières se stabilisent, et pour exclure les valeurs de $[a, b]$ non significatives.

Algorithm 1 Algorithme d'apprentissage avec quantification simulée

```

Initialiser le réseau.
Insérer la quantification simulée dans les couches et activations qu'on désire quantifier en
utilisant l'équation 18 sans toucher aux biais.
for  $epoch < epoch_{max}; epoch++$  do
    Entraîner le réseau et mettre à j les paramètres de quantification.
    if  $epochs \geq N$  then
        Quantifier les activations.
    end if
end for
Effectuer une post quantification pour obtenir les poids finaux en entiers.
  
```

Afin d'estimer l'impact de la méthode de quantification utilisée sur microcontrôleur, nous avons effectué des mesures de temps d'inférence d'un réseau Lenet5 [19] sur une carte de type STM32F446Z. Nous justifions le choix de ce réseau par l'architecture assez simple à implémenter en langage C qui peut facilement s'adapter aux contraintes mémoire de la seule carte à notre disposition. Deux réseaux ont été entraînés avec la même configuration d'apprentissage : le premier est un apprentissage standard et le deuxième est basé sur la quantification simulée. Après avoir effectué l'apprentissage les poids des deux réseaux sont extraits, puis sont utilisés sur l'environnement de développement Keil où le réseau Lenet5 a été implémenté préalablement en langage C. Le réseau est ensuite intégré sur la carte pour effectuer des mesures de temps d'inférence.

Remarque : Lors de l'implémentation du réseau quantifié en Langage C il faudra respecter le schéma d'inférence illustré par la figure 22 dont l'implémentation est détaillée dans [5] pour assurer le bon fonctionnement de l'inférence et se placer dans un cadre de test réel.

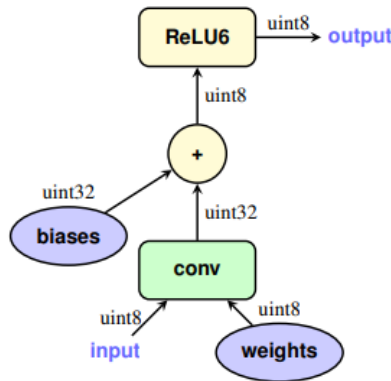


FIGURE 22 – Schéma d'inférence utilisant l'arithmétique entière uniquement [5].

3.4.2 Résultats et discussion

Nous avons entraîné un réseau Lenet5 [19] en intégrant le mécanisme de quantification simulée défini précédemment (Algorithme 1) à toutes les couches du réseau. Les paramètres d'apprentissage utilisés sont les suivants : entraînement sur 100 epochs avec un pas d'apprentissage de 0.1 avec l'optimiseur SGD, la fonction d'entropie croisée comme fonction de perte et la fonction ReLU6 comme fonction d'activation des différentes couches. La quantification des activations commence à partir de l'epoch 30.

Le réseau a été entraîné sur la base de données Mnist [20] qui contient 70000 images représentant les chiffres de 0 à 9. Ces images ont été divisées en une base d'apprentissage de 60000 images et une base de test de 10000 images.

Les résultats de l'apprentissage sont illustrés par la figure 23

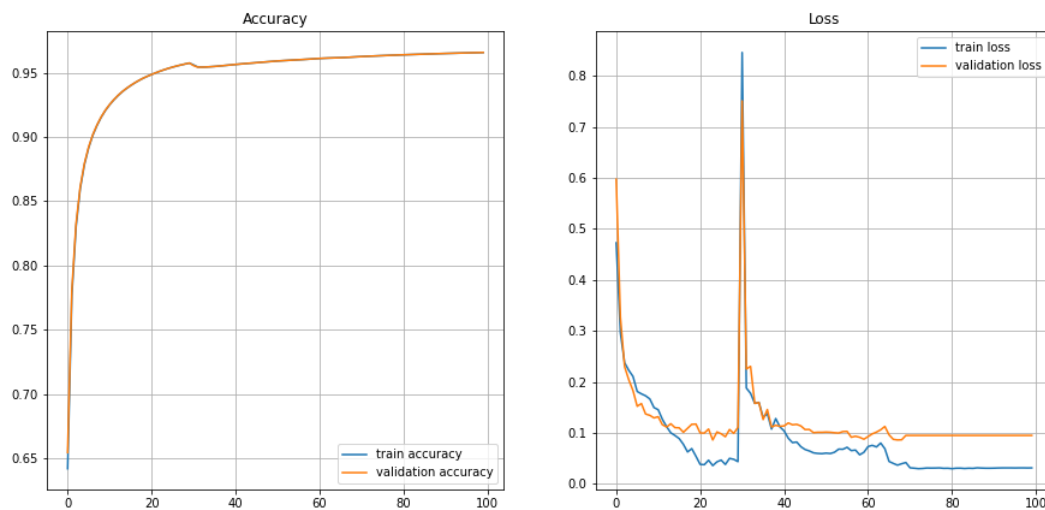


FIGURE 23 – Courbe d'entraînement d'une réseau Lenet5 avec une schéma de quantification simulée.

En analysant les courbes d'apprentissage nous avons observé ce qui suit : concernant la précision, le réseau arrive à converger vers une précision assez similaire à celle d'un entraînement standard. Après l'apprentissage et l'opération de post quantification, le réseau réussi à garder la même précision avec des poids quantifiés en INT8. Nous avons aussi observé une meilleure capacité de généralisation du réseaux quantifié comparé à celui qui ne l'était pas, et cela en ayant une meilleure précision sur la base de test.

Concernant l'évolution de la fonction de perte, on remarque que la courbe converge très bien jusqu'à l'epoch 30 à partir de laquelle on constate une divergence de la courbe, phénomène correspondant au début de la quantification des activations. La courbe ensuite arrive à se stabiliser et à compenser la divergence après quelques epochs.

Architecture	Standard FP32	Quantifiée en INT8
Temps d'inférence (ms)	58.97	46.28
Débit (cl/s)	16.96	21.61
Mémoire occupée (kbits)	247.32	62.91

TABLE 2 – Estimation de l'impact de la quantification du réseau Lenet5 sur STM32F446ZE à une fréquence de 180MHz.

Les résultats de l'estimation de l'impact de la quantification sur microcontrôleur sont résumés dans le tableau 2. Le schéma de quantification utilisé a permis d'obtenir un gain en temps d'inférence de 21.5%, ce qui impacte directement le débit de classification. En plus de cela nous avons un taux de compression mémoire de 75%.

3.4.3 Conclusion

D'après les résultats obtenus on peut en conclure que la réduction de la précision des paramètres d'un réseau de neurones peut s'avérer très efficace dans l'optimisation de l'espace mémoire occupé

par ce dernier, ce qui peut mener à une accélération matérielle non négligeable. L'algorithme que nous avons implémenté a démontré son efficacité, mais reste néanmoins très difficile à mettre en place sur carte car le schéma d'inférence utilisé nécessite des calculs intermédiaires pour gérer le débordement des valeurs résultantes des opérations arithmétiques telles que la multiplication et l'addition lors de l'inférence du réseau quantifié.

4 Élagage

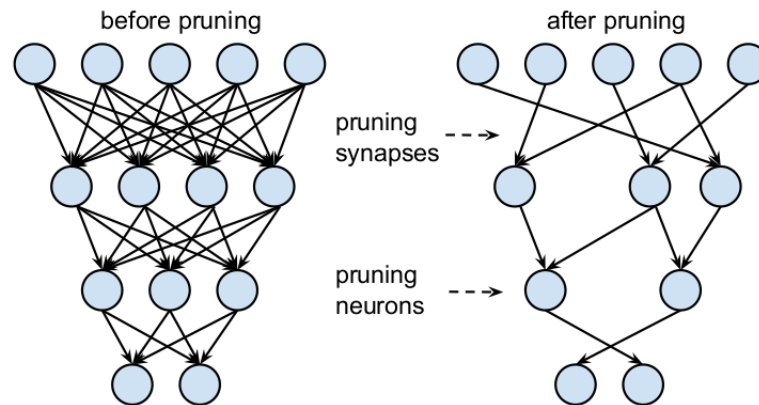


FIGURE 24 – Architecture d'un réseau de neurones avant et après élagage.[21]

Les avancements dans le domaine de l'apprentissage profond ont démontré que plus les architectures des réseaux de neurones grandissaient, plus leurs performances augmentaient. Cependant cette hausse de performances s'accompagne de coûts de calculs, de stockage et de consommation énergétique relativement importants, ce qui empêche le déploiement de tels réseaux dans des systèmes embarqués soumis à des contraintes. C'est de là qu'apparaît un besoin de techniques et de meilleures pratiques qui permettent la création de modèles optimisés. L'élagage ou plus communément appelé "*Pruning*" fait partie de ces techniques.

L'élagage consiste en la suppression de poids ou de filtres non critiques ou redondants satisfaisants des critères. Cela dans le but d'obtenir de plus petits réseaux à partir de réseaux relativement plus grands. Rechercher la meilleure combinaison de neurones à élaguer peut être un problème complexe. En outre, un réseau élagué avec une grande parcimonie peut ne pas conduire à des avantages pratiques. Par conséquent, un algorithme d'élagage réussi doit être efficace tout en réduisant la taille du modèle, en améliorant la vitesse d'inférence et en maintenant la précision.

4.1 Les approches d'élagage

Différentes méthodes ont été développées dans le but d'identifier les structures les plus pertinentes à élaguer sans dégrader la précision du modèle. Ces méthodes peuvent être classifiées en deux catégories selon leur dépendance ou non aux données d'entraînement [22, 23]. La méthode indépendante des données est plus efficace que la méthode dépendante, car l'utilisation des données d'apprentissage nécessite beaucoup plus de calculs.

Le critère le plus couramment utilisé est un critère basé sur la norme des paramètres du réseau à élaguer [23]. Ce dernier admet que les poids/filtres dotés de normes très petites peuvent être élagués à cause de leur faible contribution. Après avoir calculer les normes des poids/filtres d'un modèle un seuil prédéfini est utilisé pour sélectionner les structures à élaguer. Pour pouvoir utiliser ce critère deux conditions préalables sont nécessaires : que la variance des normes des poids/filtres soit significative, cela permet d'avoir un espace de recherche pour le seuil assez large pour séparer les poids/filtres à élaguer. Deuxièmement, Les normes des poids/filtres doivent être très proches de zéro [23]. La figure 25 illustre les conditions nécessaires d'utilisation de ce critère.

L'utilisation de ce critère permet la réduction significative des opérations flottantes *FLOPs* et la taille du modèle. Mais cela n'est pas un gage d'efficacité en pratique, car la réduction des *FLOPs* n'est pas directement liée à une accélération matérielle de l'inférence [22], surtout dans le cas des CNN profonds. En effet, la suppression des poids peut mener à des matrices parcimonieuses, ce

qui est mal supporté par certaines architectures.

Dans le but d'éviter un élagage non structuré, d'autres approches basées sur un élagage structuré ont été développées : les poids sont regroupés selon leurs similitudes avant d'être élagués. Pour les CNN des filtres peuvent être élagués complètement selon leur importance et contribution dans le réseau. Ces approches ont démontré de meilleurs résultats en matière de réduction de taille de réseaux et de conservation de la précision [22].

Remarque : Dans la littérature, il existe d'autres critères d'élagage qui ont été utilisés [22] (e.x Similarité entre poids/filtres, Suppression des activations avec un gradient plat, Minimisation de la détérioration de l'élagage en utilisant la matrice Hésienne ...). Ils ont prouvé leur efficacité, mais restent plus difficiles à mettre en place.

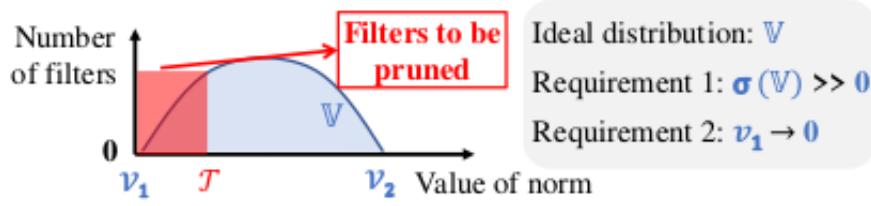


FIGURE 25 – Conditions nécessaires pour l'application du critère d'élagage basé sur la norme[23].

4.2 Partie expérimentale

Dans le but d'étudier l'impact de l'élagage sur l'intégration des réseaux de neurones dans des microcontrôleurs. Nous avons décidé de nous orienter vers l'élagage des CNN étant donnée que ce type d'architecture a démontré son efficacité par le passé et est actuellement le type de réseaux de neurones sur lequel se basent les travaux d'élagage.

Nous avons basé nos expérimentations sur les travaux de Y.He et al. [23] où une approche d'élagage structurée a été développée en se basant sur un critère de médiane géométrique pour élaguer les filtres des couches de convolution. Les chercheurs ont démontré que ce critère permettait d'élaguer les filtres d'une couche de convolution en commençant par les filtres les plus proches de sa médiane géométrique, car ces derniers contenaient des informations redondantes qui pouvaient être représentées par les filtres restants de la couche.

4.2.1 Méthodologie

La médiane géométrique x^{GM} est utilisée pour déterminer les informations communes aux filtres d'une couche de convolution i tel que :

$$x^{GM} = \arg \min_{x \in \mathbb{R}^{N_i \times K \times K}} \sum_{j' \in [1, N_{i+1}]} \|x - \mathcal{F}_{i,j'}\|_2 \quad (19)$$

Après avoir déterminé la médiane géométrique x^{GM} , on détermine le filtre de la couche de convolution i le plus proche de cette valeur :

$$F_{i,j*} = \arg \min_{F_{i,j'}} \|\mathcal{F}_{i,j'} - x^{GM}\|_2, \text{ tel que : } j' \in [1, N_{i+1}] \quad (20)$$

$F_{i,j*}$ est un filtre qui peut être représenté par les autres filtres de la couche, et le fait de l'élaguer ne dégrade pas les performances du réseau. Ce processus se applique dans le cas d'un seul filtre à élaguer. Pour pouvoir élaguer plusieurs filtres, dans le cas d'une couche avec plusieurs filtres, les filtres sont élagués selon leur distance par rapport à la médiane géométrique, de telle sorte

que les filtres les plus proches sont élagués en premier, et cela sans inclure les filtres déjà élagués (Eq. 21)

$$F_{i,x'*} = \arg \min_x g'(x), \text{ tel que : } j' \in [F_{i,j}, F_{i,N_{i+1}}] \quad (21)$$

Avec

$$g'(x) = \arg \min_{x \in \mathbb{R}^{N_i \times K \times K}} \sum_{j' \in [1, N_{i+1}], F_{i,j} \neq x} \|x - \mathcal{F}_{i,j'}\|_2 \quad (22)$$

Algorithm 2 Algorithme d'élagage de filtres par médiane géométrique

Input : Données d'apprentissage X .
 Choisir le taux d'élagage P_i , tel que ($P_i < 1$).
 Initialiser les poids du réseau W .
for $epoch < epoch_{max}; epoch++$ **do**
 Mettre à jour les poids W du réseau par rapport aux données X .
 for $i = 1; i \leq L; i++$ **do**
 Trouver les $N_{i+1} \times P_i$ filtres qui satisfassent l'Eq. 21.
 Mettre à zéro les filtres sélectionnés.
 end for
end for
Output : modèle réduit avec des poids W^* à partir de W .

Nous avons entraîné plusieurs réseaux de type Resnet avec l'algorithme décrit ci-dessus en utilisant différents taux d'élagage. Cette approche d'élagage ayant été testée uniquement pour des taux d'élagage de 30% et 40%, nous avons décidé d'aller plus loin en choisissant des taux d'élagage P_i allant de 50% à 90% dans le but d'analyser l'efficacité de cette méthode pour des élagages extrêmes et de déterminer à partir de quel taux P_i la précision d'un réseau commence à se dégrader.

Pour cela nous avons prédéfini des conditions d'entraînement à appliquer aux différents tests pour éviter l'ajout d'un biais aux résultats, lié à la configuration d'apprentissage : entraînement sur 100 epochs, SGD comme optimiseur et un pas d'apprentissage de 0.1 qui varie d'un facteur de 0.1 à l'epoch 75 puis 90. Et comme fonction de perte nous avons choisi l'entropie croisée (voir Eq.8). La base de données utilisée pour l'apprentissage est Cifar10 [16].

Dans le but d'estimer l'impact de la méthode d'élagage utilisée sur microcontrôleur, nous avons suivi la même procédure que celle utilisée pour l'estimation de l'impact de la quantification 3.4.1. Deux réseaux ont été entraînés sur la base de données Mnist [20] avec la même configuration d'apprentissage : le premier est un apprentissage standard et le deuxième est basé sur le processus d'élagage décrit précédemment avec un taux d'élagage de 50%. Les réseaux sont ensuite intégrés sur la carte pour effectuer des mesures de temps d'inférence.

4.2.2 Résultats

En analysant les résultats obtenus on peut observer que plus le réseau de neurones élagué est profond, moins il est sensible aux élagages extrêmes. Les courbes de la figure 26 illustre bien ce phénomène : on remarque que la précision des réseaux Resnet20, 26, 32, se dégrade à partir d'un taux d'élagage supérieur à 70%. Pour le Resnet50, la dégradation des performances commence à se voir à partir d'un seuil d'élagage supérieur à 80%. On peut aussi constater que même après un élagage de 90% la précision du Resnet50 est de 0.80 ce qui peut être considéré comme une précision acceptable pour certaines applications. Quant au plus petit réseau entraîné, le Resnet8, on observe très nettement la dégradation de la précision proportionnellement à la valeur du taux d'élagage.

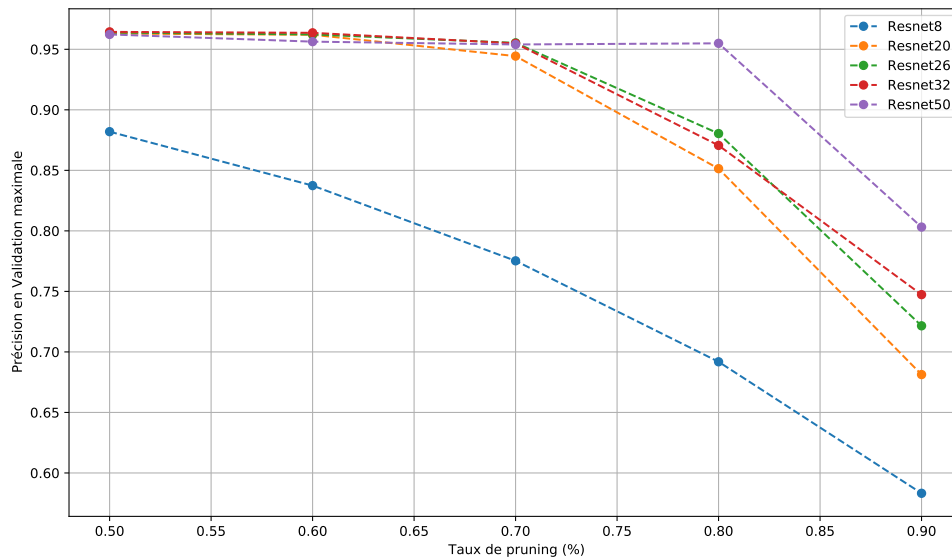


FIGURE 26 – Évolution de la précision maximale sur la base de validation en fonction du taux d’élagage.

La figure 27 illustre l’évolution de la précision des différents réseaux entraînés en fonction du nombre de paramètres non nuls restant après élagage. Chaque point des différentes courbes représentant un taux d’élagage spécifique inversement proportionnel au nombre de paramètres. On remarque qu’à partir d’un certain nombre de paramètres la précision n’augmente plus. On peut en déduire qu’à partir d’un certain seuil de paramètres il n’y a plus de gain en précision et que le reste des paramètres ne sont que des paramètres redondants qui n’apporte aucune contribution notable aux performances du réseau.

Ce phénomène de redondance est proportionnel à la taille du réseau. Cela peut être illustré en comparant le taux d’élagage à partir duquel la précision d’un réseau commence à se dégrader. En analysant les courbes des deux réseaux Resnet50 et Resnet8 (Fig. 27) on constate que la précision du réseau Resnet50 commence à se dégrader à partir d’un nombre de paramètres inférieur à 150000, soit après la suppression de 80% de ses paramètres initiaux. Pour le Resnet8 ce phénomène se produit dès la suppression de 50% de ses paramètres initiaux et on peut affirmer que cela se produit même bien au dessous d’un taux de pruning de 50% au vu de l’allure que suit la courbe. Cette dégradation de la précision est causée par le fait qu’on commence à partir d’un certain taux d’élagage à élaguer des paramètres essentiels pour la représentation des données d’apprentissage sans lesquels le réseau ne pourrait pas classifier ces données.

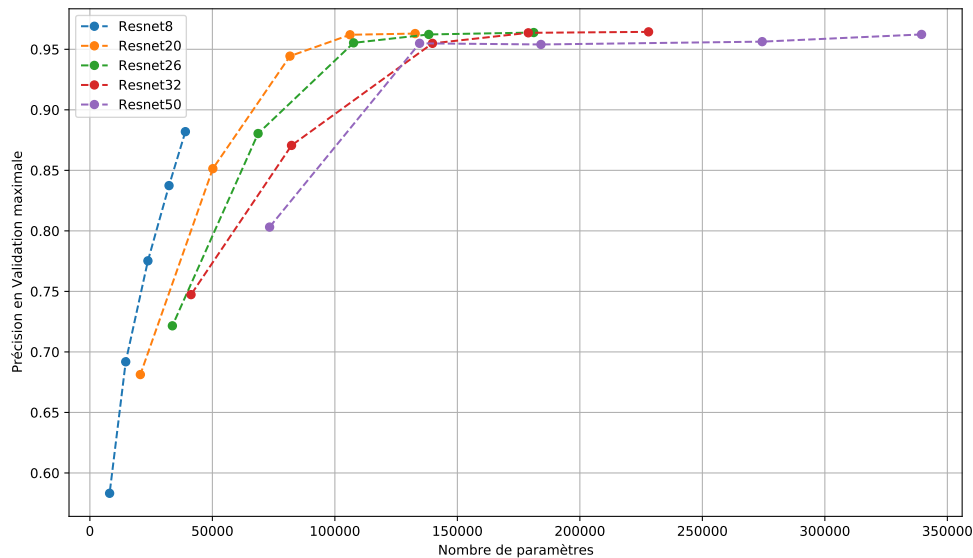


FIGURE 27 – Évolution de la précision maximale sur la base de validation en fonction du nombre de paramètres du réseau.

Les résultats de l'estimation de l'impact de l'élagage utilisé sur microcontrôleur sont résumés dans le tableau 3

Architecture	Standard	Élagage à 50% incluant les filtres nuls	Élagage à 50% excluant les filtres nuls
Temps d'inférence (ms)	58.97	58.98	25.97
Débit (cl/s)	16.96	16.95	38.51
Mémoire occupée (kbits)	247.32	247.32	239,83

TABLE 3 – Estimation de l'impact de l'élagage du réseau Lenet5 sur STM32F446ZE à une fréquence de 180MHz.

En analysant les résultats obtenus lors des mesures de temps d'inférence, on remarque qu'avec un élagage à 50% nous arrivons à réduire le temps d'inférence de 55% sur un réseau peu profond tel que Lenet5. Mais pour arriver à un tel résultat il faudra apporter une légère modification à l'architecture lors de l'implémentation du réseau en langage C en enlevant complètement les filtres élagués de l'architecture. Car le fait de laisser ces filtres, même s'ils sont mis à zéro, n'apporte aucun changement en matière d'accélération matérielle comparé à une architecture non élaguée. Concernant la mémoire, l'élagage à 50% a permis un gain de mémoire de 4%. Ce gain mémoire assez faible est dû au fait que la méthode d'élagage que nous avons implémenté ne prend en considération que l'élagage des filtres des couches de convolution et non les poids des couches entièrement connectées qui occupent généralement plus d'espace mémoire.

4.2.3 Conclusion

L'approche d'élagage utilisée lors de nos expérimentations a démontré une efficacité notable en matière d'accélération matérielle et d'optimisation de l'espace mémoire occupé par le réseau de neurones. Cependant, le réseau de neurones choisi pour les mesures ne permet pas de mettre en lumière tout le potentiel qu'offre l'utilisation d'une telle approche d'élagage, et cela est dû au fait

que le réseau Lenet5 est constitué de plusieurs couches entièrement connectées qui ne peuvent malheureusement pas être élaguées en utilisant cette approche.

5 Conclusion Générale

5.1 Comparaison des méthodes de réduction de réseaux de neurones implémentées

Méthode de réduction	Distillation hors ligne	Élagage des filtres par médiane géométrique (Taux= 50%)	Quantification INT8
Latence	N/A	+55%	+21.5%
Débit	N/A	$\times 2, 26$	$\times 1, 27$
Taux de compression	N/A	4%	75%
Précision	+2%	+3%	+0%
Facilité d'implémentation	+++++	+++	+

TABLE 4 – Tableau comparatif de l'impact des différentes techniques de réduction de réseaux de neurones implémentées.

En analysant les résultats illustrés dans le tableau 4, on constate que l'élagage est la technique de réduction de réseaux de neurones présentant le plus d'avantages. Mais ces observations restent relatives aux mesures effectuées et au réseau choisi. Cependant, cela ne veut pas dire que les autres méthodes ne sont pas utiles : on remarque que la quantification arrive à atteindre un taux de compression considérable, surpassant celui de l'élagage, sans dégradation de la précision. Quant à la distillation de connaissances, il reste très difficile d'estimer son impact sur une cible embarquée du fait qu'elle soit une technique se basant sur l'optimisation de la précision du réseau uniquement.

5.2 Perspectives d'amélioration de nos travaux

1. **Distillation de connaissances** : la distillation de connaissances n'étant pas une méthode de réduction qui permet l'optimisation d'un réseau de neurones en lui même et ne prenant pas en considération l'aspect matériel, l'association de cette méthode à la quantification ou à l'élagage pour pallier tous ses inconvénients est une piste intéressante à explorer.
2. **Quantification** : l'approche que nous avons implémenté, malgré la difficulté de la mise en place du processus d'inférence sur la carte, a donné de bon résultats. Pour aller plus loin, nous aurions voulu la tester sur des réseaux plus profonds et pour des précisions plus basses dans but de faire une étude plus approfondie et un *benchmark* plus complet.
3. **Élagage** : malgré le peu de mesures effectuées l'approche d'élagage utilisée a démontré une grande efficacité dans la réduction de CNN. Il serait intéressant de faire plus de mesures sur des réseaux plus profonds et pour des taux d'élagages plus importants.
4. **Benchmark** : par manque de temps nous n'avons pas eu l'occasion d'effectuer tous les tests et mesures voulus. Il serait très intéressant de faire les mêmes mesures que nous avons effectuées sur plusieurs réseaux dotés d'architectures différentes et si possible faire une estimation de la consommation énergétique de ces derniers dans le but d'avoir un benchmark complet des techniques de réductions de réseaux de neurones.

5.3 Difficultés rencontrées

Les difficultés majeures rencontrées lors du stage concernent principalement le manque de ressources matérielles pour effectuer l'apprentissage des réseaux de neurones. Le processus d'appren-

tissage et d'optimisation des réseaux de neurones sont des tâches très coûteuses en ressources de calcul, et le fait de ne pas avoir à notre disposition des GPUs nous a grandement ralenti lors de nos différents tests. Malgré cela nous avons réussi à exploiter le cluster CPU, certes moins performant, mais il reste utile pour gagner du temps.

Références

- [1] Ramon SANCHEZ-IBORRA et Antonio F. SKARMETA. “TinyML-Enabled Frugal Smart Objects : Challenges and Opportunities”. In : *IEEE Circuits and Systems Magazine* 20.3 (2020), p. 4-18. DOI : 10.1109/MCAS.2020.3005467.
- [2] MARTÍN ABADI et al. *TensorFlow : Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL : <https://www.tensorflow.org/>.
- [3] *AIfES : Large-Artificial intelligence for embadded systems*. 2020. URL : https://www.ims.fraunhofer.de/%20en/Business_Units_and_Core_Competencies/Electronic-Assistance%20-Systems/Technologies/Artificial-Intelligence-for-Embedded-Systems%20-AIfES.html.
- [4] Erwei WANG et al. “Deep Neural Network Approximation for Custom Hardware”. In : *ACM Computing Surveys* 52.2 (mars 2020), p. 1-39. ISSN : 1557-7341. DOI : 10.1145/3309551. URL : <http://dx.doi.org/10.1145/3309551>.
- [5] Benoit JACOB et al. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017. DOI : 10.48550/ARXIV.1712.05877. URL : <https://arxiv.org/abs/1712.05877>.
- [6] Shervine Amidi SAFSHINE AMIDI. *Pense-bête de réseaux de neurones convolutionnels*. DOI : /cs-230/pense-bete-reseaux-neurones-convolutionnels. URL : <https://stanford.edu/~shervine/1/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels>.
- [7] Benjamin PERRET. “Convolution”. In : (2017), <https://perso.esiee.fr/perretb/I5FM/TAI/convolution/index.html>.
- [8] Muhamad YANI, S IRAWAN et Casi SETIANINGSIH. “Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry’s Nail”. In : *Journal of Physics : Conference Series* 1201 (mai 2019), p. 012052. DOI : 10.1088/1742-6596/1201/1/012052.
- [9] Geoffrey HINTON, Oriol VINYALS et Jeff DEAN. *Distilling the Knowledge in a Neural Network*. 2015. DOI : 10.48550/ARXIV.1503.02531. URL : <https://arxiv.org/abs/1503.02531>.
- [10] Jianping GOU et al. “Knowledge Distillation : A Survey”. In : *International Journal of Computer Vision* 129.6 (mars 2021), p. 1789-1819. ISSN : 1573-1405. DOI : 10.1007/s11263-021-01453-z. URL : <http://dx.doi.org/10.1007/s11263-021-01453-z>.
- [11] Kaiming HE et al. *Deep Residual Learning for Image Recognition*. 2015. DOI : 10.48550/ARXIV.1512.03385. URL : <https://arxiv.org/abs/1512.03385>.
- [12] Jang Hyun CHO et Bharath HARIHARAN. “On the Efficacy of Knowledge Distillation”. In : *CoRR* abs/1910.01348 (2019). arXiv : 1910.01348. URL : <http://arxiv.org/abs/1910.01348>.
- [13] Xu CHENG et al. “Explaining Knowledge Distillation by Quantifying the Knowledge”. In : *CoRR* abs/2003.03622 (2020). arXiv : 2003.03622. URL : <https://arxiv.org/abs/2003.03622>.
- [14] Mary PHUONG et Christoph H. LAMPERT. “Towards Understanding Knowledge Distillation”. In : *CoRR* abs/2105.13093 (2021). arXiv : 2105.13093. URL : <https://arxiv.org/abs/2105.13093>.
- [15] Jing YANG et al. *Knowledge distillation via adaptive instance normalization*. 2020. DOI : 10.48550/ARXIV.2003.04289. URL : <https://arxiv.org/abs/2003.04289>.
- [16] Alex KRIZHEVSKY. *Learning multiple layers of features from tiny images*. Rapp. tech. 2009.
- [17] Antonio POLINO, Razvan PASCANU et Dan ALISTARH. “Model compression via distillation and quantization”. In : *CoRR* abs/1802.05668 (2018). arXiv : 1802.05668. URL : <http://arxiv.org/abs/1802.05668>.

- [18] Amir GHOLAMI et al. *A Survey of Quantization Methods for Efficient Neural Network Inference*. 2021. DOI : 10.48550/ARXIV.2103.13630. URL : <https://arxiv.org/abs/2103.13630>.
- [19] Y. LECUN et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In : *Neural Computation* 1.4 (déc. 1989), p. 541-551. ISSN : 0899-7667. DOI : 10.1162/neco.1989.1.4.541. eprint : <https://direct.mit.edu/neco/article-pdf/1/4/541/811941/neco.1989.1.4.541.pdf>. URL : <https://doi.org/10.1162/neco.1989.1.4.541>.
- [20] Li DENG. “The mnist database of handwritten digit images for machine learning research”. In : *IEEE Signal Processing Magazine* 29.6 (2012), p. 141-142.
- [21] Song HAN et al. “Learning both Weights and Connections for Efficient Neural Networks”. In : *CoRR* abs/1506.02626 (2015). arXiv : 1506.02626. URL : <http://arxiv.org/abs/1506.02626>.
- [22] Jiayi LIU et al. “Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications : A Survey”. In : *CoRR* abs/2005.04275 (2020). arXiv : 2005.04275. URL : <https://arxiv.org/abs/2005.04275>.
- [23] Yang HE et al. “Pruning Filter via Geometric Median for Deep Convolutional Neural Networks Acceleration”. In : *CoRR* abs/1811.00250 (2018). arXiv : 1811.00250. URL : <http://arxiv.org/abs/1811.00250>.
- [24] Matthieu COURBARIAUX, Y. BENGIO et Jean-Pierre DAVID. “Training deep neural networks with low precision multiplications”. In : déc. 2015.
- [25] Xundong WU, Yong WU et Yong ZHAO. *Binarized Neural Networks on the ImageNet Classification Task*. 2016. DOI : 10.48550/ARXIV.1604.03058. URL : <https://arxiv.org/abs/1604.03058>.