N° d'ordre NNT : 2022LYSEC017

THESE de DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de l'Ecole Centrale de Lyon

École Doctorale n°160
Électronique Électrotechnique et Automatique (EEA)
Spécialité de doctorat :
Discipline : Électronique

Soutenue publiquement le 10/05/2022, par :

DUPUIS Etienne

WEIGHT-SHARING METHODS FOR RETRAINING-FREE CNN COMPRESSION

Devant le jury composé de :

Acknowldgements

I would like to extend my first thanks to my three thesis supervisors, Alberto Bosio for his unlimited support on all tasks, from research strategy to publication and dissemination, David Novo, for always looking at problems from a helicopter's view, in addition to his very precise sense of research and writing methods, and Ian O'Connor for sharing his knowledge on research strategy and actors, and for supporting the scientific work and writing. The three of them form a very complementary team in terms of the distribution of skills and personalities, which certainly had a very positive impact on this work.

I would also like to thank my collaborators. David Briand, Olivier Sentieys, and Silviu Filip, for working together over the years in the AdequateDL consortium project, including a book chapter reviewing the literature. Marcello Traiola for having very interesting discussions on the subject of approximate computing. Thibault Allenet for sharing ideas on approximate computing for deep learning.

I would like to thank the resources and people that I have found in my close working environment. The lab and school data centers and their powerful GPU server for allowing me to develop and use all the required frameworks, and the CNRS data center for running massively parallel simulations. The infrastructure team of the lab and the school, Laurent Carrel and Laurent Pouilloux, for having indicated and given me access to different services as well as for the numerous troubleshooting. The members of my thesis follow-up committee for having followed closely my research work and my training during these years. The administrative teams of the lab, particularly Sylvie Goncalves, and Patricia Dufault for their constant help with the insane amount of registrations process.

I would like to thank everyone who provided me technical support in any way, including, but not limited to my esteemed colleagues and now friends, Mayeul Cantan, who gave me open access to his Wikipedia-level knowledge and impressive skills, Clément Zrounba, for troubleshooting and discussing work, and Arnaud Poittevin, for his scientific insights. I would also like to thank my dear brother Jean Dupuis for introducing me to various very efficient industrial tools for machine learning.

Over the years, I found emotional support in the person of my beloved wife, Alice Limouzin, and I would like to thank her for sharing her creativity, in addition to her legendary optimism with me. I would also like to deeply thank my family and my mom, for always taking the time to listen to my boring scientific discussions and for giving me invaluable motivation in the pursuit of my various studies since my childhood. I would like to thank my dear friends Dr. Adil Brik and Yazan Barazzi for their motivation and their philosophical vision on the Ph.D. And more generally, I would like to thank all my friends who have supported me in every possible way and allowed me to take time off work when necessary.

Last but not least, I want to thank the ANR AdequateDL project (ANR-18-CE23-0012) for having covered all the financial aspects and allowed this thesis to come into being.

Abstract

The outstanding performance achieved by Convolutional Neural Network (CNN) comes at the cost of extremely high computational requirements, making them out of reach for most low-power embedded devices [1]. Most of the energy cost during CNN inference comes from memory access as analyzed in [1]. This work focuses on reducing the memory footprint of CNNs to improve energy efficiency. The Approximate Computing (AxC) paradigm leverages the inherent error-resilience of CNNs to improve energy efficiency by relaxing the need for fully accurate operations. CNNs have a high degree of redundancy in terms of their structure and parameters [2], and this redundancy is not always necessary for an accurate prediction. This observation has paved the way for several highly recognized approximation techniques [1] such as pruning, quantization, low-rank factorization, and Weight Sharing (WS) [2]. WS aims to group weights into buckets or clusters sharing the same value. It allows a significant reduction of the CNN memory footprint by storing shared values in a dedicated data structure, where original weight values in the weight matrix are replaced by their corresponding indexes, represented with fewer bits. To make an example, by reducing the number of different shared values (k) to 256, the indexes can be encoded with only $log_2(k) = 8$ bits, allowing a $4\times$ Compression Rate (CR) compared to the original 32-bit counterpart. Shared weight values can be determined by using a clustering algorithm like the K-means.

WS, as well as the others approximation techniques, came at the cost of a certain accuracy loss. Although the standard application of WS [2]–[5], requires the retraining of the network to recover accuracy loss, it has been proven [6] that it is also possible to optimize the number of shared values to each layer's resilience while avoiding the costly retraining step.

ghhiExhaustively exploring every combination results in $\mathcal{O}(|k_{range}|^N)$ complexity, with $|k_{range}|$ being the set of the possible number of shared values and N the number of layers. As an example, a toy CNN with $N = 5$ layers and a $k_{range} = [1, 256]$, results in a number of scoring steps equal to $256^5 = 1.1 \times 10^{12}$. Thus, the exploration of the complete solution space would require more than three decades when considering an optimistic 1 ms evaluation step. To make things worse, the complexity rises exponentially with N, which is higher than a factor of ten higher in recent CNNs.

In this thesis, we propose a heuristic approach to achieve a scalable retraining-free WS compression. The main contributions can be summarized as (1) The extensive study of the complexity of the weight-sharing optimization; (2) A novel automatic two-step heuristic optimization to retraining-free weight-sharing.

The proposed method for CNN compression can efficiently explore the large design space to produce a set of solutions that offer very interesting trade-offs between AL and CR. These solutions achieve more than a $5\times$ compression over the baseline memory footprint in multiple state-of-the-art computer vision

CNNs on the challenging Imagenet dataset under MLPerf [7] quality target constraints. Importantly, these compression results are achieved while avoiding the prohibitively costly retraining step, which is commonly used in prior works.

The thesis dissemination consists of 4 publications in international conferences with proceedings, 1 journal Elsevier, 1 chapter of a book edited by Springer, and 4 poster dissemination in summits or workshops. The early works using greedy exploration on LeNet/MNIST were published in [8], a generalization to the Imagenet class CNNs with the use of proxy metrics to accelerate exploration were published in [9], an extension adding a multi-objective exploration was published in [6], the new divide & conquer approach results on MNIST were published in [10] and the generalization to Imagenet is published in [11]. Finally, a survey on AxC for deep learning was published in [12].

Résumé

Les performances exceptionnelles atteintes par les réseaux de neuronnes convolutionels (Convolutional Neural Network (CNN)s) se font au prix de ressources de calcul extrêmement élevées, les rendant hors de portée de la plupart des dispositifs embarqués à faible puissance [1]. La majeure partie du coût énergétique de l'inférence CNN provient de l'accès à la mémoire, comme analysé dans [1]. Ce travail se concentre sur la réduction de l'empreinte mémoire des CNNs pour améliorer l'efficacité´ énergétique. Le paradigme de calcul approximé (Approximate Computing (AxC)) tire parti de la résistance aux erreurs inhérente aux CNNs pour améliorer l'efficacité énergétique en assouplissant le besoin d'opérations totalement précises. Les CNNs ont un haut degré de redondance en termes de structure et de paramètres [2], et cette redondance n'est pas toujours nécessaire pour une prédiction acceptable. Cette observation a ouvert la voie à plusieurs techniques d'approximation très reconnues [1] telles que l'élagage (« pruning »), la quantification, la factorisation à faible rang et la Weight Sharing (WS) [2]. WS vise à regrouper les poids dans des classes partageant la même valeur dans une table de correspondance. Il permet une réduction significative de l'empreinte mémoire des CNNs en stockant les valeurs partagées dans une structure de données dédiée, où les coefficients originaux dans la matrice de poids sont remplacées par leurs index correspondants, représentés avec moins de bits. Pour prendre un exemple, en réduisant le nombre de valeurs partagées différentes (k) à 256, les index peuvent être codés avec seulement $log_2(k) = 8$ bits, permettant un ratio de compression CR de $4\times$ par rapport aux valeurs originales sur 32 bits. Les valeurs partagées des coefficients peuvent être déterminées à l'aide d'un algorithme de regroupement comme les K-means.

Le WS, ainsi que les autres techniques d'approximation, se font au prix d'une certaine perte de précision. Bien que l'application standard de WS [2]-[5], nécessite le ré-entrainement du réseau pour récupérer la perte de précision, il a été prouvé [6] qu'il est également possible d'optimiser le nombre de valeurs partagées en fonction de la résilience de chaque couche tout en évitant la coûteuse étape du ré-entrainement.

L'exploration exhaustive de toutes les combinaisons entraîne une complexité de $\mathcal{O}(|k_{range}|^N)$, avec $|k_{range}|$ la plage de valeurs partagées possibles et N le nombre de couches. À titre d'exemple, un CNN jouet avec $N = 5$ couches et un $k_{range} = [1, 256]$ résulte en un nombre d'étapes d'évaluation égal à $256^5 = 1.1 \times 10^{12}$. Ainsi, l'exploration de l'espace complet des solutions nécessiterait plus de trois décennies en considérant un pas d'évaluation optimiste de 1 ms. Pour aggraver les choses, la complexité augmente exponentiellement avec N, qui est plus au moins un ordre de grandeur supérieur dans les CNN récents.

Dans cette thèse, nous proposons une approche heuristique pour réaliser une compression WS sans ré-entraînement. Les principales contributions peuvent

être résumées comme suit : (1) L'étude approfondie de la complexité de l'optimisation du partage de coefficients ; (2) Une nouvelle optimisation heuristique automatique en deux étapes pour le partage du poids sans ré-entrainement.

La méthode proposée pour la compression de CNNs peut explorer efficacement le vaste espace de conception pour produire un ensemble de solutions qui offrent des compromis très intéressants entre la perte de précision et le CR. Ces solutions permettent d'obtenir une compression de plus de $5\times$ par rapport à l'empreinte mémoire initiale dans plusieurs CNN de vision par ordinateur de pointe sur le jeu de données Imagenet sous des contraintes de qualité MLPerf [7]. Il est important de noter que ces résultats de compression sont obtenus tout en évitant l'étape de réapprentissage prohibitive et coûteuse, qui est couramment utilisée dans les travaux antérieurs.

La diffusion des résultats de la thèse consiste en 4 publications dans des conférences internationales avec actes, 1 journal ACM, 1 chapitre d'un livre, et 4 posters diffusés dans des sommets ou des ateliers. Les premiers travaux utilisant l'exploration sur LeNet/MNIST ont été publiés dans [8], une généralisation à la classe Imagenet CNNs avec l'utilisation d'une métrique proxy pour accélérer l'exploration a été publiée dans [9], une extension ajoutant une exploration multi-objectif a été publiée dans [6], les résultats de la nouvelle approche ≪ diviser pour régner ≫ sur MNIST ont été publiés dans [10] et la généralisation à Imagenet est publiée dans [11]. Enfin, une enquête sur AxC pour l'apprentissage profond a été publiée dans [12].

French Summary

Les performances exceptionnelles obtenues par les réseaux de neurones convolutifs, ou Convolutional Neural Network (CNN) se font au prix d'exigences de calcul extrêmement élevées, ce qui les rend hors de portée de la plupart des dispositifs embarqués à faible puissance [1]. La majeure partie du coût énergétique pendant l'inférence des CNNs provient de l'accès à la mémoire, comme analysé dans [1]. Ce travail se concentre sur la réduction de l'empreinte mémoire des CNNs pour améliorer l'efficacité énergétique.

Le paradigme du calcul approximatif tire parti de la résistance aux erreurs inhérente aux CNNs, pour tirer parti d'une amélioration de l'efficacité énergétique apporté par le soulagement du besoin d'opérations totalement précises qui offre un certain niveaud de flexibilté sur le calcul. Les CNNs ont un haut niveau de redondance à la fois au niveau de leur structure et de leur paramètres, cette redondance n'est pas toujours nécessaire pour une prédiction précise [2]. Cette observation a ouvert la voie à plusieurs techniques d'approximation très reconnues [1] telles que l'élagage ou pruning, la quantification ou quantization, la factorisation à faible rang et le partage de poids ou Weight Sharing (WS).

Le WS a pour but de regrouper les paramètres, ou poids, dans des ≪ seaux ≫, ou clusters, partageant la même valeur. Il offre une réduction significative de l'empreinte mémoire du CNN ciblé en permettant de stocker les valeurs partagées dans une structure dédié, ainsi, les valeurs dans la matrice de poids sont remplacées par l'index de la structure dédiée correspondante, représentés avec moins de bits. Si l'on peut réduire le nombre de valeurs partagées différentes (k) à 256, les index peuvent être codés avec seulement $log_2(k) = 8$ bits, ce qui permet un taux de compression, ou Compression Rate (CR) de $4\times$ par rapport à l'original utilisant des valeurs de paramètres flottantes encodées sur 32 bits. Les valeurs de poids partagées peuvent être déterminées à l'aide d'un algorithme de regroupement tel que le K-means.

Bien que l'application standard de WS [2]-[5], nécessite le recyclage du réseau pour récupérer la perte de précision, il a été prouvé [6] qu'il est également possible d'optimiser le nombre de valeurs partagées pour la résilience de chaque couche tout en évitant l'étape coûteuse du recyclage.

L'exploration exhaustive de chaque combinaison entraîne une complexité de $\mathcal{O}(|k_{range}|^N)$, avec $|k_{range}|$ étant l'ensemble des nombres possibles de valeurs partagées et N le nombre de couches. À titre d'exemple, un CNN jouet avec $N = 5$ couches et un $k_{range} = [1, 256]$, résulte en un nombre d'étapes de notation égal à $256^5 = 1.1\ \times 10^{12}$. Ainsi, l'exploration de l'espace complet des solutions nécessiterait plus de trois décennies en considérant un pas d'évaluation optimiste de 1 ms. Pour aggraver les choses, la complexité augmente exponentiellement avec N, qui augmente de plus d'un facteur dix dans les CNN récents.

Objectifs d'optimisation

L'optimisation du nombre de valeurs partagées pour un CNN entraîné donné peut être représentée comme un problème d'optimisation mathématique. La définition traditionnelle du problème d'optimisation consiste à trouver les meilleures solutions, décrites par des valeurs de variables, qui maximisent les objectifs tout en respectant les contraintes. Dans le cas spécifique du réglage de WS, nous avons ce qui suit :

- Les variables peuvent être représentées par un vecteur contenant le nombre de valeurs partagées de chaque couche ;
- Les objectifs sont les métriques sélectionnées pour représenter l'efficacité d'execution et la précision ;
- La contrainte est la cible de précision décrite par MLPerf [7], bien qu'il soit possible d'en ajouter d'autres lors de l'étude d'une implémentation spécifique.

Pour définir l'espace de recherche de l'optimisation, il est nécessaire de fixer une plage du nombre possible de valeurs partagées, ou des valeurs possibles des variables d'optimisations. Cette plage peut être choisie arbitrairement, ou déterminée à l'aide de différentes techniques.

La représentation de la performance de précision est quelque chose de très courant lors du prototypage d'un CNN, elle est généralement réalisée en utilisant la précision top-1 ou top-5, c'est-à-dire le pourcentage de fois où l'étiquette attendue était présente parmi les étiquettes prédites du top-1 ou du top-5 en termes de probabilité. Une autre métrique de précision est la distance entre la sortie prédite et la sortie attendue, calculée à l'aide d'une fonction de perte. La précision et la perte sont généralement calculées en utilisant un ensemble de données de validation qui n'a jamais été utilisé pour entraîner le CNN, ce qui permet de mesurer sa capacité de généralisation. La grande majorité des articles sur l'approximation indiquent que la perte de précision, ou Accuracy Loss (AL) du top 1 ou du top 5, calculée comme la différence absolue entre la précision du CNN de base et la précision du CNN approximé. Un AL négatif signifie que le CNN approché atteint une précision plus élevé que le CNN de base, ce n'est pas habituel mais cela peut arriver avec une approximation agissant comme une régularisation pendant l'apprentissage. La métrique qui sera utilisée pour représenter la précision dans ce travail de thèse est le top-1 AL, car il donne un aperçu plus direct du comportement du CNN approximé par rapport à la référence.

Les deux métriques peuvent être obtenues à partir d'un CNN approximé, décrit par un vecteur représentant le nombre de valeurs partagées. Le CR peut être calculé à l'aide d'une formule analytique décrite par l'équation :

$$CR = \frac{W \times b_{valeurs}}{W \times b_{index} + k_i \times b_{valeurs}}, \tag{1}$$

Avec $b_{valeurs}$ le nombre de bits utilisés pour représenter une valeur partagée et b_{index} le nombre de bits utilisés pour représenter l'index d'une valeur partagée. W le nombre de poids de la couche et k_i le nombre de valeurs partagées de la couche.

D'autre part, l'obtention de l'AL nécessite une évaluation de la précision de l'inférence pour le CNN approximé donné, ce qui signifie également qu'il est nécessaire d'appliquer le partage de poids avant l'évaluation. La première étape de l'évaluation de la précision est très exigeante en termes de calcul, et vous verrez que c'est un problème commun dans les recherches menées au cours de cette thèse. En ce qui concerne les contraintes d'optimisation, le concours MLPerf [7], destiné à permettre la comparaison de plateformes hétérogènes d'apprentissage et d'inférence de CNN à l'aide de mesures de haut niveau comme le débit et la latence, a fixé un objectif de précision acceptable pour l'approximation des CNNs. Pour atteindre cet objectif de qualité, la précision du top 1 de l'approximation des CNN doit être d'au moins 99% de la précision de base. Un deuxième niveau d'objectif de qualité est fixé à 98% de la précision de base pour les CNNs ciblant l'inférence embarquée, comme MobileNetV2 [13], car ces CNN sont déjà optimisés dès leur conception et sont plus sensibles à l'approximation que les autres. Cet objectif de qualité a été largement utilisé depuis.

Il est également possible d'utiliser différentes contraintes concernant l'implémentation, comme les limites de mémoire ou de calcul pour faire respecter des métriques de haut niveau comme la latence ou le débit. Il est même possible d'utiliser des contraintes énergétiques pendant l'optimisation. Toutes ces contraintes liées à l'implémentation sortent du cadre de cette thèse.

Travaux réalisés

Deux approches ont été explorées pour l'optimisation du nombre de valeurs partagées. La première consiste en une optimisation gloutonne, dite glouton, qui peut être traduite en une optimisation locale de chaque couche du CNN dans un ordre particulier, avec un souci de conservation des effets d'approximation des couches déjà traitées. La seconde prend une approche plus heuristique avec une division du problème en un problème d'optimisation locale gloutonne et un problème d'optimisation globale à l'échelle du CNN entier. Ces deux approches sont décrites plus en détail dans les deux sous-parties suivantes.

Approche gloutonne ≪ Greedy ≫

L'utilisation d'un algorithme d'optimisation glouton permet de réduire l'espace de recherche en s'appuyant sur l'optimisation locale. L'implémentation est très simple, pour chaque couche du réseau, de la première à la dernière, le meilleur nombre de valeurs partagées k_i, est trouvé localement en évaluant tous les candidats approximatifs possibles de l'intervalle k_{range}. L'évaluation de chacun des candidats approximatifs consiste en deux étapes : (1) application du WS avec les k_i souhaités à l'aide de l'algorithme de regroupement k-means [14], et (2) évaluation du CNN résultant. L'AL est calculée sur l'ensemble des données de

Type	Top-1 AL (%)	CR
N2D2 exporté (ref)	0.00	1
WS	0.02	4.06

Table 1 : Compression de LeNet-5 [15] sur le jeu de données MNIST [16], comparaison avec la référence N2D2 [17].

test, avec les couches précédentes et la couche actuelle approximées. À l'issue de cette étape d'évaluation, le candidat présentant le meilleur AL est sélectionné. La complexité temporelle de l'algorithme proposé est :

$$\mathcal{O}(card(k_{range}) \times N). \tag{2}$$

Avec N le nombre de couches du CNN.

Pour valider la capacité de l'algorithme glouton proposé à résoudre des problèmes d'optimisation réalistes, il faut d'abord prouver qu'il peut fonctionner sur de petits CNNs. Dans cette optique, Lenet-5 [15], un CNN à 5 couches capable de reconnaître les chiffres en noir et blanc à basse résolution tirés du jeu de données MNIST [16] est utilisé comme CNN de référence. Le nombre exploré de valeurs partagées est l'intervalle linéaire $k_{range} = [2; 25]$, 25 correspond au nombre de poids d'un noyau convolutif 2D, et permet une AL inférieure à 10^4 pour toutes les couches. L'entraînement a été réalisé à l'aide du framework open-source N2D2 [17]. La description du modèle LeNet-5 que nous avons utilisé est disponible dans le framework lui-même. Il est important de mentionner une fois de plus que l'approche proposée est indépendante de l'outil adopté.

Les résultats sont présentés dans la Table 1. Ces résultats prouvent que l'algorithme proposé peut être appliqué à LeNet-5 [15], et atteindre un CR significatif de plus de 4× sous une AL négligeable.

Après la preuve de concept sur un petit CNN, il est possible d'évaluer l'approche proposée sur des CNNs plus complexes. C'est pourquoi l'évaluation de la méthode proposée est également effectuée sur des CNNs plus grands, entrainés sur le jeu de données ImageNet [18] avec 1000 classes. Trois CNNs de classification d'images ont été sélectionnés, ResNet18V2 [19], SqueezeNet [20], et MobileNetV2 [13]. L'outil précédent, qui s'appuie sur N2D2 [17] pour la mesure de l'AL des CNNs, a été adapté pour pouvoir traiter les modèles ONNX [21] provenant du zoo de modèles ONNX, et MXNET [22] est utilisé pour l'execution des CNN sur un GPU, le même GPU est utilisé pour calculer l'algorithme k-means, en utilisant la bibliothèque DeepKMCuda [23]. Pour chaque couche du CNN, le nombre de valeurs partagées est incrémenté jusqu'à ce que l'AL soit négligeable (inférieure à 10^4) pour la première couche du CNN, puis une plage est sélectionnée dans $[k_{minAL} - 20; k_{minAL} + 20]$. Pour MobileNetV2, cette plage est étendue à $[k_{minAL} - 60; k_{minAL} + 60]$ car le CNN est plus sensible à l'approximation. Cela explique pourquoi les différents CNNs ont des k_{range} différents.

Réseau de référence	K_{range}	Top-1 AL (%)	CR
ResNet18v2	[40,80]	0,22%	5,28
SqueezeNetv1.1	[40,80]	0.53%	5.17
MobileNetv2	[2,120]	1.43%	4.85

TABLE 2 : Compression des CNNs formés sur le jeu de données ImageNet [18].

Le tableau 2 présente les résultats obtenus. Chacun des réseaux de référence a été compressé avec succès jusqu'à 5×, par rapport à la référence sur 32 bits. En ce qui concerne l'acceptabilité de l'AL, il apparaît que les résultats obtenus pour ResNet18V2 [24] et SqueezeNetv1.1 [20] sont conformes à l'objectif de qualité MLPerf [7], alors que les résultats pour MobileNetV2 [13] ne le sont pas. De plus, il est important de noter que le CR dépend principalement de la k_{range}, et est plutôt constant pour les différents CNNs. Même pour SqueezeNetv1.1 [20], qui est déjà conçu pour avoir une empreinte mémoire très faible, nous obtenons un CR de 5×. En d'autres termes, même si la méthode de référence CNN a été conçue pour être efficace en termes d'empreinte mémoire, la méthode WS proposée peut encore la réduire.

La méthode a été poussée plus loin avec l'utilisation d'une métrique proxy permettant de réduire considérablement le nombre d'évaluations du CNN approximé sur le jeu de données de test. Ceci a pour but de grandement réduire le temps d'évaluation de la précision et de permettre une optimisation à plusieurs objectifs. Cette partie ne sera pas développée ici. Pour plus d'informations, veuillez vous référer au chapitre 4.

Approche Heuristique

Comme il a été expliqué précédemment, il est possible d'aborder couche par couche l'optimisation du nombre de valeurs partagées d'un CNN entraîné en utilisant une approche simple de type gloutonne. Cette approche montre cependant des limites lorsqu'il s'agit d'échapper aux minima locaux et de résoudre le problème avec des métriques à l'échelle du modèle entier, comme le CR. C'est pourquoi l'évaluation des AL et des CR de l'ensemble des CNNs approchés est nécessaire pour éviter les minima locaux et avoir une meilleure représentation du problème d'optimisation. Il est alors possible d'adopter une approche heuristique pour résoudre le problème d'optimisation. Considérant qu'aucune des explorations gloutonnes n'a pu trouver un candidat approximatif conforme à l'objectif de qualité de MLPerf [7] sur le CNN déjà bien optimisé MobileNetV2 [13], ce réseau sera le modèle de référence pour toute cette étude.

Quelques notations ont besoin d'être introduites avant de poursuivre. Pour un CNN donné avec N couches, k_i est le nombre de valeurs partagées de la couche i. k_i est borné à un ensemble de valeurs k_{range}. Par conséquent, un CNN approximé peut être caractérisé par son $k_{tuple} = \{k_i\} \in [1, N]$ représentant

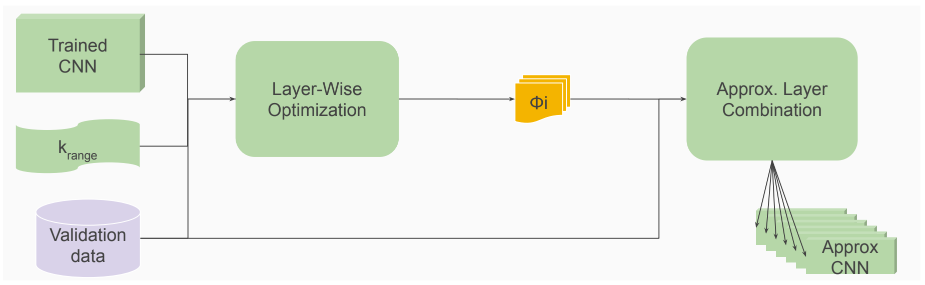


FIGURE 1 : Vue conceptuelle de l'approche en deux étapes proposée pour l'optimisation du WS.

le nombre de valeurs partagées pour chaque couche. À partir du k_{tuple}, il est possible de mesurer à la fois (1) l'AL, en évaluant le CNN approché sur un jeu de données de test large et représentatif, et (2) le CR, en utilisant Equation 1

L'exploration exhaustive de toutes les permutations possibles de k_{tuple} entraîne la complexité temporelle suivante :

$$\mathcal{O}(card(k_{range})^N). \tag{3}$$

Pour rendre le problème d'optimisation traitable, une stratégie de « diviser pour régner » peut être adoptée, en divisant le problème de complexité exponentielle d'optimisation du nombre de valeurs partagées pour chaque couche en deux sous-problèmes : (1) réduire l'espace de recherche en trouvant ϕ_i, l'ensemble des meilleurs k_i pour chaque couche du CNN et (2) appliquer une approche heuristique pour trouver un ensemble de k_{tuple} efficaces au sens de Pareto en tant que combinaison de $k_i \in \phi_i, \forall i \in N$ en considérant l'AL et le CR résultants. Une vue conceptuelle de cette approche est donnée dans la Figure 1. Cette approche s'apparente à une stratégie de « diviser pour mieux régner ».

Le premier sous-problème vise l'optimisation locale de chaque couche. Le but est de réduire l'espace de recherche en trouvant ϕ_i, l'ensemble des meilleures valeurs k_i, pour chaque couche du CNN. ϕ_i est caractérisé par (1) l'AL obtenu lorsque la couche i est approximée avec k_i valeurs partagées alors que les autres couches ne sont pas approximées, et (2) le nombre de bits requis pour stocker chaque index, défini comme $b_{index} = \lceil log_2(k_i) \rceil$. Le b_{index} est utilisé comme objectif d'optimisation locale, car il est indirectement proportionnel au CR de la couche, voir Equation 1. L'algorithme explore un par un les k_is dans le k_{range} défini par l'utilisateur, et pour chacun, il évalue à la fois l'AL et le b_{index} pour ensuite composer ϕ_i en sélectionnant le candidat avec la meilleure AL par b_{index}. Cette étape est très similaire à l'algorithme glouton, à la différence que lors de l'étude d'une couche, les autres sont maintenues inchangées. La complexité de cette recherche est linéaire au nombre de couches N et à la cardinalité du k_{range}, étant :

$$\mathcal{O}(N * card(k_{range})). \tag{4}$$

L'utilisation du b_{index} comme proxy pour le CR a pour effet de limiter la cardinalité du ϕ_i.

Le deuxième sous-problème consiste à trouver les combinaisons Pareto efficaces de CNNs approximés qui peuvent être décrites par leur $k_{tuple} = \{k_i \in \phi_i\} \forall i \in [1 : N]$. Le nombre de combinaisons possibles peut être écrit comme :

$$\mathcal{O}(\prod_{i=1}^{N+1} |\phi_i|). \tag{5}$$

Ainsi, la taille de l'espace de recherche est exponentielle au nombre de couches N, il n'est donc pas possible de l'explorer en utilisant une recherche exhaustive pour les grands CNNs. MobileNetV2 [13] avec 53 couches, donne ainsi une complexité temporelle d'environ $\mathcal{O}(n^{53})$. Pour résoudre ce problème, on propose l'utilisation d'un algorithme méta-heuristique, pour trouver un ensemble de k_{tuple} Pareto efficaces par rapport à l'AL et CR. Parmi la grande collection d'algorithmes méta-heuristiques, il est nécessaire de sélectionner un algorithme multi-objectif (AL, et CR) basé sur la population pour obtenir un ensemble de CNNs approximés efficaces au sens de Pareto. Il est proposé d'utiliser l'algorithme génétique NSGA-II [25] qui utilise un critère d'efficacité au sens de Pareto pour sélectionner les meilleurs candidats d'une population et appliquer à la fois des mutations aléatoires et des croisements aléatoires.

En plus de l'utilisation d'un algorithme génétique pour l'exploration de l'espace de conception, il est aussi possible d'utiliser un modèle de régression linéaire pour éviter l'évaluation de la précision de chaque modèle sur le jeu de données de test. Une étude poussée sur des modèles récents tels que MobileNetV2 [13], EfficientNet [26], ResNet [19] et GoogleNet [27] a été réalisée. Pour plus de détails sur les méthodes et résultats obtenus, veuillez vous référer au chapitre 5 du manuscrit.

Comparaison avec l'état de l'art

Le même flot de compression est appliqué à plusieurs CNNs entraînés sur le jeu de données ImageNet[18]. Ils peuvent être classés en deux catégories : (1) les CNN légers (MobileNetV2 [13] et les différents EfficientNets [26]) qui sont optimisés pour fonctionner sur des systèmes aux ressources limitées, tandis que (2) les CNN lourds (GoogleNet [27], ResNet50V2 [24] et InceptionV3 [28]) se concentrent exclusivement sur la précision maximale. Conformément aux recommandations de MLPerf [7], chaque catégorie a des objectifs de qualité différents (c'est-à-dire 99% et 98% de précision FP32 pour les catégories lourde et légère, respectivement). La méthode WS proposée est toujours capable de compresser de manière significative la référence de base, en obtenant systématiquement un CR supérieur de plus de 5$\times$ tout en maintenant l'objectif de qualité de l'AL. Il est important de noter que la compression est réalisée sans nécessiter l'intervention d'un expert pour le réglage manuel des k_is et sans impliquer aucune étape de ré-entraînement, de réglage fin ou de calibration.

Ce chapitre se contente de présenter les différentes comparaisons réalisées et les résultats obtenus. Pour plus de détails, se référer au chapitre 6.

La première comparaison avec l'état de l'art est faite avec l'utilisation d'un algorithme NSGA-II pour la résolution du problème d'optimisation dans son intégralité, sans avoir recours à la technique ≪ diviser pour mieux régner ≫ présentée précédemment. L'approche ≪ diviser pour mieux régner ≫ parvient à surpasser les résultats de l'approche NSGA-II, particulièrement en fournissant plus de résultats dans la zone d'intérêt de l'espace de conception.

Pour comparer la méthode proposée avec d'autres approches de WS, nous avons sélectionné deux méthodes récentes qui impliquent le ré-entrainement complet du CNN. Sur le CNN GoogleNet[27] avec la méthode de compression de l'état de l'art WS Deep-K-means [3], nous sommes en mesure de surpasser leurs résultats, car ils ne présentent pas de CR supérieur à $3\times$ sous les contraintes de l'objectif de qualité MLPERF[7]. La comparaison avec DP-NET [29] montre néanmoins que cette technique plus récente et plus poussée est capable de surpasser la méthode développée dans cette thèse, au prix d'un ré-entrainement complet qui demande l'accès aux données d'entrainement et qui a un certain coût.

Nous comparons aussi les résultats obtenus avec plusieurs méthodes d'approximation qui ne sont pas basées sur le WS, mais qui sont appliqués après l'entrainement du CNN. La comparaison avec la méthode d'élagage PTP [30] et la méthode de quantification utilisée dans TFLITE [31] montre que notre méthode est capable de trouver des solutions dominantes en termes d'AL et CR. Tandis que la méthode PWLQ [32] est capable de fournir des solutions plus précises avec un découpage de la distribution des valeurs et une approche hétérogène qui traite chaque canal indépendamment, donnant une granularité plus fine sur le réglage et de meilleurs résultats.

Conclusion

Cette thèse propose une nouvelle méthode d'optimisation du partage des poids pour la compression CNN, qui peut explorer efficacement le vaste espace de conception pour produire un ensemble de solutions qui offrent des compromis très intéressants entre AL et CR. Ces solutions permettent d'obtenir une compression de plus de $5\times$ par rapport à l'empreinte mémoire initiale dans plusieurs CNN de vision par ordinateur de pointe sur le difficile jeu de données ImageNet sous des contraintes d'objectifs de qualité MLPerf [7]. Il est important de noter que ces résultats de compression sont obtenus tout en évitant l'étape de ré-entrainement prohibitive et coûteuse, qui est couramment utilisée dans les travaux antérieurs.

En ce qui concerne la production scientifique de ce doctorat, les premiers travaux utilisant l'exploration gloutonne sur LeNet/MNIST ont été publiés dans [8], une généralisation à la classe ImageNet CNNs avec l'utilisation d'une métrique proxy pour accélérer l'exploration a été publiée dans [9], une extension ajoutant une exploration multi-objectif a été publiée dans [6], les résultats de la nouvelle approche « diviser pour régner » sur MNIST ont été publiés dans [10], et la généralisation à ImageNet (présentée dans ce résumé) est publiée dans [11].

Contents

List of Figures

List of Tables

Acronyms

ADMM Alternating Direction Method of Multipliers. 22, 24

AI Artificial Intelligence. 2, 9

AL Accuracy Loss. 5, 6, 21–23, 27, 30–35, 37–47, 49–62, 64–67, 72, 74–77, 79, 80

ASIC Application Specific Integrated Circuits. 17

AxC Approximate Computing. 1, 4

CNN Convolutional Neural Network. 1–9, 12–45, 47–57, 59–61, 64–66, 69, 70, 72, 74–76, 78–84, O, Q

CPU Central Processing Unit. 16

CR Compression Rate. 6, 24, 26–28, 30–34, 36–38, 40–44, 46, 47, 49–51, 53, 54, 59, 61, 64–67, 72, 74–77, 79, 80, 83, 84

DL Deep Learning. 9, 16, 17

DNN Deep Neural Network. 11–13

DoE Design of Experiments. 57, 59–61, 64, 65, 67, 70, 83

DSE Design Space Exploration. 6, 28, 31, 49, 70

DSP Digital Signal Processing. 30

FLOPs FLOating-Point operations. 5, 15, 16, 30

FPGA Field-Programmable Gate Array. 17, 30

GPU Graphical Processing Unit. 15, 16, O

LUT Look-Up Table. 27

MAC Multiply-ACcumulate operation. 30

ML Machine Learning. 1–3, 9

PTQ Post-Training Quantization. 22, 23

QAT Quantization-Aware Training. 22, 23

QoR Quality of Result. 2–4, 6, 7, 17, 30

RS Row-Stationary. 83

WCSS Within-Cluster Sum of Squares. 26

WS Weight Sharing. 4–8, 18, 24, 25, 27–29, 31–33, 35–38, 41, 44–46, 67, 69, 74–77, 79, 81–84

Chapter 1

Introduction

Nowadays, Machine Learning (ML) algorithms allow for addressing a large variety of problems without being specifically programmed to do so. By leveraging a bio-inspired learning process, a well-designed and trained ML algorithm can usually achieve a better quality of prediction compared to conventional approaches. This complex training relying on trial-and-error allows an ML algorithm to approximate an oracle analytical equation which is not always available. For example in computer vision, it is practically impossible to find the analytical equation allowing to distinguish a face from another in a specific image. On the other hand, it is possible to train an ML model to do so, allowing to replace humans in a lot of repetitive tasks such as mail sorting [33], quality monitoring [34]–[36], or fraud detection [37], [38], and even as public speakers [39].

In the last decades, a lot of effort has been invested in improving ML algorithms to allow for bringing intelligence to systems. Taking as an example the computer vision task, ML algorithms outperform conventional approaches since 2012, with AlexNet [40] winning the very challenging ImageNet [18] Large Scale Visual Recognition Challenge (ILSVRC) [41] featuring a high-resolution image classification problem. And since 2014 ML algorithms are known to also outperform humans [41] in such tasks. Such quality of prediction is achieved by Convolutional Neural Network (CNN), which is a subset of ML algorithms relying on the matrix convolution operation to allow detection in images. CNN achieved outstanding accuracy in computer vision tasks, more details on CNN architecture can be found in Section 2.2. This accuracy comes at the cost of large computation and memory requirements that will be detailed in the next section.

In this chapter, we will present the need for Approximate Computing (AxC) for improving the efficiency of CNN inference in Section 1.2. The related challenges are then described in Section 1.3, with the presentation of the focus of this thesis. Finally, all the scientific contributions of this work are listed in Section 1.5 and an overview of this manuscript outline is given in Section 1.6.

1.1 From AI algorithms to CNN

During the last century, different biological processes have been mimicked by Artificial Intelligence (AI) algorithms, from the genetic algorithms, inspired by the complex process of genetic selection that drives the evolution of all the species, to the simple ant colony, inspired by the way the ants find the shortest path to the food. Among these numerous different AI algorithms families, some received particular interests from both the academic and the industrial world in the last decades. By self-improving through experiment, the same way humans and most species do, the AI algorithms lying in the large Machine Learning (ML) family are among them. ML algorithms have the advantage of being able to achieve tasks without being specifically programmed to do so, by using complex learning schemes to reprogram themselves to perform better next time they are exposed to the same problem. ML algorithms are using an internal set of rules called a model to mathematically approximate the behavior of the desired oracle function. The training of ML models relies on the trial-and-error process, by leveraging a reward mechanism to compute the required modification of the model for Quality of Result (QoR) improvement. This training process can be performed either with or without supervision. Supervised learning is the process of training a model with labeled data, the same way a teacher trains a student on problems with known answers, and eventually, the student will be able to solve a new problem, this concept is called generalization. Unsupervised learning, on the opposite, allows the model to work on its own on unlabelled data using a reward function to evaluate the model, more difficult to operate but with the advantage of not relying on labeled data that are costly, error-prone, and often feature human bias. Still, supervised learning, behavioral predictability, and ease of training iteration make it very competitive against unsupervised learning.

An example of an ML application is spam filtering, deciding whether or not an email is spam based on its provenance, recipients, object, and other metadata. Learning set containing emails samples that have been labeled as spam or not constitute the training data set. Each of the samples is fed into the ML model, which in turn computes the predicted label, and depending on the predicted label correctness, the reward function then tries to tune the parameters composing the model to improve the probability that the model gives the expected labels next time. Such a model can then be used on unseen data, such as the one composing the test set, allowing to measure the capability of the model to generalize his knowledge to new data. Further to email classification, there is a plethora of use cases of varying difficulty where ML models are used, ranging from security with fraud detection, business data analysis with churn rate measurement to industry with quality monitoring, even transportation with computer vision allowing for self-driving technologies. The model inputs can be both raw data or high-level features (for instance statistical aggregates of multiple input data samples) or other complex features that are task-dependent, such as the presence of a horizontal line in an image. The ML model is charged to interpret this data and construct useful responses.

ML models achieved outstanding performance in the computer vision field, with one of the first milestones being the first use of an ML model to recognize handwritten characters for mail application in 1998, with LeNet-5 [15]. This model confirmed that matrix convolution allows recognizing digits anywhere in an image, allowing for translation invariance in the input data. This model belongs to the neural networks category, featuring a neuron-like structure with layers of neurons units connected.

1.2 Approximate Computing for Efficient CNN Inference

As it has been shown by a recent study [42], the training of state-of-the-art natural language processing deep neural networks leads to the same level of CO^2 emission as five average American cars during their complete lifetime, including manufacturing. Natural language processing models are usually orders of magnitude larger than CNN used for computer vision, but this gives an example of how much ML workload can have an impact on the environment. Considering standard supervised learning application, the training of a CNN is usually a one-time action, once it is performed, the model can be frozen and duplicated into production environments where it is executed to infer prediction. In this thesis, the focus is made on accelerating the inference of CNN, because this step is performed a lot of times, and potentially in an embedded environment with computation and energy constraints that can prohibit the use of CNN. As an example of a limitation of the use of CNN in embedded devices, Yang & al. [43] stated that "Smartphones nowadays cannot even run object classification with AlexNet [40] in real-time for more than an hour.". As it can be observed in Figure 1.1 reporting the top-1 classification accuracy, defined as the number of correct predictions measured on a labeled validation dataset, and the number of weights composing the model, the high accuracy comes at the cost of higher computation and memory requirements, for both heavy CNN optimized for accuracy and lights CNN optimized for efficiency (more details on the light and heavy class can be found in Section 4.2).

Since 2012, CNNs are improving at a very fast rate, with a focus on the quality of prediction or Quality of Result (QoR), characterized by the accuracy of the CNN measured on a test dataset. But these outstanding achievements come at the cost of a computational complexity overhead, as an ML algorithm remains a generic tool approximating an oracle analytical function. This computational complexity can make certain ML algorithms very costly or even out of reach for power-constrained embedded devices. This is particularly true for CNNs such as AlexNet [40], it has been proven in [43] that it is not possible to execute such CNN for object classification on a recent smartphone in real-time for more than an hour. This is limiting the usage of the CNNs in edge embedded devices, that are often in direct contact with the environment and would benefit from this kind of tool. For example, any autonomous vehicle whether it is a

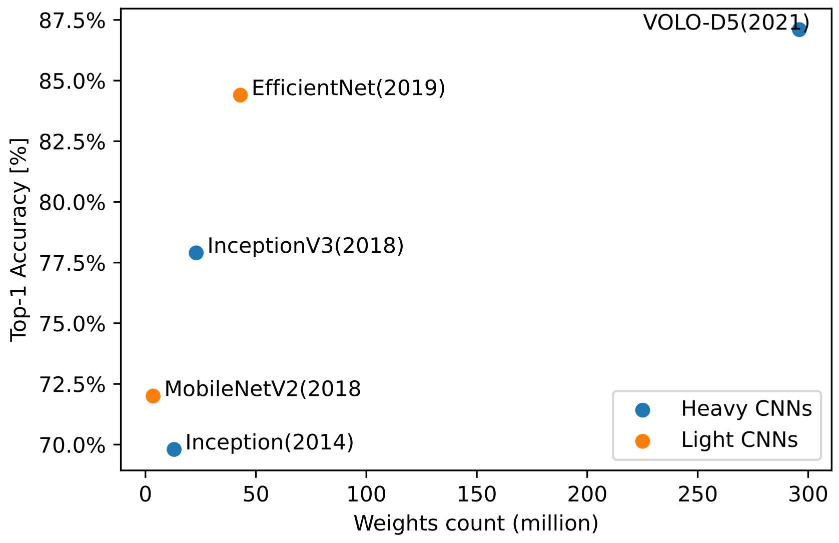


Figure 1.1: ImageNet[18] classification accuracy and number of weights of recent CNNs.

UAV, a car, or a boat, would take a lot of benefits from being able to understand its surroundings. Allowing for application such as low-cost autonomous bridges inspection with UAV or anything alike. Furthermore, the paradigm of sending sensed data to a cloud executing the CNN inference for computation has shown its limits when it comes to privacy and latency, adding even more interest in being able to execute the inference of CNN in edge devices. There is a recognized need to improve other metrics rather than only the accuracy of the CNNs, and since 2016 with the publication of MobileNet [44], a resource-efficient CNN intended to be executed in mobile devices, there is a trend for reducing the computational cost and memory footprint of CNNs inference.

Among the paradigms explored for improving the efficiency of CNNs, approximate computing alongside dedicated accelerators design is among the more promising. The approximate computing principle relaxes the need for fully-precise computation to allow for trade-offs between QoR and efficiency. This thesis explores the usage of the Approximate Computing (AxC) paradigm for CNN. With a particular focus on a specific technique, called Weight Sharing (WS), aiming at compressing the size of the CNN for enabling memory footprint reduction. The next section details the thesis focus and the related challenges.

1.3 Challenges in Approximating CNNs

Approximating CNN poses several challenges, this section is intended to define the scope of the research conducted in this thesis.

Measuring the computation requirements of a CNN is done with metrics like the memory footprint required to store the CNN weights values and the number of FLOating-Point operations (FLOPs) operations required to compute the inference of a single image. However, most of the energy cost during CNN inference comes from memory access as analyzed in [1]. It is thus important to reduce the memory footprint to improve energy efficiency. That is why the focus of this thesis is on compressing CNN weights.

When it comes down to compression, storing a sparse matrix has the disadvantage of requiring to store an extra index for the existing values, having the disadvantage of requiring large overhead compared to data size. On the opposite, quantization allows for buffering of the weights with a very small overhead and a very simple decoding step. Regarding the different quantization techniques, the Weight-Sharing WS family focus on quantizing the values of the weights on a finite number of shared values. Compared to other quantization techniques that usually quantize on levels that are computed from representation, WS has the advantage of quantizing on values that are computed from the baseline weights values, allowing for more freedom in looking for optimal quantization levels. Due to this flexibility WS is a good candidate for reaching high compression for a given trained CNN.

Although most if not all techniques of WS that can be found in the literature [2]–[5], [29], [45], [46] require the retraining of the network to recover the Accuracy Loss (AL) during the approximation process. Such post-processing has three main disadvantages:

- Computation intensity, as stated earlier, the retraining is a complex task, involving a lot of computation, raising the design costs, and limiting the exploration possibilities;
- Requires access to a large dataset, the growing concerns on privacy and security will certainly restrain the access to training datasets in important application domains;
- Require training method tuning, as the standard training method does not take into account any approximation, there is a need to tune the training method and the approximation operation must be differentiable for back-propagation. It also requires adapting the training framework, which can be costly and require expert programming skills.

On the other hand, there is a track record of post-training quantization [32], [47] that does not require any retraining, proving that a correct approximation can be made without retraining. The work conducted during this thesis focus on optimizing the compression of CNN without involving retraining.

The approach for selecting the number of shared values for existing methods is to manually tune the number of shared values [3], [5] to achieve the desired

Compression Rate (CR). The main problem with manual tuning is that it is costly and requires expert knowledge. Other methods [2], [29], [45] use a fixed number of shared values. However, the most widely adopted solution is to homogeneously vary the number of shared values (use the same number for each layer) to explore the trade-off between compression and precision [4], [48]–[50]. However, this approach is sub-optimal, as the weight sharing factor is not adapted to the differences in tolerance-to-approximation of each layer. There is a lack for automatic optimal selection of the number of shared values for achieving the best trade-off between the AL and the CR for a given baseline CNN. This is mainly due to the increasing complexity of modern CNN topologies, resulting in exponential search space size. The Design Space Exploration (DSE) required to find optimal WS approximations is thus prohibitively costly. In this thesis, there is a focus on using efficient DSE techniques to explore the trade-offs between the QoR and efficiency for a given CNN execution.

During this thesis, it is considered that the approximation is achieved to improve the efficiency of the inference for a given trained CNN on computer vision applications. The target hardware is a dedicated accelerator that could exploit the advantages of WS like the EIE [51], with a data path optimized for retrieving the shared values efficiently. However, any other hardware could benefit from an improvement in compression, and thus in memory throughput, at the cost of a small shared value read overhead (i.e., one extra level of indirection for weight look-up) that could potentially add some latency to the execution.

1.4 Thesis Statement

The goal of this thesis is to optimize the number of shared values during the application of WS without any retraining step. This would allow for significant compression of a baseline CNN with accuracy loss complying with MLPerf [7] constraints. Multiple heuristics are explored and compared with a focus on keeping the exploration time reasonable. The thesis statement is the following:

The trade-off between QoR and efficiency for compression of trained CNNs can be explored by optimizing the number of shared values of the WS and result in significant compression without any retraining.

1.5 Contributions

The main contribution of this thesis is the proposal of different exploration methods allowing to optimize the number of shared values to compress a baseline trained CNN. The resulting approximated CNNs are complying with the MLPerf [7] quality targets and are obtained without any retraining step. The following list gives more details on contributions:

- A detailed investigation of different WS granularities impact on the trade-off between efficiency and QoR is conducted with the analysis of three potential candidates. Sharing the values of the weights for a whole layer, for a convolutional filter (channel), or a 2D kernel. We have identified the layer as the optimal sharing granularity offering the best trade-offs.

- Analysing the sensitivity to the approximation of different layers of a CNN shows that there are very different resilience profiles. Meaning that there is a need to tune the approximation accordingly. We introduce the use of a very simple greedy heuristic to optimize the number of shared values for each unit. With the investigation of different proxy metrics to accelerate the greedy exploration. The acceleration allows for multi-objective optimization with a trick to keep the number of experiments in a reasonable amount. Altogether, a compression of more than 4× compared to the baseline is achieved with QoR complying with the MLPerf [7] quality constraints for two recent CNN on the ImageNet [18] dataset. Proving that it is possible to efficiently compress CNN without requiring any retraining when an appropriate level of approximation is found for each layer. These results are the object of the following incremental publications in peer-reviewed conferences [8], [9] and a peer-reviewed journal [6].

- Further improving the number of shared values of each layer requires to move from the local optimization of the greedy algorithm to a more global heuristic that targets metrics of the whole approximated CNN instead of the local users in greedy. We thus propose a method to reduce the search space complexity, enabling an efficient exploration using a meta-heuristic approach. The use of regression to accelerate the meta-heuristic algorithm, with multiple designs of experiments technique to allow for the optimal regression model training is also investigated. Altogether, the proposed divide & conquer technique allows for more than 5× compression on state-of-the-art CNNs with complying with the MLPerf [7] quality target. An extensive investigation of the application to several CNN is also conducted. These results are the object of the following incremental publication in peer-reviewed conferences [10], [11], the proposed CNN compression framework has also been made publicly available on Github [52].

- A comprehensive comparison of the achieved results with the state-of-the-art WS techniques is also conducted. As well as a comparison with others pruning and quantization methods.

1.6 Manuscript Outline

The current manuscript is structured as follows: Chapters 2, and 3 give background and details about approximation in CNN and state-of-the-art compression methods. Chapter 4-6 present the scientific thesis contributions, and Chapter 7concludes this manuscript and gives new perspectives. In Chapter 2 required background information is provided for understanding the rest of the manuscript, with details on CNN, metrics, workloads, and optimization tools.

Chapter3 presents the state-of-the-art of approximation techniques for CNN compression, with a taxonomy of the different approaches, and more details on the WS challenges. Chapter 4 presents the first work conducted with a study of the WS application granularity, as well as the use of a greedy heuristic and the use of proxy metrics to allow for multi-objective exploration in a reasonable time. Chapter 5 presents a divide & conquer heuristic approach to the optimization problem, allowing a combination combine of the power of meta-heuristic optimization algorithm with expert knowledge on WS compression. Chapter 6 gives a comprehensive comparison of the proposed methods with state-of-the-art compression techniques featuring different approaches. The conclusion of this manuscript with perspective on possible further research paths is presented in Chapter 7.

Chapter 2

Background

This chapter presents the background required for understanding the thesis work, it is structured as follows. The first Section 2.1 provides some background knowledge on neural networks, completing the introduction given in Chapter 1 and presents general ML training processes as well as the need for convolutional layers for computer vision. The second Section 2.2 presents some history of CNNs usage and introduces the different types of layers composing a CNN. Section 2.3 gives details on the DL workload and introduces the need for acceleration, as well as the different types of hardware accelerators and the techniques for model approximation.

2.1 Neural Networks

From the large field of AI, one particular set of techniques is very popular nowadays, neural networks, and particularly deep neural networks. This set of techniques belongs to the ML family of AI, focusing on giving a computer the ability to learn without being specifically programmed. As stated in [53] "A computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program.". This section is intended to give some background on neural networks structure and training process to allow for a better understanding of the remaining thesis.

Neural networks try to mimic the way a network of biological neurons, like the human brain, works. Each neuron is connected to the others, receiving input stimulation from previous neurons through dendrites and giving output stimulation to following neurons through axons, as depicted in Figure 2.1. Each of the connections of a biological neuron has a different strength, or weight, allowing the neuron to apply a function to the inputs to compute the outputs. The chemical connections between axons and dendrites required a certain level of stimulation to be triggered resulting in a non-linear response to stimuli. In artificial neural networks, this function is approximated as follows: the output stimulation is equal to the sum of each input of the neuron multiplied by a

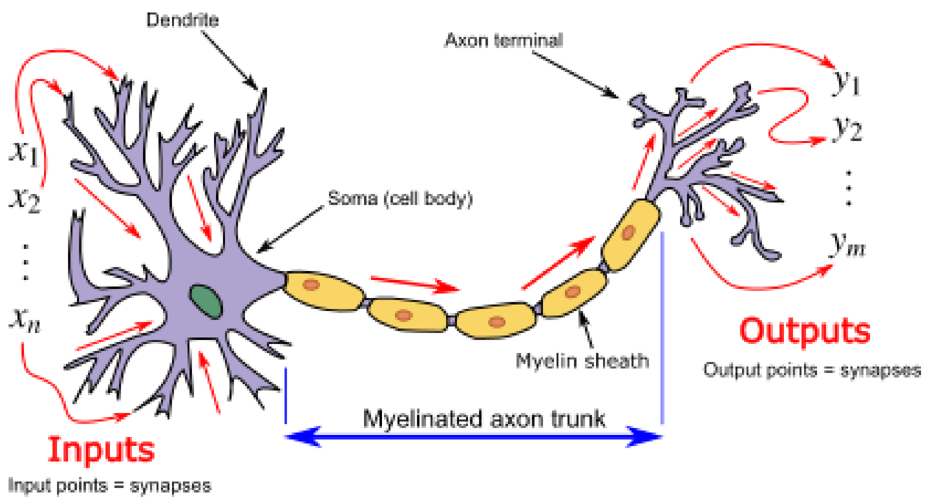


Figure 2.1: Biological neuron structure, picture from Wikipedia.

specific weight with an additional bias allowing to represent affine functions. The equivalent transfer function is the following

$$F(X, W, b) = f(\sum_{i=1}^{n} x_i w_i + b) \tag{2.1}$$

, with X the vector composed of the input stimulation, W the vector of the corresponding weights, b the vector of corresponding bias, and f a non-linear activation function used to mimic the triggering behavior of the biological neuron, some example of activation functions will be given in the following Section 2.2.

Neural networks are composed of one or more of the previously described artificial neurons, which will be called neurons for convenience. The same way a brain is composed of interconnected biological neurons, a neural network is composed of connected neurons. A simple neural network is shown in Figure 2.2, it is composed of an input layer, directly connected to the input features or data with the number of neurons corresponding to the input data size, and an output layer, connected to the prediction target, whether it is a regression (numerical), classification (categorical) or other kinds of problem. Between these two layers lies a hidden layer, with an arbitrary number of neurons. Each neuron of a layer is connected to each neuron of the previous layer, with a varying weight value, representing the strength of the connection, represented as the width of the arrow in the Figure 2.2.

Such a simple neural network structure is called a classifier, it is usually not directly connected to raw data sources, but to a previous stage called the feature extraction. The feature extraction is charged with processing the raw data to allow the highlight of high-level features, used by the classifier to make predictions. An example of a basic feature extraction process would be to extract

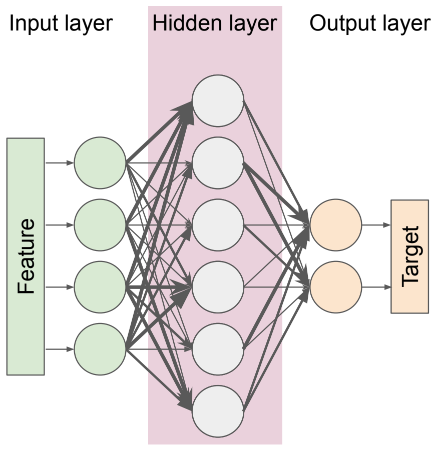


Figure 2.2: Simple neural network.

the mean of the raw values from an IMU on a given time window. This kind of feature extraction is designed by the scientist creating the system, according to his own experience of the most important features that can be extracted and used to process the raw data efficiently. This feature extraction allows the classifier to have access to refined information and perform well with a simple structure, but it also implies that there is a large bias from the scientist, some of the features might not be necessary for the model to perform well, and worst, some required features might not be present.

To this extent, it is also possible to do little to no preprocessing and feed the neural network with raw data, this task is more complex and will require a more complex classifier, instead of scaling vertically the number of neurons of the hidden layer, it is possible to scale the network horizontally, by adding multiple hidden layers between the input and the output layers. This type of extended neural network with more than one hidden layer is called a Deep Neural Network (DNN). Figure 2.3 depicts the structure of a DNN with multiple hidden layers, and the input directly connected to raw data instead of features, connection width is constant to ease the reading.

Training a DNN is done by using an iterative weights and bias update process, aiming at reducing the error between the actual behavior of the DNN and the desired behavior. The difference between supervised and unsupervised learning is the way the behavior of the model is evaluated, in supervised learning, labels from the dataset are used, whereas in unsupervised learning a reward function based on a real or a simulated acquisition environment is used, for the sake of simplicity, only supervised learning will be considered in this section. Compared to the inference process using the model to predict target from input data described until now, called the forward pass, the training involves an iterative computation of the loss from the predicted targets and the label of a given training sample. This loss can be computed using a different mathe-

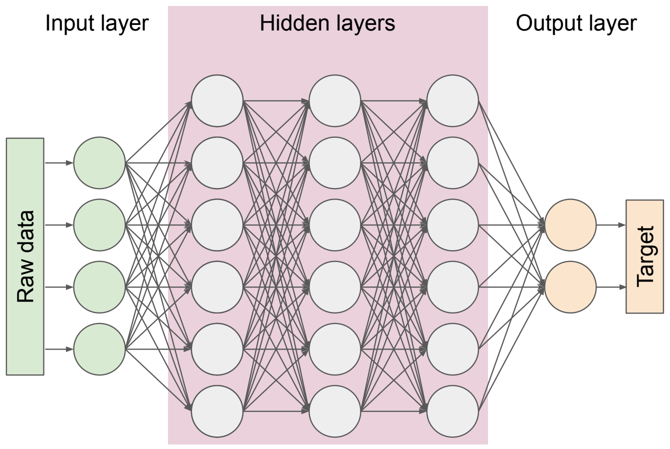


Figure 2.3: Deep neural network.

matical functions, but the principle is always to measure the difference between the current behavior and the desired one. This loss is then propagated through the network in the backward direction, from the output to the input, using the partial derivative of the graph operation. Then the final step of updating the values of the weights is performed, for each weight, the partial loss is considered as well as an update strategy called the optimizer.Figure 2.4 shows a conceptual view of the process. This process of computing the forward pass, the loss with the label, the backward propagation of the error, and the update of the values of the weights is performed multiple times using samples from the dataset. In order to accelerate the training, samples are often batched to allow for using faster matrix multiplication and reduce the number of required iterations. It is important to note that due to the computation of the loss, the backpropagation of the error, and the weight update, training a model is a much more computation-intensive task than using the model to predict a label for a specified input data, the latter is called inference. More details on DNN training can be found in the excellent survey [1].

2.2 Convolutional Neural Network (CNN) Structure

Considering the computer vision field, focusing on image recognition, the input data is structured as channels of 2D matrices, in the case of RGB images, the data has 3 layers. feeding a conventional DNN with this data means that there is a need for 3× the number of neurons in the input layer than the size of one of the 2D matrices. Allowing each layer of each pixel to be mapped to an input neuron. Training a DNN to recognize very simple data like handwritten digits perfectly centered in small images such as the samples composing the MNIST [16] dataset

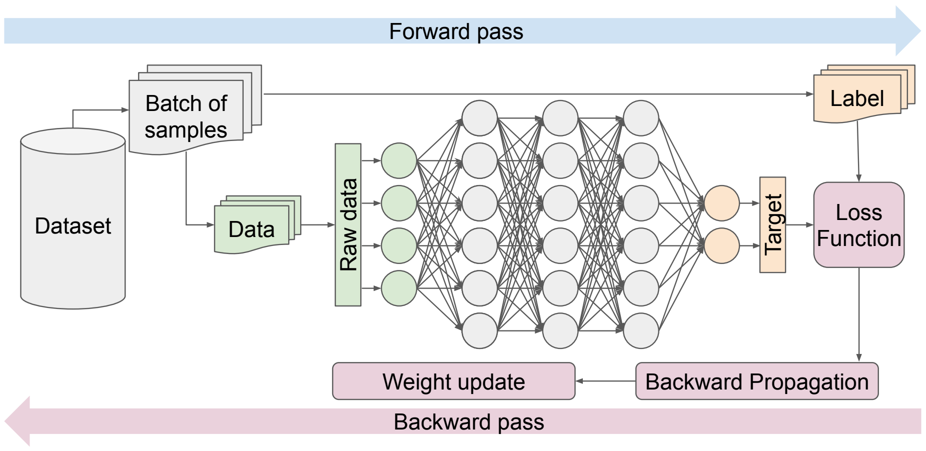


Figure 2.4: Deep neural network training process.

is possible. But when it comes to more complex images such as real-life pictures taken from a camera embedded in a vehicle, the chances are great that the subject will not be centered in the picture, requiring to process the whole high-definition image to find it. This is not possible with conventional DNN. Instead, CNNs propose to rely on the matrix convolution operation to allow the detection to be translation invariant. The data representation before and after the layer is called a feature map or activation. A convolutional layer, composing a CNN, performs the matrix convolution operation of the input feature map with kernels. Kernels are the matrices of weights composing the neurons of the convolutional layer, they have the same number of channels as the number of channels of the input feature map, and the output feature map has the same number of channels as the number of kernels in the layer. Figure 2.5 show a simplified convolutional layer, with a single layer, applying the matrix convolution to an input feature map composed of 3 channels depicting the Red, Green, and Blue (RGB) of the input image.

A CNN is thus composed of multiple layers containing kernels with trained weights. The weights values are real number, usually represented using standard 32-bit floating-point as the training process often require high precision for allowing low-magnitude changes, but we will see that there are other more efficient representations. It is also very common for CNN to feature other types of layers, like:

- fully-connected layers, which are basic neural network layers with non-convolutional neurons, are usually used to map the final convolutional layers to the output classes;
- pooling layers, using a kernel sliding on input channels almost the same way does the convolution, but computing an aggregation like mean or max instead of a matrix product, used to reduce the size of the data but keep

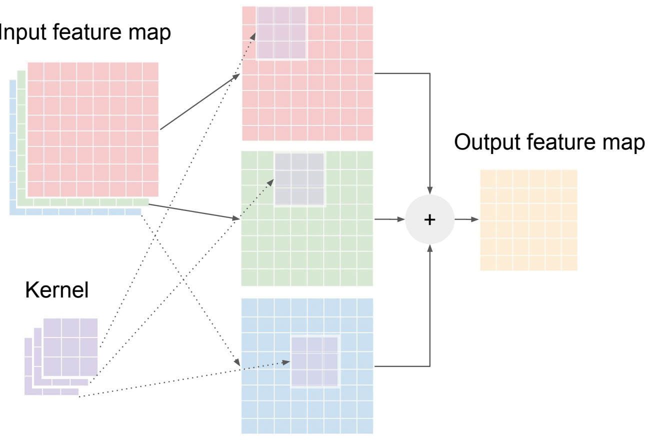


Figure 2.5: Convolutional layer simplified with a single kernel.

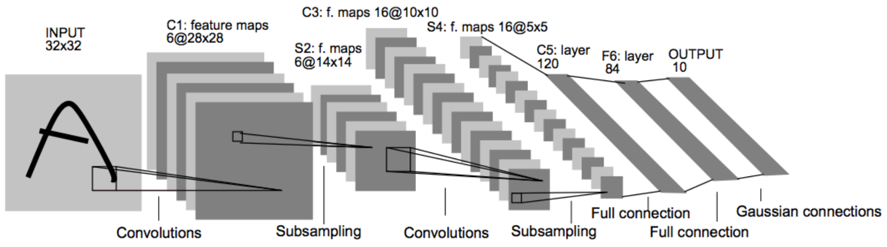


Figure 2.6: The LeNet-5 [15] Architecture.

the information;

- batch-normalization layers, normalizing input channels by using trained coefficients;
- non-linear activation functions are used to allow the model to learn for non-linear behavior.

Observing the architecture of LeNet-5 [15] in Figure 2.6, one can see that the input is a 32x32 black and white (single channel) image, then comes a convolutional layer, followed by a subsampling (pooling) layer. This structure is repeated 2 times and is followed by 3 fully-connected layers, the last one giving the classification output with 10 classes. It is important to note that despite they are not marked in the historical figure, each convolutional and fully-connected layer includes a non-linear activation function, the hyperbolic tangent.

It is only in the early 2010s that CNN proved its outstanding performance to the industrial world. With AlexNet [40] being the first CNN to outperform manually-designed models on the task of classifying high-resolution images from the ImageNet [18] dataset during the ILSVRC [41] contest in 2012. This take-off is known to have been allowed by three major factors:

- The availability of large (million samples) labeled dataset for training complex model;
- The advances in computation power of units such as Graphical Processing Unit (GPU) able to accelerate matrix computation and allowed the training to be executed in reasonable times (within weeks, not years);
- The improvement of the training technique, such as the use of overlapping pooling which allows for keeping more details during the inference rectified linear unit activation that is faster to calculate than the ones previously used, and finally, dropout [54] that allows for avoiding over-fitting the training dataset.

More recently, the training of CNN has been facilitated by the availability of open-source software frameworks like Tensorflow [55] or Pytorch [56], which allow for quick prototyping and design.

2.3 Accelerating Deep Learning Workloads

The original computational cost of training AlexNet [40] for the required 90 epochs (i.e., the number of times the full training dataset is presented to the network during the training) is so high that it has required the use of two NVIDIA Geforce GTX 580 GPUs during 6 days. Considering standard supervised learning application, the training of a CNN is usually a one-time action, once it is performed, the model can be frozen and duplicated into production environments where it is executed to infer prediction. In this thesis, the focus is made on reducing the memory footprint of the inference of CNNs, because this step is performed a lot of times, and potentially in an embedded environment with computation and energy constraints that can prohibit the use of CNN. As an example of a limitation of the use of CNN in embedded devices, Yang & al. [43] stated that "Smartphones nowadays cannot even run object classification with AlexNet [40] in real-time for more than an hour.".

To gauge the complexity of current CNNs, there is a need for a set of metrics that allow for a fair comparison between models. In this study, the metrics used are (1) the model accuracy over a validation dataset, (2) the total number of weights in the model, and (3) the number of FLOating-Point operations (FLOPs) necessary to carry out one complete inference. The accuracy is measured in terms of the frequently used top-1 and top-5 percentages (the proportion of correct predictions on the labeled validation dataset and the probability that the correct result is among the top five predictions). The number of weights allows estimating the total memory storage requirements for the model, whereas the FLOP count hints at the required computing power needed to execute the model at a certain frequency.

MLPerf category	Model Name	Year	Top-1 / Top-5 Accuracy [%]	Weights Count	FLOPs
Heavy	AlexNet [40]	2012	57.2 / 84.7	62M	1.5B
Heavy	GoogleNet [27]	2014	69.8 / 93.3	6.4M	2.0B
Heavy	ResNet50V2 [24]	2016	76.2 / 93.0	26M	4.1B
Light	MobileNetV2 [13]	2018	72.0 / 90.6	3.5M	0.3B
Light	EfficientNetB1 [26]	2019	79.1 / 94.4	7.8M	0.7B

Table 2.1: Recent evolution of CNNs for image classification on the ImageNet [18] dataset.

The Table 2.1 shows a comparison using these metrics on some popular DNNs for image classification on the ImageNet [18] dataset (adapted from the PapersWithCode [57] web resource). For a long time, the only metric of interest was the network accuracy, resulting in models that were costly to train and operate like AlexNet [40], GoogleNet [27], and ResNet50V2 [24], requiring up to 4 billion FLOPs for the inference of a single image.

The cost of training and inference became so large at one point that there is now an open engineering contest called MLPerf [7] that benchmarks DL workload and fosters innovation in the field. Thus, there is an increasing interest for faster, lighter, and overall more efficient models that are compatible with edge device resource constraints and operate more efficiently in the cloud. The last two columns in the Table 2.1 reflect this, with newer network models achieving competitive accuracy with less memory and a smaller FLOPs count, like MobileNetV2 [13], and EfficientNet [26]. The MLPerf [7] contest has selected two CNNs as representative of the "Heavyweight" workload featuring high accuracy at the cost of computation intensive and the "Lightweight" workload, with a faster model featuring lower accuracy. Depending on the intended use of CNN models, they can be found in different environments with various computing power and energy consumption characteristics. At one end of the spectrum lies edge devices, characterized by low-power and limited computational capabilities, while at the other end lies power-hungry cloud devices with a high-performance computing profile.

As stated earlier, one of the factors that allowed the take-off of CNN usage in a real-world complex application, was the advances in computing power, particularly in GPUs to accelerate the training. This move from latency-optimized computing platforms like Central Processing Unit (CPU) to throughput-optimized computing platforms like GPU is due to the highly parallel nature of the computation in DL workload. According to [58], convolutional layer operation accounts for more than 90% of the inference runtime of AlexNet [40], that is why CNN could largely benefit from computing parallelism for the matrix product. This very simple observation has paved the way for the creation of more exotic hardware, dedicated accelerators based either on reconfigurable hardware such as Field-Programmable Gate Array (FPGA) or directly on dedicated silicon such as the Google TPU[59] belonging in the Application Specific Integrated Circuits (ASIC) family.

Dedicated accelerators can be implemented either as plain or as a tensor processing unit. The plain implementation relies on unrolling for maximizing the combinatorial capabilities and needs each neuron of each layer to be physically implemented, allowing for running each in parallel with very low latency, at the cost of large resource usage as well as very limited flexibility in hardware design reuse for a different model, examples are [60]–[64] for FPGA, there is little to no advantage of using ASIC in this case because of the high design and manufacturing cost does not work with the absence of flexibility. Compute unit implementations are usually composed of memory and processing-element tiles, allowing for parallel execution of the matrix computation as well as maximizing the reuse of elements to optimize the dataflow, examples are [51], [65]–[67] for FPGA and [59], [68] for ASIC. Some accelerator designs like EyerissV2 [67] are even optimized for sparse CNN, which is obtained by pruning the least important part of CNN to enable compression and acceleration.

Such techniques lie in the approximate computing paradigm, which has emerged from the idea that by relaxing the need for full precision results, there are opportunities for improvement. That means that it could be interesting to trade some QoR for gains in terms of efficiency. In the case of sparsity, pruning the least important weights allows for benefiting in compression and even for acceleration in the specific case of removing structures like neurons, because it allows for skipping computation, this technique is called pruning. Pruning is not the only approximation technique for accelerating DL workloads, since CNN has been identified as resilient to approximation, a plethora of approximation techniques have emerged in the scientific literature. More details on different approaches are provided in the literature review found in Chapter 3.

There have been numerous different approaches to approximating CNNs, an overview of these techniques is given in Chapter 3.

2.4 Conclusions

This chapter has presented the most important parts of the required background for understanding the work conducted for this thesis. From neural networks composition and training process, followed by CNNs structure and composing layers to the need for acceleration in deep learning workloads.

Chapter 3

CNN Compression and Approximation Objectives

This chapter is dedicated to presenting a global view on the different approximation techniques for CNN compression with a focus on the state-of-the-art WS techniques. An overview of the different approximation techniques used for providing gains in performance and energy during the CNN inference is presented at the coarse grain in Section 3.1, with a detailed taxonomy. In Section 3.2, the WS techniques literature is studied with an emphasis on the difference between existing techniques and this thesis.

3.1 Taxonomy of Approximations Techniques

The taxonomy of the different approximation techniques to provide gains in performance and energy during the inference can be represented by three different families:

- The first family, **structure refinement**, aim at modifying the CNN computational structure of the CNN;
- The second family, **data refinement**, modify weights and activations values and representation, keeping most of the computational structure of the CNN intact;
- The last family, **operator refinement**, modifies the computations operator behavior to use approximation.

Refining the structure of a CNN can be done using different techniques, like architecture search, that can be performed either with automatic tools like neural architecture search [69]–[72], focusing on using optimization techniques to refine architecture, or with manual compact architecture search [13], [20], [44], [73] looking for computation or resource-efficient layer structure, and knowledge distillation [74], [75] using a large layer to help the training of smaller

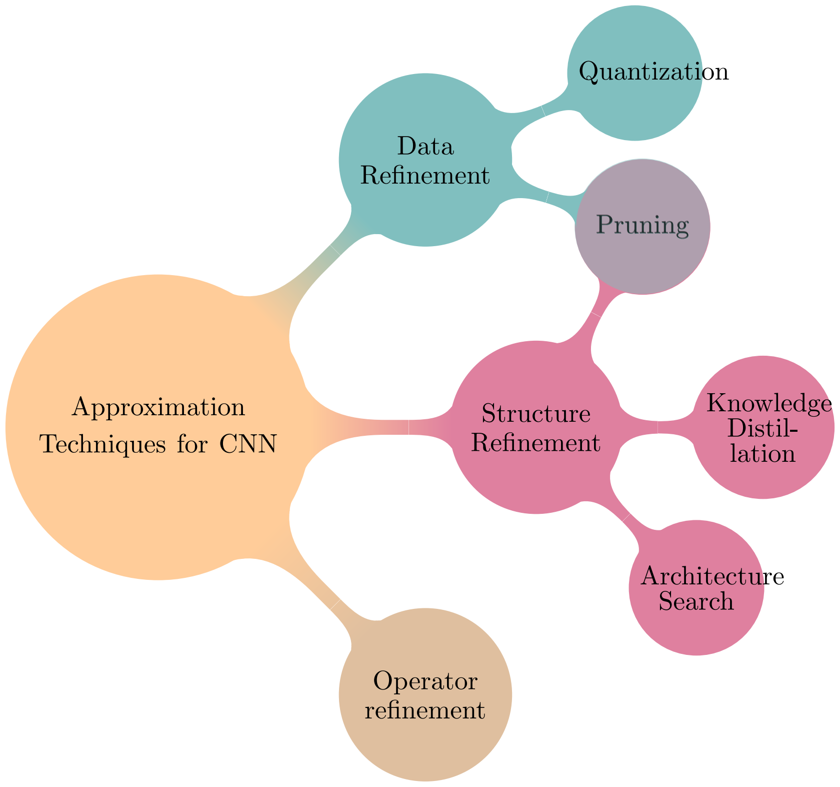


Figure 3.1: Taxonomy of most used approximation techniques for CNN compression and acceleration.

ones. Another example of structure refinement is a growing portion of the pruning techniques, aiming at removing the least important connection of a CNN. Whereas sparse pruning focuses on removing discrete weights and belongs to the data-refinement family that is presented later, structured pruning focus on removing structures such as kernels [76], [77], targeting the optimization of the computational structure, and thus, belonging to the structure refinement family.

The second family is focusing on the approximation of the data values and type with data refinement techniques. Notable examples are pruning [78], [79] aiming at removing the least important weights of the CNN, quantization [80], [81] changing the representation of the weights, and/or activations to more efficient ones. The large availability of pre-trained CNNs that can directly be transferred to another application with minor changes motivates the need for data refinement because it does not incurs large changes like the need to modify the computational structure or the hardware. For these reasons this thesis focus on data-refinement techniques and the remaining of this section gives more details on the application of such data refinement techniques.

Another approach for approximation is targeting the computational operators using operator refinement techniques. Most of the energy and the delay that is experienced during the inference of CNNs is usually coming from data movement from/to main memory (DRAM) [1]. In traditional Von-Neuman-based inference platforms, there is little room for improvement in using approximate operators to decrease the computation intensity. But still, with the recent advances in FPGA-based accelerator, which could easily benefit from approximation operators [82], [83], the computation begins to be an interesting opportunity for the future. Due to its highest energy cost and utilization in CNNs inference, multiplication is the de-facto target for approximation, this is why most methods focus on using approximate multipliers [82]–[87].

3.1.1 CNN Pruning

CNNs tend to be more complex as their top-1 accuracy improves and this complexity usually carries with it the fact that the CNN is over-parameterized. On the other hand, it has been argued for a long time [88] that structure is more important than density in CNNs, with sparse CNNs having the ability to generalize up to as well as their dense counterparts. Removing CNN weights has the direct effect of reducing the size of the CNN, but it can also be used for speeding up the inference process by reducing the number of computations. Depending on the objective, different parts of the CNN can be more interesting to prune than others. For instance, fully connected layers usually concentrate most of the CNN weights in a CNN and should be targeted for high compression. Convolutional layers, however, contain fewer weights but account for most of the computations. Since they generate the majority of data movement in the CNN, they should be targeted when CNN performance and energy efficiency are important. Pruning methods can be classified by (1) the granularity of the pruning, (2) the application process, (3) the homogeneity of the application, and finally, (4) the saliency determination approach. All these criteria are discussed in the following paragraphs.

Depending on the pruning objective (compression or performance), one can choose to focus on weight removal at various sparsity levels. The lowest pruning level is at the weight level or sparse [2], [88]. Although this generally results in the lowest accuracy loss, it does not systematically offer latency or energy improvements because sparse tensor computations are quite difficult to accelerate. Its main purpose is therefore to compress the CNN in memory. To accelerate computations, a regular sparsity pattern is usually required. This is called structured pruning and aims at removing spatially close groups of weights so that CNN inference can be simplified. it can target different regular structures, like groups of weights [89], hardware-oriented structures [90], 2D convolution kernels [91], simplifying the computational graph. The filters producing these channels in the previous layer can also be removed [92], [93]. Figure 3.2 give a conceptual view of some of the most popular pruning sparsity levels or structures. Similar to pruning weights, activations can also be pruned dynamically during the inference. This process is called dynamic sparsity and is used in many accelerators to avoid zero or near zero computations [94], [95].

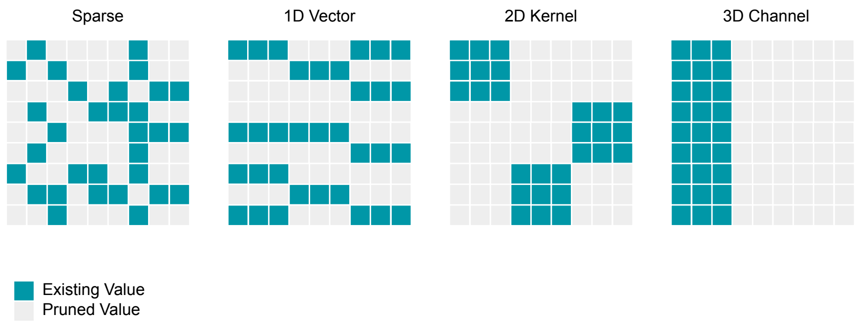


Figure 3.2: Conceptual view of some of the different pruning structures.

Most of the pruning techniques require fine-tuning of the approximated CNN for recovering the AL. Another possibility for the training is to encourage weights to group around zero using regularization. The closer weights are to zero, the less accuracy loss will be induced by removing them [96], [97]. Different regularization terms include LASSO [98], ℓ_1 regularization [99], ℓ_2 regularization [100]. In [101], feature map channels are gradually zeroed during training using a dynamic regularization factor, allowing safe removal of corresponding filters without a significant drop in accuracy. Another recent approach to optimize pruning is through architecture search. Recently, the idea that a classic network contains sub-networks that, trained from scratch, can perform as well as the original network but with fewer parameters and computation, was introduced in [102] and further explored in [103]. The issue is that none of these early studies provided a method for finding an efficient smaller architecture without doing full model training beforehand. In [104] it was thus proposed to use a bee colony exploration algorithm to find an appropriate CNN pruning scheme. It is also possible to reduce the fine-tuning cost by using an external network trained to predict weights of a certain network structure, facilitating a fast exploration of various possible architectures [105]. Finally, some outliers propose to apply the pruning without any fine-tuning, like [30].

It has been shown that some parts of CNN are more resilient to approximation than others. As such, pruning each layer at the same rate is not very efficient for accuracy. But at the same time, choosing the optimal sparsity level for the whole CNN is not an easy task. For example, [106] proposes to heuristically optimize the pruning ratio of each layer using reinforcement learning.

Removing part(s) of a CNN usually requires knowing which regions are least important for ensuring the lowest approximated CNN AL. This process is called saliency determination and there are different ways to achieve it. The simplest is the use of heuristics like the magnitude of the weights [2], [88], [107]–[110], or the ℓ_1/ℓ_2-norm for structures [111]–[113]. More complex methods involve a Taylor expansion criterion [91], weights gradient [114] or LASSO selection

[93]. It is also possible to state the saliency as an optimization problem, some examples [92] are using activations map correlation, or [43] looking at energy efficiency, more recently, the ADMM approach has been extensively used [115], [116] to determine the saliency. For more details on pruning, please refer to our book chapter [12].

3.1.2 Quantization

A full precision CNN usually relies on 32-bit floating-point values for representing weights and activations. For standard back-propagation-based training, using high precision weights makes sense since the gradient update rule generally modifies these weights by a small factor of the corresponding gradient terms. While full precision CNNs offer excellent result quality, they can generally be compressed and accelerated using lower precision arithmetic with minimal or no loss in terms of accuracy. Methods for addressing data quantization in CNNs are varied, ranging from simple binary and ternary CNNs to larger fixed-point and custom floating-point formats. This section gives an overview of the main ones. Analysis of existing approaches relies on various aspects, such as (1) what parts of the CNN are being quantized, (2) the approximation technique application process, (3) homogeneity/heterogeneity of the representation formats used inside the layers, (4) the type of representations being used.

The most obvious quantization targets are the weights of the convolutional and fully-connected layers. Reducing the number of bits used to represent them primarily brings a memory footprint reduction for on-device storage of the CNN. Latency improvements are potentially achievable with binary, ternary, and bit-shift (i.e. power of two values) quantized weights and/or activations [117]–[119]. More generally, if faster execution times are to be obtained, activations function inputs and outputs also need to be quantized. An example is [120], which proposes an efficient 8-bit integer quantization scheme for both weights and activations. Although it is out of the scope of this thesis, one can additionally quantize the weights and activations used during backpropagation [80], [121].

There are two established ways quantization can be performed for efficient inference and a third, emerging method. The first among the established approaches is Quantization-Aware Training (QAT). The idea is to use a CNN weights update procedure for several training epochs to adjust the quantized weights such that the accuracy is hopefully kept the same or is at worst minimally degraded. Much research has focused on such fine-tuning methods [80], [81], [119], [120], [122], [123], mainly because they achieve good results. Whereas the training is a powerful approach for approximated CNN AL recovery, it is not always applicable in real-world scenarios since it is costly, time-consuming and generally requires a full-size training dataset. This can be a problem when the data is proprietary, privacy and regulatory issues are in effect, for example, medical data that cannot be uploaded to the cloud for remote processing, or when using pre-trained off-the-shelf CNN for which data is no longer available. As such, there has been a push for fast Post-Training Quantization (PTQ) methods without any fine-tuning. It has been observed that for down to 8-bit word lengths, PTQ results are close to full precision ones for several CNNs [32], [47], [124], but it becomes significantly more difficult to maintain the accuracy when targeting lower precision formats. Work focused on PTQ includes [124]–[128]. A possible issue with QAT and PTQ methods is that both generate CNN that are sensitive to how quantization is carried out. As such, there has been recent work [129], [130] on methods for robust quantization that provide intrinsic tolerance of the CNN to a large family of quantization formats and policies by directly specifying it in the training loss function. Such approaches are interesting for battery-powered edge devices, where depending on the state of charge, a CNN capable of operating effectively at various quantization levels would be highly beneficial.

Initially, quantization approaches were homogeneous, with a single word length being used for the entire approximated CNN. This is the case for early works on binary [131] and ternary [132] weight CNNs, for instance. Such approaches can suffer from significant accuracy loss since different layers tend to have different sensitivities to quantization levels/noise. Subsequent work has focused more on a heterogeneous, layer-wise optimization of the quantization format [125], [133]–[137]. There have been various metrics proposed to estimate the overall effect of a quantization format inside a layer on the AL , like a signal-to-quantization-noise-ratio [138], adversarial-noise [133], [138], loss gradient [139], Hessian-based [136], [137], [140], weights entropy [141] or finally Kullback-Leibler divergence [125], [135], which is a core component for fine-tuning low precision integer weights in NVIDIA's TensorRT inference acceleration library.

On a different granularity level, [142] proposes looking at the distribution of weight values over the entire CNN to aggressively quantize weights in dense regions and more gently those in sparse ones. Compared to `float32` baselines, such an approach can achieve under 1% accuracy loss for large CNN with a 4-bit format in the dense areas and a 16 bit one for the sparse regions with $< 1\%$ of the values of the weights.

Regarding the quantization format, it is possible to denote two categories. The first category, regular quantization, groups the representation where the values are linked to a hardware state, whereas the second, weight-sharing relies on a codebook or lookup table for storing the values shared between the weights or activations.

Regular Quantization There are very different regular quantization formats, such as binary [117], [119], [131], [143] where values are bounded to $\{0, 1\}$, or the extension to ternary [118], [144] where values are bounded to $\{\pm 1, 0\}$. Those extremely low-bit quantization formats are susceptible to non-negligible AL, thus, the use of more complex representation like fixed-point [80], [124], [125], [134]–[138], [145] that features a limited dynamic range represented using an integer and a regular value. To overcome the dynamic-range limitation, it is also possible to use reduced floating-point [146]–[149], block-floating-point sharing the exponent and storing the mantissa [150], [151], or even the brain-floating-

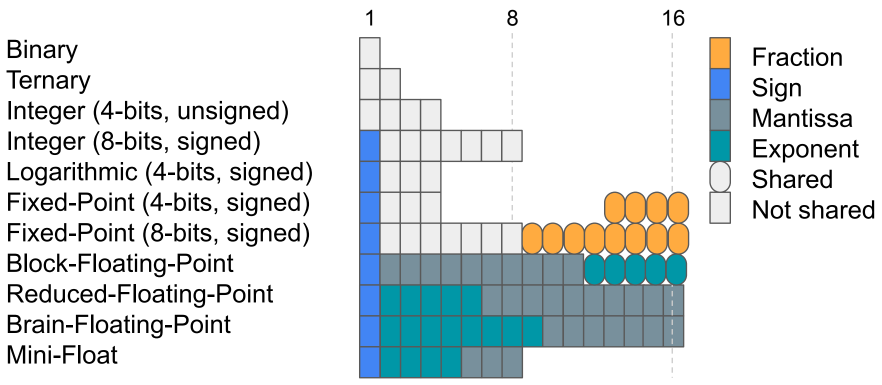
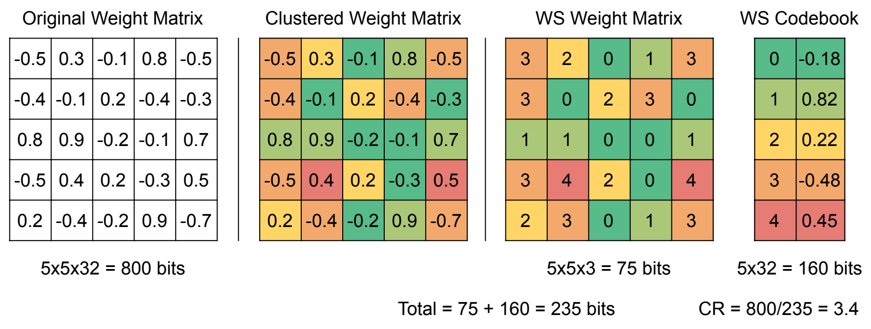


Figure 3.3: Conceptual view of the different regular quantization formats.

point format [152] dedicated to CNN training. Another regular approach is the logarithmic representation of the values [153] allowing multiplication to be replaced by inexpensive bit-shifts. Figure 3.3 gives a visual representation of the different regular quantization formats described here.

Regarding the problem of selecting and mapping the quantized values, there are numerous different ways to address it, ranging from simple heuristics like those used to convert CNN weights into binary values depending on their sign [131] or projecting real-valued weights to (one of) the closest discrete points [145], to loss functions that regularize the CNN and force weights into quantized states upon the convergence of the training algorithm [154]. More advanced approaches include incremental quantization [155], knowledge distillation [156], [157], Alternating Direction Method of Multipliers (ADMM) [158], [159], regularization terms [154] that can also be used for robust quantization [129], [130]. For more details on quantization, please refer to our book chapter [12].

Weight-Sharing Weight Sharing (WS) compresses the CNN by assigning shared values to weights and/or activations. This transforms plain weight data storage into a reduced number of shared values in a dedicated memory, together with the indices of these values in the weight matrix. A simple example is presented in Figure 3.4, compression is achieved by using 5 distinct shared-values to represent the weights, the shared-values are selected by using a clustering algorithm such as the K-means[14], allowing to address simultaneously the problem of selecting and mapping the initial values to the quantized ones. In this particular example, as there are only 5 distinct shared-values, the indexes that will be stored in the weight matrix require only 3-bits, resulting in a weight matrix size of $5 \times 5 \times 3 = 75$ bits instead of the initial $5 \times 5 \times 32 = 800$ bits, giving an overall Compression Rate (CR) of $800/(75+5*32) = 3.4$ compared to the initial matrix. a larger matrix presents more opportunities to reach high CR. More details on WS are given in Section 3.2. One of the advantages of WS compared



Original Weight Matrix				
-0.5	0.3	-0.1	0.8	-0.5
-0.4	-0.1	0.2	-0.4	-0.3
0.8	0.9	-0.2	-0.1	0.7
-0.5	0.4	0.2	-0.3	0.5
0.2	-0.4	-0.2	0.9	-0.7

WS Weight Matrix				
3	2	0	1	3
3	0	2	3	0
1	1	0	0	1
3	4	2	0	4
2	3	0	1	3

WS Codebook	
0	-0.18
1	0.82
2	0.22
3	-0.48
4	0.45

A

Figure 3.4: Basic example of weight sharing for compressing a matrix.

to regular quantization techniques, is the flexibility in shared value selection, by relaxing the presence of hardware mapping hardware constraints. This also gives the direct inconvenience of hardly benefiting from acceleration without adding dedicated logic, resulting in a computational overhead to retrieve the shared values from the memory.

3.2 Weight-Sharing in Detail

This section is intended to explain the WS principles and the different challenges and approaches observed in the literature. Following the application of the WS as described in Figure 3.4, weights are grouped into buckets or clusters sharing the same value. Observing the distribution of weights values before and after WS results in something like Figure 3.5. Depending on the number of shared values used, the distribution of the weights inside a layer will change. It is also possible to note that there is no imposed symmetry or regularity in the values as is often the case in regular quantization, offering more flexibility and potentially allowing to reach a higher compression rate. Different WS approaches can be classified by (1) the method used to group the weights, (2) the shared scope, (3) the unit shared,, (4) the possible acceleration of CNN (5) the method application process, and (6) the selection of the number of shared values.

3.2.1 Grouping Algorithm

The earliest work of the WS family, VectorQuantization [160], discovered that using the K-means [14] clustering algorithm to group the weights into clusters and use each cluster centroid values for the inference achieved surprisingly good results in compressing CNN. The K-means [14] is an unsupervised-learning clustering algorithm, that achieves iterative grouping of the samples using a distance computed between the samples and centroids samples. there is one centroid per cluster, and they are initialized either randomly or with more advanced meth-

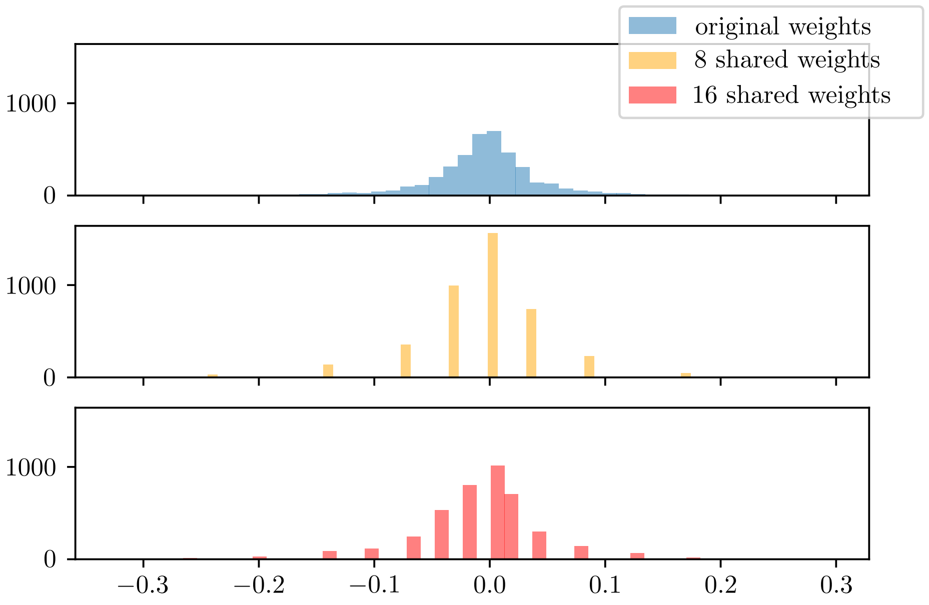


Figure 3.5: Distribution of the weights composing the first layer of a trained ResNet50V2 [19] on the ImageNet[18] dataset, original (top), with only 8 (middle) and 16 (bottom) shared values.

ods, and then iteratively refined by moving to the mean of the samples composing a cluster at the end of each iteration. By doing this, the K-means [14] algorithm minimizes the Within-Cluster Sum of Squares (WCSS) or inertia, allowing to quickly find a suitable clustering. The K-means [14] algorithm was subsequently used in several papers [2]–[4], [49], [50]. Some variation can be observed in DeepCompression [2], where multiple cluster initialization techniques are investigated. In Quantized-CNN [49], the minimization of the accumulated error across multiple layers is also implemented to improve the centroids selection. Other approaches were used to group the weights, an early work, HashedNets [161] rely on random-grouping. More advanced techniques like SoftWeightSharing [46] used mixture components. More recently, DP-NET [29] rely on dynamic programming for grouping the weights. ClusteringMCA [5] used an approach based on the dictionary-learning technique.

3.2.2 Shared units and acceleration

The unit shared has a very close relationship with the compression and acceleration potential. The earliest work, VectorQuantization [160] has investigated multiple sharing units, with first the study of sharing single weights, and then entire vectors. The metrics of interest used in VectorQuantization [160] was the CR of the resulting approximated CNN, compressing a network has the advantage of allowing to store the weights in the faster and less energy demanding, but more resource-constrained SRAM. It is also possible to benefit from acceleration with the application of computation reuse techniques, as it was proposed in Quantized-CNN [49], the shared unit is a vector for convolutional layer, which allows for higher reuse opportunities. Another example of paradigm-changing computation reuse was proposed in LookNN [50], with the use of pre-computed LUT for multiplication, allowing to execute the inference without any multiplication operation, resulting in acceleration. Another approach of computation reuse introduced in 2D-Clustering [4] with the sharing of entire kernels, allowing for fast computation, transformation invariance of the sharing was also explored in the same work. Another advanced technique is to use efficient codebook representation to reduce the computational complexity as in ClusteringMCA [5]. As a rule of thumb, sharing larger structures allow for greater acceleration and compression, at the cost of larger quantization error, which could result in AL if not compensated.

3.2.3 Sharing Granularity

Regarding the granularity of the sharing, most of the techniques[2], [3], [5], [29], [46], [49], [50], [160]–[162] proposed to use a single set of shared values for each layer of the network, allowing for better resilience to layers sensitivity. Some other techniques, like weighted-entropy [162], explored the possibility of having a single set of shared values for the whole CNN. Another interesting approach is to use an implementation-based scope, as it was explored in DeepKmeans[3], with the use of a granularity corresponding to the row-stationary dataflow [65]. Section 4.1 gives a self-investigation of the granularity opportunities and limits in terms of CR.

3.2.4 Clustering Application Process

Although the earliest WS technique, like VectorQuantization [160] proposed to apply the approximation post-training and the opposite, can be seen in HashedNets [161], who randomly group the weights and train the centroids. Most modern techniques propose to rely on less extreme processes, by applying the approximation alongside the training to allow for AL recovery. A representative example of the first approach with an iterative partial approximation of the weights and retraining of the others to recover AL is observed in DeepCompression [2]. More recently, iterative grouping and retraining are performed in [4], [5], [50]. Other non-clustering-based techniques like SoftWeightSharing [46] and weighted-entropy [162] integrate the grouping directly during the training. The same approach was used in both DeepKmeans [3] and DP-Net [29] with the use of a regularization term to encourage weight grouping during the training.

3.2.5 Selecting the Optimal Number of Clusters for Weight Sharing

Regarding the selection of the number of shared values, some existing methods [3], [5] manually tune the number of shared values to achieve the desired CR. The main problem with manual tuning is that it is costly and requires expert knowledge. Other methods [2], [29] use a fixed number of shared values. However, the most widely adopted solution is to homogeneously vary the number of shared values (i.e., use the same number for each layer) to explore the trade-off between compression and precision [4], [48]–[50]. However, this approach is suboptimal, as the weight sharing factor is not adapted to the differences in tolerance-to-approximation, or resilence of each layer.

The main objective of this thesis is to prove that it is also possible to optimize the number of shared values to each layer's resilience while avoiding the costly retraining step. However, due to the increasing complexity of modern CNN topologies, the DSE required to find optimal WS approximations is prohibitively costly. Multiple heuristic approaches will be explored in subsequent chapters to achieve a scalable retraining-free WS compression.

3.3 Conclusions

This chapter has presented the state-of-the-art of approximation techniques for CNNs compression, with a global view on the taxonomy of the different techniques, followed by a finer-grained study of the data-refinement techniques, and finally, a close view on the different WS techniques in the literature.

Chapter 4

Exploring the Use of a Greedy Optimization Process

In this chapter, a presentation of the studied approximation design space with details on variables, objectives, and constraints is given in Section 4.2. Then, the sensitivity of CNNs to approximation is explored in Section 4.1, by measuring a single layer sensitivity and evaluating different scopes for the sharing of the weights. Then, in Section 4.3 a greedy algorithm is proposed and evaluated to optimize the number of shared values per layer of a CNN regarding his local sensitivity to approximation. The use of proxy metrics to accelerate the exploration is also presented and evaluated in Section 4.4. The acceleration allows for multi-objective optimization, a last greedy algorithm is presented and evaluated in Section 4.5. Finally, the limits of the use of a greedy algorithm are studied in Section 4.6.

4.1 Studying approximated CNN sensitivity to approximation

4.2 Approximation Objectives

Optimizing the number of shared values for a given trained CNN can be represented as a mathematical optimization problem. The traditional definition of the optimization problem is finding the best solutions, described by variables, that maximize the objectives while honoring the constraints. In the specific case of WS tuning, we have the following:

- Variables can be represented as a vector containing the number of shared values of each sharing unit;

- Objectives are the selected metrics for representing efficiency and QoR;
- Constraint is the MLPerf [7] quality target, although it is possible to add others when studying a specific implementation.

To define the search space of the optimization, there is a need to set a range of the possible number of shared values, or the possible values of the variables. This range can be arbitrarily chosen, or determined with different techniques, this is explored in detail in Section 4.1. There is also a need to define the number of variables, meaning the number of sharing units, or granularity of the sharing, Section 4.1 investigate the impact of the sharing granularity on the trade-off between efficiency and QoR, with a focus on the sharing overhead and the search space size.

Representing the QoR is something very usual when prototyping a CNN, it is usually achieved using the top-1 or top-5 accuracy, meaning the percentage of time the expected label was present among the top-1 or top-5 predicted label in terms of probability. Another QoR metric is the distance between the predicted and the expected output, computed using a loss function. Both the accuracy and the loss are usually computed using a validation dataset that has never been used to train the CNN, allowing to measure its generalization capacity. The vast majority of approximation papers report the QoR as the top-1 or top-5 Accuracy Loss (AL), computed as the absolute difference between the baseline CNN accuracy and the approximated CNN accuracy. A negative AL means that the approximated CNN reaches higher QoR than the baseline CNN, this is not very usual but this can happen with some approximation acting like a regularization during the training. The metric that will be used to represent QoR in this thesis work is the top-1 AL, because it gives a more direct insight on the behavior of the approximated CNN compared to the baseline.

There are multiple usual representations of the efficiency in the literature, some describe the memory footprint reduction, as the number of weights or their accumulated size, whereas some others describe the potential acceleration as the number of FLOPs or the number of Multiply-ACcumulate operation (MAC), the last category describes the resource utilization, as the number of cells or DSP used in a FPGA design. There are also other platform-specific high-level metrics representing the efficiency of the approximated CNNs like the latency, the throughput, or the energy cost of the inference. The objective of this thesis is the compression of the weights for a given trained CNN, the most suitable metric is thus the Compression Rate (CR), defined as the ratio of the baseline CNN weights accumulated size and the approximated CNN weights accumulated size. A compression rate higher than 1 means that the approximated CNN reaches higher efficiency than the baseline, which is exactly the purpose of the approximation.

The two metrics can be obtained from an approximated CNN described by a vector representing the number of shared values for each sharing unit. The CR can be computed using an analytical formula described in Equation 4.1. On the other side, obtaining the AL requires an evaluation of the inference accuracy for the given approximated CNN, which also means that there is a need to apply the weight sharing before the evaluation. The former step of the accuracy evaluation is very computation-intensive, and you will see that this is a common problem in the investigations conducted during this thesis.

Concerning the optimization constraints, the MLPerf [7] contest, intended to allow for comparison of heterogeneous CNN training and inference platforms using high-level metrics like throughput and latency has set an acceptable accuracy target for approximating CNNs. Complying with this quality target requires the top-1 accuracy of the approximated CNN to be at least 99% of the baseline accuracy. A second quality target level is set at 98% of the baseline accuracy for CNNs targeting embedded inference, like MobileNetV2 [13], because these CNNs are already optimized by design and are more sensitive to approximation than others. This quality target has been widely used ever since.

It is also possible to use different constraints concerning the implementation, like memory or computation boundaries for enforcing high-level metrics like latency or throughput. It is even possible to use energy constraints during the optimization. All these implementation-related constraints are out of the scope of this thesis.

4.2.1 Layer sensitivity to approximation

As it has been stated in Chapter 3, there is a need to tune the number of shared values of the WS to achieve acceptable AL and CR without requiring retraining. The objective of this chapter is to investigate a simple yet effective way to search for an optimal number of shared values in terms of AL, then improve it to allow multi-objective optimization that takes into account AL and CR. In addition to DSE algorithms, this chapter also presents a case study of the granularity of the weight sharing codebook, as well as the use of a proxy metric to greatly reduce the need for computationally expensive approximate CNNs scoring to the bare minimum.

To show how can vary the accuracy of a CNN when varying the number of shared values k_i for each layer i (where $0 \leq i \leq N-1$), we present a first case study about the AL variation when the WS approximation technique is applied to the first layer $i = 0$ of a MobileNetV2 [13] trained on the ImageNet [18] dataset. For each k_0 value (i.e., the degree of approximation), the k-means [14] clustering algorithm is used to select the values of the shared weights. During the whole experiment, the other layers (from 1 to $N-1$) of the CNN remained untouched and we evaluate the AL using the validation set of the ImageNet [18] dataset. For each k_0 value in a linear range from 2 to 864 number of shared values, the CNN with the resulting approximated first layer is scored. The range has been determined empirically from the minimal number of shared values, 2 to the maximal, which is the number of weights of the layer.

The results are shown in Figure 4.1. The X-axis shows k_0 values and the Y-axis reports the achieved AL. The first observation is that the accuracy strictly depends on k_0. The second observation is that around $k_0 = 32$ the top-1 accuracy is close to the reference one with $AL = 0\%$. It is important to mention that

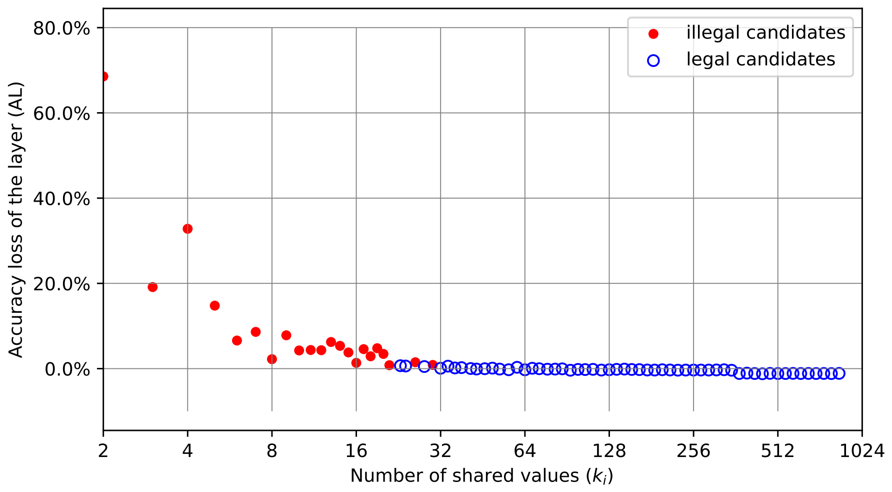


Figure 4.1: Achieved AL measured on the ImageNet [18] dataset, obtained when varying the number of shared values, k_0, for the first layer of MobileNet V2 [19].

the reference layer has 864 weights. Each weight is stored into a $b_{values} = 32$ bits floating-point variable leading to a memory footprint of 27,648 bits. Keeping $k_0 = 32$ shared values, we need 5,184 bits for storing the 864 shared values indexes of $b_{index} = \lceil log_2(k_0) \rceil = 6$ bits each plus 1,216 bits for storing the 32 shared values in the WS codebook. Resulting in a total of 6,400 bits for storing the layer weights. The achieved CR of the layer can be described using the following formula:

$$CR = \frac{W \times b_{values}}{W \times b_{index} + k_i \times b_{values}}, \tag{4.1}$$

The achieved CR of the layer is 4.32× with negligible AL. The CR changes depending on the bit-width of the CNN weights. In the above example, if 16-bit data types are used, the CR will be 2.16×, whereas for 8-bit the CR will be 1.08×. In this chapter, it is considered that the weights of the baseline CNNs are represented using a 32-bit floating-point variable to fairly compare with the state-of-the-art CNNs compression techniques.

As it has been observed in the literature, each layer of a CNN has a different resilience to approximation [3]. It can be observed in Figure 4.2 showing the sensitivity of the first and last layer of the studied CNN, MobileNetV2 [13] as an extension of the experiment conducted for Figure 4.1. This very simple observation state the need to identify the optimal number of shared values for each layer. One of the simplest approaches to optimize the number of shared values leverages the greedy heuristic, by selecting for each layer the locally best-found solution (i.e., k_i). An example of a locally best-found selection with the Figure 4.2 is selecting the number of shared values leading to the highest top-

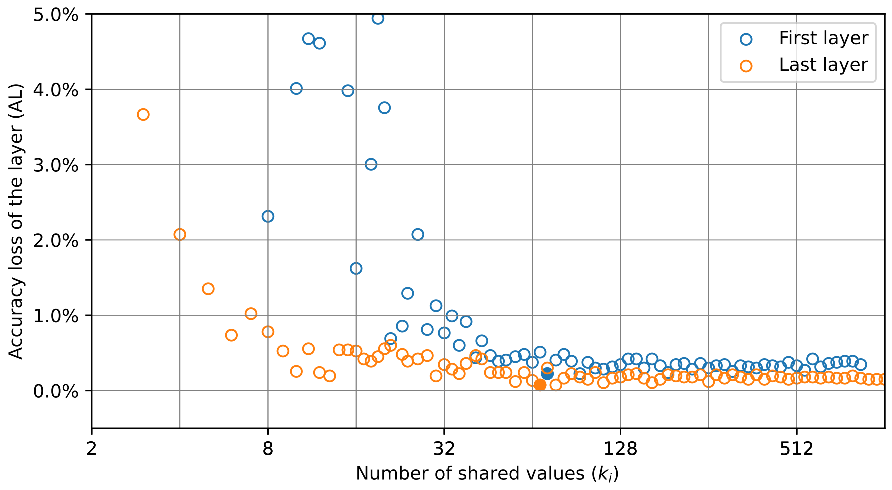


Figure 4.2: Analyzing two different layer sensitivity to approximation with the first and last layer of a MobileNetV2 [13] trained on the ImageNet [18] dataset, obtained when varying the number of shared values, k_i.

1 accuracy, highlighted with a full colored circle, the next step will be to do the same for the next layer, and continue until reaching the last layer of the CNNs. Even though a greedy algorithm suffers from local minima, it is shown in the rest of this chapter that it can produce acceptable results in a reasonable amount of time compared to an exhaustive search approach. An extension of the greedy algorithm involving multi-objective (AL, and CR) local best-found solution selection is also presented. But first, there is a need to study the impact of the weight sharing granularity, this is the purpose of the next section.

4.2.2 Impact of the Weight-Sharing Granularity

When the WS is applied, there is a need to define the granularity of the codebook, meaning the weights that will share a set of distinct values. In the example presented in Figure 4.1, a single codebook contains all the shared values for the layer. But the sharing can be achieved at different granularities, by choosing to share the weights at the level of the channels or the kernels (with a different codebook per channel or kernel respectively). Three different possible granularities are listed here from coarser to finer grain:

- *Layer*: all the weights of a layer share the same codebook;
- *Channel*: all the weights corresponding to a specific output channel (3D kernels) shares the same codebook;

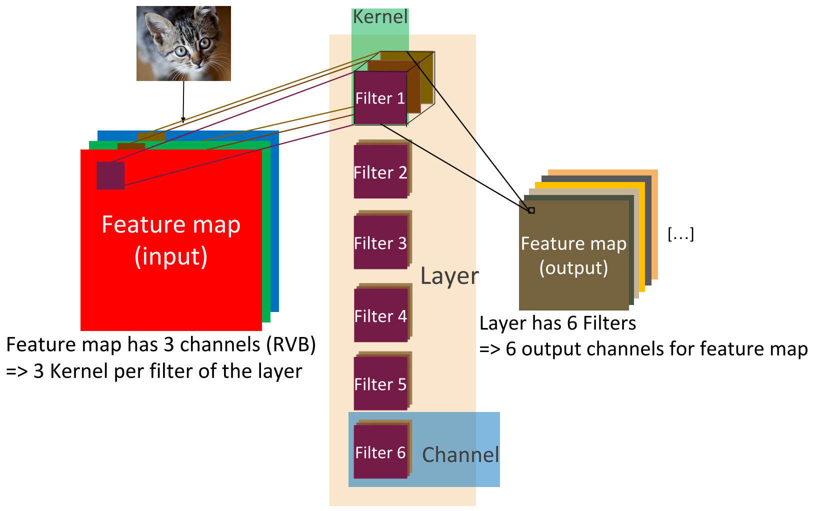


Figure 4.3: Different granularities for weight sharing. Weights can be shared at the level of the layer, the channel, or the kernel.

- *Kernel*: all the weights corresponding to both a specific input and outputs channel (2D kernels) share the same codebook.

Figure 4.3 shows a graphical example of the above granularity level for a given convolutional layer. The sharing granularity impact both the AL and the CR, smaller granularity tend to reach better AL, at the cost of lower CR and larger exploration time due to the higher number of possible solutions. Another interesting thing about the granularity is that, given an optimal number of shared values for a layer, it can virtually be found at a lower granularity (i.e. channel or kernel) by using a homogeneous number of shared values for each channel/kernel of the layer and the same codebook. The same applies between channel and kernel sharing granularity.

Each of the above granularities allows achieving different CRs (Equation 4.1) and AL w.r.t. the reference (non approximated) CNN. To investigate this AL, a very simple experiment on LeNet-5 [15] trained on the MNIST [16] dataset was carried out. for each defined granularity, the number of shared weight vary, considering that the number of shared values is the same for each structure. Figure 4.4 plots the trade-off between AL and CR obtained. The results clearly show that the “kernel” and “channel“ granularity does not lead to a good trade-off between compression and accuracy. On the other hand, the “layer” granularity allows up to 3× higher compression ratios for the same AL. For the remainder of this chapter, the **layer granularity** is considered.

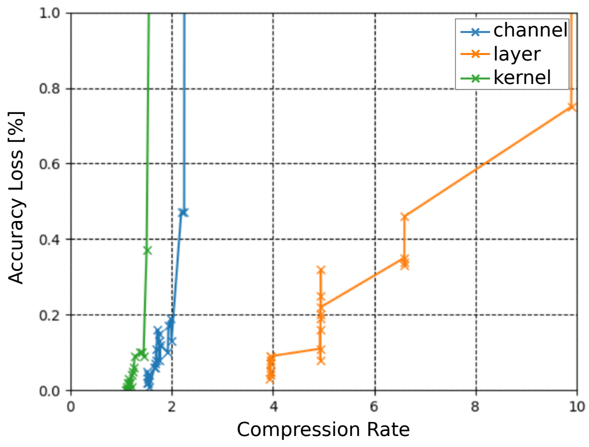


Figure 4.4: Top-1 AL and memory compression over varying numbers of clusters at different sharing granularity [8].

4.3 Greedy Optimization Algorithm

4.3.1 Introducing greedy optimization

Using a greedy optimization algorithm allows reducing the search space by relying on local optimization. The implementation is very simple, for each layer of the network from the first to the last, the locally-best-found number of shared values k_i is found by evaluating every possible approximate candidate from the k_{range}. The evaluation of each of the approximate candidates consists of two steps: (1) applying the WS with the desired k_i using the k-means [14] clustering algorithm, and (2) evaluating the resulting CNN AL on the test dataset, with the previous layers and current layer approximated. From this evaluation step, the candidate with the best AL is selected. The Algorithm 1 summarizes this implementation. The time complexity of the proposed algorithm is:

$$\mathcal{O}(card(k_{range}) \times N) \tag{4.2}$$

With N the number of layers of the CNN.

4.3.2 Proof of concept on LeNet-5

Validating the capability of the proposed greedy algorithm to solve real-world WS optimization problems, requires first, proving that it can work on small CNNs. To this extent, Lenet-5 [15], a 5 layers CNN capable of recognizing the black and white low-resolution digits from the MNIST [16] dataset is used as a baseline CNN. LeNet-5 is composed of 3 convolutional layers (CONV) followed

Algorithm 1: A greedy algorithm for optimizing the number of shared values of each layer of a CNN.

```
Input: $baselineCNN$, $BaselineAccuracy$ $k_{range}$, $testDataset$
Output: $AxCNN$, $AL$
1:  $AxCNN \leftarrow baselineCNN$;
2:  for $n \in range(len(baselineCNN.layers))$ do
3:    $AxCandidateList = []$;
                                        // Apply WS with the desired $k_i$
4:    for $k_i \in k_{range}$ do
5:      $AxCNN.layers[n].weights \leftarrow$
        $k - means(AxCNN.layers[n].weights, k_i)$;
6:      $AxCandidateList.append(AxCNN)$;
7:    end for
                    // Score approximate candidates and select the best ($bestAL$)
8:    for $AxCandidate \in AxCandidateList$ do
9:      $AL \leftarrow evaluateAL(AxCandidate, testDataset)$;
10:     if $AL < bestAL$ then
11:       $bestAL \leftarrow AL$;
12:       $AxCNN.layers[n].weights \leftarrow AxCandidate.layers[n].weights$;
13:     end if
14:   end for
15: end for
```

by 2 fully-connected (FC) layers, with a total of 61,470 parameters (of which 50% are in the convolutional layers). The explored number of shared values is the linear range $k_{range} = [2; 25]$, 25 corresponds to the number of weights of a 2D convolutional kernel, and allows for an accuracy loss bellow $1e4$ for all the layers. The training was carried out using the open-source framework N2D2 [17]. The LeNet-5 model description we used is available in the framework itself. It is important to mention once again that the proposed approach is independent of the adopted training framework. The exploration algorithm was set to iterate using previous values as initialization until it found an already identified solution, with a limit of 30 iterations. The exploration has been conducted in two different modes: natural order, from the first layer to the last; and reversed order. The execution of the optimization algorithm took 7.2 hours on a laptop (see Appendix A.2 for more details on the hardware setup). Each candidate is evaluated on the 10k images MNIST [16] test dataset.

The results are reported in the Table 4.1, the proposed WS approach is compared with two implementations, the 16-bits, and the 8-bits LeNet-5, both obtained using the post-training quantization from the N2D2 [17] framework. Interestingly, the proposed approach can reach higher CR without any calibration step that is required in the post-training quantization. There is also no significant difference between the first to the last and the last to the first order of approximation. These results prove that the proposed algorithm can be ap-

Type	Baseline Network	Top-1 AL (%)	CR
N2D2 export	16-bit (ref)	0.00	1
N2D2 export	8-bit	0.05	2
WS	16-bit first to last	0.02	4.06
WS	16-bit last to first	0	4.04
WS	8-bit first to last	0.05	4.37
WS	8-bit last to first	0.05	4.83

Table 4.1: Compressing LeNet-5 [15] on the MNIST [16] dataset, comparison is made with N2D2 [17] post-training quantization on both 16-bits and 8-bits.

plied to LeNet-5 [15], and achieve a significant compression of over 4× CR under negligible AL. The first to the last order of exploration is used in subsequent experiments.

4.3.3 Application to larger CNNs trained on the ImageNet dataset

After the proof of concept on a small CNN it is possible to evaluate the proposed approach on more challenging CNNs. That is why the evaluation of the proposed method is also performed on larger CNNs trained on the ImageNet [18] dataset, with 224x224 images and 1,000 classes output. Three image classifier CNNs have been selected, ResNet18V2 [19], SqueezeNet [20], and MobileNetV2 [13]. The previous framework relying on N2D2 for CNN scoring has been adapted to be able to process ONNX [21] models from the ONNX Model ZOO, and MXNET [22] is used for scoring the CNNs on a GPU, the same GPU is used to compute the k-means algorithm, relying on DeepKMCuda [23]. All the experiments have been executed on a GPU server (see Appendix A.2 for more details on the hardware setup). The k_{range} has been empirically determined for each network in the same way as the example reported in Figure 4.1. For each CNN layer, the number of shared values is incremented until the AL is negligible (bellow $1e4$) for the first layer of the CNN, then a range is selected within $[k_{minAL} - 20; k_{minAL} + 20]$. For MobileNetV2, this range is extended to $[k_{minAL} - 60; k_{minAL} + 60]$ because the CNN is more sensible to approximation. This explains why different CNNs have different k_{range}.

The Table 4.2 presents the obtained results. Each of the reference networks has been successfully compressed by up to 5×, compared to the 32-bits baseline, with a very small AL. Regarding the acceptability of the AL, it appears that the results obtained for both ResNet18V2 [24] and SqueezeNetv1.1 [20] comply with the MLPerf [7] quality target, whereas the results for MobileNetV2 [13] does not. Furthermore, it is important to note that the CR depends mostly on the k_{range},

Baseline Network	K_{range}	Top-1 AL (%)	CR
ResNet18v2	[40,80]	0.22%	5.28
SqueezeNetv1.1	[40,80]	0.53%	5.17
MobileNetv2	[2,120]	1.43%	4.85

Table 4.2: Compressing CNNs trained on the ImageNet [18] dataset using the Algorithm 1.

and is rather constant for the different CNNs. Even for SqueezeNetv1.1 [20], which is already designed to have a very low memory footprint, we achieve a 5$\times$ CR. In other words, even if the reference CNN has been designed to be efficient in terms of the memory footprint, the proposed WS method can further reduce it.

Regarding the execution time, execution of the algorithm 1 is dominated by more than 80% by the **scoring time** as it can be observed in Figure 4.5 reporting the breakdown of the total execution time for the clustering and the scoring of the approximate candidates in the experiments described previously. The total execution time is the following: ResNet18V2 [24] took 16 hours with 840 approximated CNN evaluations, SqueezeNetv1.1 [20] took 13 hours with 1,040 approximated CNN evaluations, and finally, MobileNetV2 [13] took 86 hours with 6,372 approximated CNN evaluations. The increase in the number of evaluations for MobileNetV2 is due to the increase in the number of explored k_i as well as the higher layer count. As the simulation time is largely dominated by the scoring of the approximate CNNs candidates on the validation dataset, it could largely benefit the use of an indirect way to estimate the accuracy of CNN without running the inference of all the validation datasets samples. This will be investigated in the next section.

4.4 Investigating the Use of a Proxy Metric

4.4.1 Correlation between clustering inertia and accuracy loss

This section investigates the use of proxy metrics to alleviate the evaluation cost and make the exploration possible for large CNNs. The key to identifying an alternative approach to avoid the score for each approximate CNN candidate relies on the analysis of the k-means [14] clustering algorithm. Indeed, it clusters the samples by trying to split them into k groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares [163] defined by the following equation:

$$Inertia = \sum_{i=1}^{k} \sum_{x=C_i} ||x - \mu_i||^2 \tag{4.3}$$

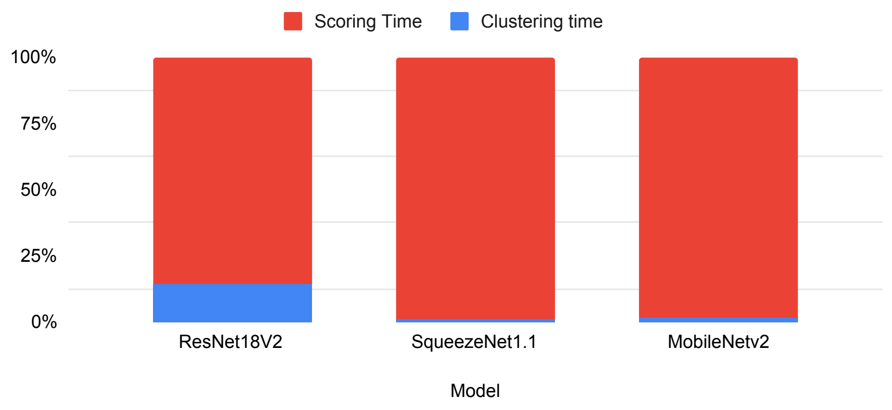


Figure 4.5: Profiling simulation time between the k-means clustering application and the scoring of approximate candidates using the validation dataset for Algorithm 1.

with k the number of clusters, C_i the set containing the samples belonging to the $i-th$ cluster, and μ_i the centroids (average value) of the $i-th$ cluster. inertia can be described as an approximation error when replacing the samples by the corresponding clusters centroids, and the lower the inertia the better the approximation. The inertia seems to be a good proxy candidate of the accuracy of the approximated CNN, the lower the inertia the higher the accuracy.

It is important to evaluate the correlation between the inertia and the AL of a CNN. To that extent, a very similar experiment to the one for Figure 4.1 is conducted, but with every layer of the CNN. Figure 4.6 reports two examples of how good is the inertia as AL estimation by showing the inertia and the AL obtained after clustering layers with $k_{range} = [2, 120]$. Both the examples are related to MobileNetV2 [13]. In particular, we reported the correlation for the first and the last layer of MobileNetV2. These layers have been chosen because they have a different complexity expressed in terms of weights count: 864 for the first layer, and 1,280,000 for the last layer. Both graphs plot the CNN AL (obtained through CNN scoring) w.r.t the inertia. The correlation in the small layer is not as strong as in the large layer, but both cases confirm that lower inertia tends to give lower AL.

As the inertia proved to be a good estimation of the approximated CNN accuracy, we modified the Algorithm 1 to use it. Instead of scoring the approximated CNN after each k-means application, in the Algorithm 2, the inertia is used to filter a subset of the most promising approximate candidates to be scored. The number of selected candidates depends on the $inertiaFilterRatio$ parameter. The complexity of Algorithm 2 is thus:

$$\mathcal{O}(card(k_{range}) \times N \times inertiaFilterRatio). \tag{4.4}$$

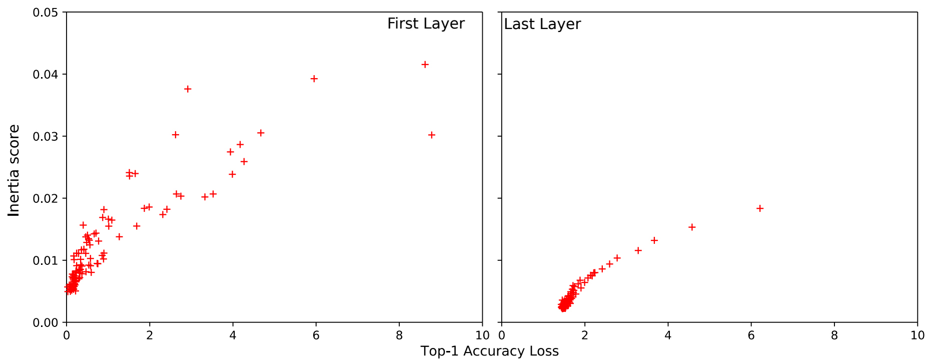


Figure 4.6: Correlation between the inertia and the top-1 accuracy obtained when varying the number of shared values for the first, and last layers of MobileNetV2 [13].

With $inertiaFilterRatio < 1$.

4.4.2 Measuring the benefit of the proxy metrics utilization

To evaluate the improvement of the Algorithm 2 experiments have been carried out under the conditions described in Section 4.3. With the $inertiaFilterRatio$ (noted Filtering in the Table 4.3), the percentage of approximate CNN versions that will be scored on the validation dataset, varying between 5%, 15%, and 33%. A special case is the $inertiaFilterRatio = 100\%$, where Algorithm 2 has the same behavior as Algorithm 1, because the subset of promising approximate candidates contains all the approximate candidates.

From the results in Table 4.3, we can see that the highest values of $inertiaFilterRatio$ generally result in lower AL values. This improvement comes at the cost of a higher optimization time due to a larger number of candidates passing through the inertia filter to the costly scoring step. Using a small filtering value (i.e., 5%, 15%) significantly reduces the overall exploration time with a limited impact on the compression rate and accuracy loss. It is particularly interesting to compare the case of MobileNetV2 [13]. With the $inertiaFilterRatio = 33\%$ the achieved AL is slightly better than the result obtained with $inertiaFilterRatio = 100\%$.

However, the most relevant impact of the proposed heuristic is on the overall exploration time, shown in Figure 4.7. In the case of $inertiaFilterRatio = 33\%$, the execution time is reduced by 50% without noticeable AL variation. With $inertiaFilterRatio = 15\%$, the AL is under 0.5% with a execution time reduced by 75%. With $inertiaFilterRatio = 5\%$ the AL is around 1% but the execution time is reduced by 90%. Looking at CR, we did not observe a significant difference with the variation of the filtering value. Achieved CRs are slightly less than those of $inertiaFilterRatio = 100\%$ but still significant.

Algorithm 2: A greedy algorithm for optimizing the number of shared values of each layer of a CNN, with inertia filtering of approximate candidates.

```
Input: $baselineCNN$, $BaselineAccuracy$ $k_{range}$, $testDataset$, $inertiaFilterRatio$
Output: $AxCNN$, $AL$
1:  $AxCNN \leftarrow baselineCNN$;
2:  for $n \in range(len(baselineCNN.layers))$ do
3:      $AxCandidateList = []$;
                                        // Apply WS with the desired $k_i$
4:      for $k_i \in k_{range}$ do
5:          $AxCNN.layers[n].weights \leftarrow$
            $k - means(AxCNN.layers[n].weights, k_i)$;
6:          $AxCandidateList.append(AxCNN)$;
7:      end for
                                        // Apply inertia filtering
8:      $AxCandidateListFiltered \leftarrow$
        $filterInertia(AxCandidateList, inertiaFilterRatio)$;
        // Score remaining approximate candidates and select the best ($bestAL$)
9:      for $AxCandidate \in AxCandidateListFiltered$ do
10:         $AL \leftarrow evaluateAL(AxCandidate, testDataset)$;
11:         if $AL < bestAL$ then
12:             $bestAL \leftarrow AL$;
13:             $AxCNN.layers[n].weights \leftarrow AxCandidate.layers[n].weights$;
14:         end if
15:     end for
16: end for
```

These results prove the efficiency of using inertia as a quick and effective metric for determining the impact of weight-sharing on CNN.

The objective of Algorithm 2 is to minimize the AL without considering other constraints. However, It would be interesting to investigate different trade-offs between the AL and the CR. The next section provides an extension of the Algorithm 2 using two objective metrics (AL, and CR).

4.5 Explore the Trade-Off Between Accuracy and Compression

4.5.1 Multi-objective approach

One can be interested to accept a higher AL for reaching a higher CR. Meaning that more objectives need to be analyzed. In this section, an extension of the Algorithm 2 with two objective metrics: (1) minimize the AL, and (2) maximize

Baseline Network	K_{range}	Filtering (%)	Top-1 AL (%)	CR
ResNet18v2	[40,80]	5	0.47%	4.93
		15	0.23%	4.95
		33	0.23%	4.94
		100	0.22%	5.28
SqueezeNetv1.1	[40,80]	5	0.57%	4.77
		15	0.55%	4.80
		33	0.49%	4.73
		100	0.53%	5.17
MobileNetv2	[2,120]	5	2.78%	4.54
		15	1.58%	4.55
		33	1.13%	4.55
		100	1.43%	4.85

Table 4.3: Compressing CNNs on the ImageNet [18] dataset using Algorithm 2.

the CR; is proposed. To achieve this modification, there is a need to enlarge the number of candidates selected during each layer exploration. Instead of selecting only the candidate showing the lowest AL, the AL vs. CR trade-off needs to be explored, using a Pareto optimal candidates selection. This implies enlarging the complexity of the exploration for the next layers. Indeed, there is a need to run the exploration for each of the selected candidates from previous layer exploration, raising the complexity of the layer-wise exploration of the layer i to $\mathcal{O}(card(\phi_{i-1}) \times card(k_{range}))$, where ϕ_i is the number of selected candidates of the layer i. The resulting complexity of the multi-objective exploration for the whole network is then:

$$\mathcal{O}(\prod_{i=1}^{N} card(\phi_{i-1}) \times card(k_{range})) \tag{4.5}$$

In the worst case (i.e., every approximate candidate is Pareto efficient from the first layer to the last, meaning that $card(\phi_i) = card(k_{range}), \forall i \in N$), the complexity becomes:

$$\mathcal{O}(\prod_{i=1}^{N} card(k_{range})^2 \iff \mathcal{O}(card(k_{range})^{2N}). \tag{4.6}$$

Taking as example a $N = 5$ layer CNN like LeNet-5 [15], and given a $k_{range} = [2, 26] \Rightarrow card(k_{range}) = 25$, and considering that the clustering step takes a very optimistic 1ms, it will take more than 3000 years to optimize the CNN. To avoid this explosion of the number of candidates, we need to restrain their count between each iteration, we can see in Equation 4.1 that in the memory footprint of the approximated model given by the formula $W * b_{index} + k_i * B$, the most important term is the first one $W * b_{index}$, indeed, W tends to be up

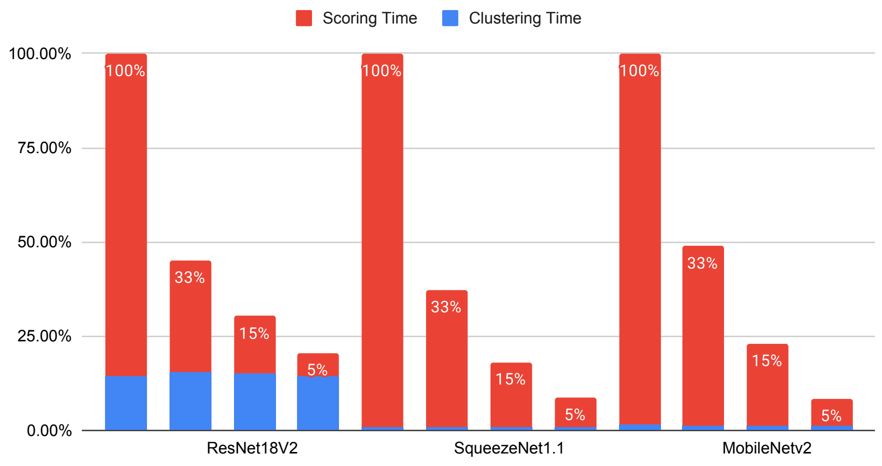


Figure 4.7: Profiling simulation time between the k-means clustering application and the scoring of approximate candidates using the validation dataset for Algorithm 2.

to several million for the largest CNNs. A way of limiting $card(\phi_i)$ is to use b_{index} as a proxy metric for the CR, meaning that for a given layer, instead of calculating the CR, the b_{index} value can be used as an estimation of the CR. Resulting in the following boundary:

$$card(\phi_i) \leq \lceil log_2(max(k_{range})\rceil - \lceil log_2(min(k_{range})\rceil. \tag{4.7}$$

The complexity of the exploration is thus reduced to:

$$O(N * \lceil log_2(max(k_{range})\rceil - \lceil log_2(min(k_{range})\rceil * card(k_{range}), \tag{4.8}$$

which our experiments show affordable even for a large network, for example, the exploration for a MobileNetV2 with an $inertiaFilterRatio = 5\%$ is about 36 hours. The resulting multi-objective hierarchical exploration algorithm is detailed in Algorithm 3. The main change is the population-based approach over the single-solution-based approach that was used in Algorithm 2. At the end of each layer exploration, a Pareto effcicient population w.r.t. AL and b_index is used as an input for the next layer exploration. The initial population for the first layer is the baseline CNN and the output population of approximated CNNs is the Pareto efficient population of the last layer.

4.5.2 Measuring the benefits of the multi-objective approach

To evaluate the improvement of the Algorithm 3 experiments have been carried out under the same conditions described in Section 4.3. To simplify the explanation, results focus on the MobileNetV2 [13] CNN, showing a larger interest

Algorithm 3: A greedy algorithm for optimizing the number of shared values of each layer of a CNN, with inertia filtering of approximate candidates and multi-objective optimization.

```
Input: $baselineCNN$, $BaselineAccuracy$ $k_{range}$, $testDataset$, $inertiaFilterRatio$
Output: $\phi$
1: $AxCNN \leftarrow baselineCNN$;
2: $\phi \leftarrow [AxCNN]$
3: for $i \in range(len(baselineCNN.layers))$ do
4:     $AxCandidateList = []$;
                                    // Iterate in $\phi_{i-1}$ with the desired $k_i$
5:     for $candidate \in \phi$ do
                          // Apply the WS with the desired $k_i$for $k_i \in k_{range}$ do
6:         $candidate.layers[i].weights \leftarrow$
           $k - means(candidate.layers[i].weights, k_i)$;
8:         $AxCandidateList.append(candidate)$;
9:       end for
10:    end for
                                                          // Apply inertia filtering
11:    $AxCandidateListFiltered \leftarrow$
       $filterInertia(AxCandidateList, inertiaFilterRatio)$;
                                      // Score remaining approximate candidates
12:    $AxCandidateAL \leftarrow []$;
13:    for $AxCandidate \in AxCandidateListFiltered$ do
14:        $AxCandidateAL.append(evaluateAL(AxCandidate, testDataset))$;
15:    end for
                  // Select the Pareto efficient candidates w.r.t. AL and $b_{index}$
16:    $\phi \leftarrow ParetoSelect(AxCandidateList, AxCandidateAL)$
17: end for
```

than the two others, due to its computational simplicity and good top-1 accuracy. The *inertiaFilterRatio* used is the same as those used in experiments of the Section 4.4.

The Figure 4.8 shows the results of the multi-objective experiments conducted on the MobileNetV2 [13] with the ImageNet [18] dataset. The chart reports on the X-axis the Compression Rate (Equation 4.1) and the Y-axis shows the AL. Three Pareto frontiers corresponding to different *inertiaFilterRatio* values are shown. As expected, a higher CR is achieved at the cost of higher AL and *vice-versa*. From these results, it is now possible to select among different solutions presenting a trade-off between AL and CR.

Figure 4.8 also compares the results of Algorithm 3 with the results of Algorithm 2. We can see that except for *inertiaFilterRatio* = 5% inertia filtering value, the solutions obtained with Algorithm 3 dominate those of Algorithm 2.

The second point of the results shown in Figure 4.8 is that the

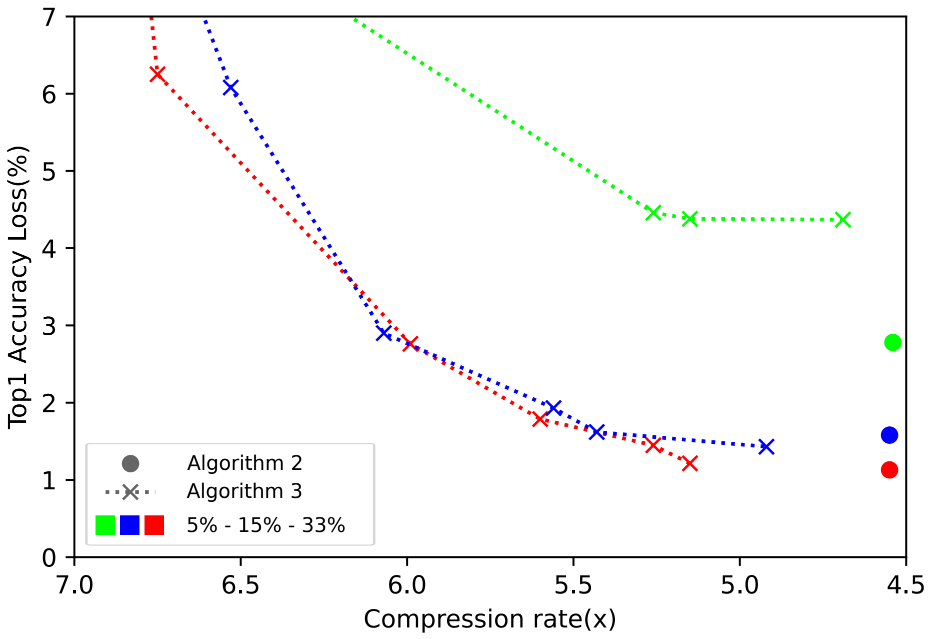


Figure 4.8: Multi-objective explorations (Algorithm 3) for MobileNetV2 [13] on the ImageNet [18] dataset with 3 different $inertiaFilterRatio$. Comparison with results obtained with the Algorithm 2.

$inertiaFilterRatio$ value impacts the results, again this was expected because using a smaller filtering value can hide interesting approximate candidates. On the other hand, we can note that the Pareto frontier obtained using $inertiaFilterRatio = 15\%$ of filtering is very close to the one related to $inertiaFilterRatio = 33\%$. This means that inertia is an efficient way to estimate the AL when WS is applied. Figure 4.9 reports the relative execution time required to obtain the three Pareto frontiers. As already pointed out in Figure 4.4, the scoring time is still dominating the overall execution time. Finally, we can safely use $inertiaFilterRatio = 15\%$ of filtering to achieve results as precise as the $inertiaFilterRatio = 33\%$ but with $2\times$ less of execution time. Once again, this confirms that the use of inertia as a sensitivity metric allows a significant speedup and paves the way to more complex exploration for WS-based approximation.

A comparison of the relative execution time of the three algorithms is presented in Figure 4.10. Looking at the relative values, we can note that executing the Algorithm 3 with $inertiaFilterRatio = 5\%$ is $2\times$ less computation-intensive than running Algorithm 1 on the same CNN, and is about the same computation intensity as running Algorithm 2. Whereas the execution of Algorithm 3 with $inertiaFilterRatio \geq 15\%$ is far more computation-intensive than others algorithms and this is mainly due to the population size during the exploration

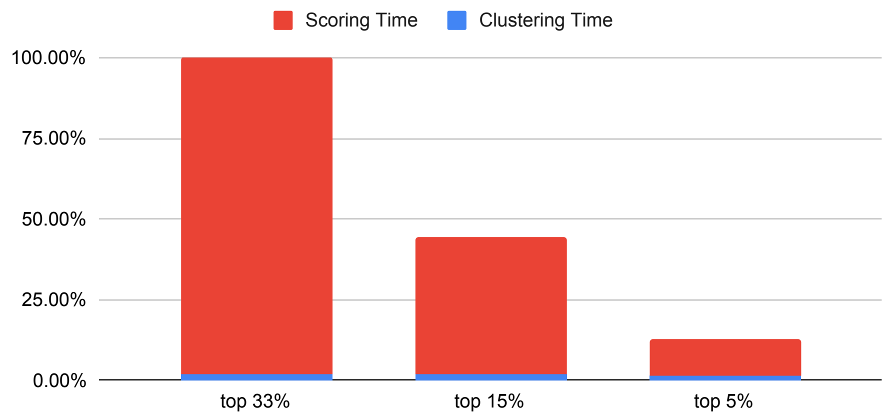


Figure 4.9: Profiling simulation time between the k-means clustering application and the scoring of approximate candidates using the validation dataset for Algorithm 3.

tending to be higher.

These results prove that (1) Using a multi-objective exploration algorithm is possible and show the trade-off between AL and CR; (2) It is possible to execute the multi-objective in affordable time by constraining the number of scored approximate candidates and the size of the population at each iteration. The next section compares the results of the proposed solution with state-of-the-art WS compression techniques and discusses the limits of the greedy approach.

4.6 Conclusions

The results presented in this chapter prove the benefits of optimizing the number of shared weights per layer as opposed to the homogeneous number of shared values used in the literature. The very simple greedy optimization is sufficient to solve this optimization problem in a reasonable time. It is also possible to rely on the inertia as a proxy metric for the AL of an approximate candidate, allowing to greatly reduce the number of required scoring, and thus the exploration time. Using a second proxy for the CR, it is also possible to perform multi-objective exploration, taking into account both the AL and the CR as optimization metrics. The last allows to explore the trade-off between AL and CR, this can prove very useful for application design.

It means that during the exploration of the layer i, there is no interest in the previous layer AL vs CR sensitivity, meaning that a sensitive layer may be over-approximated, and a resilient layer under-approximated, or both. This can be clearly observed in Figure 4.11, presenting the resulting $k_{tuple} = \{k_i\} \forall i \in [1, N]$ of the Pareto efficient candidate obtained at the end of Algorithm 3.

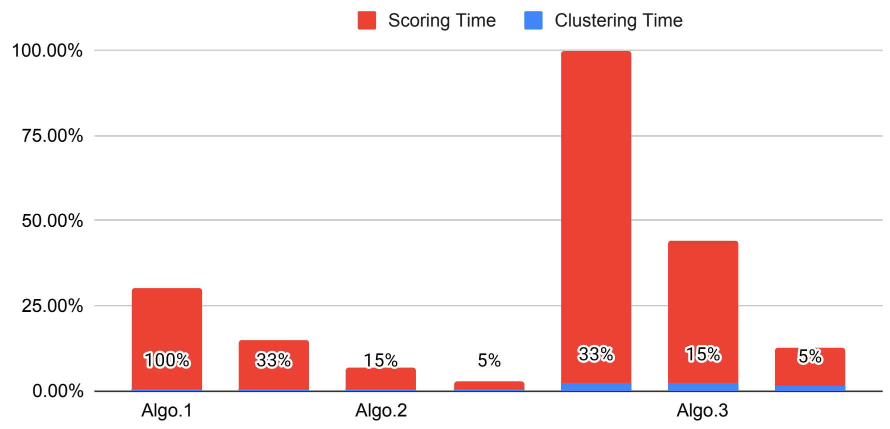


Figure 4.10: Profiling relative simulation time between the Algorithm 1, Algorithm 2, and Algorithm 3 on MobileNetV2 [13] on the ImageNet [18] dataset.

Candidates are sharing most of the k_{tuple} values, and this is due to the use of local b_index as a proxy for compression. From a decision-making perspective, the proposed greedy approach show over-exploitation and a lack of exploration. The candidate shown in blue in Figure 4.11 is the one featuring the higher CR with also the worst AL.

The greedy approach proposes to solve the global-optimization problem by focusing only on local optimization, whereas it has the advantage of being simple, it has no way to avoid falling into local minima. For example, some layers are very sensitive to approximation and have a poor correlation between AL and inertia, like the first layer of the MobileNetV2 [13] presented in Figure 4.6. These very sensitive layers would benefit from a greatly reduced approximation, whereas others would allow for more aggressive compression, the greedy algorithm can, to a limited extent, detect these constraints. But the chosen b_{index} metric used in the Pareto optimal selection of population during the multi-objective exploration presented in Algorithm 3 fails to capture the behavior of the CR of the approximated CNN during the exploration.

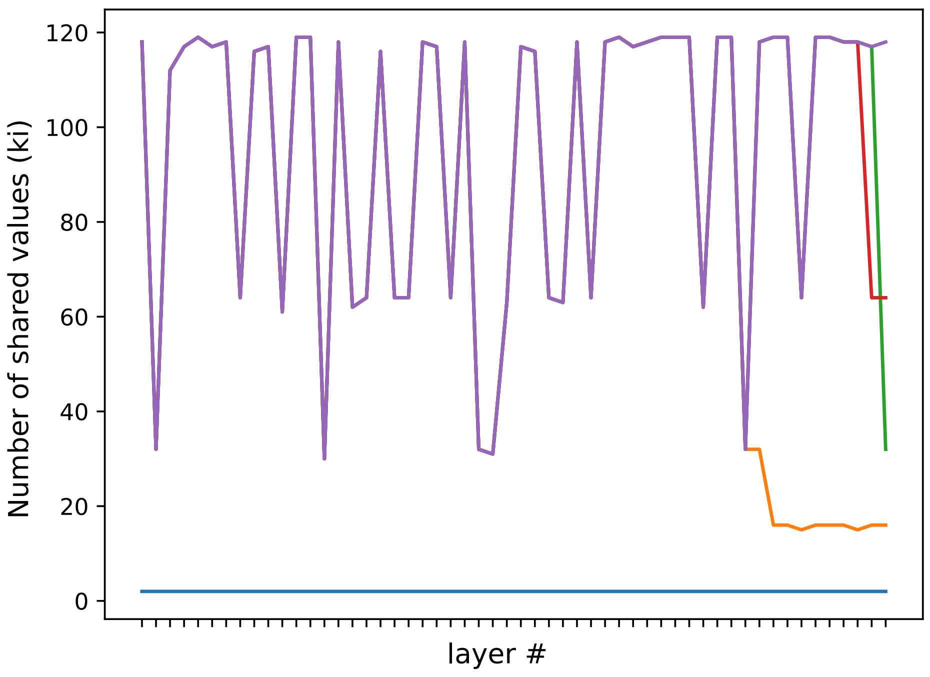


Figure 4.11: Resulting Pareto efficient k_{tuple} after the execution of Algorithm 3 on MobileNetV2 [13] on the ImageNet [18] dataset. Each color represent a different approximate CNN candidate obtained

Chapter 5

Exploring a Heuristic Approach to the Optimization problem

As it has been explained in Chapter 4, it is possible to address the optimization of the number of shared values for each layer of a trained CNN using a simple greedy approach. But this approach has shown its limits when it comes down to escaping local-minima and solving the problem with model-wide metrics. This is why evaluating the AL and CR of the whole approximated CNN is required to avoid local-minima and have a better representation of the optimization problem. It is then possible to adopt a heuristic approach to solve the optimization problem and this will be detailed in this chapter. Considering that none of the greedy exploration performed in Chapter 4 was able to find an approximate candidate complying with the MLPerf [7] quality target on the challenging MobileNetV2 [13], this network will be the baseline model for this whole study. After a brief statement of the DSE complexity in Section 5.1. The reduction of the search space to an locally-best-found number of shared values for each layer is presented in Section 5.2. The use of different techniques for exploring the combination of this reduced search space is studied in Section 5.3 with an extension including the use of statistical regression to accelerate this exploration. The proposed approach is applied to several state-of-the-art CNN to enforce scientific proof in Section 5.4. A critical analysis of the proposed compression technique is given in Section 5.5, and finally, conclusions are given on this divide & conquer strategy in Section 5.6.

5.1 DSE complexity

Using the notation from Chapter 4, for a given CNN with N layers, let k_i be the number of shared values of the layer i. k_i is bounded to a set of val-

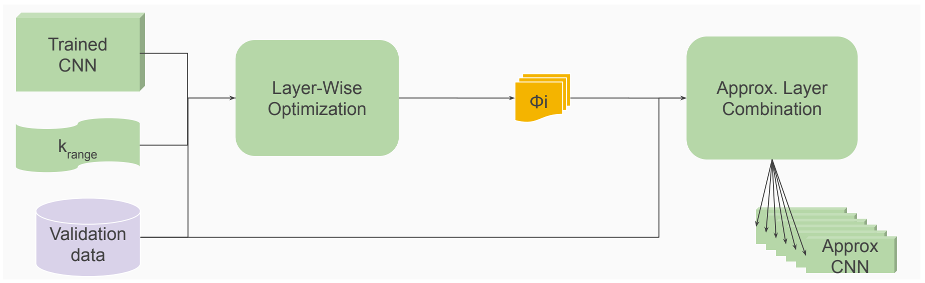


Figure 5.1: Conceptual view of the proposed weight-sharing optimization two-steps approach.

ues k_{range}. Accordingly, an approximated CNN can be characterized by its $k_{tuple} = \{k_i\} \forall i \in [1, N]$ representing the number of shared values for each layer. From the k_{tuple} it is possible to measure both (1) the AL, by scoring the approximated CNN on a large and representative test dataset, and (2) the CR, using Equation 4.1

Exhaustively exploring every possible k_{tuple} permutations results in the following time complexity:

$$\mathcal{O}(card(k_{range})^N). \tag{5.1}$$

To make the optimization problem tractable, a divide and conquer strategy can be adopted, splitting the exponential complexity problem of optimizing the number of shared values for each layer in two sub-problems: (1) reducing the search space by finding ϕ_i, the set of best-found k_i for each layer of the network and (2) applying a heuristic approach to finding a set of Pareto efficient k_{tuple} as a combination of $k_i \in \phi_i \forall i \in N$ considering the resulting AL and CR. A conceptual view of the approach is given in Figure 5.1. The next section gives details on this divide and conquers approach.

5.2 Layer-wise Optimization

The first sub-problem targets the local optimization of each layer. The goal is to reduce the search space by finding ϕ_i, the set of best-found k_i values, for each layer of the CNN. ϕ_i is characterized by (1) the AL obtained when layer i is approximated with k_i shared values while the other layers are not approximated, and (2) the number of bits required to store each index, defined as $b_{index} = \lceil log_2(k_i) \rceil$. As described in Section 4.5, the b_{index} is used as a local optimization objective as it is indirectly proportional to the CR of the layer, see Equation 4.1. The algorithm explores one by one the k_is in the user-defined k_{range} and for each, it evaluates both the AL and the b_{index} to then compose ϕ_i by selecting the candidate with the best AL per b_{index}. It is very similar to Algorithm 3, described in the previous chapter, with the difference that when

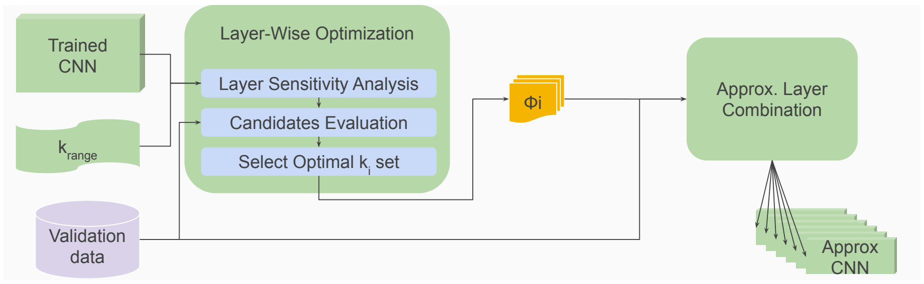


Figure 5.2: Conceptual view of our weight-sharing optimization heuristic, with layer-wise optimization detailed.

studying a layer, the others are kept unchanged, whereas, in Algorithm 3, the already clustered layers are kept clustered. The complexity of this search is linear to the number of layers N and the cardinality of the k_{range}, being:

$$\mathcal{O}(N * card(k_{range})). \tag{5.2}$$

Using the b_{index} as a proxy for the CR has the effect of limiting the cardinality of the ϕ_i, the output set of best-found k_i. Indeed, it is thus bounded as described in the Equation 4.7. There is also no need for inertia filtering as the linear complexity of the exploration has proven to be tractable for most CNNs. The complete process of the layer-wise optimization is described in Figure 5.2.

An example of such sub-problems for LeNet [15] on the MNIST [16] dataset is shown in Figure 5.3. Showing the results of the layer-wise optimization of the third layer of the CNN with a $k_{range} = [1, 1024]$ with a logarithm scale represented by 100 values. There is a need to separate legal and illegal candidates, on whether or not they are complying with the MLPerf quality constraints. For this particular example, there is a high resilience of the layer to approximation, and thus, almost every candidate shows AL that complies with the MLPerf quality targets, except for one, shown in red. From the legal candidates shown in blue, it is possible to extract ϕ_i by selecting the k_i leading to the best AL for each different b_{index} value. The resulting best-found number of shared values that will be used in further steps is thus $\phi_0 = \{4, 6, 16, 26, 46, 77, 154, 451, 580\}$. By resolving the first sub-problem of layer optimization, one can obtain the ϕ_i shown in Table 5.1. Note that layer 1 contains very few weights so the k_{range} was cropped accordingly to the total number of weights. This particular example of layer-wise optimization allows for reducing the number of possible combinations from 10 billion to only 48600, resulting in a reduction of about 6 orders of magnitude.

A more challenging target is the MobileNetV2 [13] trained on the ImageNet [18] dataset. This CNN is considered in the light categories of the MLPerf, meaning that the quality target is 98% of the baseline accuracy. This experiments uses the same $k_{range} = [2, 1024]$ that was used previously. To reduce the scoring time, a fast approximate scoring is performed, as described in Appendix A.3. Figure 5.4 shows the output of the layer-wise approximation

Layer	ϕ_i	$card(\phi_i)$
1	{2, 4, 6, 9, 23, 36}	6
2	{4, 7, 14, 18, 38, 99, 240, 329, 618}	10
3	{4, 8, 16, 19, 34, 99, 175, 272, 658}	9
4	{2, 4, 7, 15, 26, 64, 99, 145, 309, 618}	9
5	{2, 3, 5, 11, 23, 43, 82, 136, 329, 658}	9

Table 5.1: Selected ϕ_i during the layer-wise optimization for each layer of the LeNet [15] CNN on the MNIST [16] dataset.

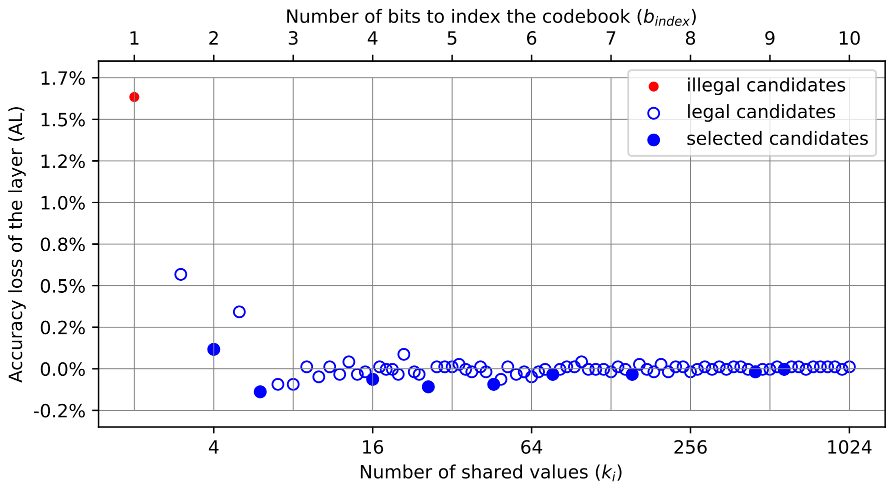


Figure 5.3: Studying the third layer of LeNet [15] sensitivity to compression by varying the number of shared values and measuring the resulting AL.

step for the first layer as an example. In the figure, illegal candidates (red circles) are the solutions leading to an AL greater than the quality target, while the legal candidates (empty blue circles) are the solutions within the AL constraint. Among the legal candidates, the selected candidates (full blue circles) are those leading to the best AL within each b_{index}. As shown in the figure, $\phi_i = \{16, 32, 39, 79, 206, 371, 569\}$ contains a significantly smaller subset of the initial k_{range}. This is exactly the objective of this layer-wise optimization, reducing the search space to only the most interesting candidates. Due to the larger number of layers compared to LeNet [15], the search space reduction is even greater.

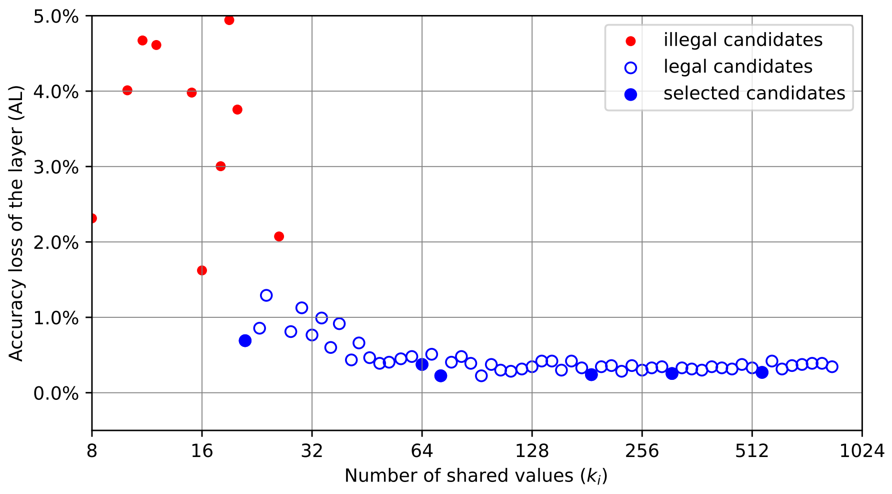


Figure 5.4: Results of the Layer-Wise exploration on the first layer of the MobileNetV2 [13], AL is obtained on a small subset of the validation set (10%).

5.3 Approximated Layer Combination

5.3.1 Stating the combination problem

The second sub-problem is to find the Pareto efficient combinations of approximated CNNs that can be described by their $k_{tuple} = \{k_i \in \phi_i\} \forall i \in [1 : N]$. The still large number of possible combinations:

$$\mathcal{O}(\prod_{i=1}^{N+1} |\phi_i|) \tag{5.3}$$

makes the exhaustive search far from tractable for a large network.

Still, for LeNet [15], the reduced number of layers ($N = 5$) allows for the exhaustive search of all the layer combinations to validate the precision of the prediction model. The LeNet [15] used as the baseline input CNN is a self-trained model achieving 0.9% top-1 accuracy on the MNIST [16] dataset, the selected MLPerf quality target is 99% of the baseline accuracy. The combination of the different $k_i \in \phi_i$ into k_{tuple} represents 48600 possible candidate solutions. To validate the proposed method the scoring of each of the solutions has been performed, taking roughly 2 hours on our GPU server (see Appendix A.2). From the results, it appears that almost 83% of the candidate solution leads to AL complying with the selected MLPerf quality target, among these, 9 of them are Pareto efficient, and thus, the most interesting for our optimization problem. Figure 5.5 shows the resulting AL and CR obtained from scoring these 48600 possible combinations. Illegal candidates that do not comply with the MLPerf

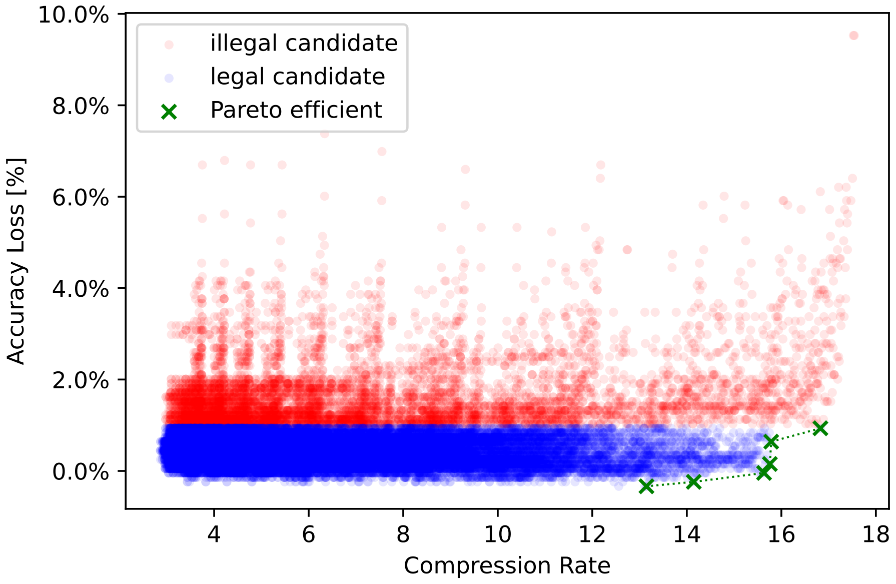


Figure 5.5: Exhaustive search of the 48 600 possible combinations of $k_i \in \phi_i$ for LeNet [15] on the MNIST [16] dataset.

quality targets are shown in red, and legal candidates in blue, it is interesting to see that among these legal candidates, a few of them are Pareto dominating the others. Some candidates achieved slightly better accuracy compared to the baseline (negative AL), the largest absolute improvement is 0.3% and is probably an artifact of noise-induced on the trained weights.

As the search space size is exponential to the number of layers N, it is not possible to explore it using an exhaustive search for large CNNs. Like MobileNetV2 [13] with 53 layers, giving a time complexity of about $\mathcal{O}(n^{53})$. To address this problem, it is proposed the use of a meta-heuristic algorithm, to find a set of Pareto efficient k_{tuple} w.r.t. AL and CR. From the large collection of meta-heuristics algorithms, there is a need for selecting a multi-objective (AL, and CR) population-based algorithm to obtain a set of Pareto efficient approximated CNNs. It is proposed to use the genetic algorithm NSGA-II [25] (see Appendix A.1.2) that uses a Pareto efficiency criterion to select the best candidates from a population and apply both random mutations to benefit from exploration improvements, and random breeding, to benefit from exploitation improvements.

There is a need to represent the proposed problem in generic optimization terms to solve it using NSGA-II. The objective metrics are both the AL, and the CR of the approximated CNNs. The parameters characterizing a candidate are the values of the k_{tuple} and are bounded to the indexes of the respective corresponding ϕ_i. To evaluate the CR, it is possible to use the analytical Equation 4.1, but evaluating the AL requires applying the clustering corresponding to the k_{tuple} and performing the scoring of the approximated CNN over the validation dataset. A genetic algorithm such as the NSGA-II requires to define a population size P and to iterate a certain number of times $\#it$ to converge towards a Pareto efficient population. Each of the iterations has the complexity of $P \times N$ layer clustering and P approximated CNN scoring. the resulting complexity of the NSGA-II algorithm is:

$$\mathcal{O}(P \times \#it), \tag{5.4}$$

it has bounded and predictable time complexity whereas the exhaustive search has a time complexity exponential to the number of layers N.

Comparing the proposed NSGA-II [25] exploration with the exhaustive search shown in Figure 5.5 for LeNet [15] trained on the MNIST [16], by using a population $P = 100$ and the number of iterations is $it = 100$ allows for quickly evaluating the fitness of the exploration. Figure 5.6 shows the resulting difference between the two Pareto fronts obtained for the legal candidates. Despite the NSGA-II [25] missing some of the best candidates, it can capture a proper representation of the design space. Regarding the convergence of the NSGA-II [25] algorithm, Figure 5.7 shows the history of the same experiment, and it is clear that the algorithm has converged as Pareto front are confused from the 10^{th} iteration. The direct conclusion of the study of this history is that increasing the number of iterations will not allow the heuristic for reaching the real Pareto-efficient candidates found by exhaustive search.

Tuning the NSGA-II hyper-parameters requires some manual exploration, the first need is to tune P, and $\#it$. The experiments are conducted on MobileNetV2 [13] using the reduced search space obtained in the experiments conducted in Section 5.2. The tested hyper-parameters set is $P \in 100$, and $\#it = 10 \times k \ \ \forall k \in [1, 10]$, there is a need to run only two experiments with varying P and using the max $\#it$ values, and reporting the results every $10\#it$. Allowing us to see the Pareto front evolution for both experiments. From the results shown in Figure 5.8, it is possible to note that the Pareto improvement at $\#it = 100$ over the random init at $\#it = 1$ is very significant, resulting in more than $8\times$ compression while complying with the MLPerf [7] quality constraints in terms of accuracy. These results are very compelling since they are proving that the compression of a challenging CNN such as MobileNetV2 [13] can be achieved using this approach.

The results of the NSGA-II [25] exploration are acceptable, but the exploration is very limited by the computational complexity of the clustering and scoring steps, composing the evaluation of the AL for candidates approximated CNNs. The number of approximated CNN evaluations that have been necessary is $P \times \#it = 10000$ and has taken a total of 6.4 hours for scoring on the GPU server (see Appendix A.2). Using both approximate scorings on 10% of the dataset and batch scoring to reduce the computational complexity of the scoring (see Appendix A.3 for more details). As an example, an exploration with $P = 100, \#it = 100$ requires the evaluation of $10k$ approximated CNNs,

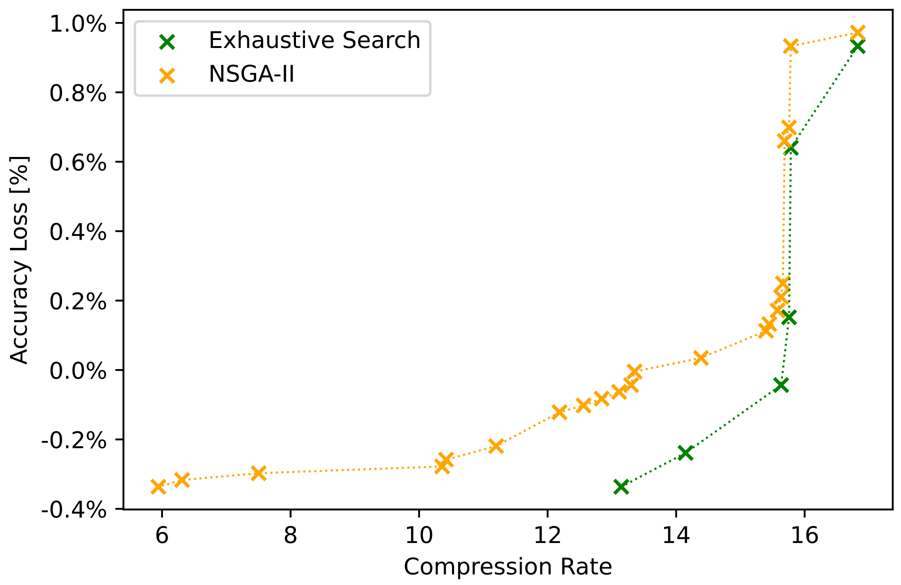


Figure 5.6: An exhaustive search of the 48 600 possible combinations of $k_i \in \phi_i$, and an NSGA-II [25] exploration for LeNet [15] on the MNIST [16] dataset.

resulting in more than 10 hours of evaluation considering the use of a 10% subset of the validation dataset for scoring and scoring the approximated CNNs in batch (see Appendix A.3 for more details). A possible solution is to make use of statistical evaluation solutions instead of computational evaluation, allowing to further improve the results obtained by allowing a more comprehensive exploration.

5.3.2 Statistical evaluation of the approximations

To alleviate the cost of the computationally intensive clustering and scoring step for measuring the AL of an approximated CNN, it is possible to use a statistical model. When studying a single layer, it is possible to use a proxy metric to evaluate an approximated CNN, like the inertia that was used in Chapter 1. Obtaining a statistical approximation of the AL for a given approximated CNN requires aggregating the proxy metrics of each layer composing it. An experiment is conducted on varying the aggregation method used on the approximated layers proxy metrics for a set of approximated CNNs which is a random subsampling of 1024 approximated CNNs among all the possible combinations of k_{tuple} composed of $k_i \in \phi_i$ obtained from the layer-wise optimization. The baseline CNN is MobileNetV2 [13] trained on the ImageNet [18] dataset. Two naive aggregation metrics are explored, the sum of each approximated layer's inertia and the weighted arithmetic average of the inertia by the number of weights of

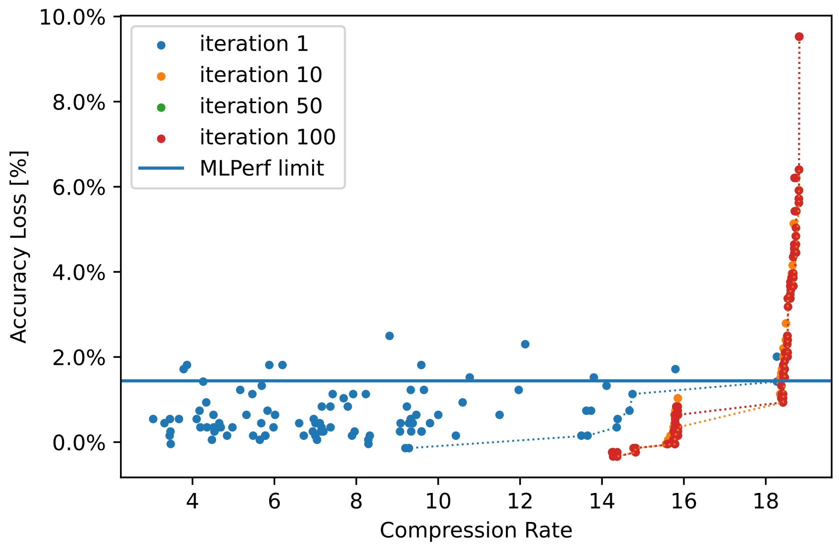


Figure 5.7: An NSGA-II [25] exploration for LeNet [15] on the MNIST [16] dataset, with history displayed. AL is obtained by scoring on 10% of the dataset, explaining the noise compared to Figure 5.6

the layer. Results are shown in Figure 5.9, it is clear that there is no correlation between the aggregated metrics and the top-1 AL.

Has been shown in Figure 4.6 the inertia and AL correlation is very different in two different layers, a direct implication is that it is not possible to make any assumption of the AL with naive metrics aggregation like the sum or the weighted average. There is a need for a more complex aggregation function. Taking into account the resilience of each layer to approximation is possible by using a coefficient for each layer, this is exactly what is done with multivariate linear regression. The resulting regression model follows the formula:

$$AL(k_{tuple}) = \sum_{i=1}^{N+1} \alpha_i * inertia_i, \tag{5.5}$$

with $inertia_i$ being the measured inertia during the local optimization for the layer $i \in N$ compressed using k_i shared values, and α_i is a trained coefficient. The linear regression model gives a statistical approximation of the resulting AL for CNN compressed with a specific k_{tuple}. But there is a need to train the *alpha*s coefficients, to that extend, training data must be collected.

Training data collection requires to sample the search space, to that extent, a large collection of Design of Experiments (DoE) techniques can be applied, from the naive uniform random sub-sampling already used for the experiment

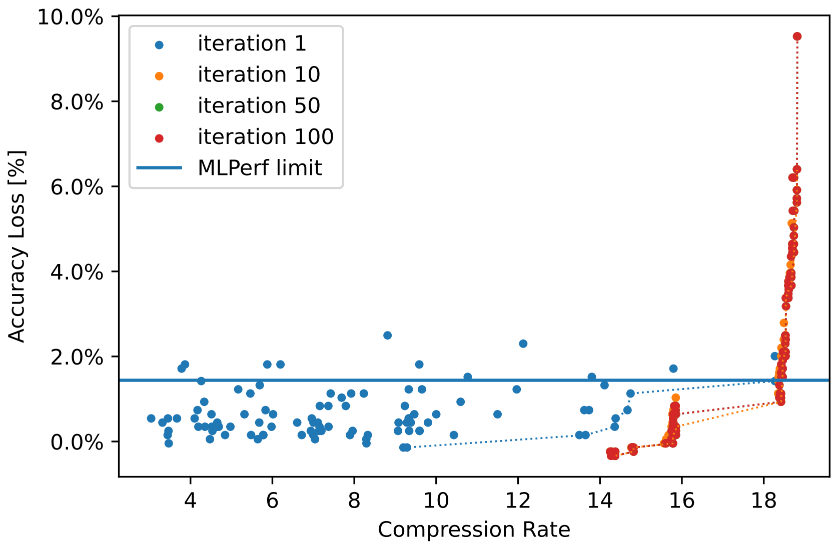


Figure 5.8: Results of the meta-heuristic exploration of the combinations for MobileNetV2 [13], AL is obtained on a small subset of the validation set (10%).

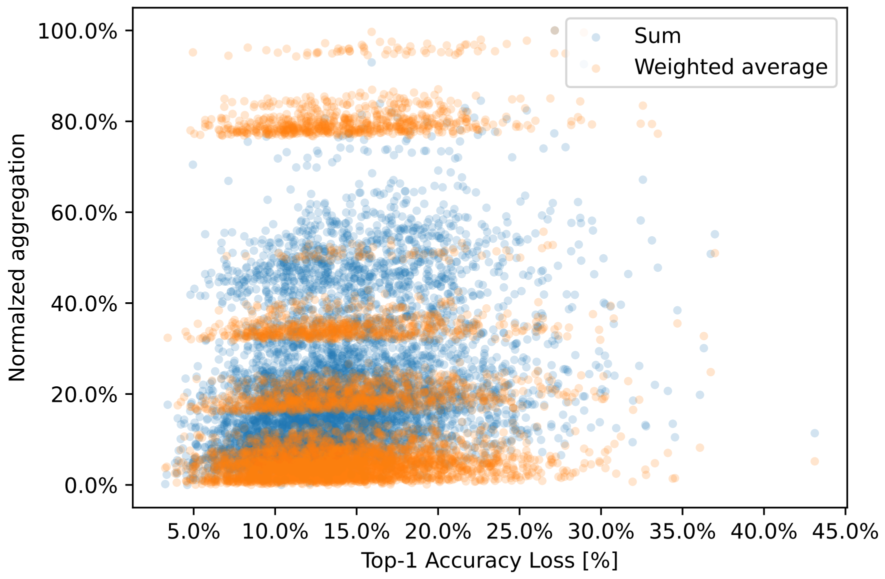


Figure 5.9: Correlation between the sum, and the weighted average of the inertia with AL for MobileNetV2 [13].

Sampling method	Regression $\mathbf{R^2}$	RMSE	MAE
Latin hypercube (simple) [165]	0.85	0.021	0.016
Latin hypercube (space-filling) [165]	0.81	0.021	0.016
Random k-means cluster [14]	0.92	0.017	0.012
Maximin reconstruction [166]	0.81	0.022	0.016
Halton sequence based [167]	0.84	0.021	0.017
Uniform random	0.81	0.021	0.016
Box-Behnken [164]	0.86	0.0045	0.0035

Table 5.2: Measuring the R^2 error obtained by training a multivariate regression model to predict the accuracy of the network from the inertia of the composing layers. Using multiple DoE techniques using MobileNetV2 [13] on the ImageNet [18] dataset. Taking the reduced search space, ϕ_i, obtained from layer-wise exploration as the input search space to be sampled.

described in Figure 5.9, to a more complex sampling method optimizing the selected samples spanning across the search space. To compare the quality of different DoE techniques, it is required to run experiments, taking the same experimental setup as the one of Figure 5.9, using MobileNetV2 [13] and the ImageNet [18] dataset. A collection of representative DoE techniques, are tested to sample the search space, if the techniques allow selecting the number of samples, it is set to 5513 which is the size of the Box-Behnken [164] for the specific problem, allowing for a fair comparison with the same number of samples for each technique. Table 5.2 presents the obtained results, the metric of evaluation for the DoE techniques is the coefficient of determination, noted R^2, of the multivariate linear regression model trained on the sampled search space, using 80% of the samples for the training of the regression model, and the 20% remaining for measuring the R^2, the Root Mean Squared Error (RMSE), and the Mean Absolute Error (MAE). From the obtained results, it appears that most sampling methods give R^2 in the range $[0.80, 0.86]$ whereas the random k-means cluster-based selection can give better results on the test dataset, with $R^2 = 0.95$. Due to the very large size of the search space, most techniques seem to have failed to capture the exploration range and are performing almost the same as random sampling (Uniform random in the table). From these experiments, it is possible to conclude that spreading the samples across the search space is difficult, but the random k-means cluster-based technique seems to perform better than others, this sampling technique will be used in subsequent experiments.

Taking a look at the obtained approximated CNNs from sampling the search space with the different DoE techniques, it appears that each produces a similar Pareto front in the objective space. The results are shown in Figure 5.10, representing the achieved CR and the AL of the Pareto efficient sampled approximated CNNs. A first observation is that most of the techniques performed

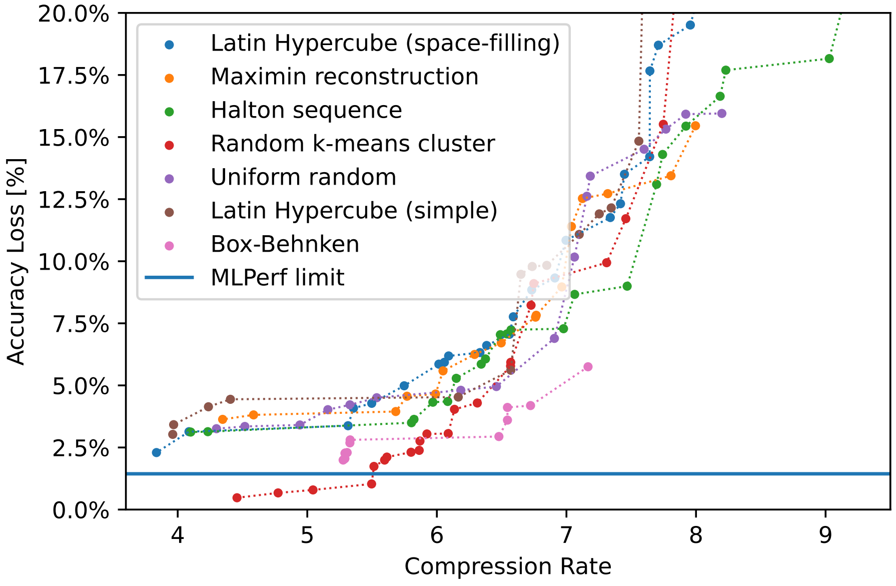


Figure 5.10: Comparing the Pareto efficient approximated CNN obtained with different DoE techniques for MobileNetV2 [13].

better than the uniform random, except for the box-Behnken, despite achieving an impressive R^2 score, it lacks some performance in the objective space, meaning that the resulting linear regression model will lack knowledge of the region of interest featuring AL complying with the MLPerf quality target. From the others techniques performing better than random uniform, it seems that random k-means is the best technique regarding the Pareto front. Unless specified, the random k-means sampling technique will be used in further experiments.

Further to the search space sampling technique, it is interesting to investigate the metrics that are used as independent variables for the multivariate linear regression model. For now, the conducted experiments relied on the use of inertia as it has been proven to correlate with the accuracy in Figure 4.6. Other metrics can also be investigated, like the k_i of the layers, or the measured AL of the approximate layer during the layer-wise exploration, $AL_{local}(k_i)$. To investigate the fitting of the metrics for regression, it is possible to compare the obtained R^2 of regression models trained using different metrics. To that extent, the same experimental setup is used (using MobileNetV2 [13] and the ImageNet [18] dataset). Table 5.3 summarizes the result of the comparison, using metrics obtained from a 5000 uniform random sampling of the search space reduced by the layer-wise exploration. It is clear that relying only on the k_i values is not enough to predict the behavior of an approximated CNN, whereas both $AL_{local}(k_i)$ and $inertia(k_i)$ give promising results. The inertia will be used in the subsequent experiments because it is less costly to measure

Metric	Regression $\mathbf{R^2}$
k_i	0.19
$inertia(k_i)$	**0.81**
$AL_{local}(k_i)$	0.79

Table 5.3: Measuring the R^2 error obtained by training a multivariate linear regression model to predict the accuracy of the network from various metrics. Using MobileNetV2 [13] on the ImageNet [18] dataset.

as it does not require a costly scoring step on the validation dataset.

Once trained, the regression model can be used to perform the prediction of the AL of an approximated CNN with a given k_{tuple} from the corresponding inertia values. It is possible to use such a model to accelerate the convergence of a heuristic algorithm such as the NSGA-II [25] used earlier. To assert the feasibility and measure the resulting speedup and quality of the results, it is required to run experiments. Using the same experimental setup (using MobileNetV2 [13] and the ImageNet [18] dataset). Figure 5.11 compares the obtained population at the end of the NSGA-II [25] with evaluating the AL both with the dataset scoring and with the statistical regression model. The Pareto front is very similar but the regression allows for the convergence of the algorithm toward the region of interest located under the MLPerf [7] quality constraint. The execution of the NSGA-II [25] optimization with regression takes only 1 minute instead of the 6.4 hours with scoring, resulting in a 380× speedup, even considering the 3.2 hours of the search space sampling required to train the regression model, the speedup is still 2×.

Further to the comparison of the heuristic exploration with and without the regression, it is also possible to evaluate the Pareto front improvement compared to the one obtained with the search space sampling used during the training of the linear regression model. Figure 5.12 shows that there is a clear improvement in the samples obtained with the NSGA-II [25] over the samples obtained with random k-means sampling of the combination space.

Fig. 5.13 shows a conceptual view of the proposed heuristic for the approximated layer combination. The input is a set of ϕ_i selected locally-best-found approximated version of layers. The output is a set of Pareto efficient approximated CNNs representing the trade-offs between the AL and the CR. The figure also shows how each sub-problem is solved in multiple sub-steps. A DoE technique is used to sample the search space, the candidates of the sampled search space are evaluated and a regression model is trained to predict the AL of each approximated CNN, then, the regression model is used to accelerate the meta-heuristic search of the Pareto efficient approximated CNN regarding AL and CR.

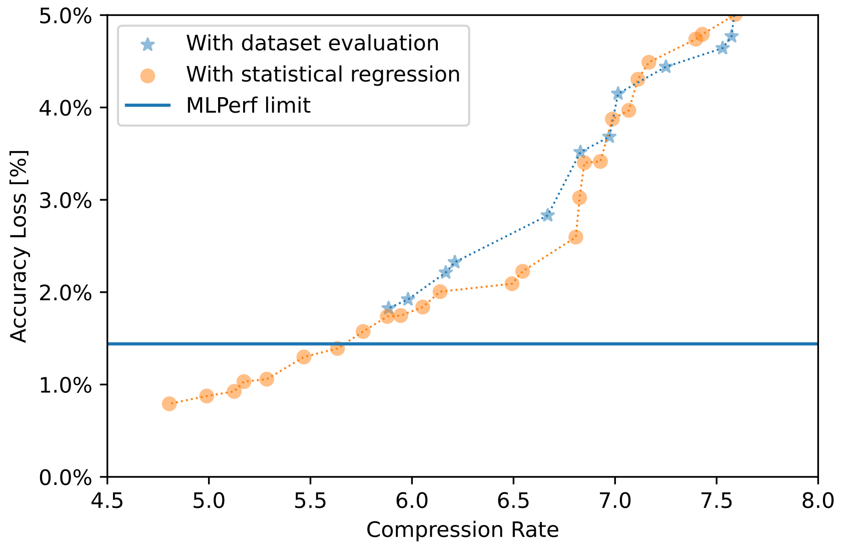


Figure 5.11: Comparing the results of the meta-heuristic exploration of the combinations for MobileNetV2 [13] with and without the regression model to predict the AL.

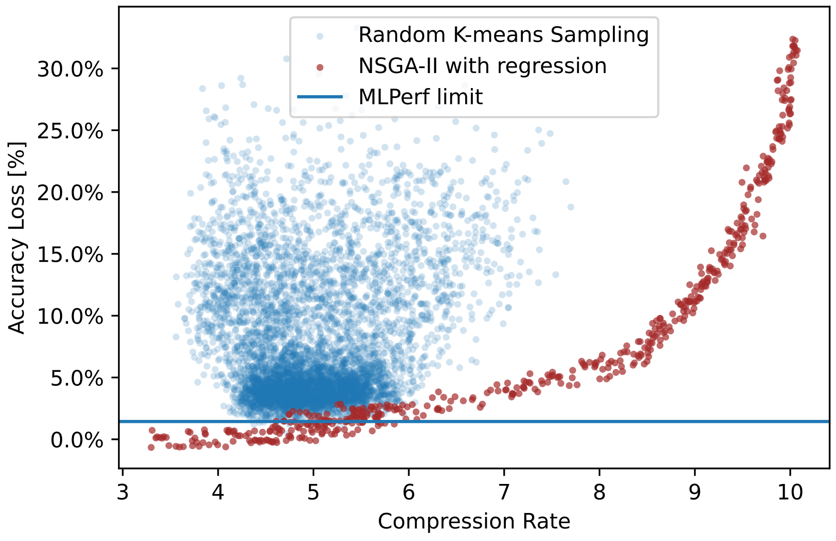


Figure 5.12: Comparing the results of the meta-heuristic exploration of the combinations for MobileNetV2 [13] using regression with the results obtained from the search space sampling, samples are scored on 10% of the ImageNet [18] validation dataset.

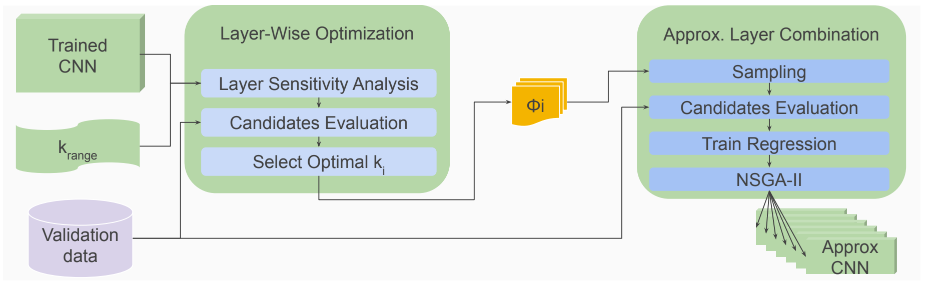


Figure 5.13: Conceptual view of our weight-sharing optimization heuristic, with approximated layer combination detailed.

5.4 Applying the Proposed Method on Several SoTA CNNs

The same compression flow is applied to all CNNs with the following parameters: the dataset used for scoring during the exploration is a subset of the ImageNet [18] validation dataset containing 10% of the 50,000 samples. The k_{range} used during the layer-wise optimization is a logarithmic range with 100 values between 2 and 1024. The number of samples used to train the linear regression model is 5,513, the selected DoE technique is random k-means sampling. The NSGA-II [25] algorithm is set to run with a population of 100 candidates for 500 generations with default mutation and crossover values of 0.2 and 0.9 respectively. All the reported AL values are measured on the entire validation dataset.

To evaluate the proposed heuristic approach, it is required to apply it to a representative set of recent image classification CNNs. The selected representative CNNs can be separated into two categories: (1) light CNNs are optimized to run on resource-constrained devices, whereas (2) heavy CNNs exclusively focus on top-1 accuracy. GoogleNet [27], ResNet50V2 [24], and InceptionV3 [28] belong to the heavy category, while MobileNetV2 [13] and the various EfficientNets [26] belong to the light category. This clustering has been performed following MLPerf [7] recommendations, each category has different quality targets (i.e., 99% and 98% of FP32 precision for heavy and light CNNs, respectively). For each targeted CNN, it is important to count the number of layers of interest that are targeted by the proposed compression method (fully connected and convolutional layers) composing the CNNs, because the count indicates the size of the search space, which grows exponentially with the number of layers.

Table 5.4 reports the results for the conducted experiments. The MLPerf [7] category of the CNN is reported first. The reported baseline memory in the table is the size required to store the 32-bit weights of the baseline CNNs in memory. The top-1 accuracy indicates the baseline top-1 classification accuracy. Finally, the min. and max. CR values represent the minimum and maximum CR values of the approximated CNNs found by the proposed compression method

Network	MLPerf category	#Layer	Baseline Mem. [MB]	Top-1 Accuracy [%]	CR min, max
GoogleNet	heavy	58	50	69.7	5.4
ResNet50V2	heavy	54	97	76.0	5.3, 5.6
InceptionV3	heavy	2.8	104	77.2	4.7, 5.3
MobileNetV2	light	53	13	71.9	4.4, 5.7
EfficientNetB0	light	82	20	76.4	4.5, 5.6
EfficientNetB1	light	116	30	78.4	4.3, 5.3
EfficientNetB2	light	116	35	79.8	3.5, 5.3

Table 5.4: Compression results on different CNNs on the ImageNet [18] dataset under MLPerf [7] quality target constraints.

complying with the corresponding MLPerf [7] quality target.

On each of the tested CNNs, the proposed WS method is always capable of significantly compressing the baseline reference, by consistently achieving over 5× CR. Importantly, the compression is achieved without requiring the intervention of an expert for manually tuning the k_is and without involving any retraining, fine-tuning, or calibration steps. Furthermore, there is little to no difference in the achieved CR for the different MLPerf [7] categories.

A graphical view of the results is shown in Fig. 5.14, with the light CNNs on the left and the heavy ones on the right. The figure depicts the association between the minimum memory size required to store all the weights and the top-1 accuracy of each CNN instance. Achieving the best trade-offs between the AL and the CR often requires exploring multiple CNN topologies with multiple levels of approximation. Accordingly, the proposed method finds Pareto optimal trade-offs between accuracy and memory footprint to facilitate the selection of the most suitable CNN instance for any given application.

5.5 Regression of the approximated CNN without layer-optimization

One can argue that a DoE technique can be used to sample the exhaustive search space instead of sampling the search space of the combination of $k_i \in \phi_i$ after the layer-wise optimization. To evaluate the quality of the regression model trained with different samples obtained using different DoE techniques, the same experiments as the one for Table 5.2 on MobileNetV2 [13] with the ImageNet [18] dataset are executed without prior layer-wise optimization knowledge. Table 5.5 reports the coefficient of determination R^2 for both experiments, with and without prior knowledge of the layer-wise optimization. The regression without the layer-wise optimization is outperformed by the one with layer-wise optimization. As the coefficient of determination states the variance expression, one can conclude that variance is better expressed when the design space is reduced before sampling.

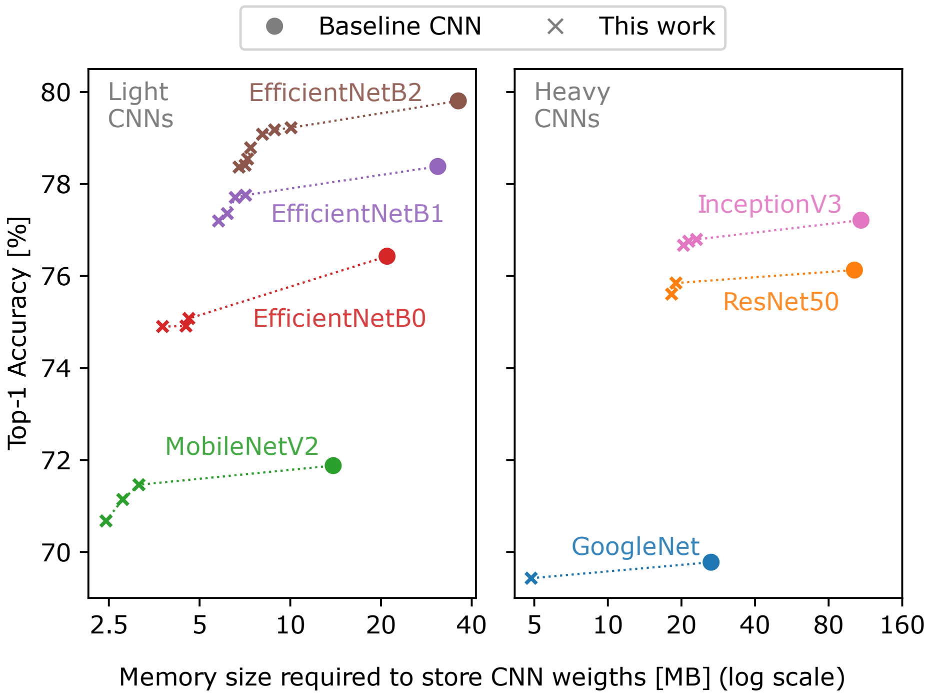


Figure 5.14: Comparison of different approximated CNNs characterized by their top-1 accuracy and memory requirements on the ImageNet [18] dataset.

5.6 Conclusions

The objective of this approach compared to the previously presented greedy approach in Chapter 4 is to use global objective metrics such as the CR and AL of the whole approximated CNN instead of local objective metrics that can fail to capture the behavior of the approximation. Using a two-step optimization process with first a search space reduction, and second, an efficient exploration based on the use of design space sampling and regression allows for such global optimization of the number of shared values as a vector and not as an iterative process.

The proposed approach was able to compress ImageNet [18]-class CNNs over 5×, for both lightweight and heavyweight application targets such as the representative MobileNetV2 [13] and ResNet50 [19]. Keeping the AL complying with MLPerf [7] quality target.

Regarding the obtained k_{tuple} at the end of the exploration, shown in Figure 5.15, the approximated CNNs do not share most of the k_i values as it was the case when executing the Algorithm 3, and as it can be observed in Figure 4.11. A conclusion is that the proposed algorithm can efficiently explore the search space at a higher level, with a balance between exploration and exploitation. This can be explained by the use of high-level objective metrics like

Sampling method	Number of Samples	Regression R^{2} with ϕ_i
Latin hypercube (simple) [165]	0.85	0.335
Latin hypercube (space-filling) [165]	0.81	0.47
Random k-means cluster [14]	0.92	0.48
Maximin reconstruction [166]	0.81	0.52
Halton sequence based [167]	0.84	0.40
Uniform random	0.81	0.47
Box-Behnken [164]	0.86	0.83

Table 5.5: Measuring the R^2 error obtained by training a multivariate regression model to predict the accuracy of the network from the inertia of the composing layers. Using multiple DoE techniques using MobileNetV2 [13] on the ImageNet [18] dataset. Taking the reduced search space obtained from layer-wise exploration (See \ref{}) as the input search space to be sampled.

the CR and AL directly during the exploration.

Despite the use of a meta-heuristic optimization, the proposed approach still depends on a k_{range} possible number of shared values, adding a human bias in the equation. It is clear that some of the layers in Figure 5.15 saturate at the maximum possible number of shared values, 1024, and would have potentially benefited from an increase of the search range, inducing enlarging the search space and the complexity of the exploration.

A first comparison of the proposed approach with some variations including the use or not of a design space sampling and a regression model for the exploration was proposed in Section 5.3. There is a need for further comparison with other meta-heuristic approaches, as well as other state-of-the-art WS and non-WS based compression approaches. This comparison is the object of the next Chapter 6.

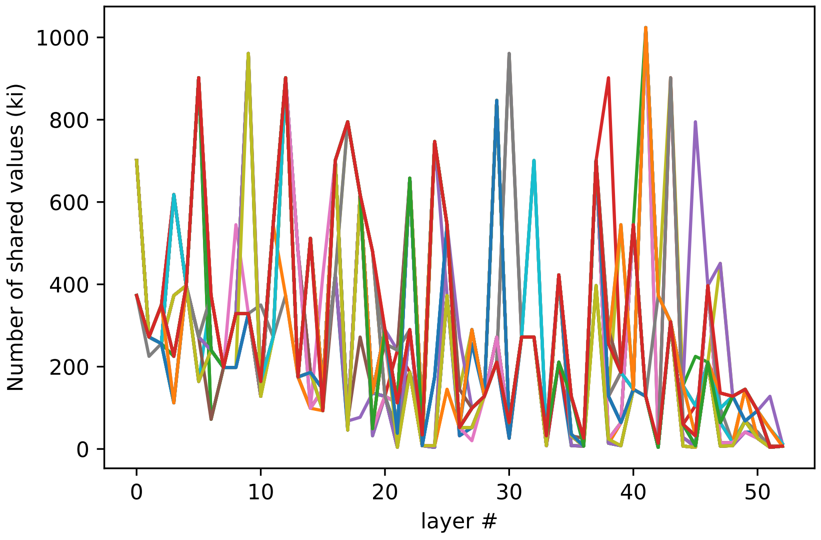


Figure 5.15: Resulting Pareto efficient k_{tuple} after the execution of the proposed divide & conquer approach on MobileNetV2 [13] on the ImageNet [18] dataset.

Chapter 6

Comparison state of the art compression techniques

A comparison with other compression techniques is required to quantitatively measure how the proposed approaches can efficiently explore the design space. The comparison will be made first in Section 6.1 on the same problem of tuning the number of shared values for each layer of a network using different approaches, including the approaches proposed in Chapter 4 and Chapter 5 with the state of the meta-heuristics approaches, such as the NSGA-II [25].Then the proposed compression flow will be compared with others WS techniques involving retraining in Section 6.2. Finally, an overture will be made by comparing the proposed approach with non-WS-based state-of-the-art post-training compression techniques in Section 6.3.

6.1 Comparison of the proposed flow with meta-heuristic optimization

The first proposed comparison of the proposed greedy and divide & conquer approaches, is with a pure meta-heuristic approach, the already introduced genetic algorithm NSGA-II [25].For a fair comparison, each algorithm is executed on the same hardware with the same Pytorch [56] backend, and the same baseline CNN, MobileNetV2 [13]. The exploration range is $k_{range} = [2, 1024]$, approximate scoring on 10% of the ImageNet [18] validation dataset is used, as well as batched scoring of the network.

The greedy exploration algorithm has been set to use a filter value *inertiaFilterRatio* = 15% when applicable, as it was proven in Chapter 4 that it provides the best trade-off between exploration time and exploration quality. The exploration range used is linear. Two different versions of the algorithm are applied: (I) single objective with the use of a proxy to filter out the worst candidate, as described in Algorithm 2; and (II) multi-objective with the use of a proxy to filter out the worst candidate and the local number of bits for reducing the complexity of the search algorithm, as described in Algorithm 3.

The same divide & conquer approach described in Chapter 5 is applied, with first a search space reduction with a layer-wise optimization and then a combination exploration with the use of a meta-heuristic algorithm. Two different processes are reported, (III) involving the use of approximate CNN scoring during the NSGA-II [25]; and (IV) involving the use of DoE to sample the search space and train a regression model, used during the NSGA-II [25] exploration. The parameters for the search space reduction are the same for both (III) and (IV), a logarithmic k_{range} is used with 100 points. Concerning the combination, the same mutation and crossover parameters are used, but the population size and the number of iterations are greatly reduced for (III) to allow for keeping the scoring count in a reasonable range. The population size and the number of iteration is respectively $(P = 100, \#it = 100)$, and $(P = 500, \#it = 200)$ for (III) and (IV).

The compared heuristic, (V) is a pure NSGA-II [25] exploration, the genetic algorithm can efficiently explore the DSE as it has been proven by the work in Chapter 5, here it is used without any prior search space reduction technique or even DoE for sampling the search space and training a regression model. The population size and the number of iterations is $(P = 100, \#it = 100)$, and the default mutation and crossover values are used.

id	Approach	Section	Scoring Count	Scoring Time [hh:mm]	Clustering Time [hh:mm]	Algorithmic Overhead	Total Time [hh:mm]
I	Greedy SO (proxy)	4.4	7 874	04:06	00:38	2.48%	04:51
II	Greedy MO	4.5	56 322	29:14	02:60	1.94%	32:51
III	Divide & Conquer	5.3.1	14 352	07:27	00:56	2.48%	08:35
IV	Divide & Conquer (regr.)	5.3.2	10 268	05:20	00:32	7.15%	06:19
V	NSGA-II [25]	-	10 098	05:15	00:60	2.41%	06:23

Table 6.1: Comparing the optimization computational cost for applying the 4 explored approaches on MobileNetV2 [13] trained on the ImageNet [18] dataset. A scoring on 10% of the validation set is used for all approaches.

Each approach has been implemented in the same framework and running on the same GPU server (see Appendix A.2) to guarantee execution time consistency. Table 6.1 reports the computation time for the different optimization approaches with previously stated hyper-parameters. The scoring count reports the number of approximated version of the CNN that is scored over the validation dataset. The scoring time and clustering time report the total time spent scoring or clustering an approximated version of the CNN. The algorithmic overhead reports the portion of the time spent in other tasks than scoring and clustering. Finally, the total time reports the whole duration of the complete optimization. There is a clear difference in exploration time between single candidate exploration, represented by (I), and population-based exploration, represented by (II-VI), single candidate exploration is faster, but lacks representation of the trade-off between AL and CR. From the population-based exploration, (IV) and (V) are the fastest approaches, with almost the same number of scoring around 10000, resulting in a very similar execution time. The main difference between (IV) and (V) regarding the execution time is the number of clustering and the algorithmic overhead, because in (IV), the acceleration of the NSGA-II [25] evaluation with the statistical representation of the design space allows for larger exploration with increased population size and iteration count, resulting in a larger algorithmic overhead. Whereas in (V), the evaluation of the candidates during the exploration requires more clustering steps than required during the search space reduction step of (IV), considering that every layer needs to be clustered for each approximated CNN in (V).

To compare the different heuristics on the quality of the exploration, there is a need to compare the obtained Pareto-fronts of approximate CNNs. Figure 6.1 shows the different Pareto-fronts obtained with each of the previously described heuristics I-V. Regarding the greedy algorithms, the single objective (I) was able to find the solution featuring the lowest AL among all heuristics, but this solution does not give a high compression rate, because the single objective is not capable of capturing the trade-off between AL and CR. Whereas the multi-objective approach (II) is not showing a competitive Pareto-front, mostly since local evaluation is not able to capture the behavior of the entire approximated CNN, and the drastic filtering required to keep the number of trials tractable hides potentially interesting candidates during the exploration. Considering the divide & conquer algorithms, the one relying on real evaluation of the approximate CNNs is showing results outside of the region of interest, with approximate CNNs not able to comply with the MLPerf [7] quality targets. The last comparison with the proposed divide & conquer with statistical regression approach (IV) is the NSGA-II [25] exploration (V), similar to (IV) but without the prior search-space reduction. The divide & conquer approach (IV) shows better exploration capabilities than the pure heuristic NSGA-II[25] (V), with about 20% higher CR under the same quality targets constraints. Furthermore, (IV) is particularly efficient at finding solution with lowest AL, which are the most interesting solutions.

The results shown in this section prove that the proposed approach is able to compete with meta-heuristic algorithms when it comes down to design space

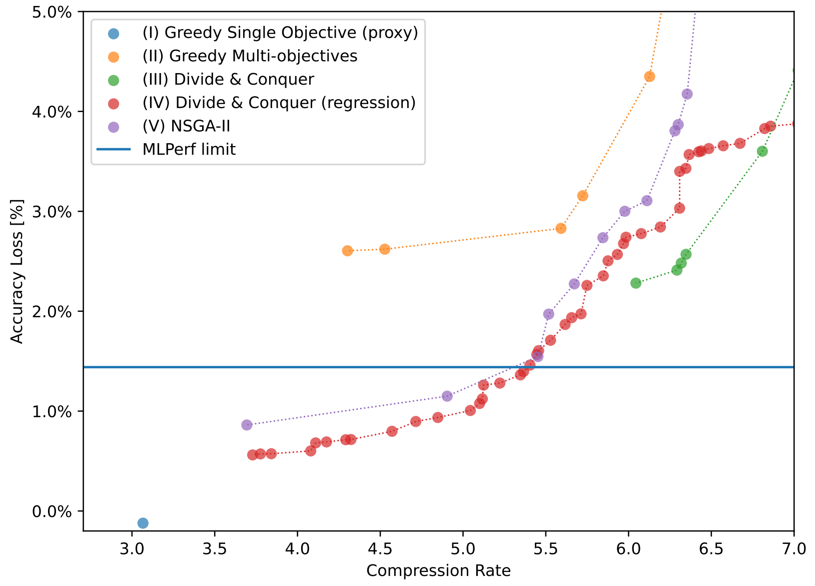


Figure 6.1: Results of the meta-heuristic exploration of the combinations for MobileNetV2 [13], AL is obtained on a small subset of the validation set (10%).

exploration efficiency. The proposed divide & conquer approach allows for appealing results and can be used to compress significantly any of the tested ImageNet [18] class CNNs.

6.2 Comparison with other WS techniques

To assess the quality of the proposed method to existing works, three recent WS methods using GoogleNet [27] on the ImageNet [18] dataset are used as comparison points.

Deep k-means [168] use the k-means objective metric (i.e., inertia, see Section 4.4) as a regularization term of the loss function used during training. Applying such WS-aware training encourage weight grouping during the training, which reduces the error introduced in the subsequent WS step. Deep k-means specify the number of shared values of each layer using a fixed *cluster rate* defined as the number of shared values k_i divided by the number of weights of the layer N. They report both the AL and the CR for GoogleNet [27] using their method with and without retraining, the reported results are used for comparison

The second method, named DP-Net [29], uses dynamic programming to achieve the clustering instead of the k-means clustering algorithm. This brings a performance improvement as well as a guarantee of the convergence of the clustering compared to the highly-variable k-means complexity. The method Also relies on a WS-aware training with a regularization term encouraging the weight grouping during the training, inspired by Deep k-means [3].

Both methods reported results on the GoogleNet [27] CNN with the ImageNet [18] dataset. The proposed Algorithm 2 is applied on the same CNN model and dataset. More in detail, we use the available GoogleNet from the Torchvision model zoo, achieving 69.78% top-1 accuracy on the ImageNet [18] dataset with standard input pre-processing including normalization, re-scaling, and center-crop. The $inertiaFilterRatio = 15\%$ has been chosen because it has been proven in previous sections that it can show the best trade-off between computation time and result. Finally, it is important to stress the fact that a fair comparison is only possible with Deep k-means [3] since it is the only approach that reports results without using retraining.

The Table 6.2 shows the results of the comparison. The first column depicts the method, the second specifies if the retraining phase is applied or not. Finally, the third and fourth columns report the AL and CR respectively. As already pointed out, the proposed approach does not use retraining, Deep k-means [3] give results with and without it, while DP-Net [29] always uses retraining. The proposed method achieves a higher CR than Deep k-means [3] at a lower accuracy loss, even when compared with the approach involving retraining. For similar CR (about 4x) the proposed approach achieved 0.83% AL while Deep k-means [3] achieved 1.95% of AL. The DP-Net [29] approach can find a more efficient solution by better leveraging the retraining phase.

Results of Table 6.2 showed that our approach is comparable with state-of-

Method	Retraining	Top-1 AL (%)	CR
Deep K-means(2018)	**No**	1.22	1.5
	No	3.7	2
	No	13.72	3
	No	48.95	4
	Yes	0.26	1.5
	Yes	0.17	2
	Yes	0.36	3
	Yes	1.95	4
DP-Net (2020)	Yes	-0.3	7
	Yes	1.56	10
Greedy (proxy)	**No**	0.83	4.6
Divide & Conquer	**No**	0.35	5.4

Table 6.2: Comparing our result with other weight sharing techniques on GoogleNet [27] on the ImageNet [18] dataset using Algorithm 2.

the-art, even if no retraining is applied. Unfortunately, no data are reported regarding the execution time of Deep k-means [3] and DP-Net [29] for further comparison. However, the execution time of the proposed Algorithm 2 took less than 6h on our on-premise GPU server (see Appendix A.2 for more details). On the same machine with the same CNN model and dataset, 5h corresponds to the time required to run about 5 epochs of retraining. We can thus project that the exploration time of the proposed approach is far less than the required retraining step of Deep k-means [3] and DP-Net [29], respectively 90 and 30 epochs.

6.3 Comparison with others non-WS-based CNN Compression Techniques

Further to compare the obtained results with similar meta-heuristics approaches and other WS techniques, it is possible to compare to state-of-the-art non-WS compression techniques such as pruning and quantization techniques. As it can be observed in Table 6.2, a fair comparison is only possible on post-training compression techniques, that is why only post-training pruning and quantization will be considered in this section. To that extent, three state-of-the-art post-training compression techniques have been selected for comparison: (1) Post-Training Pruning, (2) TFLite, and (3), PieceWise Linear Quantization.

The first comparison of the proposed method is with Post-Training Pruning (PTP) [30], a state-of-the-art technique that achieves compression via pruning. PTP includes MobileNetV2 [13]. In detail, PTP proposes a data-free weight pruning approach based on automatically-generated synthetic fractal images for retraining. This pruning step is then followed by post-training quantization, resulting in an 8-bit sparse matrix representation. Theoretically, this would lead to 6.7× CR on MobileNetV2 [13]. However, in practice, it is also necessary to store indexes next to the sparse weight values. Without those indexes, it would be impossible to recover the position of the weights. Accordingly, in this comparison, the ideal case of just adding one 8-bit index per weight is considered.

The second comparison is with the post-training quantization implemented in the Tensorflow Lite [31] framework. The quantization scheme is based on affine mapping of the real value onto quantized values, using two constant parameters to manage the scale and the offset of the values. A different set of quantized parameters is used for each layer, it is obtained using a small set of samples allowing to perform a calibration of the quantized values. This quantization scheme has the advantage of requiring very minimal hardware adaptation to benefit from compression and acceleration. The paper includes compression results on MobileNetV2 [13].

The third comparison is with PieceWise Linear Quantization (PWLQ) [32], a post-training quantization technique designed for tensor values showing bell-shaped distribution with long tails as they are often seen in CNN weights. The method is based on assigning the quantization level using non-overlapping region breaks with the same number of levels, allowing to distribute of the quantization level over the distribution of the values with significantly more levels in low magnitude regions with most of the values than the tails regions. The number of regions directly impacts the hardware requirements and the accuracy of the quantized representation, thus the problem can be stated as an optimization problem. This quantization scheme allows for post-training quantization from 8-bit to 4-bit with very minimal AL. The paper includes compression results on MobileNetV2 [13].

Figure 6.2 shows the comparison of the proposed divide & conquer approach with the previously introduced non-WS-based post-training compression techniques PTP [30], TFLite [31], and PWLQ [32]. The top-1 AL and the memory size required to store the weights are considered as the comparison metrics.

The figure shows that the PTP [30] solution is Pareto-dominated by the proposed divide & conquer WS compression technique. Considering that the PTP [30] compression estimates are based on very optimistic assumptions, we conclude that our method provides better accuracy-compression trade-offs. Regarding the comparison with TFLite [31], it is also Pareto-dominated by the proposed divide & conquer WS compression technique.

Concerning the comparison with PWLQ [32], the method allow for better accuracy-compression trade-offs, by showing very minimal AL. This can be explained by the fact that the weights are quantized per channel in PWLQ [32], allowing a finer-grained approximation than the per-layer granularity used in our experiments, resulting in a reduced approximation error.

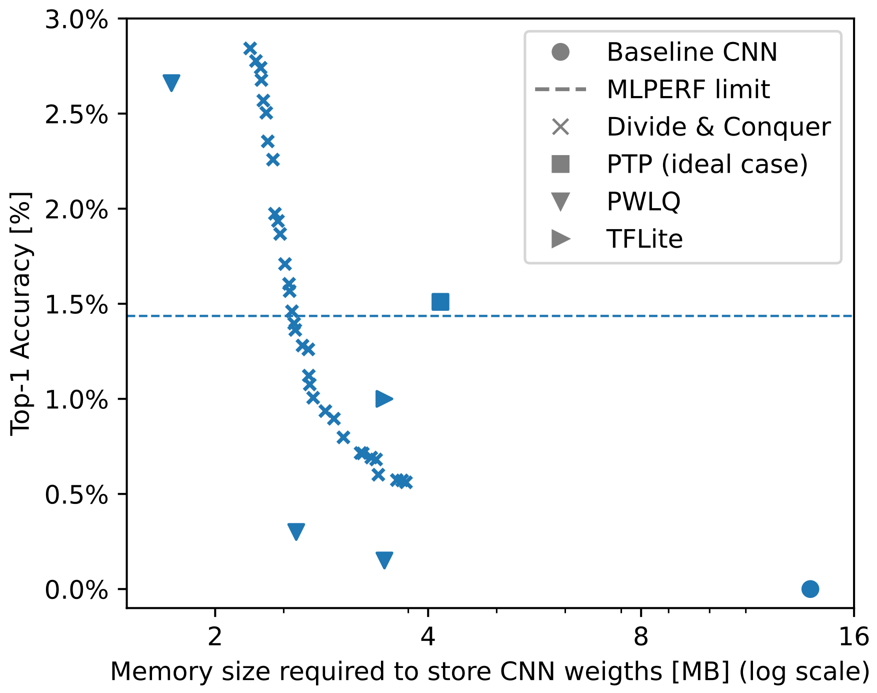


Figure 6.2: Comparison of obtained approximated CNNs with the proposed method and PTP [30], PWLQ [32], and TFLite. CNNs are characterized by the absolute AL measured on the ImageNet [18] and CR compared to the baseline full precision version.

6.4 Conclusions

A broad comparison with different compression approaches has been presented in this chapter, from the closest approach involving different meta-heuristics addressing the same problem of optimizing the number of shared values for each layer, to more distant techniques addressing the problem of optimally grouping the weights with retraining, and finally, other post-training compression techniques that do not involve WS.

From the current comparison, it is clear that the proposed method could benefit from improvements such as the use of fine-tuning of the shared values after the compression or the application of the WS per-channel instead of the coarser grain per-layer application currently in use. Proving that the direction taken after the study described in Figure 4.4, of sharing the values of the weights inside the layer, is possibly not the optimal direction, and that sharing at the level of the channel can result in significantly better approximation, both in terms of AL and CR. This can be explained by the fact that the study was conducted only on a small CNN that is highly resilient to approximation and the extension to a larger CNN like MobileNetV2 [13] is required to conclude on the appropriate granularity.

Chapter 7

Conclusions and Perspectives

As a reminder, the goal of this thesis is to optimize the number of shared values during the application of WS without any retraining step. The goal is to achieve a significant compression of a baseline CNN with accuracy loss complying with the MLPerf [7] quality constraints accepting accuracy over 99% of the baseline CNN accuracy for heavyweight CNNs and 98% of the baseline CNN accuracy for lightweight CNNs. Multiple heuristics have been explored and compared with a focus on keeping the exploration time reasonable. This chapter summarizes the main technical contributions of this thesis in Section 7.1, gives an exhaustive list of the scientific dissemination of the conducted research in Section 7.2, and concludes this thesis with perspectives on improving the proposed approaches in Section 7.3.

7.1 Summary of the Technical Contributions

The first contribution of this thesis is the clear identification of the trade-offs between AL and CR involved in the selection of the most suitable granularity, as described in Section 4.1.

This early investigation work paved the way for the development of a framework for automatic tuning of the number of shared values for each layer of a given baseline CNN. This first framework was relying on the C-export of trained CNN generated by the software N2D2 [17] to measure the AL of a certain approximation, and the use of sklearn [169] for applying the K-means [14] clustering to the weights. This framework was mainly used to compress LeNet-5 [15] on the MNIST [16] dataset, reaching CR over 9× at a very small AL of 0.05%. The obtained results were the object of a publication [8]. The greedy algorithm presented in Section 4.3 was used.

Scaling the framework to be able to use GPU for accelerating the inference required for evaluating the AL motivated the move from C-export from

N2D2 [17] to a more generic CNN representation like ONNX [21] using the GPU runtime from MXNET [22]. Allowing to apply the compression technique to ImageNet [18]-class CNN like ResNet [19] or MobileNet [44]. The orders of magnitude increase in the number of weights also forced us to move from sklearn [169] for applying the K-means [14] to the GPU, relying on the framework kmcuda [23]. Further to these hardware and software moves, the increased depth of CNN enlarged the search space and raised the problem of the algorithmic time complexity. The use of proxy metrics to accelerate the approximation of ImageNet [18]-class CNNs as it was described in Section 4.4 was the object of a conference publication [9]. This exploration process was able to achieve CR over 5× at AL complying with the MLPerf [7] quality target for ResNet18v2 [24] and SqueezeNet1.1 [20].

This last framework also paved the way to multi-objective optimization, offering the possibility to evaluate quickly a compression level. The second metric of interest representing the compression level of the layer was introduced as described in Section 4.5, requiring the move from single-point optimization algorithm to population-based with a local Pareto efficient selection strategy. The use of the greedy optimization algorithm allowed for quickly setting up the compression framework and obtaining very interesting results showing the trade-offs between AL and CR inside the output population. Altogether with previously obtained results, these results were the object of a journal publication [6] giving all details on the work conducted with the greedy algorithm.

The local search nature of the greedy algorithm lacks global knowledge of the approximation impact on the entire approximated CNN, and this has an impact on the obtained results, as it is not capable of compressing efficient CNN such as MobileNetV2 [13] under the MLPerf [7] quality target. There is a need to move from the greedy optimization algorithm with a local scope to a meta-heuristic approach relying on global metrics of interests such as the AL and the CR of the whole approximated CNN. This is the object of subsequent works.

The issue with global optimization is the search space size, we addressed this problem by creating an original divide & conquer strategy to first reduce the search space and second efficiently explore it using a meta-heuristic optimization algorithm such as the NSGA-II [25], allowing compression up to 6×, or merely 2× more than the greedy algorithm, still complying with the MPLerf [7] quality target. This work was described in Section 5.2, and 5.3, the early results of the application of this strategy on LeNet-5 [15] were the object of a workshop publication [10].

Scaling the baseline CNN to ImageNet [18]-class CNN required the acceleration of the combination step that has a larger time complexity than the search space reduction step. To this extent, we have targeted the reduction of the number of required scoring by using a statistical regression model trained on a sampled search space as it was described in Section 5.3. Allowing to compress MobileNetV2 [13] and others recent CNN over 5×, and complying with the MLPerf [7] quality target. The results and an extensive study of the application to a large pool of CNNs were the objects of a conference publication [11].

With the framework reaching the limits of available trained CNN in the

ONNX [21] representation, there was a need to move to other frameworks to gain access to a community-backed model zoo as well as advanced scoring features. We evolved the framework to use either Pytorch [56] or Tensorflow [55] as backends for model representation and scoring. This last version of the framework is also using Fast Kmeans Pytorch [170] to apply the K-means [14] algorithm, because it proved itself to be faster than kmcuda[23] for 1D clustering, with an additional reduced number of copies in memory if the backend of the CNN is Pytorch [56]. This last framework allowed for large-scale experiments on state-of-the-art CNNs like EfficientNet [26] or Inception-V3 [28]. This last framework was open-sourced on GitHub under Apache-2.0 license and is intended to be reused as a block in a compression or deployment pipeline [52].

Further to proposing the divide & conquer strategy, there was a need to compare it with another meta-heuristic optimization algorithm, like the NSGA-II [25]. The two points of comparison were the execution time and the exploration capabilities of the different heuristics, showing that the proposed divide & conquer methods are very competitive heuristics on both execution time and exploration capabilities. These comparisons are described in Section 6.1.

Another comparison was proposed, with other state-of-the-art WS techniques involving retraining. Both DeepKmeans [3] and DP-Net [29], based on regularized training that encourages weight grouping during training, were selected as representative of the recent WS techniques. The proposed approach was able to outperform DeepKmeans [3] on both execution time and compression but was not able to outperform the very effective DP-Net [29] on compression. Still, our methods have the major advantage of not requiring any retraining step or access to a large training dataset. This comparison is the object of Section 6.2.

Another relevant comparison was made with other non-WS-based techniques, this is the object of Section 6.3. Three post-training compression techniques were considered one sparse pruning, PTP [30] and two quantization [31], [32], the only technique that was showing Pareto-dominating results over the ones obtained using the proposed method was PWLQ [32], which has the advantage of having a finer granularity over our method, being the grouping of weights per-channel instead of the per-layer grouping that we have adopted.

7.2 Scientific Dissemination

This section gives a chronological enumeration of the scientific dissemination of the work conducted for this thesis. Unless specified, my contribution to each of the following is the development of the main idea, implementation, and execution of experiments, writing, and presenting the paper.

1. The very first dissemination of the work has been a poster presentation of the early results for compressing LeNet-5 [15]. The venue was the annual symposium of the French research group GDR SOC2 on systems-on-chip, embedded systems, and connected devices in June 2019.

2. The complete set of results for compressing LeNet-5 [15] using a heuristic approach, proving that it is possible to apply WS without retraining has been published as an interactive paper for the international conference DATE 2020, the format was a poster session transformed into video presentation due to the COVID crisis. The paper is available in the proceedings [8]

3. The extension of this early work to ImageNet [18] class CNNs relying on the use of inertia as a proxy metric for accelerating the exploration was published at the international conference DDECS 2020, the format was a conventional presentation, transformed into a video presentation with a live discussion session. The paper is available in the proceedings [9].

4. This last conference publication gave access to an special issue in the journal Microelectronics Reliability, in which an extension of the work featuring multi-objective exploration was published. The paper is available in the journal [6]

5. In the meantime, the new divide & conquer approach has been developed, with a publication of the early results in the form of a poster at the System-level Design Methods for Deep Learning on Heterogeneous Architectures (SLOHA) workshop happening as part of the DATE 2021 conference, the format was a video presentation due to the COVID crisis. The paper is available in open-access on ArXiv [10].

6. The joint effort of the AdequateDL consortium, aiming at accelerating deep learning inference on hardware accelerator, in which this thesis registered, has been the object of a special presentation for the DDECS 2021 conference. The format was a video presentation in which I was presenting my work, as well as the works of the other partners. The paper is available in the proceedings [171]. My contribution to this paper is co-writing the paper and presenting it.

7. The early results obtained with the divide & conquer method on ImageNet [18] class CNNs was the object of a live poster session at the 2021 edition of the annual symposium of the French research group GDR SOC2.

8. The application of the proposed divide & conquer approach to a larger number of CNN was the object of a live poster presentation at the HIPEAC 2021 Computing Systems Week.

9. The complete work on the divide & conquer approach was the object of a conference presentation at the ASP-DAC 2022 conference, giving more details on the approach as well as first comparison with other non-WS based compression techniques such as pruning. The paper is available in the proceedings [11]

10. An extensive survey of the acceleration techniques for the acceleration of the training or the inference of CNNs was conducted collaboratively

with members of the AdequateDL consortium and was the object of the publication of a chapter in an approximate computing book. The chapter is available in the book [12].

11. The summary of this thesis work has been the object of a poster presentation at the DATE 2022 Ph.D. Forum.

12. The extension of the divide & conquer method with a focus on studying different DoE techniques and comparison of the proposed approach with standard approach highlighting the benefits of the design & conquer strategy, as well as the use of regression to accelerate the training, has been the object of an open-access journal publication in IEEE ACCESS 2022, currently under the revision process.

To quantitatively summarize the dissemination of the work conducted during this thesis, please find details in Table 7.1.

Book Chapter	1
Journal	1 (+1)
International Conferences with Proceedings	4
Other Conferences and Symposiums	5

Table 7.1: Quantitative summary of the scientific dissemination.

7.3 Perspectives

The work conducted in this thesis is among the first using heuristic approaches to optimize the WS for CNN compression, tackling the problem of the retraining cost with an adapted approximation. Paving the way for several more research directions, this section is intended to give an overview of the possible research directions.

1. The very first research direction can complete the work achieved using the greedy algorithm, by investigating another approach than n-bit selection for greedy multi-objective, such as the k-means algorithm for grouping the candidates and keeping the closest candidate to the centroids of each cluster, allowing to limit the number of the explored path the same way the number of bits does, but with high-level metrics such as the CR.

2. Another path that will be very interesting to explore is changing the sharing granularity to use hardware-oriented sharing granularity, the same way Deep-k-means [3] used RS dataflow oriented granularity. A heuristic approach such as the ones used in this thesis could potentially allow for even more memory footprint compression at the same quality targets compared to the use of a homogeneous or manually tuned number of shared values.

3. Further to this retraining-free approach, a calibration or small fine-tuning can be achieved, allowing to benefit of the fast exploration of the design space and the enhancement of the best solutions found, with a highly reduced number of candidates.

4. It would also be interesting to investigate the use of other metrics than CR like the weight loading cost as described in Deep-k-means [3]. This has not been explored during this thesis as this requires selecting a target platform and to optimize the CNN for this platform, whereas the approach taken in this thesis was to explore the potential of the design space exploration, without focusing the contribution on a specific hardware target.

5. The proposed divide & conquer approach has been developed in a very network agnostic fashion, allowing it to be applied to virtually any CNN featuring weights. But it is also very close to approximation-agnostic, potentially allowing the application to any hyper-parameter tuning for other non-WS approximation techniques such as pruning or quantization, as long as it targets CNNs.

Bibliography

[1] V. Sze, Y. Chen, T. Yang, and J. S. Emer, 'Efficient processing of deep neural networks: a tutorial and survey', *Proceedings of the IEEE*, 2017. DOI: 10.1109/JPROC.2017.2761740.

[2] S. Han, H. Mao, and W. Dally, 'Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding', *arXiv: Computer Vision and Pattern Recognition*, 2016.

[3] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, and Y. Lin, 'Deep k-means: re-training and parameter sharing with harder cluster assignments for compressing deep convolutions', *ArXiv*, vol. abs/1806.09228, 2018.

[4] S. Son, S. Nah, and K. M. Lee, 'Clustering convolutional kernels to compress deep neural networks', in *ECCV*, 2018.

[5] E.-V. Pikoulis, C. Mavrokefalidis, and A. Lalos, 'A new clustering-based technique for the acceleration of deep convolutional networks', Dec. 2020, pp. 1432–1439. DOI: 10.1109/ICMLA51294.2020.00222.

[6] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, 'CNN weight sharing based on a fast accuracy estimation metric', *Microelectronics Reliability*, vol. 122, 2021, ISSN: 0026-2714.

[7] P. Mattson, C. Cheng, C. Coleman, *et al.*, *MLPerf training benchmark*, 2020. arXiv: 1910.01500 [cs.LG].

[8] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, 'On the automatic exploration of weight sharing for deep neural network compression', in *Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, ser. DATE '20, Grenoble, France: EDA Consortium, 2020, pp. 1319–1322, ISBN: 9783981926347.

[9] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, 'Sensitivity analysis and compression opportunities in dnns using weight sharing', *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 1–6, 2020.

[10] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, 'Fast exploration of weight sharing opportunities for cnn compression', *ArXiv*, vol. abs/2102.01345, 2021.

[11] ——, 'A heuristic exploration of retraining-free weight-sharing for cnn compression', *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, vol. 134-139, 2022.

[12] E. Dupuis, S.-I. Filip, O. Sentieys, D. Novo, I. O'Connor, and A. Bosio, 'Approximations in Deep Learning', 2022. [Online]. Available: `https://hal.archives-ouvertes.fr/hal-03494874`.

[13] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, 'MobileNetV2: inverted residuals and linear bottlenecks', *Proceedings of CVPR*, 2018.

[14] J. A. Hartigan and M. A. Wong, 'A k-means clustering algorithm', *JSTOR: Applied Statistics*, vol. 28, no. 1, pp. 100–108, 1979.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, vol. 86, no. 11, 1998, ISSN: 0018-9219. DOI: `10.1109/5.726791`.

[16] Y. LeCun and C. Cortes, 'MNIST handwritten digit database', 2010. [Online]. Available: `http://yann.lecun.com/exdb/mnist/`.

[17] CEA-LIST, *N2D2*, `https://github.com/CEA-LIST/N2D2`, [Accessed: Dec-2019]. (visited on 09/04/2019).

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, 'ImageNet: A Large-Scale Hierarchical Image Database', in *Proceedings of CVPR09*, 2009.

[19] K. He, X. Zhang, S. Ren, and J. Sun, 'Deep residual learning for image recognition', *CoRR*, vol. abs/1512.03385, 2015. arXiv: `1512.03385`.

[20] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, 'Squeezenet: alexnet-level accuracy with 50x fewer parameters and ¡1mb model size', *CoRR*, vol. abs/1602.07360, 2016. arXiv: `1602.07360`. [Online]. Available: `http://arxiv.org/abs/1602.07360`.

[21] J. Bai, F. Lu, K. Zhang, *et al.*, *Onnx: open neural network exchange*, `https://github.com/onnx/onnx`, 2019.

[22] T. Chen, M. Li, Y. Li, *et al.*, 'Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems', *CoRR*, vol. abs/1512.01274, 2015. arXiv: `1512.01274`. [Online]. Available: `http://arxiv.org/abs/1512.01274`.

[23] V. Markovtsev and M. Cuadros, *Src-d/kmcuda: 6.0.0-1*, version v6.0.0, Feb. 2017. DOI: `10.5281/zenodo.286944`. [Online]. Available: `https://doi.org/10.5281/zenodo.286944`.

[24] K. He, X. Zhang, S. Ren, and J. Sun, 'Identity mappings in deep residual networks', *ArXiv*, vol. abs/1603.05027, 2016.

[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, 'A fast and elitist multiobjective genetic algorithm: NSGA-II', *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, 2002.

[26] M. Tan and Q. V. Le, 'EfficientNet: rethinking model scaling for convolutional neural networks', *ArXiv*, vol. abs/1905.11946, 2019.

[27] C. Szegedy, W. Liu, Y. Jia, *et al.*, 'Going deeper with convolutions', *CoRR*, vol. abs/1409.4842, 2014. arXiv: `1409.4842`.

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, 'Rethinking the Inception architecture for computer vision', *Proceedings of CVPR*, 2016.

[29] D. Yang, W. Yu, A. Zhou, H. Mu, G. Yao, and X. Wang, 'DP-Net: dynamic programming guided deep neural network compression', *ArXiv*, vol. abs/2003.09615, 2020.

[30] I. Lazarevich, A. Kozlov, and N. Malinin, 'Post-training deep neural network pruning via layer-wise calibration', *ArXiv*, vol. abs/2104.15023, 2021.

[31] B. Jacob, S. Kligys, B. Chen, *et al.*, 'Quantization and training of neural networks for efficient integer-arithmetic-only inference', *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[32] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. Hassoun, 'Post-training piecewise linear quantization for deep neural networks', in *ECCV*, 2020.

[33] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, 'Machine learning for email spam filtering: review, approaches and open research problems', *Heliyon*, vol. 5, 2019.

[34] H. Lu and X. Ma, 'Hybrid decision tree-based machine learning models for short-term water quality prediction.', *Chemosphere*, vol. 249, p. 126 169, 2020.

[35] S. L. Shylaja, S. Fairooz, J. Venkatesh, D. Sunitha, R. P. Rao, and M. R. Prabhu, 'Iot based crop monitoring scheme using smart device with machine learning methodology', *Journal of Physics: Conference Series*, vol. 2027, 2021.

[36] K. R. Dahal, J. N. Dahal, H. Banjade, and S. Gaire, 'Prediction of wine quality using machine learning algorithms', *Open Journal of Statistics*, vol. 11, pp. 278–289, 2021.

[37] D. Varmedja, M. Karanovic, S. Sladojevic, M. Arsenovic, and A. Anderla, 'Credit card fraud detection - machine learning methods', *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–5, 2019.

[38] M. N. Ashtiani and B. Raahemi, 'Intelligent fraud detection in financial statements using machine learning and data mining: a systematic literature review', *IEEE Access*, 2021.

[39] G. Venture, B. Muraccioli, M.-L. Bourguet, and J. Urakami, 'Can robots be good public speakers?', *Sixteenth International Conference on Tangible, Embedded, and Embodied Interaction*, 2022.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks', *Communications of the ACM*, vol. 60, pp. 84–90, 2012.

[41] O. Russakovsky, J. Deng, H. Su, *et al.*, 'ImageNet Large Scale Visual Recognition Challenge', *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: `10.1007/s11263-015-0816-y`.

[42] E. Strubell, A. Ganesh, and A. McCallum, 'Energy and policy considerations for deep learning in nlp', *ArXiv*, vol. abs/1906.02243, 2019.

[43] T.-J. Yang, Y.-h. Chen, and V. Sze, 'Designing energy-efficient convolutional neural networks using energy-aware pruning', *Proceedings of CVPR*, 2017.

[44] A. Howard, M. Zhu, B. Chen, *et al.*, 'MobileNets: efficient convolutional neural networks for mobile vision applications', *ArXiv*, vol. abs/1704.04861, 2017.

[45] Y. Hu, J. Li, X. Long, *et al.*, 'Cluster regularized quantization for deep networks compression', in *Proceedings of ICIP*, 2019.

[46] K. Ullrich, E. Meeds, and M. Welling, 'Soft weight-sharing for neural network compression', *ArXiv*, vol. abs/1702.04008, 2017.

[47] P. Wang, Q. Chen, X. He, and J. Cheng, 'Towards accurate post-training network quantization via bit-split and stitching', in *ICML*, 2020.

[48] Y. Gong, L. Liu, M. Yang, and L. Bourdev, 'Compressing Deep Convolutional Networks using Vector Quantization', *arXiv*, 2014. (visited on 09/14/2019).

[49] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, 'Quantized convolutional neural networks for mobile devices', *CoRR*, vol. abs/1512.06473, 2015. arXiv: `1512.06473`. [Online]. Available: `http://arxiv.org/abs/1512.06473`.

[50] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, 'LookNN: Neural network with no multiplication', in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, IEEE, 2017, pp. 1775–1780.

[51] S. Han, X. Liu, H. Mao, *et al.*, 'Eie: efficient inference engine on compressed deep neural network', *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254, 2016.

[52] E. Dupuis, I. O'Connor, D. Novo, and A. Bosio, 'A Heuristic Exploration to Retraining-free Weight-Sharing for CNN Compression', version 1.0.0, Nov. 2021. [Online]. Available: `https://github.com/e-dupuis/retraining-free-weight-sharing`.

[53] A. L. Samuel, 'Some studies in machine learning using the game of checkers', *IBM J. Res. Dev.*, vol. 3, pp. 210–229, 1959.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 'Dropout: a simple way to prevent neural networks from overfitting', *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: `http://jmlr.org/papers/v15/srivastava14a.html`.

[55] M. Abadi, P. Barham, J. Chen, *et al.*, 'Tensorflow: a system for large-scale machine learning', in *Proceedings of OSDI*, 2016.

[56] A. Paszke, S. Gross, F. Massa, *et al.*, 'Pytorch: an imperative style, high-performance deep learning library', in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[57] R. Stojnic, R. Taylor, V. Kerkez, and L. Viaud, *Papers with code, state of the art models on the imagenet dataset*, 2020. [Online]. Available: `https://paperswithcode.com/sota/image-classification-on-imagenet`.

[58] J. Cong and B.-Y. Xiao, 'Minimizing computation in convolutional neural networks', in *ICANN*, 2014.

[59] N. P. Jouppi, C. Young, N. Patil, *et al.*, 'In-datacenter performance analysis of a tensor processing unit', *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2017.

[60] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, 'Logicnets: co-designed neural networks and circuits for extreme-throughput applications', *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 291–297, 2020.

[61] T. Murovi and A. Trost, 'Massively parallel combinational binary neural networks for edge processing', 2019.

[62] J. M. Duarte, S. Han, P. C. Harris, *et al.*, 'Fast inference of deep neural networks in fpgas for particle physics', *ArXiv*, vol. abs/1804.06913, 2018.

[63] S. Tridgell, M. Kumm, M. Hardieck, *et al.*, 'Unrolling ternary neural networks', *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, pp. 1–23, 2019.

[64] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, 'Lutnet: rethinking inference in fpga soft logic', *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 26–34, 2019.

[65] Y. Chen, J. Emer, and V. Sze, 'Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks', 2016.

[66] Y. Umuroglu, N. J. Fraser, G. Gambardella, *et al.*, 'Finn: a framework for fast, scalable binarized neural network inference', *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[67] Y.-h. Chen, J. S. Emer, and V. Sze, 'Eyeriss v2: a flexible and high-performance accelerator for emerging deep neural networks', *ArXiv*, vol. abs/1807.07928, 2018.

[68] S. Agarwal, E. Hervas-Martin, J. Byrne, A. K. Dunne, J. L. Espinosa-Aranda, and D. Rijlaarsdam, 'An evaluation of low-cost vision processors for efficient star identification', *Sensors (Basel, Switzerland)*, vol. 20, 2020.

[69] H. Fan, M. Ferianc, Z. Que, *et al.*, 'Algorithm and hardware co-design for reconfigurable cnn accelerator', *ArXiv*, vol. abs/2111.12787, 2021.

[70] Y. Li, C. Hao, X. Zhang, *et al.*, 'Edd: efficient differentiable dnn architecture and implementation co-search for embedded ai solutions', *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[71] H. Fan, M. Ferianc, S. Liu, Z. Que, X. Niu, and W. Luk, 'Optimizing fpga-based cnn accelerator using differentiable neural architecture search', *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 465–468, 2020.

[72] L. Sekanina, 'Neural architecture search and hardware accelerator co-search: a survey', *IEEE Access*, vol. 9, pp. 151 337–151 362, 2021.

[73] A. G. Howard, M. Sandler, G. Chu, *et al.*, 'Searching for mobilenetv3', *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

[74] G. Hinton, O. Vinyals, and J. Dean, 'Distilling the Knowledge in a Neural Network', *arXiv*, Mar. 2015. [Online]. Available: `http://arxiv.org/abs/1503.02531` (visited on 09/13/2019).

[75] J. Tang, R. Shivanna, Z. Zhao, *et al.*, 'Understanding and Improving Knowledge Distillation', *arXiv preprint arXiv:2002.03532*, 2020.

[76] S. Anwar, K. Hwang, and W. Sung, 'Structured pruning of deep convolutional neural networks', *ACM JETC*, vol. 13, 2017.

[77] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, 'Autocompress: an automatic dnn structured pruning framework for ultra-high compression rates', in *AAAI*, 2020.

[78] Y. LeCun, J. S. Denker, and S. A. Solla, 'Optimal brain damage', in *NIPS*, 1989.

[79] J. Frankle and M. Carbin, 'The lottery ticket hypothesis: finding sparse, trainable neural networks', *arXiv: Learning*, 2019.

[80] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, 'Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients', *arXiv preprint arXiv:1606.06160*, 2016.

[81] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, 'Incremental network quantization: towards lossless cnns with low-precision weights', *ArXiv*, vol. abs/1702.03044, 2017.

[82] M. E. Elbtity, H.-W. Son, D.-Y. Lee, and H. Kim, 'High speed, approximate arithmetic based convolutional neural network accelerator', *2020 International SoC Design Conference (ISOCC)*, pp. 71–72, 2020.

[83] M. Cho and Y. Kim, 'Fpga-based convolutional neural network accelerator with resource-optimized approximate multiply-accumulate unit', *Electronics*, 2021.

[84] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, 'Energy-efficient neural computing with approximate multipliers', *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, pp. 1–23, 2018.

[85] V. Mrázek, Z. Vasícek, L. Sekanina, M. A. Hanif, and M. A. Shafique, 'Alwann: automatic layer-wise approximation of deep neural network accelerators without retraining', *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[86] C. D. la Parra, A. Guntoro, and A. Kumar, 'Full approximation of deep neural networks through efficient optimization', in *ISCAS 2020*, 2020.

[87] V. Mrazek, L. Sekanina, and Z. Vasícek, 'Libraries of approximate circuits: automated design and application in cnn accelerators', *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, pp. 406–418, 2020.

[88] Y. L. Cun, J. S. Denker, and S. A. Solla, 'Optimal brain damage', in *Advances in Neural Information Processing Systems 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 598–605, ISBN: 1558601007.

[89] Y. Ji, L. Liang, L. Deng, Y. Zhang, Y. Zhang, and Y. Xie, 'Tetris: tile-matching the tremendous irregular sparsity', in *NeurIPS*, 2018.

[90] J. Yu, A. Lukefahr, D. Palframan, G. S. Dasika, R. Das, and S. Mahlke, 'Scalpel: customizing dnn pruning to the underlying hardware parallelism', *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 548–560, 2017.

[91] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, 'Pruning convolutional neural networks for resource efficient transfer learning', *CoRR*, vol. abs/1611.06440, 2016. arXiv: `1611.06440`. [Online]. Available: `http://arxiv.org/abs/1611.06440`.

[92] J.-H. Luo, J. Wu, and W. Lin, 'Thinet: a filter level pruning method for deep neural network compression', *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5068–5076, 2017.

[93] Y. He, X. Zhang, and J. Sun, 'Channel pruning for accelerating very deep neural networks', *CoRR*, vol. abs/1707.06168, 2017. arXiv: `1707.06168`. [Online]. Available: `http://arxiv.org/abs/1707.06168`.

[94] Y. Huan, Y. Qin, Y. You, L. Zheng, and Z. Zou, 'A multiplication reduction technique with near-zero approximation for embedded learning in iot devices', in *2016 29th IEEE International System-on-Chip Conference (SOCC)*, IEEE, 2016, pp. 102–107.

[95] ——, 'A low-power accelerator for deep neural networks with enlarged near-zero sparsity', *arXiv preprint arXiv:1705.08009*, 2017.

[96] V. Lebedev and V. Lempitsky, 'Fast convnets using group-wise brain damage', *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2554–2564, 2016.

[97] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, 'Learning structured sparsity in deep neural networks', *ArXiv*, vol. abs/1608.03665, 2016.

[98] M. Yuan and Y. Lin, 'Model selection and estimation in regression with grouped variables', *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 68, pp. 49–67, 2006.

[99] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, 'Learning efficient convolutional networks through network slimming', *CoRR*, vol. abs/1708.06519, 2017. arXiv: `1708.06519`. [Online]. Available: `http://arxiv.org/abs/1708.06519`.

[100] X. Ding, G. Ding, J. Han, and S. Tang, 'Auto-balanced filter pruning for efficient convolutional neural networks', in *AAAI*, 2018.

[101] J.-H. Luo and J. Wu, 'Autopruner: an end-to-end trainable filter pruning method for efficient deep model inference', *Pattern Recognit.*, vol. 107, p. 107 461, 2020.

[102] J. Frankle and M. Carbin, 'The lottery ticket hypothesis: training pruned neural networks', *CoRR*, vol. abs/1803.03635, 2018. arXiv: `1803.03635`. [Online]. Available: `http://arxiv.org/abs/1803.03635`.

[103] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, 'Rethinking the value of network pruning', *ArXiv*, vol. abs/1810.05270, 2019.

[104] M. Lin, R. Ji, Y.-x. Zhang, B. Zhang, Y. Wu, and Y. Tian, 'Channel pruning via automatic structure search', *ArXiv*, vol. abs/2001.08565, 2020.

[105] Z. Liu, H. Mu, X. Zhang, *et al.*, 'Metapruning: meta learning for automatic neural network channel pruning', *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3295–3304, 2019.

[106] Y. He and S. Han, 'ADC: automated deep compression and acceleration with reinforcement learning', *CoRR*, vol. abs/1802.03494, 2018. arXiv: `1802.03494`. [Online]. Available: `http://arxiv.org/abs/1802.03494`.

[107] B. Hassibi and D. Stork, 'Second order derivatives for network pruning: optimal brain surgeon', in *NIPS*, 1992.

[108] S. Srinivas and R. V. Babu, 'Data-free parameter pruning for deep neural networks', in *BMVC*, 2015.

[109] Y. Guo, A. Yao, and Y. Chen, 'Dynamic network surgery for efficient dnns', in *NIPS*, 2016.

[110] S. Narang, G. Diamos, S. Sengupta, and E. Elsen, 'Exploring sparsity in recurrent neural networks', *ArXiv*, vol. abs/1704.05119, 2017.

[111] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf, 'Pruning filters for efficient convnets', *ArXiv*, vol. abs/1608.08710, 2017.

[112] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, 'Towards efficient model compression via learned global ranking', *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. DOI: `10.1109/cvpr42600.2020.00159`. [Online]. Available: `http://dx.doi.org/10.1109/cvpr42600.2020.00159`.

[113] ——, 'Legr: filter pruning via learned global ranking', *CoRR*, vol. abs/1904.12368, 2019. arXiv: `1904.12368`. [Online]. Available: `http://arxiv.org/abs/1904.12368`.

[114] X. Dai, H. Yin, and N. K. Jha, 'Nest: A neural network synthesis tool based on a grow-and-prune paradigm', *CoRR*, vol. abs/1711.02017, 2017. arXiv: `1711.02017`. [Online]. Available: `http://arxiv.org/abs/1711.02017`.

[115] T. Zhang, S. Ye, K. Zhang, *et al.*, 'A systematic DNN weight pruning framework using alternating direction method of multipliers', *CoRR*, vol. abs/1804.03294, 2018. arXiv: `1804.03294`. [Online]. Available: `http://arxiv.org/abs/1804.03294`.

[116] S. Ye, T. Zhang, K. Zhang, *et al.*, 'Progressive weight pruning of deep neural networks using ADMM', *CoRR*, vol. abs/1810.07378, 2018. arXiv: `1810.07378`. [Online]. Available: `http://arxiv.org/abs/1810.07378`.

[117] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, 'Binarized neural networks', in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 4107–4115.

[118] F. Li and B. Liu, 'Ternary weight networks', *CoRR*, vol. abs/1605.04711, 2016. arXiv: `1605.04711`. [Online]. Available: `http://arxiv.org/abs/1605.04711`.

[119] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, 'XNOR-Net: ImageNet classification using binary convolutional neural networks', pp. 525–542, 2016.

[120] B. Jacob, S. Kligys, B. Chen, *et al.*, 'Quantization and training of neural networks for efficient integer-arithmetic-only inference', *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[121] S. Wu, G. Li, F. Chen, and L. Shi, 'Training and inference with integers in deep neural networks', *arXiv preprint arXiv:1802.04680*, 2018.

[122] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, 'Pact: parameterized clipping activation for quantized neural networks', *arXiv preprint arXiv:1805.06085*, 2018.

[123] D. Zhang, J. Yang, D. Ye, and G. Hua, 'Lq-nets: learned quantization for highly accurate and compact deep neural networks', in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[124] R. Banner, Y. Nahshan, and D. Soudry, 'Post training 4-bit quantization of convolutional networks for rapid-deployment', in *Advances in Neural Information Processing Systems*, 2019, pp. 7950–7958.

[125] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, 'ZeroQ: a novel zero shot quantization framework', in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 169–13 178.

[126] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, 'Low-bit quantization of neural networks for efficient inference', in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, IEEE, 2019, pp. 3009–3018.

[127] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, 'Data-free quantization through weight equalization and bias correction', in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1325–1334.

[128] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang, 'Improving neural network quantization without retraining using outlier channel splitting', *arXiv preprint arXiv:1901.09504*, 2019.

[129] M. Alizadeh, A. Behboodi, M. van Baalen, C. Louizos, T. Blankevoort, and M. Welling, 'Gradient ℓ_1 regularization for quantization robustness', *arXiv preprint arXiv:2002.07520*, 2020.

[130] M. Shkolnik, B. Chmiel, R. Banner, *et al.*, 'Robust quantization: one model to rule them all', *arXiv preprint arXiv:2002.07686*, 2020.

[131] M. Courbariaux, Y. Bengio, and J.-P. David, 'BinaryConnect: training deep neural networks with binary weights during propagations', in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[132] F. Li, B. Zhang, and B. Liu, 'Ternary weight networks', *arXiv preprint arXiv:1605.04711*, 2016.

[133] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, 'Adaptive quantization for deep neural network', *arXiv preprint arXiv:1712.01048*, 2017.

[134] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, 'Mixed precision quantization of convnets via differentiable neural architecture search', *arXiv preprint arXiv:1812.00090*, 2018.

[135] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, 'Haq: hardware-aware automated quantization with mixed precision', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[136] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, 'Hawq: hessian aware quantization of neural networks with mixed-precision', in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 293–302.

[137] Z. Dong, Z. Yao, Y. Cai, *et al.*, 'Hawq-v2: hessian aware trace-weighted quantization of neural networks', *arXiv preprint arXiv:1911.03852*, 2019.

[138] D. Lin, S. Talathi, and S. Annapureddy, 'Fixed point quantization of deep convolutional networks', pp. 2849–2858, 2016.

[139] S. Khoram and J. Li, 'Adaptive quantization of neural networks', in *International Conference on Learning Representations*, 2018.

[140] S. Shen, Z. Dong, J. Ye, *et al.*, 'Q-bert: hessian based ultra low precision quantization of bert.', in *AAAI*, 2020, pp. 8815–8821.

[141] X. Zhu, W. Zhou, and H. Li, 'Adaptive layerwise quantization for deep neural network compression', in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, IEEE, 2018, pp. 1–6.

[142] E. Park, S. Yoo, and P. Vajda, 'Value-aware quantization for training and inference of neural networks', in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 580–595.

[143] W. Tang, G. Hua, and L. Wang, 'How to train a compact binary neural network with high accuracy?', in *AAAI*, 2017, pp. 2625–2631.

[144] A. Kundu, K. Banerjee, N. Mellempudi, *et al.*, 'Ternary residual networks', *arXiv preprint arXiv:1707.04679*, 2017.

[145] B. Jacob, S. Kligys, B. Chen, *et al.*, 'Quantization and training of neural networks for efficient integer-arithmetic-only inference', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[146] S. O. Settle, M. Bollavaram, P. D'Alberto, *et al.*, 'Quantizing convolutional neural networks for low-power high-throughput inference engines', *arXiv preprint arXiv:1805.07941*, 2018.

[147] C. Wu, M. Wang, X. Chu, K. Wang, and L. He, 'Low precision floating-point arithmetic for high performance fpga-based cnn acceleration', *arXiv preprint arXiv:2003.03852*, 2020.

[148] C. Wu, M. Wang, X. Li, J. Lu, K. Wang, and L. He, 'Phoenix: a low-precision floating-point quantization oriented architecture for convolutional neural networks', *arXiv preprint arXiv:2003.02628*, 2020.

[149] T. Tambe, E.-Y. Yang, Z. Wan, *et al.*, 'Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference', in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[150] Z. Song, Z. Liu, and D. Wang, 'Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design', *arXiv preprint arXiv:1709.07776*, 2017.

[151] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, 'High-performance FPGA-based CNN accelerator with block-floating-point arithmetic', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[152] D. Kalamkar, D. Mudigere, N. Mellempudi, *et al.*, 'A study of bfloat16 for deep learning training', *arXiv preprint arXiv:1905.12322*, 2019.

[153] D. Miyashita, E. H. Lee, and B. Murmann, 'Convolutional neural networks using logarithmic data representation', *arXiv preprint arXiv:1603.01025*, 2016.

[154] Y. Choi, M. El-Khamy, and J. Lee, 'Learning sparse low-precision neural networks with learnable regularization', *IEEE Access*, 2020.

[155] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, 'Incremental network quantization: Towards lossless CNNs with low-precision weights', *arXiv preprint arXiv:1702.03044*, 2017.

[156] H. Bai, J. Wu, I. King, and M. Lyu, 'Few shot network compression via cross distillation', *arXiv preprint arXiv:1911.09450*, 2019.

[157] A. Polino, R. Pascanu, and D. Alistarh, 'Model compression via distillation and quantization', *arXiv preprint arXiv:1802.05668*, 2018.

[158] S. Chen, W. Wang, and S. J. Pan, 'Deep neural network quantization via layer-wise optimization using limited training data', in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3329–3336.

[159] C. Leng, H. Li, S. Zhu, and R. Jin, 'Extremely low bit neural network: squeeze the last bit out with admm', *arXiv preprint arXiv:1707.09870*, 2017.

[160] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, 'Compressing deep convolutional networks using vector quantization', *ArXiv*, vol. abs/1412.6115, 2014.

[161] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, 'Compressing neural networks with the hashing trick', *CoRR*, vol. abs/1504.04788, 2015. arXiv: `1504.04788`. [Online]. Available: `http://arxiv.org/abs/1504.04788`.

[162] E. Park, J. Ahn, and S. Yoo, 'Weighted-entropy-based quantization for deep neural networks', *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7197–7205, 2017.

[163] S. P. Lloyd, 'Least squares quantization in pcm', *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 1982.

[164] G. E. P. Box and D. W. Behnken, 'Some new three level designs for the study of quantitative variables', *Technometrics*, vol. 2, no. 4, pp. 455–475, 1960, ISSN: 00401706. [Online]. Available: `http://www.jstor.org/stable/1266454`.

[165] M. D. McKay, R. J. Beckman, and W. J. Conover, 'A comparison of three methods for selecting values of input variables in the analysis of output from a computer code', *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979, ISSN: 00401706. [Online]. Available: `http://www.jstor.org/stable/1268522`.

[166] M. Johnson, L. Moore, and D. Ylvisaker, 'Minimax and maximin distance designs', *Journal of Statistical Planning and Inference*, vol. 26, no. 2, pp. 131–148, 1990, ISSN: 0378-3758. DOI: `https://doi.org/10.1016/0378-3758(90)90122-B`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/037837589090122B`.

[167] J. H. Halton, 'Algorithm 247: radical-inverse quasi-random point sequence', *Commun. ACM*, vol. 7, no. 12, pp. 701–702, 1964, ISSN: 0001-0782. DOI: `10.1145/355588.365104`. [Online]. Available: `https://doi.org/10.1145/355588.365104`.

[168] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, and Y. Lin, 'Deep k-Means: Re-Training and Parameter Sharing with Harder Cluster Assignments for Compressing Deep Convolutions', *arXiv*, Jun. 2018. [Online]. Available: `http://arxiv.org/abs/1806.09228` (visited on 09/14/2019).

[169] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, 'Scikit-learn: machine learning in Python', *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[170] @DeMoriarty, *Fast pytorch kmeans*, `https://github.com/DeMoriarty/fast_pytorch_kmeans`, 2020.

[171] O. Sentieys, S.-I. Filip, D. Briand, *et al.*, 'Adequatedl: approximating deep learning accelerators', *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 37–40, 2021.

[172] D. Arthur and S. Vassilvitskii, 'K-means++: the advantages of careful seeding', in *SODA '07*, 2007.

Appendix

A.1 Algorithms

A.1.1 K-means clustering

From the very different clustering approaches, K-means [14] is among the most popular. Its aim at identifies a certain number of groups or clusters in the samples by using a distance-based iterative approach to the problem of selection and mapping. For each iteration the samples are grouped by their distance to the centroid of each cluster, different distance metrics can be used such as the norm or the cos. The second step is to compute the new centroid of each cluster by averaging the samples composing the cluster. This iterative process is repeated until convergence or for a certain number of iterations. There are different possibilities for the initialization of the centroid of the clusters, linear initialization spanning across the distribution of the sample, random initialization, or even kmeans++ [172], a selection method that allows for faster convergence.

A.1.2 NSGA-II meta-heuristic optimization

In multi-objective optimization, there is a need to explore the trade-offs between the objectives in the design space. One of the numerous algorithms allowing such exploration is the NSGA-II [25] algorithm. Using a genetic approach to problem-solving, involving iterating over a population, by selecting the best candidates and crossing their genes, and applying random mutation over the genome. The genes are the variable of the optimization problem, and the selection of the best candidates is performed using the Pareto front, thus the name of the NSGA-II, Non-Dominated Sorting Genetic Algorithm. The Pareto front is selected successively, from the first front composed by the non-dominated candidate, then the second front that is composed of the non-dominated candidate when not considering the first front, this process is repeated until the last candidate, and a certain number of candidates are kept, whereas the others are discarded. Mutation and crossover are applied to the remaining population, and the newly obtained population is ready for another Pareto front selection.

A.2 Thesis Compute Resources

The work conducted during this thesis required the use of different computing platforms for running the approximation of the approximated CNNs. The early work was conducted on a commercial laptop featuring an INTEL i7 with 8 cores at 1.9Ghz. mainly used for development and early testing on LeNet-5 [15]. At some point, there was a need for more computing power and a GPU to be able to efficiently score the approximated CNNs. A server with an Intel Xeon 4210 with 40 core at 2.2Ghz and a Tesla V100 GPU with 32GB of memory was used . Other remote resources were also involved in the last step of the thesis when there was a need to apply the compression on several different CNN topologies. The PMCS2I cluster of Ecole Centrale de Lyon and the Jean-Zay cluster of the french national research agency were used.

A.3 Fast CNN scoring

Scoring an approximated CNN over a validation dataset such as the 50k images validation set of ImageNet [18] requires several steps. First, there is a need to process the raw data from disk, apply pre-processing transforms to normalize the size and color distribution, convert values into floating-point, and aggregate batches of samples together so that they can be fed to the CNN. Then the output of each layer is successively computed and fed to the next layer until the final output, which is compared with labels from the validation dataset. Comparing every output with the corresponding labels, or expected outputs, allows for computing the top-1 accuracy as well as other metrics that will not be detailed here. This process is computation-intensive and could benefit from some optimization in our specific framework.

A.3.1 Dataflow optimization

When training a CNN a single candidate is improved during the whole training process, that is why the validation step is executed on a single CNN at the end of each training iteration. In our case, several CNNs needs to be tested at the same moment because we are optimizing a population of approximated CNNs. This different framework allows for some optimization in terms of dataflow, when profiling a CNN scoring, it appears that the process is often input-bound, because pre-processing time is long and even using multiple workers for loading the data, the GPU appears to be starving. To avoid this GPU starving, it is possible to score a population of approximated CNNs with a single data processing, by computing each approximated CNN output for each batch of samples processed sequentially. Allowing for performance gains as it can be seen in Figure A.1, scoring a batch composed of up to 20 approximated CNN sequentially, where the scoring throughput is $1.6\times$ more important than scoring a single CNN at a time.

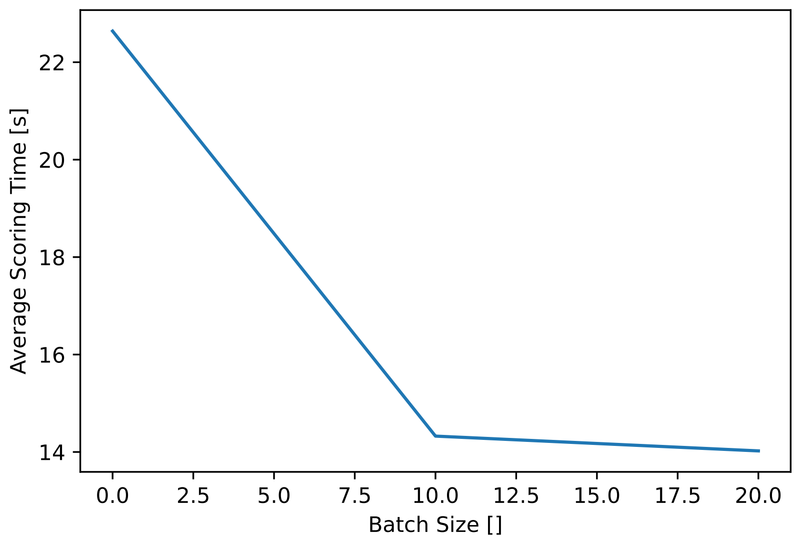


Figure A.1: Reduction of the scoring time caused by using dataflow optimization with population scoring (or batch scoring) for MobileNetV2 [13] on the ImageNet [18] dataset.

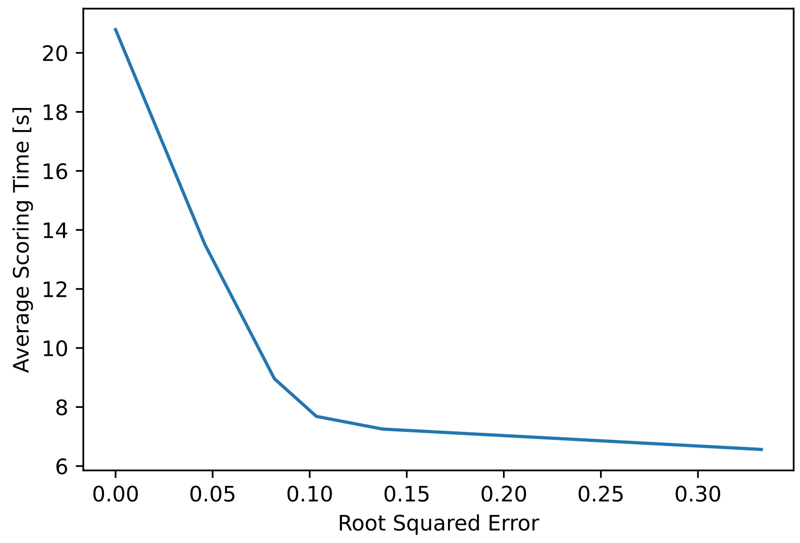


Figure A.2: Variation of the incertitude (root squared error) on the top-1 accuracy obtained by scoring multiple approximated CNN on the ImageNet [18] dataset using a portion of the samples in the validation set.

A.3.2 Approximated scoring

Further to dataflow optimization, it is also possible to use approximate scoring to benefit of scoring acceleration at the cost of scoring accuracy. The simplest way of achieving such approximate scoring is by reducing the number of samples used to score an approximated CNN. Taking as example ImageNet [18], and its 50k samples validation set, it is possible to measure the acceleration provided by varying the number of samples used for scoring the approximated CNN, and evaluate the induced incertitude over the measured top-1 accuracy. Figure A.2 show the result of the scoring of various approximated CNN derived from MobileNetV2 [13] with a different number of samples, shown as the ratio of the number of samples of the initial 50k samples validation set.