Deep Neural Networks hardware-algorithmic enablers for compact ASIC design towards embedded image/video processing

Van-Thien Nguyen

▶ **To cite this version:**

Van-Thien Nguyen. Deep Neural Networks hardware-algorithmic enablers for compact ASIC design towards embedded image/video processing. Signal and Image processing. Université Grenoble Alpes [2020-..], 2022. English. NNT : 2022GRALT114 . tel-04339285

HAL Id: tel-04339285
https://theses.hal.science/tel-04339285

Submitted on 13 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : Signal Image Parole Télécoms
Unité de recherche : Laboratoire d'Electronique et de Technologie de l'Information (LETI - CEA)

Conception de réseaux de neurones profonds pour le traitement embarqué d'image/vidéo, compatible avec une architecture micro-électronique frugale

Deep Neural Networks hardware-algorithmic enablers for compact ASIC design towards embedded image/video processing

Présentée par :

Van-Thien NGUYEN

Direction de thèse :

Gilles SICARD — Directeur de thèse
Ingénieur / Chercheur, Université Grenoble Alpes
William GUICQUERO — Co-encadrant de thèse
CEA LETI

Rapporteurs :

Jacques-Olivier KLEIN
PROFESSEUR DES UNIVERSITES, UNIVERSITE PARIS-SACLAY
Eva DOKLADALOVA
INGENIEUR HDR, ESIEE NOISY LE GRAND

Thèse soutenue publiquement le **12 décembre 2022**, devant le jury composé de :

Jacques-Olivier KLEIN — Rapporteur
PROFESSEUR DES UNIVERSITES, UNIVERSITE PARIS-SACLAY
Eva DOKLADALOVA — Rapporteure
INGENIEUR HDR, ESIEE NOISY LE GRAND
Stéphane MANCINI — Examinateur
MAITRE DE CONFERENCES HDR, GRENOBLE INP
Dominique GINHAC — Président
PROFESSEUR DES UNIVERSITES, UNIVERSITE DE BOURGOGNE

Invités :

William GUICQUERO
INGENIEUR DOCTEUR, CEA-LETI
Gilles Sicard
INGENIEUR HDR, CEA-LETI

Remerciements

Je tiens tout d'abord à remercier les deux rapporteurs, Jacques-Olivier Klein et Eva Dokladalova, pour avoir pris le temps de relire ce manuscrit, pour leurs retours constructifs et leurs conseils valables. Je voudrais remercier également les deux examinateurs, Stéphane Mancini et Dominique Ginhac, pour leur intérêt sur mes travaux et leurs questions très enrichissantes permettant d'avoir un autre point de vue sur la problèmatique et les propositions dans ce manuscrit.

Je voudrais ensuite remercier mon encadrant de thèse, William Guicquero, pour son soutien, sa patience, sa disponibilité et ses judicieux conseils dans la recherche scientifique. La confiance immédiate qu'il m'a accordée m'a construit en tant qu'un jeune chercheur et m'a indéniablement fait grandir. Il m'a introduit auprès du domaine de Tiny Machine Learning qui devient pour moi une grande passion pour la suite de mes recherches. Un grand merci pour son soutien décisif dans tous les moments clés de ma thèse, non seulement au niveau scientifique, mais aussi au niveau personnel.

Je tiens à remercier Gilles Sicard, mon directeur de thèse, pour son support à travers le stage puis la thèse. Merci d'avoir calmé mes doutes lorsqu'il le fallait et de m'avoir donné des conseils tout au long de la route. Je retiendrai son accessibilité, ses expériences enrichissantes et ses conseils importants pendant ces années.

Je voudrais remercier tous les membres de l'équipe L3I où j'ai eu l'occasion de travailler depuis environ quatre ans, tout d'abord en tant qu'un stagiaire, puis qu'un doctorant et maintenant qu'un ingénieur de recherche. Merci à Fabrice Guellec, notre chef du labo, qui m'a accueilli avec bienveillance dans son équipe et m'a supporté constamment dès mon premier jour au L3I. Merci Laurent Alacoque de m'avoir choisi pour son sujet de stage en traitement d'images, et de m'avoir recommandé pour cette thèse. Je retiendrai sa simplicité, ses idées intuitives et notamment sa bonne humeur. Merci à Wissam Benjilali pour les conseils qu'il m'a donnés avant et pendant la thèse. Merci à Valentin et Mathieu - les doctorants avec qui j'ai eu beaucoup de discussions intéressantes au bureau comme en dehors des heures de travail. Merci également les personnels du DOPT qui m'ont aidé tout au long du déroulement de ma thèse.

À titre personnel, je tiens à remercier mes amis qui m'ont accompagné pendant ces années, depuis mon arrivée en France en 2015. C'est un long parcours et leur soutien est indispensable pour moi. Merci à Hoa, Linh, Thao, Luu pour avoir pris le temps de participer à ma soutenance, malgré la distance et les contraintes de temps à la fin d'année. Merci à Phat pour son amitié et pour toutes nos discussions dans ce domaine que l'on partage la même grande passion. Dernier point, mais pas le moindre, merci à mes parents et mon petit frère, qui m'ont tout donné pour que je m'épanouisse. Ce sont toujours les grandes sources de motivation pour ma vie.

Contents

Remerciements i

Table of Contents iii

Résumé **1**

0.1 Introduction 1
0.2 Co-conception matérielle-algorithmique des réseaux de neurones 3
0.2.1 Quantification des poids avec égalisation de l'histogramme des poids . . . 3
0.2.2 DNNs à précision mixte pour multi-tâche non-corrélé 4
0.2.3 Réseaux de neurones résiduels avec les portes logiques et factorisation de convolution compacte 5
0.2.4 Génération des poids de DNNs avec des modèles auxiliaire et PRNG . . . 6
0.3 Conclusion et ouvertures 7

1 Introduction **9**
1.1 Problematic 11
1.2 Contributions and Thesis Outline 12

2 Background **15**
2.1 Overview of Deep Neural Networks 16
2.1.1 Layers 17
2.1.2 Activation functions 20
2.1.3 Loss function and regularization for DNNs 22
2.1.4 Learning procedure 24
2.1.5 Datasets 27
2.1.6 Overfitting and training tricks 28
2.1.7 Neural Network Architectures 29
2.1.8 Deep learning frameworks 34
2.2 DNN accelerators 35
2.2.1 General-Purpose Accelerators (GPA) 35
2.2.2 AI-Dedicated Accelerators (AIDA) 36
2.2.3 Application-Specific Accelerators (ASA) 36
2.2.4 Discussion 37
2.3 DNN hardware-algorithmic enablers state-of-the-art 38
2.3.1 Efficient architecture designs 38
2.3.2 Model compression techniques 42
2.3.3 Training trick for compact models: Knowledge Distillation 50
2.4 Conclusion 50

3 Histogram-Equalized Quantization (HEQ) for low-precision weighted networks **51**
3.1 Context 52
3.2 State of the art in linear symmetric quantization 53
3.2.1 Quantization error minimization methods 53

3.2.2 Task loss gradient-based methods 54
3.2.3 Statistics-based methods 54
3.3 Linear symmetric quantization and the unbalanced histogram 55
3.4 Histogram-Equalized Quantization (HEQ) 55
3.5 Experiments 58
3.6 Conclusion and Perspective 60

4 Mixed-precision Deep Neural Networks for image classification and patch-based compression 63
4.1 ASIC designs: energy efficiency versus flexibility 64
4.2 Related works 65
4.2.1 Model quantization 67
4.2.2 Alternatives to the Fully Connected layer (CNN bottleneck) 68
4.2.3 Autoencoder for patch-based image compression 68
4.3 Nonlinear quantized encoder 69
4.3.1 Half-Wave Most-Significant-Bit (HWMSB) activation 70
4.3.2 Layer-shared BitShift-based Normalization (BSN) 71
4.3.3 Pre-defined pruning with Group-wise Convolution (GC) 72
4.3.4 Compression of the bottleneck dense layer 73
4.4 Image reconstruction from patch-based quantized measurements with PURENET 73
4.5 Simulation results 75
4.5.1 Image classification on CIFAR-10 75
4.5.2 Comparison with prior works 76
4.5.3 Full-frame image compression 79
4.6 Conclusion and Perspective 81

5 Hardware-aware Residual Networks with logic-gated skip connections and light-weight convolutional factorization 89
5.1 Context 90
5.2 Logic-gated Residual Neural Networks 91
5.2.1 Skip connections with OR and MUX gates 91
5.2.2 Experimental results 92
5.3 Compact CNN with light-weight convolutional factorization 93
5.3.1 MUX-Residual Block (MRB) 95
5.3.2 Convolution factorization leveraging CA-generated weights 96
5.3.3 Experiments 97
5.4 Conclusion 100

6 From image to video: Binarized Conv3D-LSTM model for efficient video inference 101
6.1 Context 102
6.2 Related works 102
6.3 BILLNET 103
6.3.1 Conv 3D Factorization 103
6.3.2 3D MUX-OR Residual (MOR) block 105
6.3.3 Fully-quantized LSTM 105
6.4 Multi-stage quantization training algorithm 107
6.5 Experiments 108
6.5.1 Settings 108
6.5.2 Results 109
6.6 Conclusion and Perspectives 113

7 Model compression via weight generation network and PRNG **115**
7.1 Hypernetworks: a bridge between random input and DNN weights 116
7.2 Practical Weight generation network designs 118
7.2.1 Weight generation network for FC layer 118
7.2.2 Conv2D layer with weight generation network 120
7.3 Experiments 123
7.3.1 MLP on MNIST 123
7.3.2 VGG-7 on CIFAR-10 124
7.4 Discussion 124
7.4.1 Configurations of the PRNG 124
7.4.2 Weight generation network architecture 126
7.4.3 Application 126
7.5 Conclusion and Perspective 127

8 Thesis Conclusion & Perspectives **129**
8.1 Thesis summary 129
8.2 Summary of the Contributions 129
8.3 Perspectives 131

Table of figures **134**

List of tables **137**

Publications & Patent **140**

Bibliography **I**

Résumé

0.1 Introduction

Récemment, les avancées dans le matériel de calcul, *e.g.*, la vitesse des cartes graphiques (GPU), ont profondément supporté le grand succès de l'apprentissage profond, en particulier des réseaux de neurones profonds (DNNs). Ces derniers sont des algorithmes de calcul très complexes, qui se composent de nombreux éléments appelés *couches*, dont chacune contient un grand nombre de paramètres et d'opérations, pour pouvoir imiter dans quelque sens le fonctionnement neurologique du cerveau humain. Les DNNs sont devenus la référence dans l'état de l'art concernant une large palette de cas d'usages tels que la vision par ordinateur [1], [2], le traitement du langage naturel [3], les mécanismes en oeuvre pour la voiture autonome [4] ou la robotique au sens large [5]. Ces performances exceptionnelles sont d'une part liées aux avancées dans le domaine des algorithmes d'optimisation, et d'autre part soutenues par l'augmentation substantielle de la complexité des réseaux de neurones. Ce dernier point se retrouve au travers divers éléments de dimensionnement de la topologie du réseau, sa profondeur (*i.e.*, le nombre des couches), sa largeur (*i.e.*, la nombre de neurones de chaque couche), le nombre de branches et l'hétérogénéité de connexions entre les couches...

Actuellement, la majorité de déploiement des DNNs repose sur l'informatique déporté (*cloud computing*) où des serveurs reçoivent les données brutes issues de capteurs, en vue de les traiter à l'aide de plateformes matérielles puissantes dédiées, typiquement, des GPUs. Cependant, porter l'inférence des DNNs au plus proche des capteurs semble de nos jours de plus en plus pertinent, notamment pour des raisons d'efficacité énergétique, de préservation de la vie privée, de latence de calcul et même lié aux problèmes de limitation de bande passante. Dans ce contexte, il est nécessaire de faire face aux obstacles liés à la grande taille des DNNs, en particulier concernant le coût mémoire et la charge conséquente de calculs qui se trouve dépasser souvent la capacité matérielle disponible dans les systèmes embarqués. Cela attire une grande attention sur le traitement efficace des DNNs. Différentes approches ont été proposées, en fonction des besoins en termes de versatilité du système, des tâches d'inférence ciblées, du budget matériel disponible (mémoire et complexité de calculs) et du niveau de la performance algorithmique en usage concret. Une des approches consiste à considérer la topologie de l'architecture du réseau comme une boîte noire en se focalisant uniquement sur l'accélération des noyaux de calcul et/ou sur l'optimisation du flot de données au sein des processeurs, en vue d'atteindre le meilleur niveau de d'efficacité énergétique de calcul (exprimé en TOPs/W). Par ailleurs, les travaux récents abordent également la parcimonie au sein des DNNs, *i.e.*, pour mieux profiter des paramètres/activations ayant une valeur nulle. Enfin, d'autres travaux se focalisent sur des plateformes de calcul qui permettent de supporter des architectures de réseaux impliquant différents types de précisions de calculs. Malgré le fait que ces approches peuvent servir à une large palette de DNNs dans diverses applications, ce avec une amélioration prometteuse en terme de consommation énergétique et d'efficacité de calcul, le niveau d'efficacité est toujours limité à cause des contraintes inhérentes à l'architecture elle-même. En effet, du fait du surparamétrage lié au grand nombre de paramètres et d'opérations des DNNs, ce qui cause une forte redondance dans les poids du réseau et implique de facto le phénomène de sur-apprentissage. En outre, puisque les architectures de DNNs sont conçues tout d'abord pour les applications logicielles sans avoir une perspective matérielle, un effort considérable est souvent nécessaire au niveau de la phase d'implémentation matérielle. Toutes ces raisons

nous mène à un besoin identifié en co-conception matérielle-algorithmique afin de pouvoir débloquer le niveau d'efficacité tout en renforçant le compromis entre la performance algorithmique et l'efficacité énergétique de calculs. Cette approche a ouvert la voie à un champs de recherches portant sur l'optimisation de la conception d'architectures de réseaux dédiées ainsi que la compression d'architectures pré-existantes en vue de leur intégration sous contraintes. Cet axe de recherche a notamment pour objectif final la conception de puces micro-électroniques dédiées à des applications spécifiques (*i.e.*, Application-Specific Integrated Circuits, ASICs).

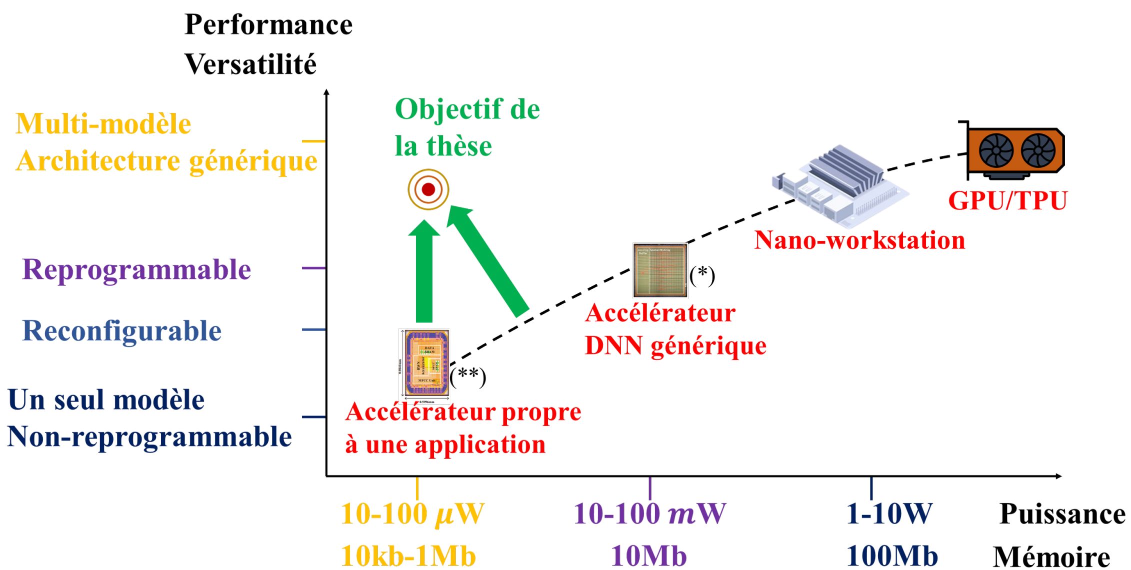


Figure 1: Compromise matérielle-algorithmique de différentes plateformes d'accélération de DNNs. (∗): Die micrographe de l'accélérateur générique Eyeriss [6]. (∗∗): Die micrographe d'un chip de 4.53µW dédié pour la vérification de voix et le keyword spotting [7].

La Figure 1 présente une taxonomie des plateformes d'accélération de DNNs, en fonction de l'ordre de grandeur des ressources matérielles, du niveau de performance algorithmique et du degré de reprogrammabilité/versatilité. Les plateformes génériques comme les GPUs/TPUs ou les nano-workstation permettent le déploiement d'une large gamme d'applications, ce avec une haute performance algorithmique et un support de calcul générique permettant la programmation de n'importe quelle type d'architecture de réseau. Cependant, ce type de plateforme requiert typiquement un budget élevé en mémoire (*i.e.*, >100Mb) et nécessite un niveau élevé de puissance (*i.e.*, ~1-10W). A l'opposé, une méthodologie de conception type ASIC permet de satisfaire au mieux les différentes contraintes matérielles applicatives extrêmes dans un contexte de système embarqué (*i.e.*, ~10-100µW, ~10kb-1Mb), mais au sacrifice de la versatilité algorithmique. Par ailleurs, à l'interface des deux types de dispositifs précédents, on constate l'émergence de processeurs compacts dédiés à l'IA, exhibant un niveau de reprogrammabilité relativement élevé permettant le déploiement d'une diversité d'architectures, sous des contraintes matérielles restant limitées (*i.e.*, ~10-100mW, ~10Mb).

Dans le cadre particulier de cette thèse, nous visons à améliorer encore le compromis algorithmique-matériel des accélérateurs de DNNs de type ASIC, en utilisant les méthodes les plus avancées à l'état de l'art pour la compression de DNNs et de conception des modèles efficaces.

Dans la littérature scientifique, il est possible d'identifier deux grandes classes de méthodologies de conception de réseaux compacts. La première consiste à dimensionner directement une

architecture compacte lors de sa conception. Cette architecture neuronale peut s'appuyer sur des blocs élémentaires efficaces comme montrés par les modèles de MobileNet [8] ou ShuffleNet [9], soit automatiquement en utilisant la recherche d'architectures (NAS) proposée par [10]. Par opposition, il est aussi possible de compresser un modèle existant pour réduire sa taille mémoire ainsi que la complexité de calcul. Premièrement, la quantification (paramètres et données, *i.e.*, des poids et des activations) [11], [12], [13] semble un levier essentiel pour le portage frugal d'architectures. Ensuite, l'élagage [14], [15], [16] est une autre technique utile à la simplification algorithmique, qui vise à retirer des opérations de calculs, des poids ou des activations au sein de la topologie. La forme de l'élagage peut se faire d'une manière statique [16], ou bien dynamique en prenant en compte les caractéristiques des données en entrée [17], [18]. Par ailleurs, la factorisation est une technique consistant à décomposer chaque tenseur de poids du DNNs en plusieurs sous-tenseurs de tailles plus petites [19], [20]. Enfin, pour réduire spécifiquement le nombre de paramètres du modèle, les travaux récents [21], [22] introduisent un modèle auxiliaire de plus petite taille générant à la volée les poids du modèle principal. A noter que le chapitre 2 de cette thèse détaille ces différents leviers algorithmiques en vue d'obtenir des modèles très compacts.

Dans ce manuscrit, nous avons choisi la quantification comme étant la technique pivot et centrale, permettant de réduire la complexité du portage matériel d'une topologie de réseau. Tandis que la plupart des travaux de l'état de l'art ne se focalisent que sur la quantification des poids et des activations, nous proposons une approche plus globale permettant de s'assurer de l'absence de couches non quantifiées présentes dans les DNNs modernes tel que la normalisation (typiquement la Batch Normalization, BN [23]) et les connexions résiduelles [24]. Ce choix s'appuie notamment sur l'hypothèse d'une plateforme matérielle n'embarquant que des opérateurs de calculs à arithmétique réduite. Ces propositions sont présentées au travers de différents chapitres de cette thèse, du Chapitre 3 au Chapitre 6 et en combinaison avec des blocs de convolution compacts (Chapitre 5 et Chapitre 6). L'objectif de ce travail réside dans l'amélioration des compromis entre performance algorithmique et efficacité en mémoire/calcul, tout en favorisant un haut niveau de compatibilité matérielle des topologies investiguées. Enfin, dans le Chapitre 7, nous présentons une ouverture sur des travaux préliminaires portant sur la compression de DNNs exploitant l'usage de modèles auxiliaires et des générateurs automatiques de nombres pseudo-aléatoires (PRNG).

0.2 Co-conception matérielle-algorithmique des réseaux de neurones

0.2.1 Quantification des poids avec égalisation de l'histogramme des poids

La conception des modèles à faible précision est une approche qui vise à réduire la précision des poids et des activations de 32-bit à 8-bit voire bien en deçà. Cela permet de réduire avantageusement à la fois le besoin de mémoire et aussi la complexité de calcul pour l'implémentation des MACs (Multiply and Accumulate) dédiés. Il existe deux façons de quantifier un réseau, soit la quantification post-entraînement soit l'usage d'opération de quantification lors de l'entrainement. En général, la seconde technique fournit de biens meilleurs niveaux de performances algorithmiques, en appliquant directement des opérations de quantification aux valeurs latentes dans leur représentation flottantes (pour les poids et les activations) lors de la phase d'apprentissage. Cela a pour effet de forcer le comportement du réseau à s'adapter en prenant en compte les effets causés par la faible précision de calcul. Pour augmenter encore la performance de ces modèles dits quantifiés, il est possible d'ajuster la fonction de quantification, ce pendant cette phase d'entraînement. Les paramètres de la quantification peuvent donc s'adapter sous la minimisation de l'erreur de quantification estimée [13], [25], ou bien d'une fonction de coût associée à la tâche d'inférence ciblée [26].

Le Chapitre 3 présente une nouvelle méthode d'entraînement s'appuyant sur une quantification

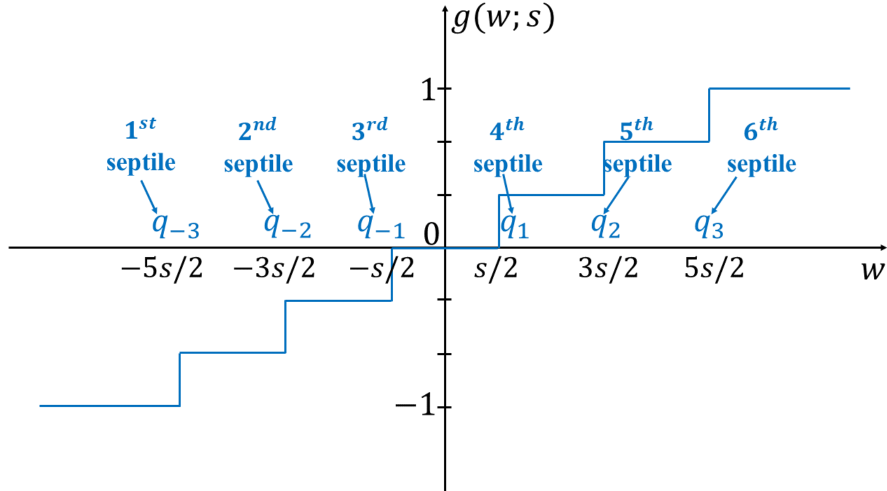


Figure 2: Quantification linéaire symmétrique avec une distribution uniforme lorsque n-quantiles (q_{-i}, q_i) sont symmétriques and coincident avec les seuils de quantification.

linéaire symétrique des poids. Concrètement, la fonction de quantification linéaire symétrique est paramétrée par un facteur d'échelle s qui définit le pas entre des seuils de quantification. En partant du constat que les méthodes de quantification présentes dans l'état de l'art n'exploitent pas au mieux tous les niveaux de quantification disponibles (pouvant causer une limitation de l'efficacité algorithmique), notre méthode appelée *Histogram-Equalized Quantization* (HEQ), réalise une mise à jour de s en fonction de statistiques tirées sur la distribution des poids flottants de façon à forcer les poids dans leurs vues quantifiées suivent une distribution plus uniforme. Par conséquent, HEQ cherche en quelque sorte à maximiser l'entropie des poids quantifiés, de sorte à profiter au maximum de tous les niveaux de quantifications disponibles. Figure 2 présente la condition pour avoir une équi-distribution entre n niveaux de quantification, cela équivaut à forcer la coïncidence entre les seuils de quantification et les n-quantiles (*i.e.*, $n-1$ points qui partionnent l'histogramme des poids flottants en n morceaux égaux). Prenant en compte cette condition, nous proposons une approximation de s en fonction de n-quantiles, *i.e.*, $s = \frac{4}{(n-1)^2}\sum_{i=1}^{\frac{n-1}{2}}(|q_{-i}| + q_i)$. Cette approche est générique et a l'avantage d'être compatible avec la plupart des architectures de DNNs. Les résultats expérimentaux ont réussi à démontrer que HEQ permet d'obtenir des performances d'état de l'art voire au delà. A noter que cette technique HEQ est utilisée pour entraîner divers types de réseaux présenté dans cette thèse, allant de réseaux à précision mixte (Chapitre 4) à des modèles compacts intégrant des couches récurrentes (Chapitre 6).

0.2.2 DNNs à précision mixte pour multi-tâche non-corrélé

Comme mentionné précédemment, les accélérateurs propres à une application sont strictement limités en terme de la flexibilité. Concrètement, ces accélérateurs sont conçus spécifiquement pour ne servir qu'à une seule tâche ou à un certain nombre de tâches corrélées telles que la détection du visage et la localisation de repères faciaux [27]. Dans un contexte où nous avons besoin de réaliser des tâches non supposément corrélées telles que la classification et la compression des images, il est difficile de concevoir une seule et même architecture de DNN dans un contexte de ressources restreintes (*e.g.*, mémoire, calcul). Le Chapitre 4 vise à proposer une architecture augmentant d'une part la versatilité applicative d'accélérateurs dédiés à une application, et d'autre part améliorer le compromis algorithmique-architecture (Figure 3). Dans ce cas, deux tâches algorithmiques partagent une unique architecture d'encodeur, dont les poids peuvent être mis à jour en fonction de la tâche d'inférence ciblée.

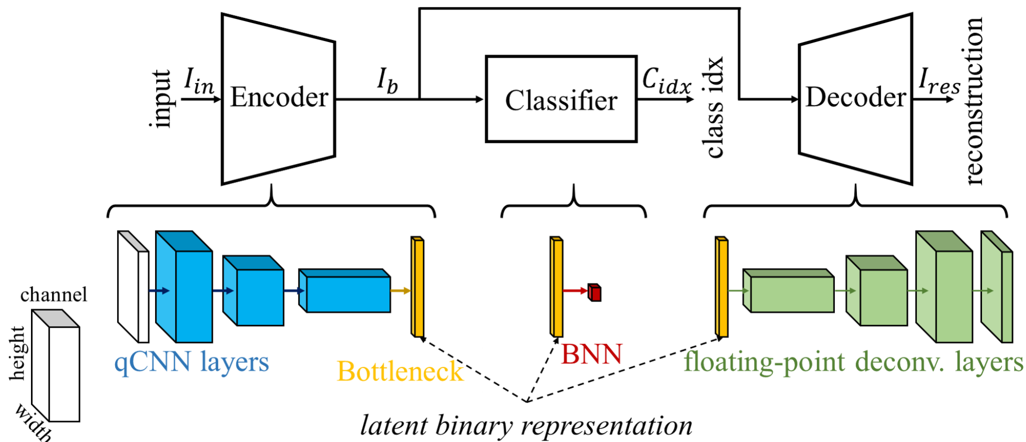


Figure 3: Description schématique de l'encodeur multi-tâche à précision mixte, du classificateur binarisé et du décodeur flottant pour la reconstruction d'image entière à partir des vecteurs de compression par blocs.

L'élément clé dans ce chapitre est donc une structure dite d'encodeur qui se trouve être à précision mixte, conçu et dimensionné de façon précise. Cette topologie à précision mixte, construit manuellement, intègre des poids quinaires (5 niveaux), ternaires (3 niveaux) et binaires ainsi que des activations entre 1 et 2bit. Puisque les premières couches sont cruciales pour extraire directement des informations discriminantes à partir de données en entrée, plus de précision y est accordé de façon à mieux guider l'entraînement du modèle vers un haut niveau de performance. En plus de l'usage de la méthode HEQ (en Chapitre 3), cette topologie exploite une nouvelle fonction de quantification des activations hautement compatible avec une implémentation matérielle (*i.e.*, position du bit de poids fort). Ensuite, nous adressons l'obstacle des normalisations par batch dans le contexte de la quantification, en proposant une approximation en puissance de 2 équivalente à un déplacement de bit (bitshift). L'ensemble de ces propositions favorise de manière notable la compatibilité matérielle de l'encodeur pour une implémentation de type RTL. Dans une configuration où l'encodeur requiert une taille de mémoire de seulement 1Mb, nous obtenons un taux de prédictions correctes de 87.5% sur la base de données Cifar10. D'un autre côté, nous démontrons aussi qu'il est possible d'utiliser cette architecture d'encodeur pour compresser des images par blocs tout en assurant une reconstruction de l'image entière à l'aide de notre décodeur PURENET. Concrètement, PURENET peut rendre les images presque sans artéfacts de blocs, ce sous un flux de données très faible, surpassant certaines méthodes existantes sous la contrainte d'un bitrate constant par bloc.

0.2.3 Réseaux de neurones résiduels avec les portes logiques et factorisation de convolution compacte

Malgré le fait que les méthodes de compression de DNNs ont obtenu des résultats exceptionnels, la compatibilité des modèles compressés reste une question ouverte. Une des raisons est que la méthode de compression ne prend pas –pleinement– en compte la correspondance entre l'algorithme et son implémentation en matérielle. Une autre raison est liée à l'architecture du modèle à compresser elle-même. Un exemple concret de cette limitation est l'implémentation matérielle des connexions résiduelles (*e.g.*, ResNet [24]) lorsque le modèle est fortement quantifié.

Par ailleurs, dans les Chapitre 5 et 6, nous proposons des architecture intégrant des chemins résiduels compacts, dont les connexions se font uniquement à l'aide de portes logiques comme OU, MUX et/ou des opérations compatibles avec le matériel comme le bitshift ou le bitcount (Fig-

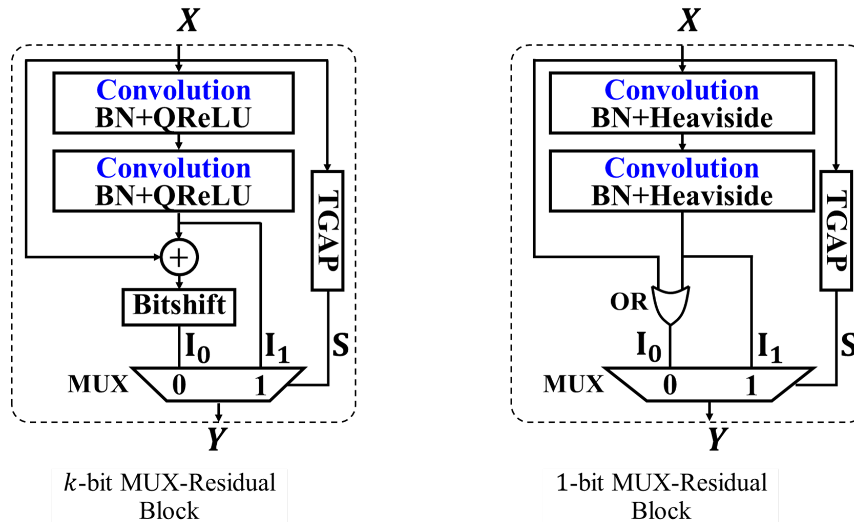


Figure 4: Le bloc résiduel élémentaire avec des portes logiques OU, MUX et l'opération bitshift.

ure 4). En détail, il s'agit d'un multiplexeur 2 vers 1, dont les entrées I_0 et I_1 correspondent à deux chemins de calculs et dont le signal de contrôle binaire S est issu d'une unité de calcul TGAP agissant directement sur l'entrée X. Cette structure est proposée en particulier pour des couches de convolution 2D utilisées au sein d'architectures de réseau pour des tâches de classification d'images dans le Chapitre 5 ainsi que pour des couches de convolution 3D dans le Chapitre 6.

En parallèle, et pour réduire drastiquement la taille du modèle, nous proposons en outre une factorisation de convolution similaire à celle présentée dans ResNext [28], consistant à effectuer une convolution par groupe 3×3 entre deux convolution 1×1. Cependant, cette factorisation n'inclut ni les activations ni les normalisations. Enfin, afin de réduire encore le besoin en mémoire, nous utilisons une structure d'automate cellulaire permettant de générer à la volée la matrice des poids de la deuxième convolution 1×1. Cette combinaison de couches, adossée aux méthodes de quantification déjà mentionnées permet une réduction drastique de la taille des réseaux ciblés, les rendant plus fortement compatibles avec des systèmes embarqués contraints. Les résultats expérimentaux reportés dans ce manuscrit confirment l'intérêt de cette optimisation conjointe, exploitant différents leviers de conception de topologies, et ce pour divers types de couches de réseau (convolution 2D/3D, Long Short-Term Memory, LSTM).

0.2.4 Génération des poids de DNNs avec des modèles auxiliaire et PRNG

Le Chapitre 7 se concentre sur des travaux préliminaires portant sur une approche de compression des DNNs exploratoire, davantage en rupture par rapport à l'état de l'art ainsi que les chapitres précédents de cette thèse. Le concept général de cette approche est présenté dans la Figure 5, dont les poids $\boldsymbol{W}$ du modèle principal ne sont pas directement stockés dans la mémoire, mais générés à la volée à partir de la sortie d'un modèle auxiliaire qui prend quant à lui en entrée des séquences aléatoires issues d'un PRNG (Pseudo-Random Number Generator). Deux déclinaisons de cette approche sont rapportés, pour les couches entièrement connectées et pour les couches de convolution 2D. Les résultats expérimentaux de ce chapitre démontrent qu'il est possible de réduire le besoin de mémoire via cette approche, cependant, le surcoût calculatoire associé peut y être un frein. C'est pourquoi cette approche requiert encore à l'heure actuelle davantage d'investigations scientifiques, notamment s'agissant de son couplage avec les techniques de quantification, d'élagage dynamique ou de mécanismes d'attention.

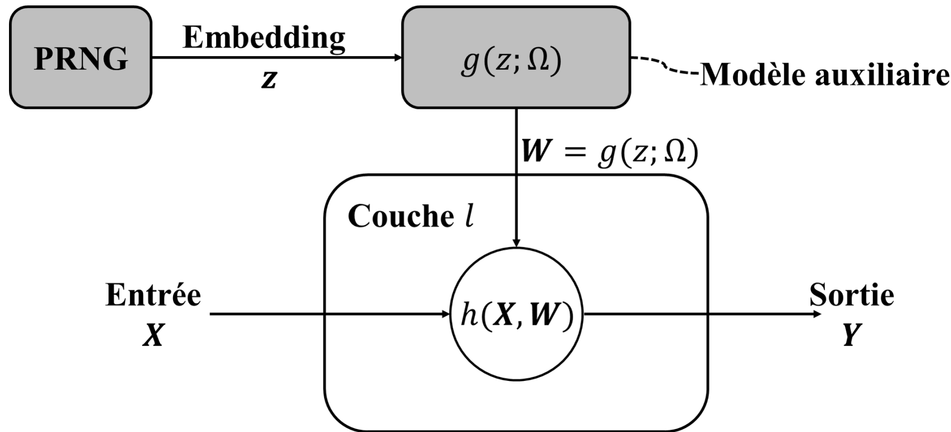


Figure 5: Génération des poids d'une couche à l'aide d'un modèle auxiliaire et générateur automatique de nombres pseudo-aléatoire (PRNG).

0.3 Conclusion et ouvertures

Ce manuscrit aborde donc la question du portage de l'inférence s'appuyant sur les techniques d'apprentissage profond, au sein des systèmes embarqués à fortes contraintes matérielles, et bénéficiant d'une conception dédiée. Pour cela, il s'est tout d'abord agit d'analyser ce qui se fait dans le domaine des puces d'accélération de réseaux de neurones, et leurs spécifications, en particulier leur consommation en mémoire, en puissance et leur niveau requis en termes de versatilité applicative. Ces propriétés sont mentionnées en lien avec les méthodes d'optimisation tant au niveau de l'implémentation matérielle qu'au niveau de la conception algorithmique. L'importance de la conception conjointe entre l'algorithme et le matériel a pu être –encore une fois– démontré, notamment grâce au déploiement concret des techniques de quantification étendues à l'architecture dans son ensemble (y compris concernant les étages de normalisation, les connexions résiduelles et les couches récurrentes). Ces propositions algorithmiques, compatibles avec un portage matériel frugal, sont appliquées sur des architectures neuronales légères (factorisation de convolutions et usage de poids générés à la volée) dans le but d'obtenir des modèles certes compacts mais restant efficaces. Enfin, ce manuscrit propose une étude préliminaire sur l'utilisation de modèles auxiliaires couplés à des PRNGs pour réduire la taille en mémoire des réseaux.

Différentes perspectives sont proposées tout au long de ce manuscrit pour continuer à améliorer nos contributions. Au vu du travail de cette thèse mais aussi des avancées récentes de l'état de l'art à ce sujet, il est désormais indéniable que les méthodes de quantification jouent un rôle crucial dans la co-optimisation des réseaux de neurones en vue de leur portage sur cible matérielle. La littérature scientifique semble s'accorder sur le fait de l'importance d'adapter les fonctions de quantification au cours de l'entraînement du modèle afin d'en augmenter sa capacité de représentation des données. Réduire le gap entre le modèle flottant et le modèle quantifié reste une problématique importante, pouvant être abordé notamment en utilisant une quantification à précision mixte, ou bien des connexions résiduelles à précision plus élevée pour mieux préserver la propagation de l'information au travers du réseau. Par ailleurs, un point très peu abordé dans ce travail est l'apprentissage par distillation des connaissances [29] qui semble cependant être aussi une voie prometteuse et pertinente à explorer pour ce problème.

La recherche automatique d'architecture (Neural Architecture Search, NAS) est un des champs de recherche les plus actifs en apprentissage profond actuellement. Et pour cause, ces algorithmes ont pour tâche de permettre la conception et l'optimisation automatique des architectures, avec le moins d'intervention humaine dans la boucle. Les avancées dans ce domaine viennent d'ouvrir la

possibilité d'effectuer ces procédures de NAS avec un coût plus abordable que par le passé [30], dépassant les solution issues d'un dimensionnement à la main. Les blocs et les modèles efficaces proposés dans ce manuscrit peuvent donc bien évidemment s'intégrer dans le cadre d'un algorithme de NAS, permettant d'élargir l'espace de recherche et de trouver le modèle le plus optimal pour une tâche donnée, sous contraintes matérielles. En outre, et par rapport aux travaux développés dans cette thèse, nous pouvons également envisager d'utiliser des approches de NAS pour optimiser l'architecture du modèle auxiliaire (cf. Chapitre 7).

Par ailleurs, le partage des poids [31], [32] est aussi un levier algorithmique encore assez peu exploré pour réduire la redondance des paramètres de réseaux. A noter que le partage de paramètres peut s'établir à trois niveaux: à l'intérieur d'une même couche, entre les couches d'un réseau ou même entre différents réseaux. Dans le cadre de ce manuscrit, une extension possible s'appuierait plutôt sur le deuxième niveau, en utilisant une seule et unique base de filtres convolutifs pour une variété de réseaux auxiliaires. D'un autre côté, un partage de poids entre réseaux peut être avantageux, typiquement en ce qui concerne un accélérateur multi-tâche (cf. Chapitre 4), en vue de réduire le besoin mémoire total nécessaire au stockage des paramètres pour toutes les tâches à adresser.

Enfin, au regard des résultats présentés dans ce manuscrit, il est difficile de remettre en cause l'intérêt confirmé de combiner les différents leviers algorithmiques de l'état de l'art en vue d'atteindre les meilleurs compromis performances/conso pour un portage matériel d'un réseau de neurones profond. Cela nécessite non seulement une conception conjointe entre l'approche de compression et l'architecture du réseau, mais aussi d'adapter en conséquence la procédure d'apprentissage et les moyens de contournement utilisés pour limiter les effets du sur-apprentissage.

1 Introduction

Recent advances in computational hardware platforms have greatly supported the remarkable success of Deep Learning (DL) and Deep Neural Networks (DNNs) in the last few years. Nowadays, DNNs have complex architecture that embeds a huge number of parameters and complex operations, in a sense to mimic the ability of human brain on extracting the relevant features from the data. The performance of DNN models have been improving rapidly and can be typically observed based on the start-of-the-art prediction accuracy on ImageNet [33] benchmark: from 63.3% in 2012 (AlexNet [1]) to 90.88% in 2021 (CoAtNet-7 [34]), *i.e.*, 25% after only a decade. Along with the improvement of optimization algorithms, this remarkable improvement also relies on a common rule of thumb: scale up the model in terms of depth (*i.e.*, the number of layers), width (*i.e.*, the number of feature maps), the number of branches and skip connections. Such breakthrough performance of DNNs results in an explosion of artificial intelligence (AI) applications in different fields, including computer vision [1], [2], machine translation [3], speech recognition [35], self-driving cars [4] or robotics [5], without mentioning all data analysis use cases [36].

Although the mainstay deployment of DNNs is sending raw sensory data to cloud or server equipped by generic-purpose and robust graphics processing units (GPUs), pushing DNN inference to the edge (*e.g.*, smartphone, wearable devices, sensors) is increasingly demanded, for the reason of energy-efficiency, privacy, latency or even limited bandwidth. However, this time we have to deal with several hardware-related overheads of these resource-constrained systems. The first key bottleneck is the memory-related cost. Modern DNNs have hundreds up to billions of parameters: AlexNet contains 60M while CoAtNet-7 has up to 2.4G weights. Generally, each weight is represented using floating-point (32b) representation, this means that the on-chip memory requirement to store the whole model, at the time of writing this thesis, is much larger than 1Gb. These numbers simply do not fit on-chip storage of low-power embedded systems, which typically contain 1-10Mb of memory size. Consequently, off-chip costly DRAM accesses is required, which causes stringent impacts in terms of energy efficiency and latency.

Another key bottleneck is the computational cost, measured by both the number of operations and the complexity of each operations. For instance, ResNet-50 [24] takes 3.9G Multiply-ACcumulate (MAC) operations to process a single 224×224 image. Running this model with data from a much higher resolution camera thus requires a huge computation throughput which is far beyond the capacity of embedded systems. Moreover, since the DNNs conventionally contain floating-point 32-bit weights and activations, they also consume a large amount of computational

resources. Concretely, a 32b MAC requires hundreds of Xilinx FPGA slices [37], hence disadvantageously increasing the footprint area and the energy consumption to perform the whole computation.

Despite the aforementioned challenges, the field of efficient DNN processing has drawn great attention in the last few years. Depending on the versatility, the targeted applications, the available hardware resources and the expected level of algorithmic performance, different approaches have been proposed. One approach is to treat DNN model as a black box and only focus on accelerating the kernel computation and/or optimizing the dataflow of accelerator platforms to achieve the highest tera-operations per second per watt (TOPS/W) as the main efficiency Figure of Merit [38], [6]. Recent works also focus on the sparsity of the models, *i.e.*, efficiently treating the computations of zero-valued weights and activations [39]. Although this option can serve for general-purpose DNN processing with promising improvement in terms of energy and computational efficiency while preserving the algorithmic performance, the efficiency is still strictly upper-bounded by the architecture of the models. Indeed, one of the main reasons for the robustness of DNN is due to its over-parameterization with an increasingly huge number of parameters and operations. The models usually exhibit redundant computations and large overfitting, *i.e.*, they are biased by small variations of the training examples and do not well generalize on unseen data. Moreover, since the original model architectures are firstly designed only for software application without hardware specification perspective, it requires considerable efforts on the implementation stage. Therefore, jointly optimizing the DNN algorithm with hardware implementation perspective is ubiquitous to unblock the efficiency limitation and boost the hardware-algorithm trade-offs. This approach has created a large space for researches on DNN compression and efficient model architecture design, mainly related to the design of Application-Specific Integrated Circuits (ASICs) for a certain range of targeted tasks.

Figure 1.1 depicts the order of required budget versus algorithmic performance-genericity level of different DNN processing platform. Despite the ability of serving for a wide range of applications with highest algorithmic quality and generic computational support for different types of DNNs, general-purpose platforms such as large GPU/TPUs and nano-workstations require a large amount of memory (*i.e.*, ~100Mb), power consumption (*i.e.*, ~1-10Mb or even more) and large-sized footprint area. On the other hands, ASIC designs, including generic DNN accelerators such as Eyeriss [6] and application-specific DNN accelerators such as the voice processing system in [7], can better fulfill the resource-constrained devices, however, this is achieved by sacrificing the scalability as well as the application-versatility of the system. Concretely, generic DNN chips with regrogrammability/reconfigurability can be modified to adapt to different DNN architectures or applications, while ensuring a lower power (*i.e.*, ~10-100mW) and memory (*i.e.*, ~10Mb) requirement compared to GPUs/TPUs or nano-workstation. In the most extreme case, application-specific accelerators, with a highest level of hardware-algorithm customization, are specifically designed for some targeted tasks. They achieve a lowest power consumption level (*i.e.*, $\sim 10-100\mu W$) and smallest memory budget (*i.e.*, ~10kb-1Mb), by sacrificing even the reconfigurability while optimizing both hardware and DNN algorithms according to the estimated complexity of the targeted tasks. In this thesis, we aim at improving the hardware-algorithmic trade-offs of DNN-based ASIC accelerators, by leveraging different model compression and efficient architecture design methods.

Model quantization is a relevant approach to compress the model by reducing the precision of model's weights and/or activations from the original 32-bit encoding to 8-bit [40] or less [41], [12]. This way, compared to the full-precision model, we advantageously reduce the memory-related cost so as the computational complexity, the power consumption and footprint.

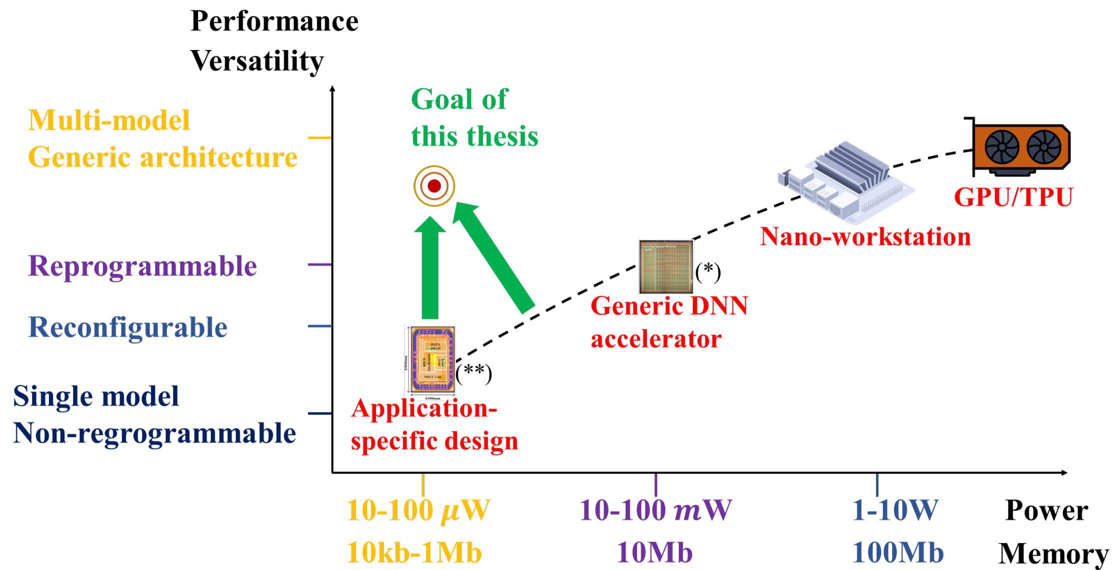


Figure 1.1: Typical hardware-algorithmic trade-offs of different DNN processing platforms. (∗): Die micrograph of Eyeriss DNN accelerator [6]. (∗∗): Layout of the 4.53µW accelerator enabling both speaker verification and keyword spotting [7].

Network pruning is another relevant approach to alleviate the hardware overheads by reducing the number of operations, operands and weights in DNNs. This approach is proposed based on the observation that DNNs usually exhibits sparsity and redundancy, and the models are largely overfitted with a densely-connected architecture. The pruned network contains less parameters and operations compared to its densely-connected counterpart, and in the most general case can be accelerated by sparse-aware matrix calculator [42], [43].

Efficient architecture design aims at alleviating the hardware-expensive of canonical models, mainly the convolutional layers, by introducing alternative architectures leveraging less parameters and computations [8], [9] or favoring the feature maps reuse [44]. Beyond the aforementioned hand-crafted designs, recent works turn to Neural Architecture Search (NAS [10]) enabling us to automatically searching for the best architecture given some hardware-related constraints (*i.e.*, model size, number of MACs, energy or latency).

These model compression methods and efficient architecture design schemes will be discussed later in detail in Chapter 2.

1.1 Problematic

Despite tremendous progress with remarkable results in the last few years, there still exists many shortcomings for the design of hardware-aware DNNs and model compression methods for ASIC-based platforms. For instance, most of the model compression methods focus on existing large models to achieve extremely high compression rates, however, the compressed network may not be easily implemented from a hardware point of view. In some cases, the additional elements introduced by the compression procedure may result in a considerable deficiency of the DNN processing. A practical question has been proposed: should we continue to use this paradigm, or is it better to firstly propose a new hardware-compliant, light-weight model architecture to which we further apply compression methods at a moderate rate?

There is another issue that is more related to the applicability of the ASIC designs. Currently,

most of ASIC accelerators are dedicated to a single task (*e.g.*, image classification), or some correlated tasks (*e.g.* face detection and alignment) thanks to the feature-sharing property of these applications. An open question is whether an application-specific accelerator can be used to deploy DNN processing of uncorrelated tasks such as the mid-level image classification and the low-level image compression, while always capping hardware constraints. Since the task's complexity and the computational complexity of these uncorrelated tasks are different, it is a real challenge to build a single hardware-constrained model topology that can fulfill the required algorithmic performance of both tasks.

This thesis makes use of network quantization as one of the key model compression technique for the design of hardware-compliant DNNs. Although obtaining excellent performance, the hardware compatibility of some previously proposed quantizers is still questionable. Besides, how to determine the optimal precision of weights/activations of every layers is challenging. Moreover, most of existing works only focus on the quantization of model's weights and activations, while there is a lack of attention to other key elements in modern DNNs such as the Batch Normalization (BN) and the skip connections. For low-precision accelerators, these operations may introduce new hardware overheads even though the weights and activations are all quantized.

Beyond network acceleration techniques that are well studied like quantization and pruning, there are rooms for proposing new research directions, for example the use of weight-generation network [21] and online-generated weights. The works in this thesis attempt to answer all the aforementioned questions and pave the way to future research directions.

1.2 Contributions and Thesis Outline

This thesis explores the design of light-weight DNNs taking advantages of a hardware-algorithm co-optimization to perform on-chip image/video processing. To this end, we firstly propose a novel linear symmetric quantization for weights, so-called Histogram-Equalized Quantization (HEQ), to favor more information-carrying capacity in the quantized weights. This quantization scheme is then applied to obtain the ternary and quinary weights in our proposed models. We also replace the hardware-expensive BN by the bit-shift approximation for low-precision fully-quantized neural networks. Moreover, we introduce two novel logic-gated residual connections using the OR and the Multiplexer gates, allowing to perform the skip connections with negligible implementation cost while keeping a homogeneous bit-width of intermediate data. To reduce furthermore the model size and the computational cost, we also propose a light-weight convolutional factorization which leverages on-line generated weights via Cellular Automaton. These hardware-algorithmic enablers, when combined together, enable us to design hardware-compatible networks with model size and computational cost fitting embedded systems. Finally, we explore the use of weight generation networks along with Pseudo Random Number Generator (PRNG) to compress the weights of DNNs. This thesis is organized as follows:

Chapter 2 presents a background of Deep Learning, the current DNN accelerators and the model compression/acceleration landscape. It first introduces the key concepts of the DNNs, including basic components such as layers and operations, existing model architectures, datasets and training systems that we used in this thesis. Next, we survey some common types of DNN accelerators, focusing on the compromise between hardware resources and algorithmic flexibility. Next, we provide a non-exhaustive list of related works in model compression and acceleration techniques, involving quantization, pruning, efficient architecture design/search, weight kernel decomposition and weight generation network...

Chapter 3 proposes a novel quantization method favoring the uniform distribution between output levels to increase the information-carrying capacity of the quantized weights, namely Histogram-

Equalized Quantization (HEQ). It automatically adjusts the step size of the quantization mapping during the training process based on the statistics of the real-valued proxy weights, such that the resulting quantized values are more equally distributed. This weight equalization scheme can be considered as a regularizer applied indirectly to the quantized weights. We provide experimental results to demonstrate the effectiveness of this adaptive QAT method compared to SOTA weight quantization schemes.

Chapter 4 shows that a compact DNN-based ASIC design can still serve for two uncorrelated applications: image classification and patch-based compression. The central proposition of this chapter is a mixed-precision, fully-quantized encoder topology making use of the proposed HEQ and binary networks along with the hardware-friendly Half-Wave Most-Significant-Bit (HWMSB) activation quantization and Bit-Shift Normalization. The reconstruction of full-resolution image is performed remotely by a dedicated decoder. Several simulation are carried out to show the possible degree of application-versatility for a compact neural network architecture towards ASIC designs. It also highlights the relevance of a careful hardware/algorithm co-design to reach the best compromise between hardware implementation complexity and algorithmic performance.

Chapter 5 firstly tackles the hardware mapping of the skip connections in the context of quantized neural networks. It typically introduces new residual networks of n-bit activations, whose skip connections can be easily implemented by integer operations, bitshift and logic gates (MUX, OR) rather than floating-point arithmetic hardware. Besides, we also propose a light-weight convolution factorization leveraging fixed-"random" weights which are generated on-the-fly via a Cellular Automaton. These hardware-compliant architectures, when combined with the proposed HEQ and other quantization techniques, enable an efficient model which may achieve better model size-accuracy trade-offs compared to previous works, while offering a better implementation compatibility. This also highlights the importance of a co-design of the model architecture and the compression techniques in improving the efficiency of compact neural networks.

Chapter 6 extends the proposed model architecture in Chapter 5 for embedded video inference. We first propose a Conv3D-LSTM model architecture, whose the convolutional part is a 3D version of the light-weight model in Chapter 5. In particular, to obtain a fully-binarized model, we also propose a method for quantizing both weights and hidden values of the Long Short-Term Memory (LSTM) layers. To the best of our knowledge, this is one of the first works on fully quantizing the LSTM for video classification. In order to alleviate severe performance degradation, we present a multi-stage training algorithm which gradually replace full-precision components of the model by their corresponding quantized version, then fine-tune the model to recover the performance level. Simulation results show the possibility of designing extremely low-precision Conv3D-LSTM model for embedded video processing.

Chapter 7 exploits the possibility of compressing DNNs through the use of a weight generation network, which transforms the random embedding issued from PRNG to the primary model's parameters. We present two examples of the weight generation network for the case of Fully-Connected and 2D Convolution layers. These preliminary results show that it is possible to obtain a significant compression rate, however, the additional computational cost is a major downside and needs to be addressed by integrating other compression techniques such as quantization or dynamic pruning into the weight generation model.

Chapter 8 summarizes the thesis problematic and contributions before discussing possible extensions that are not fully addressed through this thesis, as well as future direction for DNN compression/ acceleration towards efficient embedded inference.

2 Background

In this chapter, we first introduce some basic notions of Deep Learning, from the common layers to the complex model architectures and the learning process. We also present the datasets as well as the Deep Learning frameworks used throughout this thesis. We then provide a review on current DNN accelerators which can be categorized into three classes based on their application-versatility and flexibility. Finally, we systematically present state-of-the-art hardware-algorithm enablers facilitating embedded DNN processing.

Contents

2.1 Overview of Deep Neural Networks . . . 16
2.1.1 Layers . . . 17
2.1.2 Activation functions . . . 20
2.1.3 Loss function and regularization for DNNs . . . 22
2.1.4 Learning procedure . . . 24
2.1.5 Datasets . . . 27
2.1.6 Overfitting and training tricks . . . 28
2.1.7 Neural Network Architectures . . . 29
2.1.8 Deep learning frameworks . . . 34
2.2 DNN accelerators . . . 35
2.2.1 General-Purpose Accelerators (GPA) . . . 35
2.2.2 AI-Dedicated Accelerators (AIDA) . . . 36
2.2.3 Application-Specific Accelerators (ASA) . . . 36
2.2.4 Discussion . . . 37
2.3 DNN hardware-algorithmic enablers state-of-the-art . . . 38
2.3.1 Efficient architecture designs . . . 38
2.3.2 Model compression techniques . . . 42
2.3.3 Training trick for compact models: Knowledge Distillation . . . 50
2.4 Conclusion . . . 50

2.1 Overview of Deep Neural Networks

Neural network (NN) is a biologically-inspired computation algorithm which emulates the functionality of the brain for learning and solving problems. Figure 2.1 shows the typical scheme of a simple neural networks. The basic element in NN is *neuron*. Each neuron receives the input signals from predecessor neurons, performs computations on those signals and generates an output. This output may then branch out and connect to several successor neurons. Each connection is characterized by a *weight* corresponding to the scaling factor applied to the input signal. In the most popular case, neuron's computation consists of a weighted sum of the inputs followed by a nonlinear *activation function*. Neurons are organized into *layers*. Generally, neurons in the same layer share the same type of operation and are not connected. The neurons in the input layer receive signals from the input data, propagates them to the neurons in the hidden layers of the model, before ultimately reaching the output layer with the desired outcomes to the user. The number of neurons (*i.e.*, dimensionality) in each layer defines the *width* while the number of hidden layers gives the *depth* of the model. A model is considered as deep neural network (DNN) if the number of hidden layers is large (*e.g.*, typically more than three). Modern deep neural networks may have up to thousands of layers.

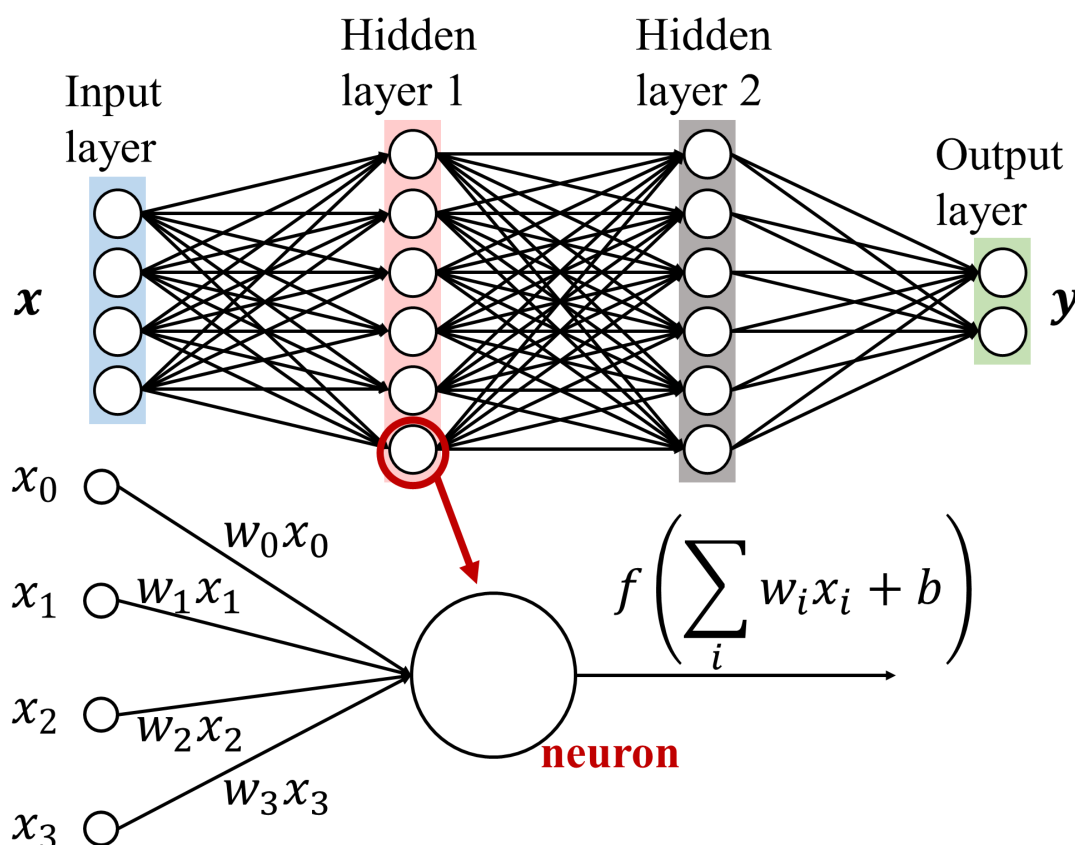


Figure 2.1: Example of a simple neural networks with 2 hidden layers. Neuron's computation involves applying a nonlinear activation function f to the weighted sum of the input values.

The NN depicted in Figure 2.1 contains only fully-connected layers, where each neuron in a layer is connected to all neurons of the previous layer. Depending on the nature of data, it is necessary to introduce different types of layer such as the 2D Convolution to extract the local correlation of signals, or a recurrent neural networks (RNN) to keep track long-term dependencies. Designing the model *architecture* consists of determining the number of layers, dimensionality of each layer, type of layers and connections between layers, type of activation functions... This architecture remains unchanged when the model learns to perform a given task. At the beginning,

these weights are randomly initialized but during the learning process or *training*, they are adjusted in response to some learning stimulus. The training of DNNs is a difficult task and deeply depends on the initialization of the parameters and the model architecture as well. Concretely, due to the non-convex optimization nature of the training process, an initial point can allows for the model to converge or not. Advanced initialization techniques such as [45] and [46] consider the impact of activation functions and the layer's dimensionality to define the statistics of the random initialization.

2.1.1 Layers

Let us consider a layer with index l which performs a linear operation h followed by a nonlinear activation function f. Denote $\boldsymbol{X}_l$ the input of the layer, $\boldsymbol{W}_l$ the layer's kernel and $\boldsymbol{b}_l$ the layer's bias, the output $\boldsymbol{X}_{l+1}$ can be described as follows:

$$\boldsymbol{X}_{l+1} = f(\boldsymbol{Z}_{l+1}) = f(h(\boldsymbol{X}_l, \boldsymbol{W}_l) + \boldsymbol{b}_l) \tag{2.1}$$

The transformation $\boldsymbol{Z}_{l+1} = h(\boldsymbol{X}_l, \boldsymbol{W}_l) + \boldsymbol{b}_l$, where $\boldsymbol{Z}_{l+1}$ is termed as the pre-activation, defines the layer type and in general, it can be equivalently represented under the form of a matrix-to-vector multiplication where the weight $\boldsymbol{W}_l$ can be organized into a transformation matrix. For the sake of simplicity, in the following subsections, we omit the bias $\boldsymbol{b}_l$ and only focus on the operation $h(\boldsymbol{X}_l, \boldsymbol{W}_l)$ of the common layers.

Fully connected (Dense) layer

Let us consider a fully connected (FC) layer which takes the input $\boldsymbol{x}_l \in \mathbb{R}^{n_l}$ and outputs $\boldsymbol{x}_{l+1} \in \mathbb{R}^{n_{l+1}}$. The weight matrix of this FC layer is $\boldsymbol{W}_l \in \mathbb{R}^{n_l \times n_{l+1}}$, corresponding to $n_l n_{l+1}$ learnable parameters. The following equation describes the linear operation $h(\boldsymbol{x}_l, \boldsymbol{W}_l)$ of this layer:

$$h(\boldsymbol{x}_l, \boldsymbol{W}_l) = \boldsymbol{W}_l^T \boldsymbol{x}_l \tag{2.2}$$

Since this layer performs n_{l+1} projections of support size n_l, the number of MAC operations is therefore $n_l n_{l+1}$. This way, each output unit is a weighted sum of all input features. The dense layer is often used at the end the model, where the high-level features are already extracted and we want to combine them to obtain the final output of the network.

Convolutional layers

Convolutional layers make use of the convolution in place of the linear operation h. In practice, this specialized operation is employed to process the data having a grid-like topology such as image (with 2-D grid of pixels) or video (with 3-D time-space grid). It consists in sliding a filter (or *filter kernel*) along all positions of the data and compute the dot products with the local units. Mathematically speaking, the discrete convolution can be represented as a multiplication by a block-circulant matrix, where the matrix is very sparse (*i.e.* many entries are equal to zero since the kernel size is smaller than that of the image) and has several entries constrained to be equal to the others. This operation is desired because it allows to keep track where the local features appear inside the data while using the same parameter for all locations. A neural network consisting of convolutional layers is generally termed as Convolutional Neural Network (CNN).

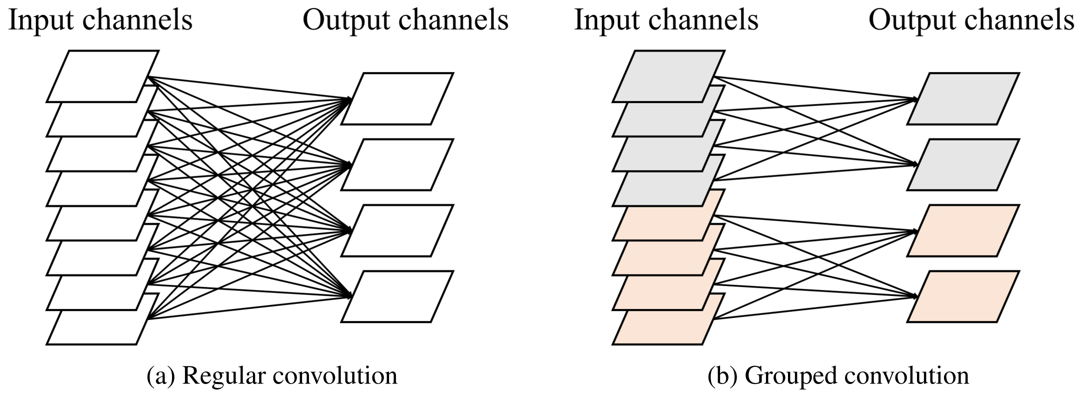
The 2D convolutional (Conv2D) layer is used in the context of images, where the input $\boldsymbol{X}_l$ is a 3D tensor $\boldsymbol{X}_l \in \mathbb{R}^{H_l \times V_l \times C_l}$, with C_l denotes the number of input channels (or *feature maps*), H_l and V_l are respectively the height and the width of the input feature maps. The weight tensor (also called *kernel*) $\boldsymbol{W}_l \in \mathbb{R}^{h \times v \times C_l \times C_{l+1}}$ is a 4D tensor where $h \times v$ represents the filter size and C_{l+1} denotes the number of output channels. This kernel contains hvC_lC_{l+1} learnable parameters. Each unit of the

pre-activation $\boldsymbol{Z}_{l+1} \in \mathbb{R}^{H_{l+1} \times V_{l+1} \times C_{l+1}}$ is then computed as follows:

$$Z_{l+1,i,j,c} = \sum_{k=1}^{c_l} \left(\boldsymbol{W}_{l,:,:,k,c} * \boldsymbol{X}_{l,:,:,k} \right)(i,j) = \sum_{k=1}^{c_l} \sum_{m=1}^{h} \sum_{n=1}^{w} W_{l,m,n,k,c} X_{l,i+m,j+n,k} \tag{2.3}$$

where $H_{l+1} \times V_{l+1}$ is the spatial resolution of the output feature maps and $*$ denotes the convolution. Each output feature map is connected to all input channels in a fully-connected manner similar to the case of FC layer (Figure 2.2a). The number of MACs of this Conv2D layer is $H_{l+1}V_{l+1}C_{l+1}hvC_l$. To reduce the number of parameters and MAC operations due to this dense connection scheme, the grouped convolution (GConv) has been introduced, in which the computation of each output channel only relates to the input channels within the group. Figure 2.2b depicts a grouped convolution where the connection pattern is defined in a structured manner. In the more general case, the group structure can be defined randomly or through the learning process [47], [48].

The 3D convolution (Conv3D [49]) is another type of convolutional layers where the kernel slides in three dimensions as opposed to two dimensions in the case of Conv2D. One example use case is video processing where each sample contains several contiguous frames in time. Therefore, it is necessary to expand the convolution along the temporal direction to learn the spatio-temporal features.



(a) Regular convolution

(b) Grouped convolution

Figure 2.2: Computational relationship between input and output channel maps in the case of a 2D convolution with 8 input channels and 4 output channels. a) regular convolution with dense connection; b) structurally grouped convolution with sparser connections. The number of connections is reduced by a factor equal to the number of groups ($g = 2$).

Depthwise and separable convolution layers

In the case of depthwise convolution (DWConv) layer, the spatial convolution is separately performed on one channel at a time, therefore there is no information mixture across different input channels. Given the input tensor $\boldsymbol{X}_l \in \mathbb{R}^{H_l \times V_l \times C_l}$, a depthwise convolution layer with the *depth multiplier* d_m contains the kernel $\boldsymbol{W}_l \in \mathbb{R}^{h \times v \times C_l \times d_m}$. The output of this layer is computed as follows:

- Split the input into C_l individual channels. Each channel can be considered as a 3D tensor of size $H_l \times V_l \times 1$.
- Convolve each input channel $\boldsymbol{X}_{l,:,:,c}$ with the corresponding depthwise kernel $\boldsymbol{W}_{l,:,:,c,:}$ to obtain d_m output channels.
- Concatenate these outputs along the channel dimension to obtain a single output of $d_m C_l$ feature maps.

Depthwise separable convolution (DSConv [50]) consists of a depthwise convolution with the depth multiplier $d_m = 1$ followed by a 1×1 (*pointwise*) convolution mapping the number of channels from C_l to C_{l+1}. It is proposed by hypothesizing that the spatial correlation and the cross-channel correlation of the regular convolution can be separable. In this case, the depthwise convolution is used to extract the spatial features while the pointwise convolution ensures the correlation between all input channels. Compared to the regular Conv2D, this DSConv layer contains only $hvC_l + C_lC_{l+1}$ parameters and requires only $H_{l+1}V_{l+1}C_lhv + H_{l+1}V_{l+1}C_{l+1}C_l$ MACs.

Pooling layers

Pooling is a way to downsample the output of the convolutional layer. Typically, the pooling function summarizes the data over a whole neighborhood by using the statistical features of that region. For example, the max pooling operation takes the maximum value over a window whereas the average pooling outputs the average value within that window. Since the maximum value and the average value do not change if we shuffle the values within a window, these pooling operations helps to obtain the feature representation that is invariant to small translations of the input. This property is useful in some context, for example image recognition, where we only need to know the presence of some certain features rather than their exact location. Moreover, the cascade of convolution and pooling layers allows to learn compact representation of the data with more general and abstract features. Since the pooling reduces the dimensionality of the data, it also helps to decrease the computational cost as well as the memory needs to store data for the deeper layers. More concretely, the spatial resolution of data is gradually decreased using pooling layers while the number of channels is generally increased along the depth dimension of a CNN.

Normalization layers

Modern deep neural networks has complex architectures with several layers. This also makes their training difficult because the distribution of the layer's inputs may easily change with the parameter update during training. Besides, since the input features may have different ranges, the network is biased towards the features of higher numerical values. Normalizing the layer inputs is a relevant approach to resolve these aforementioned issues and accelerate the learning process.

Batch Normalization (BN [23]) is the first introduced and also the most common type of normalization in deep learning. It standardizes the inputs to a layer for every mini-batch during the training procedure, using the statistics mean and variance of every feature in the mini-batch. Consider a mini-batch of size M. Denote $\boldsymbol{X}^{(m)}$ as the input tensor of the sample m ($1 \leq m \leq M$), and $x_i^{(m)}$ is a feature. During the training phase, the BN transform can be described as follows

$$\mu_i = \frac{1}{M} \sum_{m=1}^{M} x_i^{(m)} \tag{2.4}$$

$$\sigma_i^2 = \frac{1}{M} \sum_{m=1}^{M} \left(x_i^{(m)} - \mu_i \right)^2 \tag{2.5}$$

$$y_i^{(m)} = \gamma_i \frac{x_i^{(m)} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} + \beta_i \tag{2.6}$$

where μ_i and σ_i^2 are respectively the mean and the variance of the feature computed over the current mini-batch. The scale γ_i and bias β_i are learnable parameters which make sure that the transformation can represent the identity transform, hence preserving the representation power of the network. The constant ε is a small positive value to avoid dividing by zero. At inference time, the BN layer normalizes its input features using the moving mean $\mathbb{E}[x_i]$ and the moving variance

$\mathrm{Var}[x_i]$ recorded during the training process. These statistics are observed over all mini-batches of the training data set and fixed during inference. Therefore, at inference time, the BN simply corresponds to a linear transform applied to each input feature

$$y_i^{(m)} = \frac{\gamma_i}{\sqrt{\mathrm{Var}[x_i] + \varepsilon}} x_i^{(m)} + (\beta_i - \frac{\gamma_i \mathbb{E}[x_i]}{\sqrt{\mathrm{Var}[x_i] + \varepsilon}}) \tag{2.7}$$

In pratice, if the BN is used after a Conv2D layer, the statistics μ_i and σ_i^2 are separately computed on every channel, *i.e.*, $x_i^{(m)}$ is a 2D feature map. If the BN is used after the FC layer, these statistics are independently computed for every single neuron.

The use of BN has significantly eased the training of DNNs. One of the most popular hypothesis for BN's success is that it allows smoothing the optimization landscape [51]. Additionally, in order to keep the distribution of the intermediate values stable, the model only needs to learn the parameters β and γ rather than the whole weights and biases of several layers. As a result, BN reduces layer interdependence and simplifies the job of the optimizer. Note that, while BN computes the mean and the variance across the batch dimension, other techniques have also been proposed which normalize the features in a batch-independent manner. For example, Layer Normalization [52] computes the mean and the variance along all hidden units of the layer, while Instance Normalization [53] computes these statistics independently for each feature map and Group Normalization [54] operates in a group-wise manner. Besides these feature normalization techniques, [55] proposes to normalize the weights as a reparameterization to improve the optimizability of the network. In the scope of this thesis, we make use of the BN as the preference choice of model's normalization.

2.1.2 Activation functions

If a neural network only contains linear layers such as fully-connected layers and convolutional layers, then it can be equivalently constructed by using only a single linear layer. Therefore, it is necessary to insert the non-linearity between the linear operations, in order to improve the representation power of the model and establish a nonlinear mapping between the input and the output, which is generally the behavior of most systems in nature, including the brain. In neural networks, this non-linearity is introduced by using the element-wise activation function. The notion of *activation* comes from the biological inspiration of neuron: if the weighted sum of the input signals and the bias exceeds a certain threshold, the neuron is activated and outputs a response. Generally speaking, an activation function is a scalar, non-linear, monotonic and differentiable function.

Common continuous activation functions can be loosely divided into two categories: saturated functions and non-saturated functions. Intuitively, the graph of saturated functions stays within a horizontal band, while the graph of unbounded counterparts does not. Apart from these continuous functions, recent works also make use of quantized/discrete mappings to obtain low-precision DNNs satisfying hardware-related constraints.

Saturated activation functions

One of the most popular activation function is the *sigmoid* function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

The sigmoid function is a continuous function with the output value bounded in the interval $(0, 1)$. One noticeable property of this function is that it works in linear regime around the zero while

saturating (with vanishing gradient) when it comes far from the zero area. This hardens the learning process and limits the use of sigmoid function in deep neural networks, except for the output layer. Besides, its computation is also complicated as it leverages the exponential function.

To simplify the computation of the sigmoid, an approximated version called *hard sigmoid* has been proposed:

$$\mathrm{hardsigmoid}(x) = \mathrm{Clip}(\alpha x + 0.5, 0, 1) \tag{2.9}$$

where the Clip function is defined as $\mathrm{Clip}(x,a,b) = \max(a, \min(x,b))$ with $a < b$ are two clipping thresholds. The constant α controls the slope of the linear regime.

Hyperbolic tangent (or *tanh*) is another popular bounded activation function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.10}$$

This function is also differentiable everywhere and has output range between −1 and 1. Compared to the case of sigmoid, the hyperbolic tangent function has a linear regime around the zero but with a greater slope, while the saturation region is larger. This strengthens the vanishing gradient problem when input values come far from the zero point. Similar to the case of sigmoid, this function has a simplified version called *hard tanh* defined as follows

$$\mathrm{hardtanh}(x) = \mathrm{Clip}(\alpha x, -1, 1) \tag{2.11}$$

Softsign is another saturated activation function that is used in this thesis

$$\mathrm{softsign}(x) = \frac{x}{|x| + 1} \tag{2.12}$$

Non-saturated activation functions

One of the main drawback of these aforementioned saturated functions is that they allow to easily activate the neuron (with a nonzero output value) while do not well preserving the magnitude of the input signal, making it difficult to learn high relevant feature representations. To address these issues, the Rectifier Linear Unit (ReLU) has been introduced

$$\mathrm{ReLU}(x) = \max(x, 0) \tag{2.13}$$

which applies the identity to positive arguments while zeroing out all negative values. This way, the positive-part function lets positive input pass without modulating it, hence preserving the magnitude (hence the importance) of the features. The negative input is cut down, keeping the neuron inactivated and hence helping the model to learn the presence of some features within the data. Another benefit of ReLU is that both the function itself and its derivatives are computationally inexpensive compared to the previous saturated counterparts.

However, since the gradient of the negative part is strictly zero, ReLU function is also prone to the gradient vanishing problem. The LeakyReLU [56] (or LReLU) has been proposed to resolve this issue, by introducing a small slope (typically $\alpha = 0.01$) for the negative part and therefore sacrificing the sparsity of the output

$$\mathrm{LReLU}(x;\alpha) = \begin{cases} x & \text{if } x \geq 0, \\ \alpha x & \text{otherwise.} \end{cases} \tag{2.14}$$

Furthermore, [46] introduces the Parametric ReLU (PReLU) which learns the slope parameter α rather than fixing it.

Quantized activation functions and the Straight-Through Estimator (STE)

Although there exists a biological inspiration that human brain stores information in a discrete form [57], [58], quantized activation functions are mainly used in modern DNNs for the reason of hardware efficiency within restricted computational resources. Clearly, moving from full-precision representations to low-precision values allows to drastically reduce both the memory requirement, latency and computational complexity.

One of the most popular quantized activation functions is the binary *sign* function:

$$\operatorname{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{2.15}$$

It is easy to see that the derivative of this piece-wise constant function is zero almost everywhere, making it inappropriate to be used in the neural networks. Concretely, due to the chain rule, the gradient of the cost function with respect to the weights and activations before the sign operation are also zero, preventing the model from learning. One of the most popular approach to address this issue is using the Straight-Through Estimator (STE) technique [59]. Concretely, STE firstly chooses a continuous approximation for the discrete mapping, then replaces the almost-everywhere-zero derivative of the quantization function with the derivative of this approximation. In the case of the sign function, we choose the clipped identity (hardtanh in Eq. 2.11 with $\alpha = 1$) as its approximation (Figure 2.3a). The STE derivative of the sign function is computed as follows

$$\frac{\partial \operatorname{sign}(x)}{\partial x} = \begin{cases} 1 & \text{if } |x| \leq 1, \\ 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

which allows the gradient to pass through in the clipping range $[-1, 1]$ (see Figure 2.3b). Although this STE derivative does not allow to obtain the exact gradient with respect to the cost function, [60] proves that the resulting coarse gradient still provides a descent direction for minimizing the cost. In that context, it is noteworthy mentioning that the hyperbolic tangent and the softsign can be considered as soft approximations of the sign function.

Another well-known binary activation function is the *heaviside*

$$\operatorname{heaviside}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

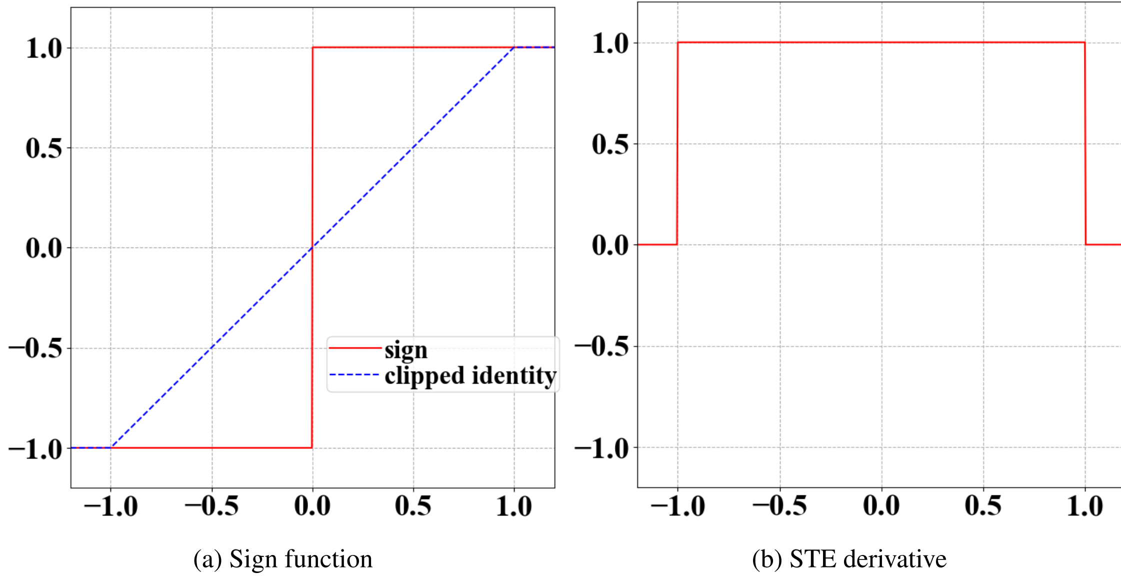
which promotes the sparsity (like ReLU) as it throws away all negative values. This function can be approximated by the hard sigmoid function (cf. Eq. 2.9) with the slope $\alpha = 0.5$. The STE gradient of this function is defined as follows

$$\frac{\partial \operatorname{heaviside}(x)}{\partial x} = \begin{cases} 1 & \text{if } |x| \leq 1, \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

where we still keep the derivative equal to 1 in the clipping range $[-1, 1]$ rather than 0.5, allowing to pass through the gradient without decreasing its magnitude (hence slowing down the parameter update).

2.1.3 Loss function and regularization for DNNs

Denote $\boldsymbol{Y}^{(1)}, \boldsymbol{Y}^{(2)} \ldots \boldsymbol{Y}^{(N)} \in \mathbb{R}^{n_L}$ the set of a DNN's output associated to a training dataset of N samples, where n_L is the dimensionality of the output layer. In the supervised learning, each training sample is associated with a target output $\hat{\boldsymbol{Y}}^{(i)}$. The per-example loss function ℓ measures



(a) Sign function

(b) STE derivative

Figure 2.3: Sign function and its approximated derivative using STE technique. a) the clipped identity is considered as a continuous approximation of the sign function. b) STE derivative of the sign function allowing the gradient to pass through within the clipping range $[-1,1]$.

the difference between the predicted output $\boldsymbol{Y}^{(i)}$ and the target $\hat{\boldsymbol{Y}}^{(i)}$ corresponding to that input. The goal of the learning algorithm is to reduce the empirical risk L_{sup} computed as the average loss over the training datasets:

$$L_{\text{sup}} = \frac{1}{N}\sum_{i=1}^{N}\ell\left(\boldsymbol{Y}^{(i)},\hat{\boldsymbol{Y}}^{(i)}\right) \tag{2.19}$$

The following loss function are commonly used to train the DNNs:

- **Euclidean distance:** this loss is commonly used for regression rather than classification. In this case, the empirical risk is also called as Mean Squared Error (MSE).

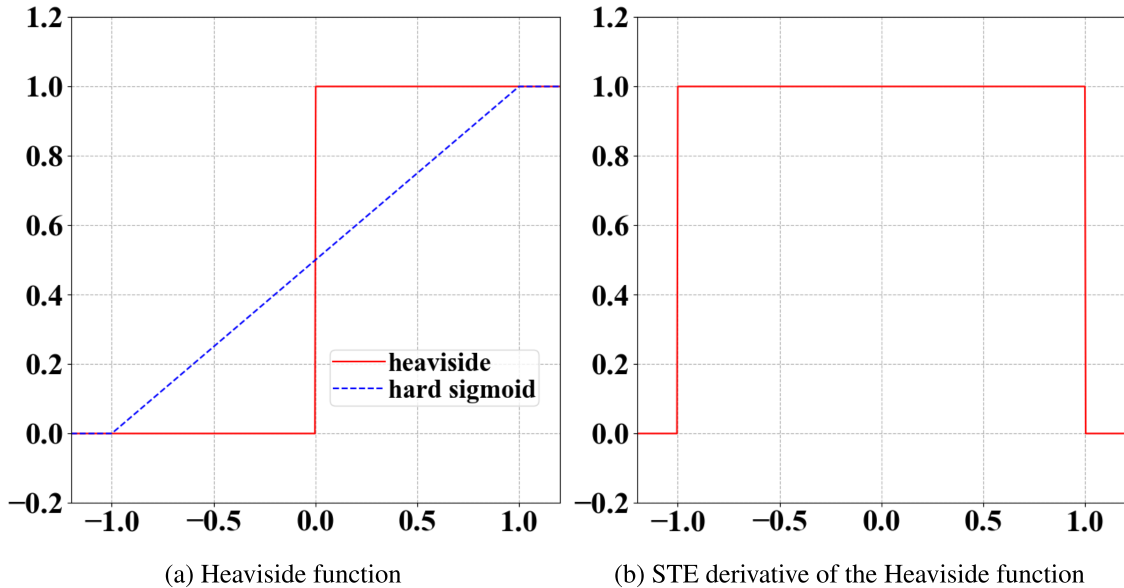
$$\ell\left(\boldsymbol{Y}^{(i)},\hat{\boldsymbol{Y}}^{(i)}\right) = \sum_{j=1}^{n_L}(Y_j^{(i)} - \hat{Y}_j^{(i)})^2 \tag{2.20}$$

- **Cross-entropy:** this is one of the most common choice for classification problems. This loss requires that the output of the model is normalized in the range of $[0,1]$ to represent the predicted probability, by setting the output activation function as the softmax in the case of multi-class classification or the sigmoid in the case of binary classification.

$$\ell\left(\boldsymbol{Y}^{(i)},\hat{\boldsymbol{Y}}^{(i)}\right) = -\sum_{j=1}^{n_L}\left(\hat{Y}_j^{(i)}log(Y_j^{(i)}) + (1-\hat{Y}_j^{(i)})log(1-Y_j^{(i)})\right) \tag{2.21}$$

- **Squared hinge loss:** this loss is used for maximum-margin classification (*e.g.*, for support vector machines [61]). Unlike the cross-entropy loss, the squared hinge loss does not require that the final output is normalized in $[0,1]$. Denote y_i as the class label of the i-th training example. The squared hinge loss imposes a safety margin (typically one) between the score of the correct category and the incorrect ones as follows:

$$\ell\left(\boldsymbol{Y}^{(i)},\hat{\boldsymbol{Y}}^{(i)}\right) = \sum_{j\neq y_i}\left[\max\left(0, Y_j^{(i)} - \hat{Y}_{y_i}^{(i)} + 1\right)\right]^2 \tag{2.22}$$



(a) Heaviside function

(b) STE derivative of the Heaviside function

Figure 2.4: Heaviside function and its approximated derivative using STE technique. a) the hard sigmoid is considered as a continuous approximation of the Heaviside function. b) STE derivative of the Heaviside function allowing the gradient to pass through within the clipping range $[-1,1]$.

Note that, minimizing the empirical risk is not the only goal of the learning procedure. As previously mentioned, the generalizability is also an important aspect when training the model. Typically, an over-parameterized model may rapidly achieve a very good performance on training set, while predicting poorly on unseen data. It is thus crucial to apply regularization techniques to force the model to learn only the most representative features, or to reduce the generalization gap. In both cases, this can be done through the use of data augmentation [62], or dropout techniques [63], [64]. Another choice is to add a penalty term P to the objective function that we aim at minimizing, where P may be a function of the weights or activations. Generally, the objective function can be expressed as follows

$$L = L_{\text{sup}} + \lambda P \tag{2.23}$$

where λ is the hyper-parameter defining the relative contribution of the regularization term.

2.1.4 Learning procedure

Most of the learnable parameters of the network, except for scales and offsets in the Batch Normalization, are randomly initialized before being adjusted during the learning process, based on the minimization of the lost function on the training set. For this purpose, we make use of *gradient descent*, a first-order iterative algorithm for finding the local minimum of a differentiable function, by moving in the opposite direction of the gradient (exact or approximated). Since the training set usually contains a large number of examples, the computation of the cost function and its partial gradients over all examples is very expensive. Such a complete pass of the whole training set through the gradient descent algorithm is called an **epoch**. In practice, at every epoch, we randomly divide the training set of N examples into several non-overlapping mini-batches of size M, then successively taking the loss and gradients over a single mini-batch at every iteration (or step). This ways, the parameter M controls the compromise between the accuracy of the computed gradient (hence the stability of the training) and the generalization of the network (due to the stochasticity added to the learning process).

A typical gradient descent-based NN training process includes 3 stages: forward propagation,

error back-propagation and weight update [65]. In the forward propagation, we takes a mini-batch of inputs, compute the network's output for every samples, and obtain the mini-batch empirical risk based on those outputs and the given labels. In the error back-propagation, we compute the partial gradients of the loss function with respect to the weights using the chain rule [66]. Finally, once all partial gradients are estimated, the weights are updated using information of the gradient computed at the current iteration or accumulated so far through the learning.

Forward propagation

Considering a mini-batches of size M including the set of input data $\{\boldsymbol{X}_0^{(i)}\}$ and the corresponding target set $\{\hat{\boldsymbol{Y}}^{(i)}\}$, the forward propagation processes the input data through all layers of the neural networks to obtain the set of output $\{\boldsymbol{Y}^{(i)}\}$. The loss function L computes the average error on these M examples as follows:

$$L = \frac{1}{M}\sum_{i=1}^{M} \ell\left(\boldsymbol{Y}^{(i)}, \hat{\boldsymbol{Y}}^{(i)}\right) + \lambda P \tag{2.24}$$

Figure 2.5a depicts the computational graph of a simple neural network. Note that it is necessary to store intermediate values of the networks, (*e.g.* $\boldsymbol{Z}_1, \boldsymbol{X}_1, \boldsymbol{Z}_2$ in Fig. 2.5a) for later use in gradient computation.

Error back-propagation

Based on the loss function which measures the difference between the predicted outputs and the targets, the learning process computes the partial gradients of the loss w.r.t the weights $\boldsymbol{W}_l$ of layer l, denoted as $\frac{\partial L}{\partial \boldsymbol{W}_l}$. The term *back-propagation* or *backward pass* comes from the fact that these gradients are derived using the chain rule of calculus, *i.e.*, passing values backwards through the network to estimate how the loss is affected by each weight. Figure 2.5b illustrates the backward pass of the network. From the Equation 2.1 that describes relationship between the input $\boldsymbol{X}_l$ and output $\boldsymbol{X}_{l+1}$ of a layer, we can derive the gradients w.r.t the kernel $\boldsymbol{W}_l$, the bias $\boldsymbol{b}_l$ and the input $\boldsymbol{X}_l$ of that layer:

$$\frac{\partial L}{\partial \boldsymbol{W}_l} = \frac{\partial L}{\partial \boldsymbol{X}_{l+1}} \frac{\partial \boldsymbol{X}_{l+1}}{\partial \boldsymbol{W}_l} = \frac{\partial L}{\partial \boldsymbol{X}_{l+1}} \odot \frac{\partial f(\boldsymbol{Z}_{l+1})}{\partial \boldsymbol{Z}_{l+1}} \frac{\partial \boldsymbol{Z}_{l+1}}{\partial \boldsymbol{W}_l} \tag{2.25}$$

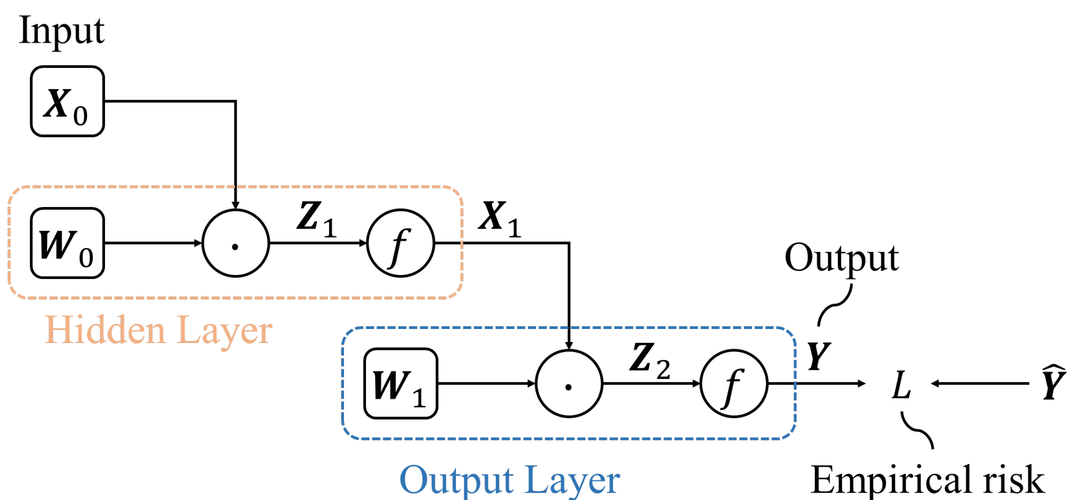
$$\frac{\partial L}{\partial \boldsymbol{b}_l} = \frac{\partial L}{\partial \boldsymbol{X}_{l+1}} \frac{\partial \boldsymbol{X}_{l+1}}{\partial \boldsymbol{b}_l} = \frac{\partial L}{\partial \boldsymbol{X}_{l+1}} \odot \frac{\partial f(\boldsymbol{Z}_{l+1})}{\partial \boldsymbol{Z}_{l+1}} \tag{2.26}$$

$$\frac{\partial L}{\partial \boldsymbol{X}_l} = \frac{\partial L}{\partial \boldsymbol{X}_{l+1}} \frac{\partial \boldsymbol{X}_{l+1}}{\partial \boldsymbol{X}_l} = \frac{\partial L}{\partial \boldsymbol{X}_{l+1}} \odot \frac{\partial f(\boldsymbol{Z}_{l+1})}{\partial \boldsymbol{Z}_{l+1}} \frac{\partial \boldsymbol{Z}_{l+1}}{\partial \boldsymbol{X}_l} \tag{2.27}$$

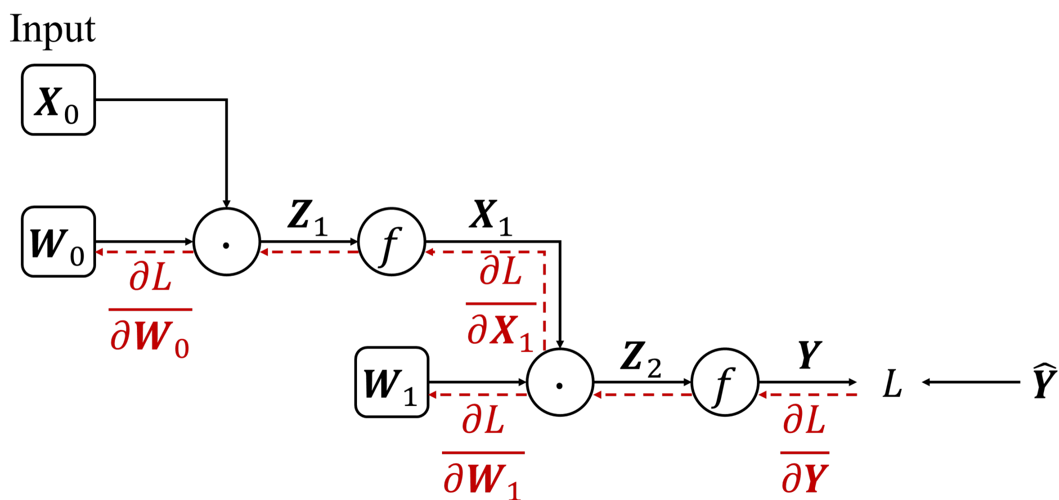
where $\odot$ denotes the element-wise multiplication since the nonlinear activation function f is applied element-wise. Given the gradient w.r.t the output $\frac{\partial L}{\partial \boldsymbol{X}_{l+1}}$, we can compute the gradient w.r.t the learnable parameters X_l, B_l and the input X_l of that layer. This way, the gradient can be backwardly propagated from the output layer to the first layer, allowing us to compute the gradients w.r.t all parameters of the network. The term $\frac{\partial f(\boldsymbol{Z}_{l+1})}{\partial \boldsymbol{Z}_{l+1}}$ justifies the need of storing the values of the pre-activations (*e.g.*, $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ in Fig. 2.5b) while $\frac{\partial f(\boldsymbol{Z}_{l+1})}{\partial \boldsymbol{Z}_{l+1}}$ and $\frac{\partial \boldsymbol{Z}_{l+1}}{\partial \boldsymbol{X}_l}$ require the storage of the hidden units $\boldsymbol{X}_l$. This also implies that the training has increased the storage requirements compared to the inference which consists only the DNN processing (forward pass) to obtain the output.

Weight update and different gradient descent variants

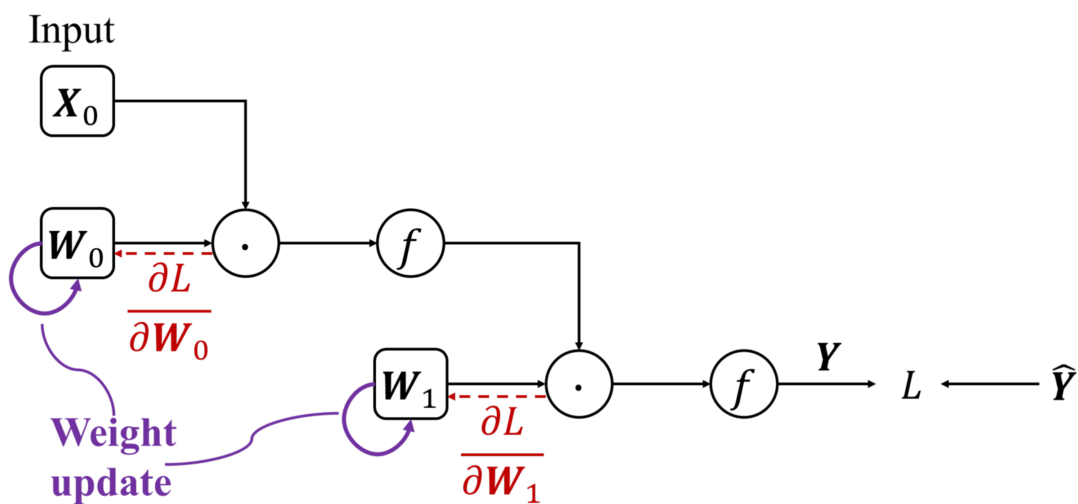
As mentioned before, the gradients w.r.t the weights are used to determine the minimization direction of the loss function. Different gradient descent-based algorithm (also called **optimizers**) share the same principle in forward and backward pass, and only differs from each other in the



(a) Forward propagation.



(b) Backward propagation.



(c) Weight update.

Figure 2.5: Three stages of a typical gradient descent iteration.

weight update strategy. In general, there are two kinds of optimizers based on the way the gradient is used: non-accumulated gradient-based optimizer and accumulated gradient-based optimizer.

- **Non-accumulated gradient-based optimizer** This is the simplest kind of optimizer, where the descent direction is determined by only using the gradient at the current iteration. Probably the most popular algorithm is Stochastic Gradient Descent (SGD [67]). Denote $W_l^{[t]}$ and $\left(\frac{\partial L}{\partial W_l}\right)^{[t]}$ as the weight of the layer indexed l and its gradient at the iteration t. Let us denote also the small constant η as the **learning rate** which defines how much the weight will be moved along the gradient-descent direction. The weight update of the SGD can be described as follows
$$W_l^{[t+1]} = W_l^t - \eta \left(\frac{\partial L}{\partial W_l}\right)^{[t]} \tag{2.28}$$

- **Accumulated gradient-based optimizers** Learning with SGD may be slow due to the high variance of the gradient computed over a small number of examples. Moreover, the SGD treats all parameters (or directions) in the parameter space equally with the same learning rate, while in practice, the loss function is often more sensitive to some directions and less sensitive to others. Therefore, different variants have been proposed to accelerate the learning, reduce the unstable effect due to the mini-batches and even offer a flexible learning rate to each direction/parameter based on the recorded history of the gradient. Let us denote also the variable $\Delta W_l^{[t]}$ as a moving record of the gradient w.r.t the weight W_l at the iteration t, and κ is its updating function. The weight update process of an accumulated gradient-based optimizer can be generally described as follows
$$\Delta W_l^{[t]} = \kappa\left(\Delta W_l^{[t-1]}, \left(\frac{\partial L}{\partial W_l}\right)^{[t]}\right) \tag{2.29}$$
$$W_l^{[t+1]} = W_l^t - \Delta W_l^{[t]} \tag{2.30}$$
Well-known accumulated gradient-based optimizers include Nesterov momentum [68], AdaGrad [69], RMSProp [70] and Adam [71]. While the Nesterov momentum optimizer only makes use of the exponentially decaying moving average as the only recorded history of the gradient, the others additionally introduce a second-order accumulated squared gradient to properly adjust the learning rate regarding direction in the parameter space.

2.1.5 Datasets

In this section, we present the datasets used to test the performance of our model compression techniques as well as the proposed model architectures. In details, we make use of MNIST [72], CIFAR-10/100 [73], STL-10 [74] for image classification; DIV2K [75] for image compression; and Jester Dataset V1 [76] for hand gesture recognition.

MNIST dataset is a large collection of handwritten digits with 60000 training examples and 10000 test examples. The images are grayscale and have a fixed size of 28×28. There are 10 classes, corresponding to digits from 0 to 9. This dataset is relatively simple and small with low dimensionality and less variations to learn, making it easy to train the model up to the convergence. Moreover, it does not require a complex network architecture to obtain a high accuracy. Therefore, this dataset is more appropriate for prototyping a new idea to see if it works, rather than verifying its effectiveness or comparing it with state-of-the-art methods.

CIFAR-10 and CIFAR-100 are datasets containing tiny RGB images of size 32×32. Each dataset consists of 60000 images in total. CIFAR-10 contains 10 classes, each class has 5000 training images and 1000 test images. CIFAR-100 contains 100 classes, each class contains 500

images for training and 100 images for testing. In particular, the 100 classes in the CIFAR-100 are grouped into 20 superclasses. Although having properties of a toy dataset (in the sense of small-sized images and small number of examples), these datasets still offer space for testing novel network architectures and competition between different models, with state-of-the-art performance of 99.01% accuracy on CIFAR-10 [77] and 89.46% [78] on CIFAR-100.

STL-10 is an image dataset derived from ImageNet, containing 100000 unlabeled images along with 13000 labeled images from 10 object classes. These annotated images are partitioned into 5000 images for training and 8000 images for testing. Each image is an RGB image with the resolution of 96×96. Although more popularly used for unsupervised feature learning and self-taught learning, STL-10 can still be considered as one competitive image classification dataset, with the state-of-the-art accuracy of 95.48% [79].

DIV2K is a well-known image dataset firstly introduced for the image super-resolution challenge NTIRE2017 [80]. It contains 1000 images with diverse scenes, contents and resolution. The DIV2K dataset is divided into 800 for training, 100 for validation and 100 for testing. The images are of high quality with low noise and have 2K pixels on at least one of the two axes. Since the 100 testing images are not publicly available, we only make use of 900 images for evaluating the performance of models in Chapter 4. The images are then cropped and resized to meet the target resolution in our image compression task.

The Jester Dataset V1 is a video classification dataset including 148092 labeled video clips of humans performing basic, pre-defined hand gestures in front of a laptop camera or webcam. The clips belong to 27 classes of human hand gestures, and splitted into training, validation and test set with a ratio of $8:1:1$. The video clips have diverse lengths and spatial resolutions. Since the labels of the test set are not publicly available, we evaluate our model in Chapter 6 using the validation examples. This dataset is the current largest-scale benchmark for hand gesture recognition, allowing to build models learning spatio-temporal features (*e.g.*, using Conv3D and recurrent layers like LSTM).

2.1.6 Overfitting and training tricks

In practice, a dataset is divided into three parts: training set, validation set, and test set. The training set is used to train the model while the validation set is used to monitor the generalizability of the model on the unseen data during the training process. Unlike these two parts, the test set is usually unknown and then annotated using the trained model. Note that in some cases the dataset only contains two parts (*e.g*. CIFAR-10/100), the test set and the validation set are the same. The model's generalizability is usually measured by the difference between the model's performance on the training and the validation data, and this is popularly termed as the **generalization gap**. If the training set and the validation are both drawn from the same distribution, it is expected that a well-trained model achieve a high performance on the training set while the generalization gap is small. If the model performs much better on the training set than on the validation set, it is called **overfitted**. This overfitting phenomenon can be explained by an over-parameterized design causing the model to be biased by small variations of the training set that are not present in the validation set.

While the most direct approach is design a more compact neural network, there exists several training tricks to alleviate the overfitting issue. One common approach is applying some regularization techniques (*e.g.*, ℓ_1, ℓ_2 penalty) to the model's weights/activations during the training. Another model-related approach is Dropout [63], which randomly shuts down a portion of neurons at every learning iteration. Data augmentation is also a relevant approach to reduce the general-

ization gap, which increases the diversity of the training set and reinforces the learning of only relevant features rather than the biased ones. Typically, in the case of image classification, this can be done by adding synthetically modified data obtained through random translation, zooming, rotation or noise addition.

2.1.7 Neural Network Architectures

In this part, we give an overview of different types of neural networks, from Multi-Layer Perceptron (MLP) to Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNN).

Multi-Layer Perceptron (MLP)

A MLP contains only fully connected layers, each followed by a non-linear activation function. The major computation in MLP is the matrix-vector multiplication, in which the matrix is dense since every output unit is connected to all input units. Therefore, when working with high-dimensional data, the MLP will contains a large number of parameters. This model architecture is more relevant for data without grid-like topology or local correlation. Therefore, in modern DNNs, an MLP is mainly used at the end of the feature extractor for the sake of combining high-level relevant features. Some models even alleviate the parameter-heaviness of MLP by leveraging the parameter-free pooling layers.

Convolutional Neural Networks (CNNs)

As a simplest definition, a CNN is a neural network that contains at least one convolutional layer. As mentioned in the section 2.1.1, the convolution is a favorable operation for processing data having local-correlation nature like time series, images or video. Concatenating several convolutional layers enables the neural network to learn more complex features from the local structure of data. Therefore, in CNN, the convolutional layers are usually plugged in the first part of the model to form a so-called feature extractor. During the last few years, CNNs have been remarkably successful in various practical applications. One of the main reason for this success is the tremendous progress of CNN architectures. Since most of the advances in model architectures are originally focused on image-related benchmark before being applied to other data (*e.g.*, time series or video), it is reasonable to see the development path of convolutional networks through the lens of 2D CNNs.

AlexNet [1] (2012) is well-known because of drawing first attentions to the use of CNN-based solution for the ImageNet challenge. In details, AlexNet contains 5 Conv2D layers and 3 fully-connected (dense) layers. Some of the Conv2D layers are followed by a max pooling. The first two Conv2D layers have large-sized kernels of respectively 11×11 and 5×5, while the three latter make use of only 3×3 kernels. It employs the ReLU activation function in place of the traditional choices sigmoid and tanh. AlexNet consists of 61M parameters and requires 724M MACs for each inference.

VGGNet [81] (2014) attempt to improve the performance of AlexNet by increasing the number of convolutional layers (hence the depth) of the model. Instead of using large kernel sizes like AlexNet, [81] suggests that a stack of several 3×3 convolutional layers can enlarge the receptive field as well, while incorporating more nonlinearity into the model and decreasing the number of parameters and MACs. The popular version with 16 layers, *i.e*. VGG16 (cf. Fig. 2.7), consisting of 13 convolutional layers and 3 dense layers, has 138M learnable parameters and requires 15.5G MACs for each inference.

The trend of making the model deeper has given rise to a question regarding its impact on the learning. If a model has a suitable depth, adding more layers to it may leads to higher train-

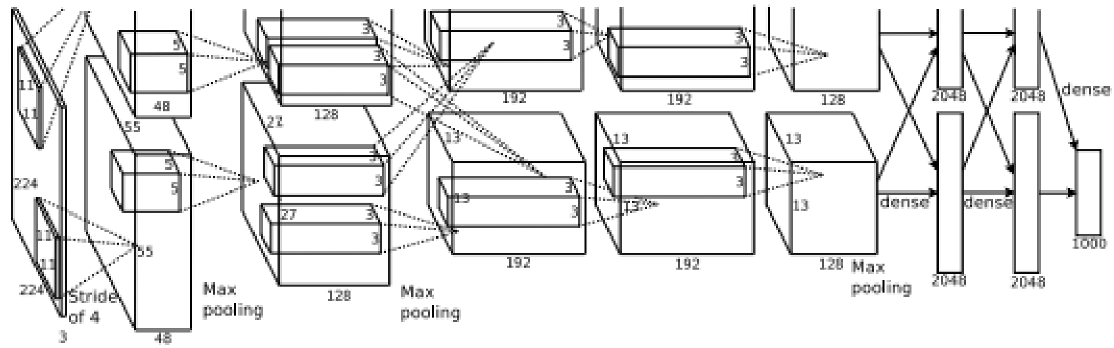


Figure 2.6: AlexNet architecture. Note that this figure is originally introduced in [1].

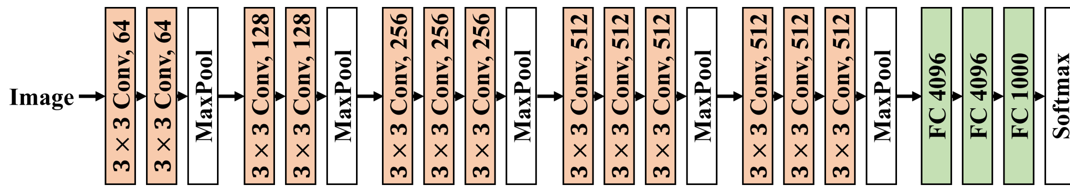


Figure 2.7: VGG16 [81] architecture.

ing error rather than overfitting, due to the gradient vanishing/exloding problem. To address this issue, [24] introduces ResNet, a network architecture with the identity mapping or *residual/skip connection* (Figure 2.8). Apart from ensuring the gradient flow, the skip connections also increase the representation power of the model by favoring the feature reuse. ResNet-50 contains 49 convolutional layers and 1 fully-connected layer but has only 25.5M learnable parameters and requires 3.9G MACs per inference. Compared to VGG16, ResNet-50 is smaller-sized and less computationally expensive while offering higher algorithmic performance (*i.e.* 95.3% vs 92.64% accuracy on CIFAR-10). This has motivated the design of later efficient CNN architectures by leveraging more branches and skip connections to increase the representation power of the model. However, a drawback of ResNet and its variants is that during inference, it requires the simultaneous storage of several intermediate feature maps, hence increasing the cache memory needs with respect to its corresponding plain model counterpart.

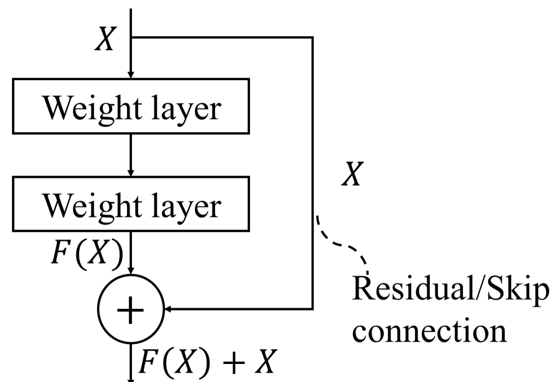


Figure 2.8: A general residual block.

ResNext [28] additionally introduces another dimension into the CNN configuration, namely *cardinality* - the size of the set of transformations inside each residual block. In details, they propose a building block consisting of two 1×1 convolutions and a grouped 3×3 convolution inserted in-between (Figure 2.9). Here the cardinality is defined through the number of groups.

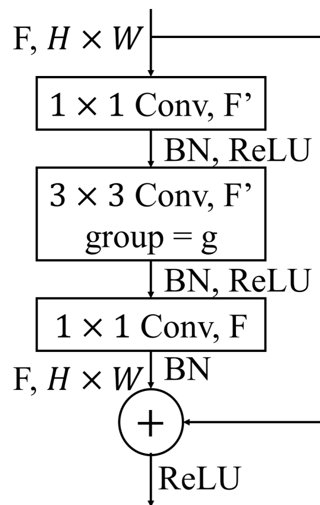


Figure 2.9: Building block of ResNext [82].

SqueezeNet [82] is one of the first work focusing on the design of an efficient CNN architecture with fewer parameters but on-par accuracy compared to existing models. The building block of SqueezeNet is called "Fire module" (Figure 2.10), which consists of a 1×1 convolution for squeezing the channel dimension, and an expansion stage with 1×1 and 3×3 convolutions concatenated together. SqueezeNet has shown promising results, achieving an equivalent accuracy to AlexNet with 50× fewer parameters.

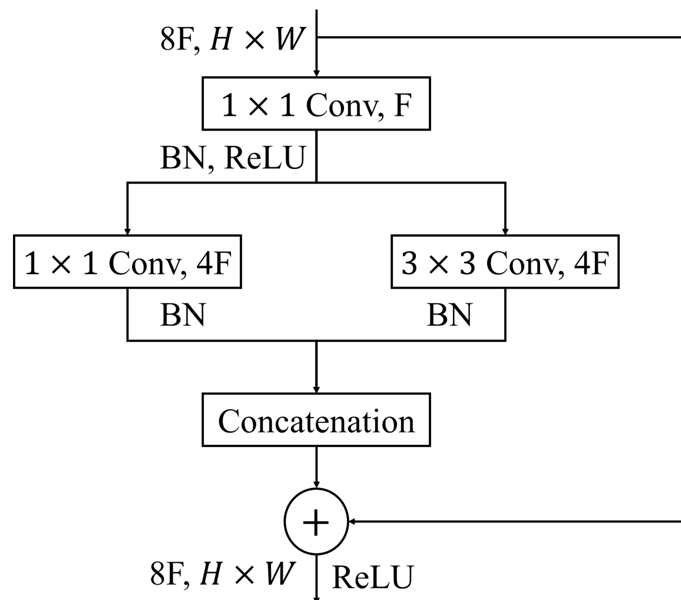


Figure 2.10: Fire module in the SqueezeNet [82].

[83] originally introduces the notion of attention residual learning for image classification task that mainly relies on an Attention Module demonstrated in Figure 2.11. The key element of this module relies on the soft mask branch $M(\boldsymbol{X})$ with output values range from $[0,1]$ which works as a feature selector. The transformation branch $T(\boldsymbol{X})$ is simply a cascade of convolutional layers. [84] proposes a channel-wise feature attention mechanism through the so-called Squeeze-and-Excitation (SE) block (Figure 2.12). In details, the channel-wise attention branch of SE block consists of a squeeze step which collects global information of each channel separately, and an

excitation step capturing channel-wise dependencies to re-calibrate the features accordingly.

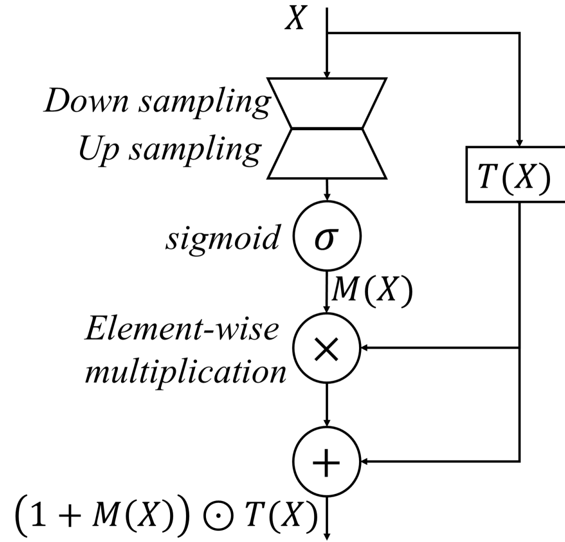


Figure 2.11: Residual Attention block [83].

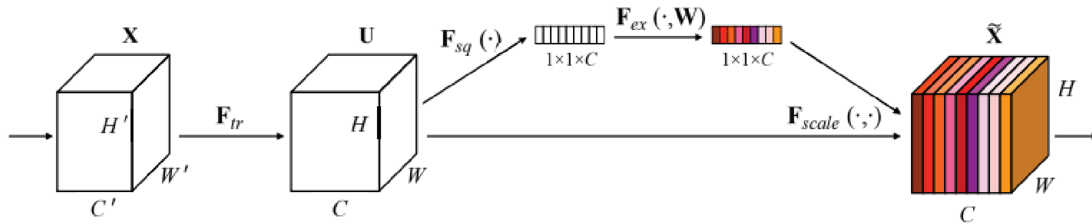


Figure 2.12: Squeeze-and-Excitation (SE) block [84]. Note that this figure is originally introduced in [84].

It is noteworthy mentioning that all of these aforementioned network architectures are manually designed and often requires significant efforts along with expert knowledge. The development path of CNN architectures also gives a hint about the potential design scheme for building blocks of CNNs: embrace several branches and connections between different operations. However, such complex model architectures rely on a vast design space while human knowledge is limited. This has motivated a novel paradigm for automatically designing DNN architectures, namely Neural Architecture Search (NAS [10]). In a NAS-based framework, each component of the model is chosen under the global perspective of the network, *e.g.*, building an optimal DNN architecture with highest performance within a given computational budget. More details about NAS is discussed in section 2.3.1.

Recurrent neural networks (RNNs)

Recurrent neural networks is a class of neural networks to process sequential data, allowing us to exhibit long-term dependencies. These models are commonly used for various tasks, including speech/video recognition, natural language processing or image captioning. The term **recurrent** comes from the fact that the output of this layer is computed by taking into account the prior elements within the sequence. Just as convolutional layers share the same parameters across spatial position, recurrent layers share the same weights across different time steps, forcing the learned

features to be independent to the temporal position.

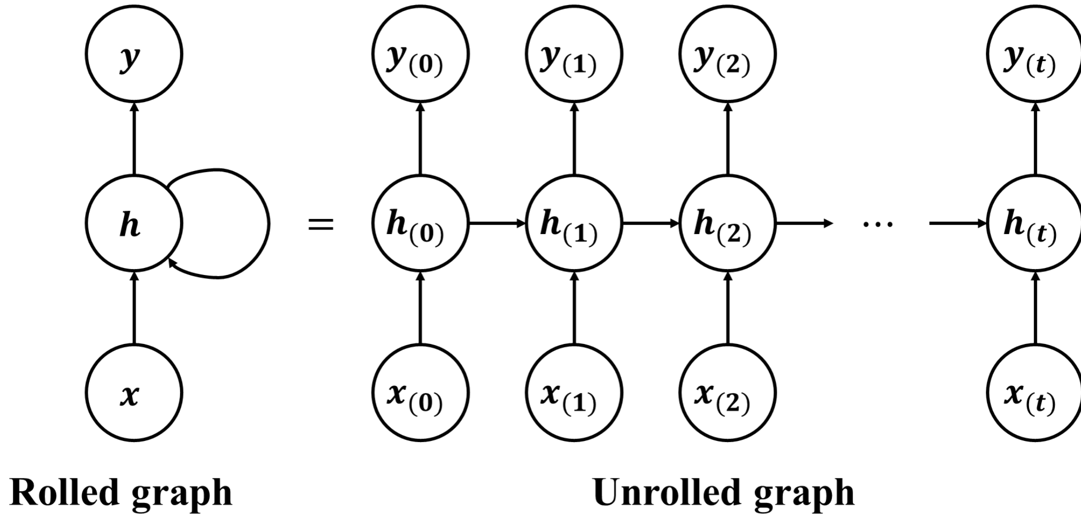


Figure 2.13: An example of recurrent neural network. This model receives the sequential input data $\boldsymbol{x}$ and combines it with the hidden state $\boldsymbol{h}$ which is passed through the time, to obtain the output $\boldsymbol{y}$. *Left*: the folded graph represents the whole neural network. *Right*: the unrolled graph depicts the whole computation graph, allowing us to see the information flow in the forward pass and also the gradient propagation in backward pass.

Figure 2.13 depicts a recurrent layer which processes sequential inputs $\boldsymbol{x}_0, \boldsymbol{x}_1 ... \boldsymbol{x}_t$ and outputs $\boldsymbol{y}_0, \boldsymbol{y}_1 ... \boldsymbol{y}_t$. This recurrent network can be represented by recurrent connection from the hidden state $\boldsymbol{h}$ of the previous time step, or by an unrolled computation graph, where each node is related to one time instance. The simplest form of RNN is firstly proposed in [85] to handle sequences of vector as follows:

$$\boldsymbol{h}_t = \sigma_h \left(\boldsymbol{W}_h [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h \right) \tag{2.31}$$

$$\boldsymbol{y}_t = \sigma_y \left(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y \right) \tag{2.32}$$

where $[.,.]$ is the vector concatenation, $\boldsymbol{W}_h, \boldsymbol{W}_y, \boldsymbol{U}_h$ are the weight matrices, $\boldsymbol{b}_h, \boldsymbol{b}_y$ are the biases and σ_h, σ_y are activation functions. The particular problem of recurrent networks is the vanishing/exploding gradient w.r.t the weight matrices due to their sharing across the temporal dimension. One of the most relevant approaches to address this issue is to propose more complex RNN architecture with internal gates to adaptively gather information over a long duration. One of the most popular gated RNNs is the Long Short-Term Memory (LSTM [86]) depicted in Figure 2.14 which can be described as follows:

$$\boldsymbol{i}_t = \text{sigmoid} \left(\boldsymbol{W}^i . [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}^i \right) \tag{2.33}$$

$$\boldsymbol{f}_t = \text{sigmoid} \left(\boldsymbol{W}^f . [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}^f \right) \tag{2.34}$$

$$\boldsymbol{o}_t = \text{sigmoid} \left(\boldsymbol{W}^o . [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}^o \right) \tag{2.35}$$

$$\tilde{\boldsymbol{c}}_t = \tanh \left(\boldsymbol{W}^c . [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}^c \right) \tag{2.36}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{2.37}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{2.38}$$

where $\boldsymbol{i}_t, \boldsymbol{o}_t, \boldsymbol{f}_t, \boldsymbol{c}_t$ are the input, gate, forget gates and the cell memory, respectively. The cell memory state $\boldsymbol{c}_t$ has a linear self-connection controlled by the forget gate $\boldsymbol{f}_t$. This mechanism

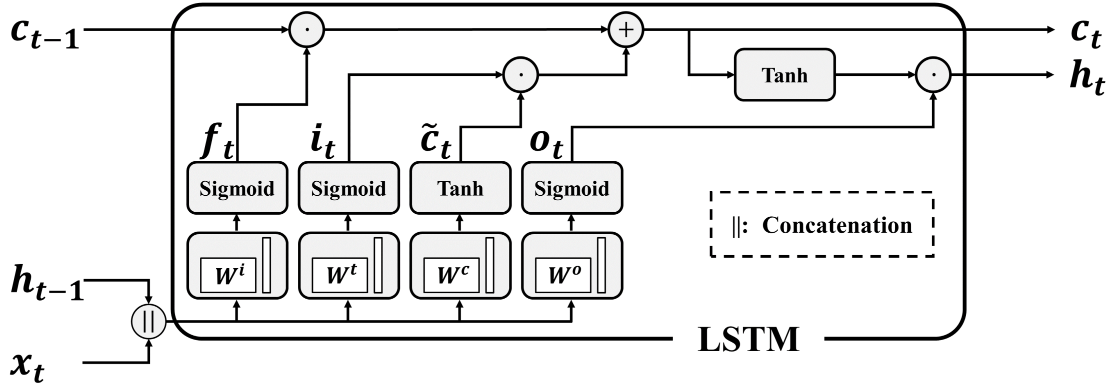


Figure 2.14: Computational graph of the LSTM cell.

enable the model to learn how long to remember the information from the past and when to clear these past states. Assuming that $\boldsymbol{x}_t \in \mathbb{R}^{n_l}$ and $\boldsymbol{h}_t \in \mathbb{R}^{n_{l+1}}$, then the LSTM consists of four weight matrices of size $n_{l+1} \times (n_l + n_{l+1})$. In the case where both input and output sequences are high-dimensional, this will result in a memory bottleneck. To reduce the complexity and the cost of LSTM, other lighter variants have been proposed, including the Gated Recurrent Unit (GRU) or the recent STAckble Recurrent (STAR) cell [87]. Other works overcome this memory burden by exploiting the sparsity [88], [89], or decomposing the weight tensors [90] of the LSTM.

It is noteworthy mentioning that all RNNs mentioned above is dedicated to the processing of 1D sequences by a fully-connected manner. This restricts their utilization in DNNs which aim at extracting spatio-temporal features (*i.e.* in the case of video processing). To handle this type of data, [91] proposed ConvLSTM which integrates the convolutions into the input-to-state and state-to-state transitions. Although ConvLSTM can capture the long-term spatio-temporal dependencies, this ability comes with a significantly increased computational complexity and memory burdens. Therefore, ConvLSTM is not a preferable choice for video processing compared to the standard yet effective 3D CNN-LSTM architectures. Besides, similar to the case of CNNs, recent works has also been leveraged the NAS framework in order to search for novel RNN architectures [92], [93].

2.1.8 Deep learning frameworks

Deep Learning frameworks provide abstractions of all neural network's elements such as layers, activation functions and basic computational operations along with tools for data pre-processing and the learning procedure, therefore practitioners only need to write the code describing the network architectures and the ML workflow.

Throughout this thesis, we utilize the TensorFlow [94] framework as the baseline development tool for building models, preparing data, optimizing the networks and sometimes customizing an element in the ML workflow. However, since the model compression techniques like quantization generally require additional operations inside the layers, we need another framework with complementary abstraction for developing such algorithms or customizing existing operations/layers. To this end, we make use of the Larq framework [95] which is an open-source Python library built upon Tensorflow for training low-precision DNNs. Compared to other basic Deep Learning frameworks, Larq provides a key abstraction called *quantizer* which defines the way how real-valued input is transformed into discrete output. It also allows the definition of the STE gradient (see section 2.1.2) for the backward propagation. They also provides customized quantized layers integrating the use of those quantizers for obtaining the low-precision input and the weight as well (Figure 2.15).

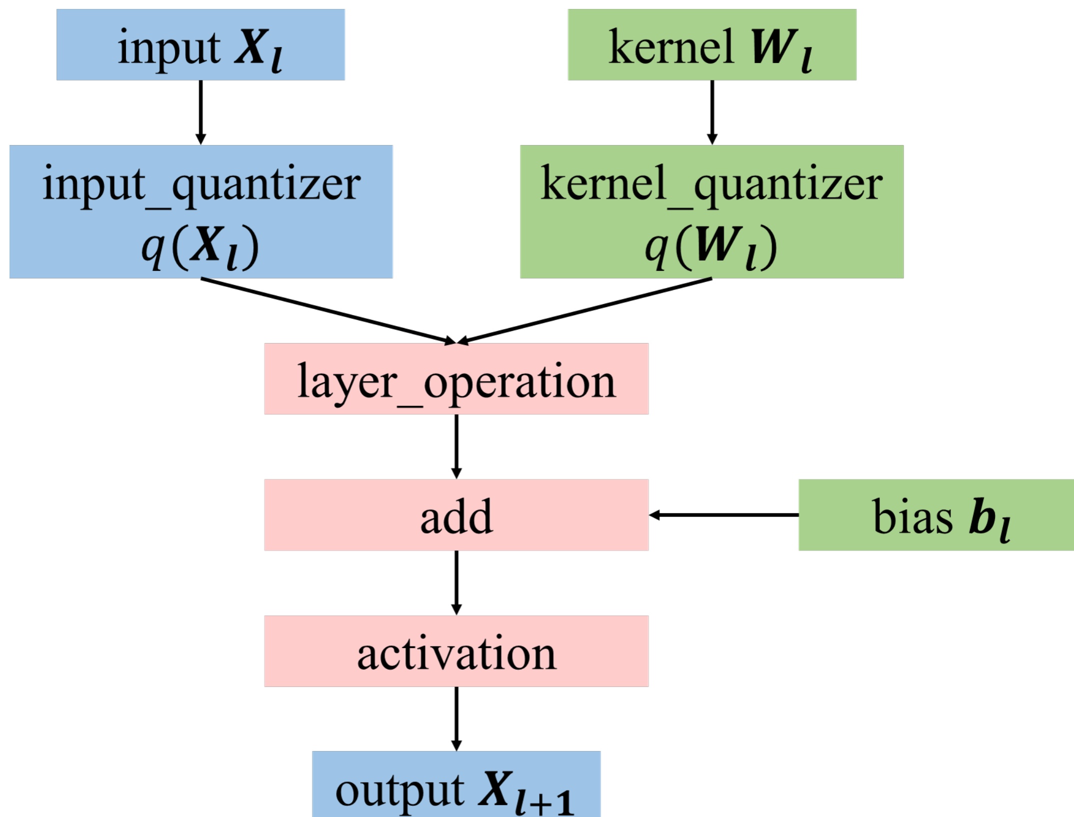


Figure 2.15: Computation graph of a typical quantized layer in Larq. In details, the quantization is done through the *input_quantizer* and *kernel_quantizer* arguments.

2.2 DNN accelerators

The increasing number of large-scale datasets and tremendous advances of computing hardware has empowered the success of AI and especially Deep Learning (DL). Reciprocally, the exceptional development of DL with complex DNNs requires specialized hardware to support and accelerate these increasingly complex algorithms during both training and inference time. We can categorize AI accelerators into 3 classes based on their application-versatility: general-purpose accelerators, AI-dedicated programmable accelerators and application-specific accelerators.

2.2.1 General-Purpose Accelerators (GPA)

A general-purpose accelerator, as it is termed, can serve for different type of computations and tasks. As the most versatile computing core with widespread availability and recently increasing performance, the Central Processing Units (CPUs) are also employed for AI-based processing [96], [97]. Modern CPUs are equipped with multi-core parallelism [98] and SIMD (single instruction, multiple data) technique, which allows to perform the MAC operations that dominate the processing of DNNs. Current accelerations over CPU platforms can be done by optimizing graph-based frameworks [99], [100], [97] which map DNN computations onto CPU execution, or by reducing cache misses and re-arrange data throughput in an architecture-friendly manner [101], [102].

Graphical Processing Units (GPUs), thank to their great capability to handle parallel computing, have been one of the major factors enabling the development of DL. While desktop and

server GPUs are mainly used for training without power constraints, mobile and edge GPUs are mostly used for inference for the reason of latency or privacy. Nowadays, several smartphones are equipped with GPU-based neural engines for smart applications, for example iPhone 13 with A15 Bionic engine [103], or Samsung with Exynos [104]. Besides, embedded GPUs like NVIDIA Jetson Nano [105] also target low-power inference application at the edge.

While GPU still remains the first choice for DNN processing, FPGA-based accelerator has recently drawn much research attention. Compared to CPUs and GPUs, FPGAs are more flexible in terms of architecture, and can be customized to perform any desired operations and meet resource constraints as well. In order to improve the performance and the energy efficiency of FPGA-based accelerators, current techniques may focus on the computation unit level, loop unrolling level and system level. At the computation unit level, the designs aim at increasing the peak performance, by simplifying the calculation with low-precision arithmetic [106], [107], using fast convolution algorithm [108] or increasing the working frequency [109]. The loop unrolling designs target the high level of parallelism during processing [108], [110], or optimize the data transfer [111]. At the system level, designs may fuse layers together to avoid the intermediate data transfer and alleviate the costly access of external memory [112].

2.2.2 AI-Dedicated Accelerators (AIDA)

While the above hardware platforms that can support any kinds of computations, AI-dedicated accelerators (AIDA) [113], [108], [114], [115] partly sacrifice the computation-versatility in their design by a higher level of hardware customization to improve the processing efficiency of different AI algorithms, in particular DNNs. Early approaches [116], [6] consider memory hierarchy as one of the key points in the design of energy-efficient AI accelerators and propose dataflow schemes that increase the data reuse. These works has motivated the development of dataflow-aware AI chips in the industry [117], [118]. [119] presents an unified NN processor which can support different types of NN models and accelerate the MAC operation in frequency domain based on the block-circulant algorithm [120].

Emerging approaches improve the efficiency of accelerators by leveraging three main features: low-precision computation, sparsity, and computation-in-memory (CIM). [121] presents a DNN accelerator that can support variable weight precision from 1 to 16 bit. [122] then proposes a CNN-based AI processor with static random access (SRAM)-CIM unit macro with multi-precision weights/activations. [123] introduces an SRAM-based all-digital CIM macro to handle multi-bit MAC operations of different CNN architectures. Besides low-precision computation, exploiting sparsisty of DNN's weights and activations is another way to increase the efficiency of accelerators. [124] presents an energy-efficient neural processing unit (NPU) incorporating a feature map zero-skipping mechanism to improve computation efficiency. [125] combines zero skipping and weight compression to minimize memory access and computation of an application-versatile accelerator.

2.2.3 Application-Specific Accelerators (ASA)

If the targeted task is well defined, we can reduce furthermore the power consumption and computation cost of accelerators. Compared to those aforementioned AI-dedicated chips, these application-specific accelerators (ASA) even requires a higher level of customization in response to the specific usage and a careful design for every element of the DNN model leveraging hardware-algoritm co-optimization. Besides, the efficiency of these ASAs also comes at the cost of limited reconfigurability and programmability. For instance, [126] proposes an always-on low-power processor for both face detection and face recognition featuring shared layer and ternary quantization. [127] presents a 184µW real-time hand gesture recognition system with an edge-CNN core to extract hand feature features from the edge data. [128] proposes a voice activity detector featuring a

sparsity-aware time-domain CNN feature extractor combined with a binary neural network-based classifier. [7] proposes a 4.53µW accelerator enabling both speaker verification and keyword spotting sharing most of the network's binarized parameters. [27] proposes a multi-task CNN accelerator that can serve for both face detection and alignment.

The remarkable hardware/algorithm trade-offs of these above ASAs are crucially based on the proper definition of the use case scenario and hence the application complexity, allowing us to better estimate the computational complexity of the DNN algorithm. Through a fine-grained analysis of every element in the DNN model, we can leverage hardware-algorithmic enablers such as efficient DNN architectures and model compression techniques, to design an ASA that meet the target resource constraints. Nevertheless, it is noteworthy mentioning that ASAs are limited in terms of application and scalability, *i.e.*, only supporting a task or certain correlated tasks as previously demonstrated.

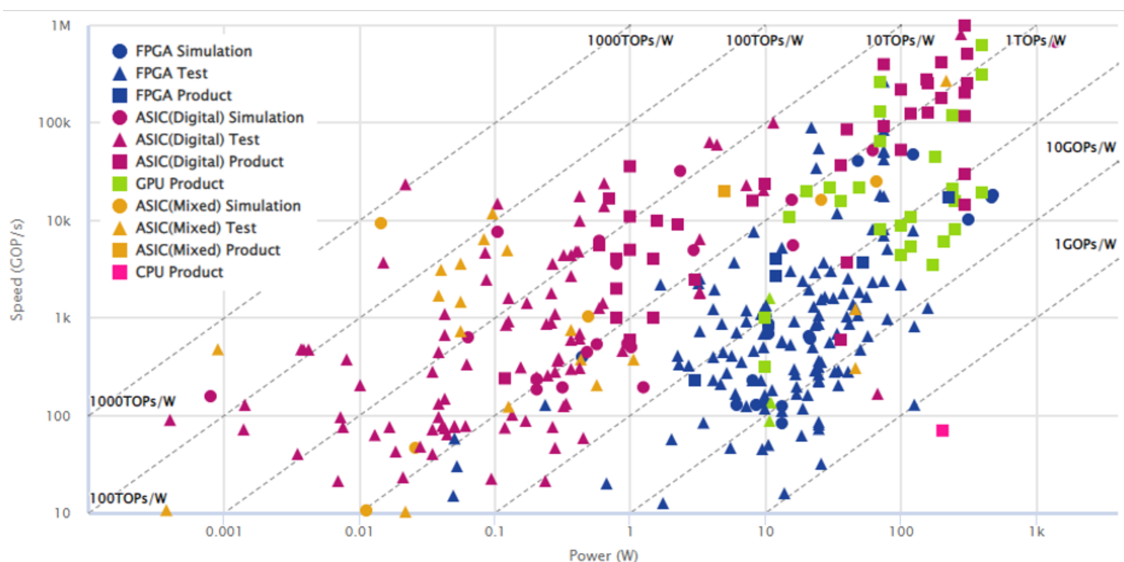


Figure 2.16: Power versus computational speed of different DNN accelerator platforms. Note that this figure is originally introduced in [129].

2.2.4 Discussion

The three classes of DNN accelerators offer different levels of computation throughput, power consumption, versatility and scalability. Figure 2.16 depicts the power versus computation speed of different accelerator platforms, including CPUs, GPUs, FPGAs and ASICs. Despite their ubiquitousness and maturity, CPUs and GPUs has major drawbacks in efficiency. Although FPGA-based processors require lower power consumption than CPUs and GPUs, they still has lower computational efficiency compared to ASIC designs. It is clearly demonstrated that GPAs stay in the less optimal zone with lower TOPs/W zone. On the other hands, ASIC designs -mostly relating to AIDAs- mainly stay close to higher TOPs/W lines.

Note that, apart from this dynamic power consumption which is proportional to the number of tera operations per second (#TOPs) by the factor $\frac{1}{F_{\text{TOPs/W}}}$, there exists a static power consumption P_{static} deeply depending to the choice of platform and its corresponding technology (*e.g.*, type of transistors, frequency ...). The total power consumption can be thus described as follows

$$P_{\text{total}} = P_{\text{static}} + \frac{\#\text{TOPs}}{F_{\text{TOPs/W}}} \tag{2.39}$$

Figure 2.17 depicts the total power consumption of these DNN accelerators as a function of the number of tera operations per second. Although GPUs and AIDAs has higher TOPs/W thank to

their optimized computational cell, their static power is several orders higher than that of ASAs. As a result, for compact models with small #TOPs, ASAs have lower power consumption than GPU and AIDA platforms. This makes ASAs favorable as a target inference platform in our works, with the objective being improve the application-versatility as well as the hardware-algorithmic trade-offs.

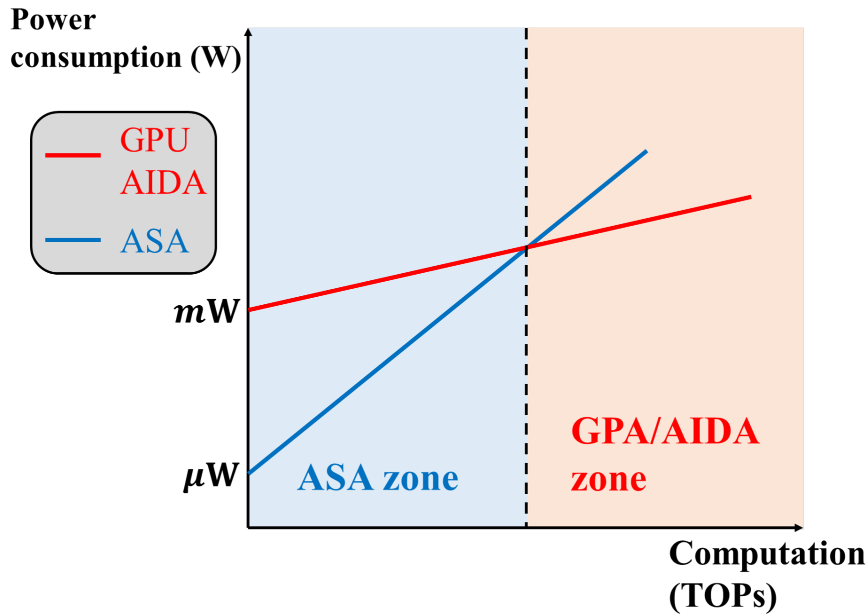


Figure 2.17: The total power consumption of different platforms as a function of #TOPs.

2.3 DNN hardware-algorithmic enablers state-of-the-art

As discussed in section 2.1.7, Deep Neural Networks, in particular Convolutional Neural Networks, contains a huge number of weights and requires intensive computations, thus hindering their deployment in resource-constraint edge devices. Therefore, it is necessary to design compact models -regarding the memory and computational requirement- without significantly degrading the inference performance of the model. This has motivated researches on efficient model design and network compression, with tremendous progress in the last few years.

2.3.1 Efficient architecture designs

The simplest way for matching CNNs with hardware-related constraints and accelerating the inference is to design light-weight architecture from the beginning, rather than compressing existing well-known model to obtain small model. The success of small-sized CNNs such as SqueezeNet [82], MobileNet [8] and ShuffleNet [9] has demonstrated the crucial role of the network architecture as well as the opportunities of designing more efficient CNNs towards embedded inference. As mentioned in section 2.1.7, the networks can be designed specifically by human intuition and experience, or automatically by NAS [10]. Therefore, the efficient model designs are can be divided based on these two approach. Since the convolutional layers are the major factor causing the huge number of MAC operations in modern CNNs, most of the works focus on simplyfing the structure of the convolution layers, either by a simpler convolution layer or a cell combining several light-weight convolution layers.

Hand-crafted model designs

The output of the regular convolution is a combination of spatial and cross-channel correlation between all input channels. Early works [130] propose to factorize the standard $d \times d$ convolution into a stack of $d \times 1$ (horizontal) and $1 \times d$ (vertical) convolution. However, this approach still has some drawbacks such that the efficiency gain is limited when the kernel size d is small, and some types of features (*e.g.* diagonal edges) are not suitable for being decomposed into 1D features. However, the idea of using these 1D spatial convolutions are then employed in ACNet [131], which replaces each 3×3 convolution at the training time by an Asymmetric Convolutional Block (ACB) of 3×3, 3×1 and 1×3 convolution layers, and their outputs are summed up to obtain the output of the block. At inference time, each ACB can be equivalently represented by a single 3×3 convolutional layer, therefore, if the model is small-sized, this scheme is a promising way to ease the training and improve the model's performance. Other works focus on reducing the cost introduced by the cross-channel correlation. For example, MobileNet V1 [8] makes use of the depthwise separable convolution (DSConv) (cf. section 2.1.1) including a depthwise and a pointwise convolutions. Note that each convolution layer is followed by a Batch Normalization (BN) and a ReLU activation. [132] reduces furthermore the cost of DSConv by inverting the order of the pointwise and depthwise convolution then performing a low-rank approximation of the pointwise matrix. Besides, grouped convolution [28] is another way to reduce the dense connection between input and output channels (cf. section 2.1.1 and Figure 2.2b). Unlike the original version where the groups are unchanged and structurally partitioned, [48] learns the group structure by an end-to-end manner, adding more flexibility into the model. However, in hardware implementation, this introduces additional cost and efforts to encode the resulting unstructured connections. [133] dynamically selects the input channels for each group based on the saliency of all input channels.

Another approach for designing light-weight CNNs is to build an optimal computation cell including efficient convolutions (GConv, DWConv, DSConv) and skip connections (Add or Concatenation), inspired from the success of SqueezeNet [82]. The simple DSConv in MobileNet V1 is then replaced by a residual block in MobileNet V2 [134], where the residual mapping consists of two pointwise convolutions and a DWConv inserted in-between (Figure 2.18). ShuffleNet V1 [9] introduces a normal block (Figure 2.19a) when the block's output has the same dimension with the input, and a reduction block (Figure 2.19b) when using convolution layer with strides of 2. Unlike MobileNet V2, the two regular 1×1 convolutions in the residual branch are replaced by the group-wise version to reduce furthermore the number of parameters and MACs. Moreover, to alleviate the side effects due to the use of grouped convolutions, a channel shuffle mechanism is inserted after the first grouped pointwise layer. ShuffleNet V2 [44] returns with the standard pointwise convolutions in the residual branch, besides an additional channel splitting at the beginning of the normal block. The output of the two branches are concatenated to obtain the block's output (Figure 2.20).

These aformentioned CNN architectures are designed taking into account the model size/accuracy and computational cost/accuracy trade-offs. The common point in their basic building blocks is the proper combination between light-weight convolution (GConv, DWConv, DSConv) to ensure both spatial correlation and cross-channel correlation like in standard convolution while not affecting the training of the network. This task requires expert knowledge as well as considerable effort on empirically choosing the configuration for each element in the blocks.

Neural architecture search (NAS)

Despite the success of manually-designed CNN architectures, building an optimal model is a challenging task and limited by human knowledge as well. An alternative approach is to automatically searching for efficient neural network architectures with limited human intervention. In general, a

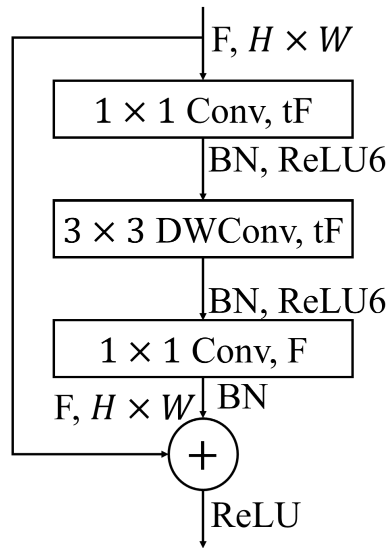
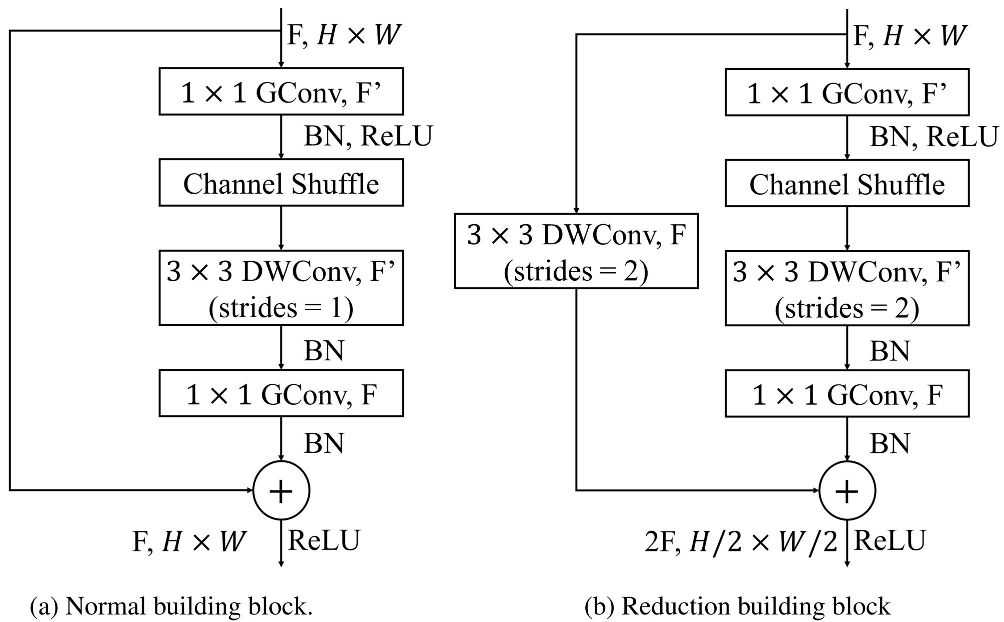


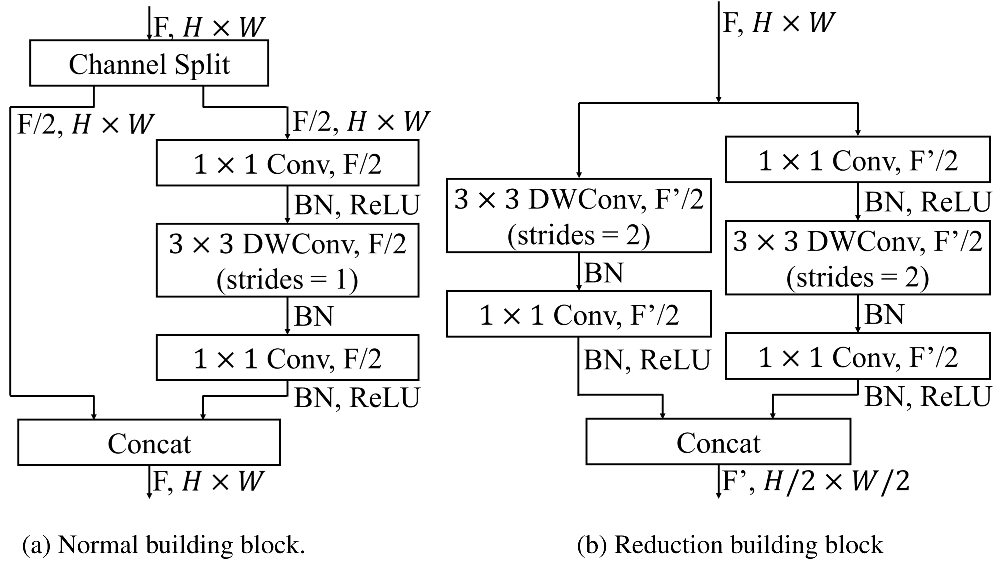
Figure 2.18: The building residual block of the MobileNet V2 [134]. The parameter t is the expansion factor for increasing the number of intermediate channels.



(a) Normal building block.

(b) Reduction building block

Figure 2.19: Building blocks of ShuffleNet V1.



(a) Normal building block.

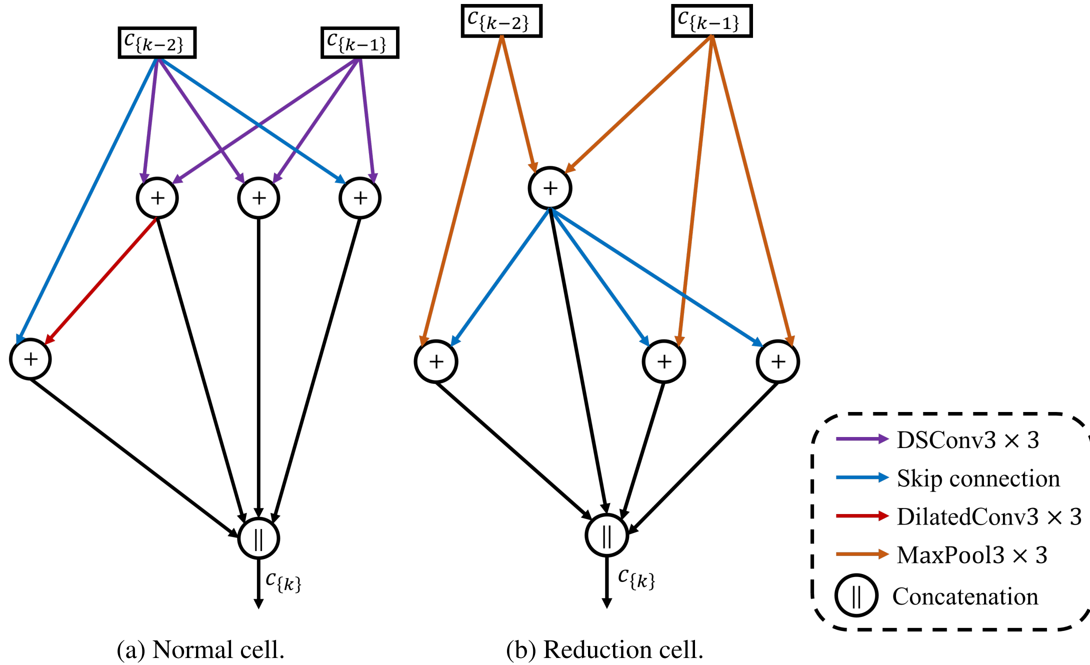
(b) Reduction building block

Figure 2.20: Building blocks of ShuffleNet V2.

NAS consists of three main parts: search space, search algorithm and the performance evaluator. The early work of [10] figures out that neural networks can be represented by strings issued from an RNN controller, then using the accuracy of the sampled RNN as a reward signal to update the controller. Since the number of network samples in the search space grows exponentially if we search for every element's configuration along with their inter-connections, it is crucial to restrict the search space, *i.e.* adding some human preliminary definition into the network architecture.

One of the popular approach for restricting the search space is to pre-define the overall CNN architecture as a stack of basic cells repeated many times but with different weights, then we only need to search for the optimal cell architecture. Besides the time-consuming reinforcement learning-based approach like [92], DARTS [93] proposes a gradient descent-based search algorithm, where the search space can be parameterized by a set of continuous variables representing the possibility of choosing one operation between intermediate data in the network (Figure 2.21). This idea is then improved by SNAS [135] and P-DARTS [136]. It is clear to see that the optimal cells found by these methods has complex structures and cannot be easily explained based on human knowledge.

Unlike the aforementioned cell-based approaches which mainly rely on the connections inside the cells, other works focus more on the global architecture and pay more attention to the hyper-parameters (*e.g.* depth, width) of the cells. MnasNet [137] decomposes the model into several blocks, where each block is a stack of repeated identical cells. This way, the search space is factorized into several per-block sub search space consisting of the convolutional operation (ConvOp) and hyperparameters such as kernel size, skip operations, number of output filters and the number of repeat times. Since the ConvOp is chosen from basic convolutions and existing architecture (*e.g.* MobileNet block [134], SE block [84]), the resulting cell architecture is quite close to existing model architecture. ProxylessNAS [138] then proposes a binary-gated DARTS-based method for learning both parameters and architectures. Moreover, it also incorporates a latency-aware regularization to drive the search towards different targeted hardware platforms. EfficientNet [139] firstly applies a similar approach like [137] but with FLOPS-aware objective to find a mobile-size baseline architecture, then scales this model by a small grid search over expansion factor for network depth, width, and resolution. Besides latency or FLOPS, existing works also make use of other hardware-aware metrics such as energy consumption [140] or area [141].



(a) Normal cell.

(b) Reduction cell.

Figure 2.21: Optimal cells learned by DARTS on CIFAR-10. Note that the operations are characterized by colors. $c_{\{k-1\}}$ and $c_{\{k-2\}}$ are outputs of the two previous cells.

2.3.2 Model compression techniques

While efficient model design directly creates a compact neural network to meet the hardware-related specifications from the beginning, model compression techniques mainly aim at compressing a given model architecture to alleviate the memory and computation overheads. In practice, the linear operations $Z = h(W,X)$ in Equation 2.1 , *e.g.* convolution or matrix-vector multiplication, dominate the number of MACs as well as contributes most of the model's weights. Therefore, compressing the model generally relies on applying additional operations to the weights W and/or the input data X of each layer. Based on the goal of these additional operations, we have identified five classes of compression approaches as depicted in Figure 2.22. More concretely, network quantization makes use of quantization mappings q_1, q_2 to reduce the bitwidth of the weights/activations. Network pruning aims at reducing the number of parameters and layer interconnections, which can be equivalently represented by an element-wise multiplication between the weight matrix and a fixed binary mask M. Dynamic network also simplifies the computational graph by dynamically choosing a small portion of input data during the feed-forward, but this time the binary mask is computed based on the input data itself rather than a fixed pattern. Low-rank decomposition splits tensor weights into more smaller ones via a linear transformation h_1. Finally, weight generation networks leverage auxiliary small-sized models $g(W,\Omega)$ to generate the weights of the main model.

Quantization

One of the most prominent approach in DNN compression is network quantization which attempts to reduce the precision of the data representation in DNNs from the conventional full-precision 32b to 8b or less, hence advantageously reducing the memory requirements as well as the computational complexity for hardware implementation and acceleration. This method consists of applying the nonlinear quantization mappings to different objects of a DNN during the entire back-propagation procedure: weights (W), activations (A), gradient (G) or even update (U). Since this thesis mainly leverages model compression approaches for embedded inference, we will focus

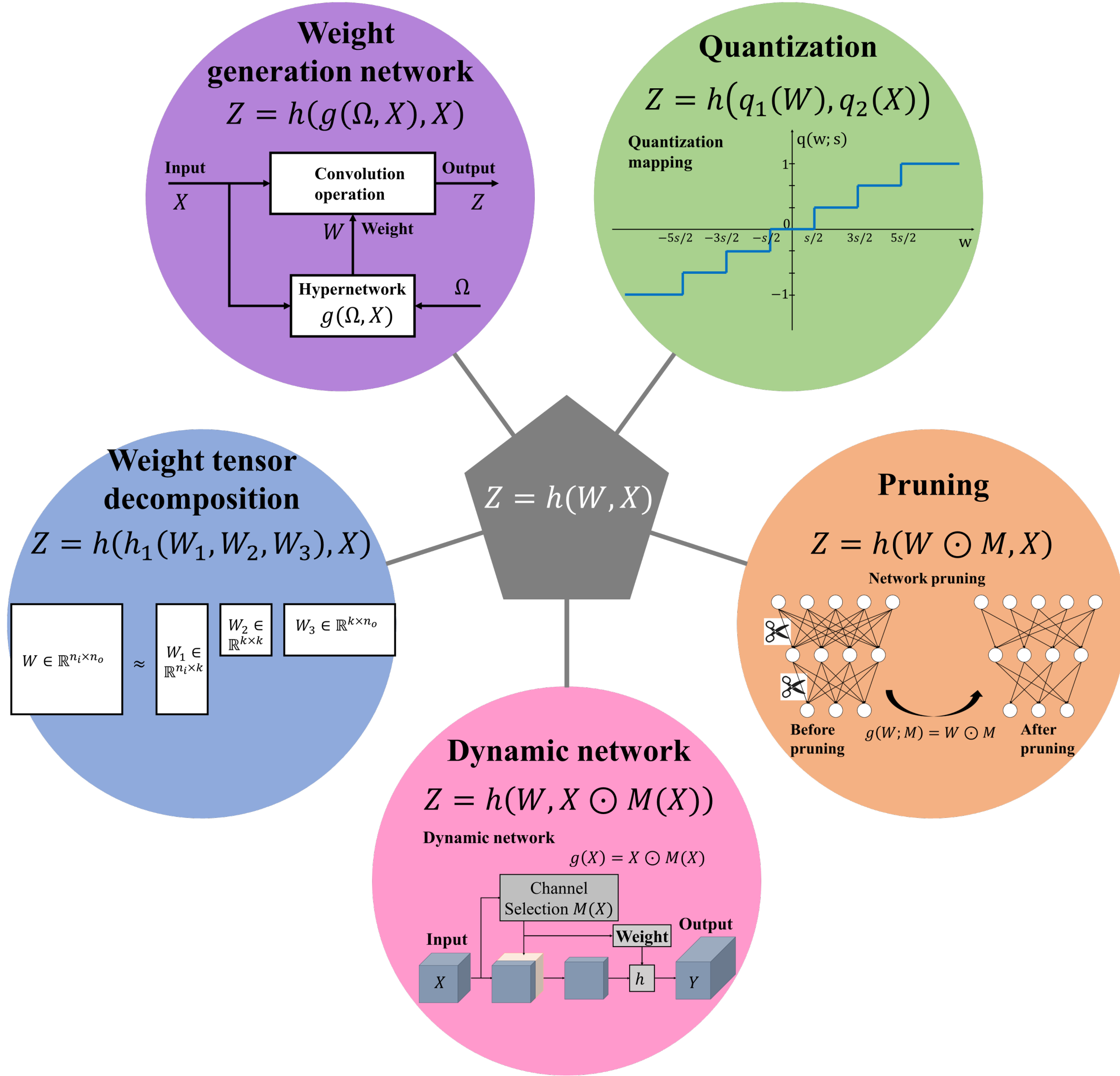


Figure 2.22: Taxonomy of model compression approaches. The central operation causing the hardware-related bottleneck of DNNs is the linear operation $\boldsymbol{Z} = h(\boldsymbol{W}, \boldsymbol{X})$ (*e.g.* matrix multiplication or convolution). Compression techniques consists of applying additional operations to the operand $\boldsymbol{X}$ and $\boldsymbol{W}$ in order to lighten this operation or simply reduce the model size.

only on the quantization of W and A. In Figure 2.22, the quantization is depicted by applying the mapping q_1 and q_2 to the weight tensor W and the input data X, respectively.

The quantization is defined by a projection which maps full-precision values in a continuous space onto low-precision values in a discrete space. If the quantization thresholds and the resulting quantized values are non-uniformly spaced, it is called *non-uniform quantization*. This can be done by either using a logarithmic representation [142], entropy-based clustering [143], code-book [11] or learnable approach [144]. These constraint-free mappings increase the representation power of the quantized values and allow better capturing the distributions of the full-precision inputs, hence limiting the performance degradation due to the quantization. However, non-uniform quantization schemes are typically difficult to be implemented and deployed on hardware, since they usually requires additional modules to store and access quantized values as well as computing the matrix-vector multiplication between quantized values. For this reason, designing uniform (or linear) quantization is still the de-facto choice thank to its simplicity and hardware compatibility. In this case, the quantization mapping can be described as follows:

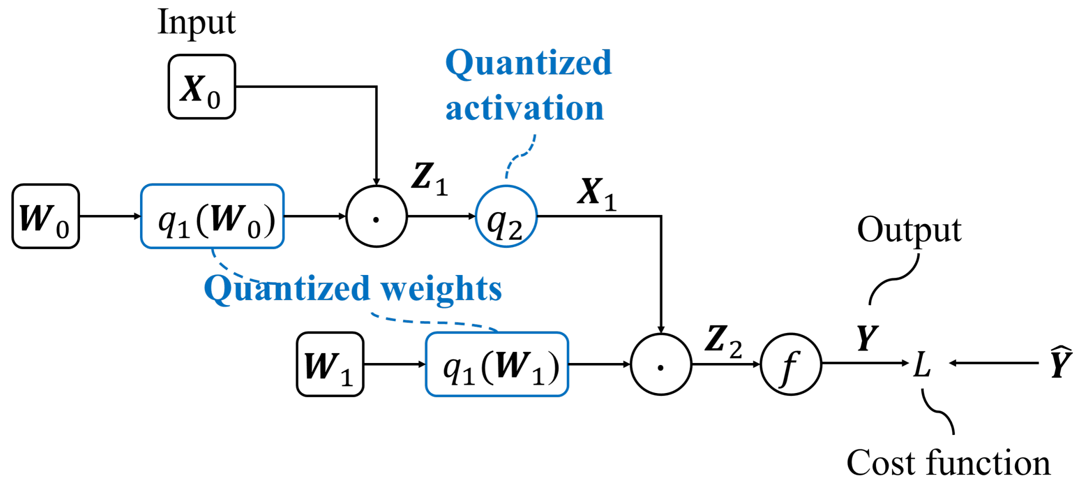
$$x_q = q(x) = s\ \text{Clip}\left(\lfloor \frac{x}{s} \rceil, \alpha, \beta\right) = \begin{cases} s\lfloor \frac{x}{s} \rceil & \text{if } \alpha \le \lfloor \frac{x}{s} \rceil \le \beta, \\ s\alpha & \text{if } \lfloor \frac{x}{s} \rceil < \alpha, \\ s\beta & \text{if } \lfloor \frac{x}{s} \rceil > \beta \end{cases} \tag{2.40}$$

where x is full-precision value, s is the scaling factor defining the quantization step size, α, β are the integers defining the clipping range $[\alpha, \beta]$. The factor s outside of the Clip function aims at keeping the initial dynamic range of the input value x. In some case, in order to obtain straightforward integer values, we can use the integer version without s. This mapping requires $log_2(x_{max} - x_{min} + 1)$ bits to represent each quantized values. When the clipping range is symmetric with respect to the origin, *i.e.* $-\alpha = \beta$, the mapping is called *symmetric quantization*. Otherwise, it is called *asymmetric quantization*. Choosing the clipping range deeply depends on the quantization object in order to fully capture all discrete values. For example, since the distribution of weights usually has symmetric "bell curve" shape, it is better to choose a symmetric range, while quantizing ReLU activation strictly requires a non-negative clipping range with $\alpha = 0$. More importantly, how to define the step size s plays a crucial role in the resulting quantized values. If the step size is too small, most of the quantized values stay near the clipping range. On the other side, a large step size may results in the dominance of low-magnitude discrete levels. In pratice, these parameters can be determined based on data distribution [145] or gradient-based techniques [26].

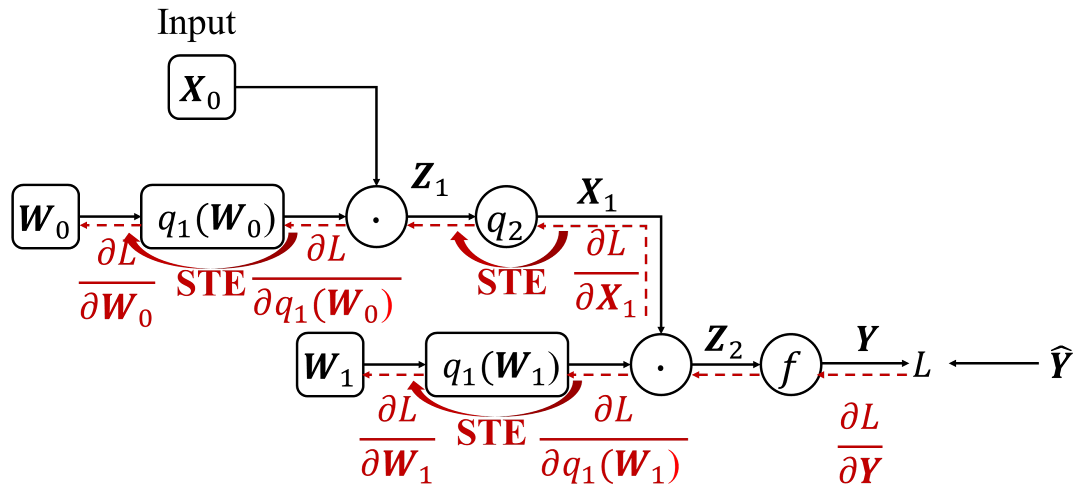
Inserting the quantization into the forward pass will certainly modify the output of the model and causes performance degradation, it is thus crucial to determine the quantized weights/activations such that the degradation is limited as much as possible. This recovery process can be done after the training of the full-precision model [146], [147], namely Post-Training Quantization (PTQ), or during the training with the quantization inserted in the forward pass [148], [149], [26], namely Quantization-Aware Training (QAT). Although being more hardware-expensive and time-consuming at training time, QAT generally offers better algorithmic performance with limited to negligible loss compared to PTQ method. As this thesis focus on improving the hardware efficiency/algorithmic performance trade-offs at inference time, we mainly focus on QAT approaches. Unfortunately, the round function $\lfloor . \rceil$ is a piece-wise flat operator, its derivative is thus almost everywhere zero, preventing the model from learning. To circumvent this issue, the mainstream approach in literature [145], [150], [151] is to use a coarse gradient based on the STE scheme (see section 2.1.2). Intuitively, STE ignores the rounding operation (or any other quantized operations like ceil $\lceil . \rceil$, floor $\lfloor . \rfloor$ or sign) and approximates it with an identity which allows the gradient flows to pass through the quantization during backward pass, *i.e.* $\frac{\partial \lfloor x \rceil}{\partial x} = 1$. Consequently, the STE derivative of the uniform quantization mapping in Equation 2.40 can be computed as follows:

$$\frac{\partial x_q}{\partial x} = \begin{cases} s\frac{\partial \lfloor \frac{x}{s} \rceil}{\partial x} & \text{if } \alpha \le \lfloor \frac{x}{s} \rceil \le \beta, \\ \frac{\partial s\alpha}{\partial x} & \text{if } \lfloor \frac{x}{s} \rceil < \alpha, \\ \frac{\partial s\beta}{\partial x} & \text{if } \lfloor \frac{x}{s} \rceil > \beta \end{cases} = \begin{cases} 1 & \text{if } \alpha \le \lfloor \frac{x}{s} \rceil \le \beta, \\ 0 & \text{if } \lfloor \frac{x}{s} \rceil < \alpha, \\ 0 & \text{if } \lfloor \frac{x}{s} \rceil > \beta \end{cases} \tag{2.41}$$

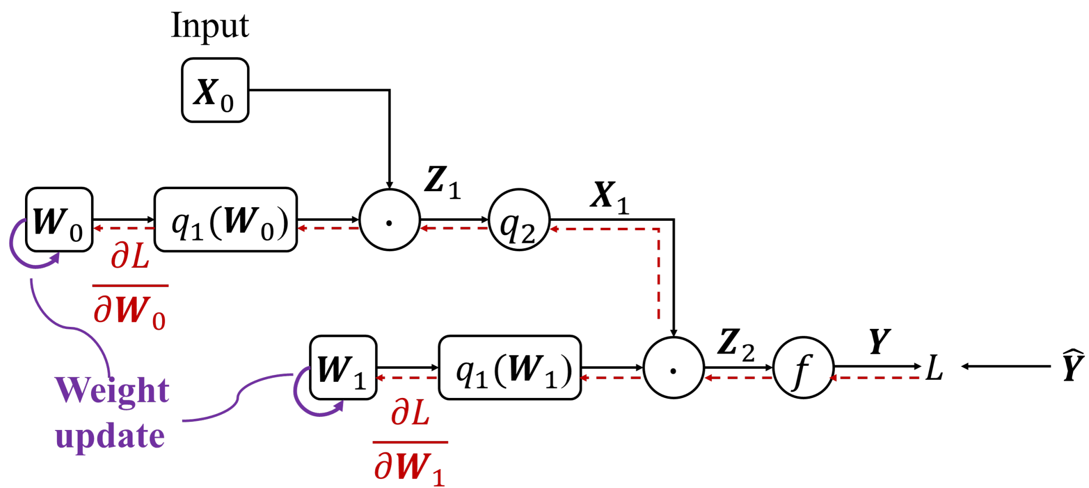
which is equivalent to a clipped identity. Figure 2.23 demonstrates a gradient descent iteration of a quantized neural networks (QNNs) using STE scheme. During forward pass (Figure 2.23a), the weights W_0 and W_1 do not directly participate in the matrix multiplication, but rather their quantized version $q_1(W_0)$ and $q_1(W_1)$. Therefore, they are also called *proxy weights*. The gradients of the cost function w.r.t the weights are computed using the chain rule and STE to bypass the quantization operations applied to both weights and activations (Figure 2.23b). Finally, the proxy weights are adjusted, hence updating the quantized weights (Figure 2.23c). Apart from the STE-based methods, [152] also uses pulse shapes like rectangle or triangle to better approximate the impulse derivative of the quantization mapping at discontinuous points. [153] introduces a differentiable soft approximation of the uniform quantization which is gradually sharpened during



(a) Forward propagation.



(b) Backward propagation.



(c) Weight update.

Figure 2.23: Three stages of a STE-based gradient descent iteration of a quantized neural network.

the learning process, aiming at providing more accurate gradients compared to the STE-based approaches. [154] alleviates the quantizer's gradient vanishing problem by using a gradually-adjusted blending between the proxy weights and the quantized weights in forward pass which gives rise to non-zero gradients w.r.t the proxy weights in backward pass. [155] represents the weights as a learnable probability distribution over the discrete space from which the quantized weights can be sampled. Despite remarkable progress in recent years, these STE-free approaches require a higher level of customization during the learning procedure compared to the baseline STE, therefore STE still remains the most popularly used approach.

In the most extreme case, model's weights and activations can be represented by only 1-bit as proposed in Binarized Neural Networks (BNNs [12]) using the previously mentioned Sign (Equation 2.15) or Heaviside functions (Equation 2.17). [156] then even demonstrates that these BNNs can be compressed furthermore since the binarized kernels of each layer tend to distribute over a small subset rather than the whole binarized space. Figure 2.24 illustrates the power consumption versus computation throughput of accelerators of different precisions. It is clearly shown that binary and ternary precision dominate the higher TOPs/W zone. With this in mind, extremely low-precision ASAs may achieve a higher TOPs/W, allowing to increase furthermore the gain in terms of power consumption. Despite several hardware-related benefits, using ultra low-precision quantization may significantly reduce the algorithmic performance of the model, especially if the task requires more complex information (*e.g.* object detection or image compression). Besides, the role of every layer in the model changes accordingly to their positions as well as the model architecture. This gives rise to another important question: how to determine the optimal bit-width for every layer's weights and activations in the model for a certain task? Existing works address this challenge by estimating the layer's contribution level to the model's performance [157] or gradient-based optimization [158] or NAS-based approach [159], [160]. Nevertheless, these methods often require a large amount of time and additional computation overheads for the search of proper bitwidth.

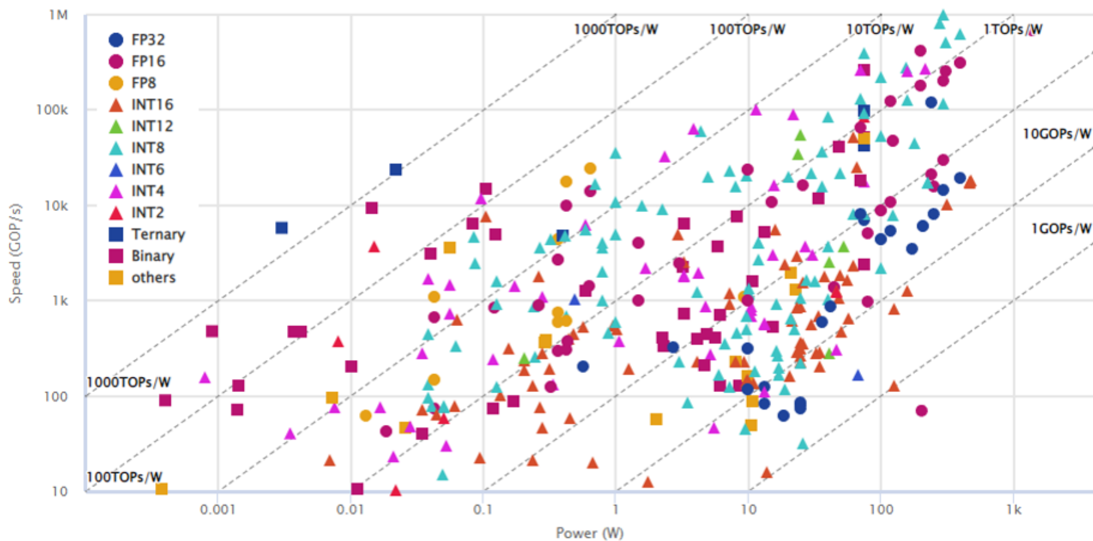


Figure 2.24: Power versus Computation Speed of different DNN accelerator platforms based on precision. Note that this figure is originally introduced in [129].

It is noteworthy mentioning that most of existing works mainly focus on the quantization of weights and activations in DNNs while ignoring other elements such as BN and skip connections. This may results in other hardware bottlenecks, in particular for hardware platforms supporting only low-precision integer computations.

Pruning

DNNs are typically over-sized and exhibits overfitting, in which only a portion of neurons and weights is important, while the rest has limited impact to the output of the models. This has motivated the core idea of model pruning: remove irrelevant operations, neurons and weights to improve the hardware-algorithm compromise [14]. This operation can be described by an element-wise multiplication between the weight tensor W with a binary mask M in Figure 2.22. Originally, given a pre-trained DNN, a typical model pruning procedure consists in performing pruning and fine-tune to obtain a final pruned model. To determine which parameters to be removed, some evaluation criterion is employed. [161] and later [162] believe that magnitude can be used to measure the relative importance of weights by arguing that smaller weights tend to produce weak activations. [163] empirically observes that ℓ_2-norm allows better pruning results than ℓ_1-norm. Besides, [164] quantifies the importance by directly approximating the impact of pruned weights to the loss function.

Once the pruned parameters are identified and removed, it is crucial to retrain the model to recover its performance. The whole pruning and fine-tuning process can be performed recursively or one-shot. [162] shows that iterative pruning may achieves better results than one-shot counterparts. [15] later proposes an iterative method to identify the remaining parameters within a DNN which allows to learn faster while achieving on-par accuracy with the original network. Compared to the time-consuming iterative approach, one-shot pruning [16], [165] is more training-efficient which proposes to prune the model once at initialization prior to the training.

According to the pruned structure, we can categorize CNN pruning techniques into two kinds: non-structured pruning and structured pruning. The first kind mainly relies on the notion of weight pruning [161], *i.e.* zeroing unimportant individual connection without considering the structure of the pruned weights. This allows a high level of flexibility but results in irregular structures which is not favorable in hardware implementation. Concretely, the pruned weights will be simply considered as zero values to be consistent with the hardware, hence discarding memory and computation benefits offered by the pruning. Therefore, we need to introduce extra modules to store and read the position of the remaining weights, leading to problems on memory access and cache. Although recent accelerator platforms also focus on the sparse multiplication, it still requires a considerable effort on hardware design. For these reasons, applying structured pruning [166] is the preferable option thank to its hardware compatibility. To avoid common ambiguity in the taxonomy of structured pruning methods, let us remind the Conv2D layer (cf. section 2.1.1) with 4D weight tensor of size $h \times w \times C_l \times C_{l+1}$ where h, w, C_l, C_{l+1} are the dimensions of the l-th weight tensor along the axes of spatial height, spatial width, channel (*i.e.* input channels) and filters (*i.e.* output channels), respectively. Channel pruning [167], [166], [165] aims at removing the input channels from C_l to a desired number C_l' $(0 \leq C_l' \leq C_l)$ by discarding the corresponding channels of the weight tensor. On the other hand, filter pruning [162], [168], [169] involves removing the number of output channels from C_{l+1} to C_{l+1}' $(0 \leq C_{l+1}' \leq C_{l+1})$ by zeroing out the corresponding filters. Subsequently, these removed feature maps result in a channel pruning in the next convolutional layers and the architecture of the pruned model is also modified. Since the channel pruning has less damage to the CNN structure than the filter-wise approach, it may naturally offer a compact model with less performance degradation. However, in terms of hardware mapping, the filter pruning provides higher pruning rate while being more compliant with the implementation. Concretely, using a filter pruning approach, we can obtain a compact model with conventional CNN structure that is compatible with off-the-shell deployment platforms. Whereas in the case of channel pruning, we still need additional modules to store the indices of the pruned channels.

Dynamic networks

Dynamic neural networks adapt their architectures or weights based on the input during the inference stage. It is biologically inspired from the belief that the brain process information in a dynamic way [170] depending on the given information itself. Compared to the static counterpart, dynamic networks exhibit several favorable properties such as an increasing representation power due to the enlarged parameter/computation space [171], better interpretability [172] and especially hardware efficiency [173], [17], [18]. If targeting efficient inference, dynamic networks adjust the computational graph (*i.e.* along the axes of depth or width) such that small sub-networks are used for easy input sample while more complex ones are dedicated to difficult inputs. It is desirable that paying attention to the input sample to adapt the processing/inference can boost the learning efficiency and enable us reducing the average inference cost.

In the same vein as static pruning (*c.f.* section 2.3.2), dynamic pruning [17], [18] aims at reducing the number of computations in CNNs by using a variable subset of convolution weights. However, unlike the case of static pruning where the pruning patterns are fixed and data-agnostic for all input samples, dynamic pruning adapt the pruned weights in an input-dependent manner. Figure 2.22 depicts a case of dynamic pruning where the binary mask $M(\boldsymbol{X})$ is a function of the input sample $\boldsymbol{X}$. [17] proposes a dynamic channel pruning scheme taking only a subset of input channels to generate the partial sum of the output and decides whether to discard or keep the computations in the rest of input channels. [174] makes use of an assistance model to adaptively predict the pruned filters for every input. Model's depth is another dimension to be explored in dynamic networks, which proposes to either exit early if the output prediction has high confidence [173] or skip residual blocks based on the feature maps of previous layers [175].

In terms of hardware efficiency, both dynamic pruning and dynamic depth do not reduce the memory requirements for storing the model, since all parameters are necessary for later access once the selection is done. On the contrary, it needs extra memory for the storage of selector network. The advantages of dynamic networks mainly relate to the computational reduction, this under the constraint that the computation cost of selector network is negligible compared to that of the main network.

Weight tensor decomposition

Weight tensor decomposition aim at factorizing the weight tensor of each layer into smaller tensors, hence reducing the model size as well as the number computations. This operation is illustrated in Figure 2.22 by decomposing the weight tensor W into three smaller tensors W_1, W_2 and W_3. Early works propose to use Singular Value Decomposition (SVD [19]) or Principal Component Analysis (PCA [176]) to take advantages of the redundancy within the convolutional weight tensor by clustering filters then forcing the weight-sharing in their approximations. [177] then presents a cross-filter low-rank approximation which decomposes the canonical $h \times w$ Conv2D layer with C_{l+1} output channels into another $h \times w$ Conv2D layer with only m output channels ($m \ll C_{l+1}$) followed by a pointwise convolution projecting these m channels into C_{l+1} output channels. In particular, [178] and later [20] propose a unified framework for low-rank decomposition and filter pruning by inserting a regularized coefficient matrix into the normal convolutional layer. Recently, advanced tensor decomposition techniques, including tensor train [179] and tensor ring [180], have attracted much attention and become an active line of research. Besides, common convolutional blocks such as DSConv [50] or the bottleneck design in ResNet [24] can be equivalently expressed as light-weight decompositions of the weight tensor.

Unlike pruned or dynamic networks, low-rank decomposed models only involve primitive operations which can be implemented by of-the-shell architectures at inference time. The drawback

of these approaches is that they introduces additional hyperparameters (*i.e.* the ranks) that need to be carefully chosen, and the original model need to be trained under low-rank regularization in order to obtain a better compression rate. Beside, since the decomposition increases the model's depth with stacks of linear layers, the model is more prone to gradient vanishing, hence complicating the training.

Weight generation networks

In canonical DNNs, each layer contains weight tensors to convert the raw inputs to some desired representations via a linear transformation (called primary operation). If both inputs and outputs are high-dimensional tensors, the weight tensors will contain a considerable amount of parameters, causing memory-related overheads especially for memory-constrained systems. An unconventional approach to cumbersome this issue is to generate the layer's weights from a much smaller-sized neural network inside the layer, termed as weight-generation networks or weight net. This idea is illustrated in the Figure 2.22 by an auxiliary function $g(\Omega, X)$ with parameter Ω and a dependency of the input X. Hypernetworks [21] originally uses a commonly-shared small network to generate the weights for all layers in the main network, allowing to reduce the number of learnable parameters while achieving acceptable accuracy level. Whereas [21] introduces a static generation approach, [181] proposes to dynamically generate the main layer's weights taking into account the layer's input. In the same vein, [182] then makes use of the input activation to generate one filter for each spatial position. WeightNet [22] incorporates an input-specific weight net with grouped fully-connected layer to generate the weights of the main convolutional layer.

Unlike previously mentioned model compression schemes which may reduce both model size as well as computational complexity, weight nets can only target the model size, since they also introduce extra computation for generating the main model's weights. For the sake of efficiency, this additional computational cost must be negligible compared to that of the original model, hence limiting the design space of the weight nets. From the practical deployment point of view, such limitations are undesirable, in particular for systems with constrained computational capacity. Table 2.1 summarizes the impact of those presented model compression techniques to the memory requirement and the computational cost during DNN inference.

Table 2.1: Qualitative comparison of the impact of hardware-algorithmic enablers to memory and computation costs during DL inference. ☺: reduced; ☹: increased; —: unchanged.

Method \ Metrics	On-chip memory	Feature-related memory	#MACs	MAC complexity
Efficient architecture designs	☺☺	☺	☺	—
Quantization	☺	☺	—	☺☺
Pruning	☺	☺	☺☺	—
Dynamic network	☹	☺	☺	—
Tensor decomposition	☺☺	☺	☺	—
Weight generation network	☺☺	—	☹	—

2.3.3 Training trick for compact models: Knowledge Distillation

The lack of representation power makes training compact DNNs much more difficult to learn highly relevant features directly from the input data. Knowledge distillation is a training trick aiming at using the knowledge from a larger model as an extra supervision to guide the learning of a smaller one, thus easing the training of this small model. This approach is deeply inspired by the way the teacher explains complex notions to the student using simple analogies. The larger DNN is therefore also called as *the teacher* while the smaller DNN is termed *the student*. The student network is commonly chosen as a smaller-sized network compared to the teacher, which can be built by reducing the width [183] and the depth [184]; quantizing [29] or pruning [185] the teacher model; or using light-weight compact model [186]. Different types of information from the teacher network can be used as the knowledge to be distilled to the student network, including the output logits [187], the intermediate features [29] or even the weights [188].

Along with the rapid development of efficient DNN architecture design and model compression techniques, knowledge distillation has drawn considerable attention in the last few years. In the context of compact efficient DNNs where training the model is a real challenge, knowledge distillation offers a great trick to ease the learning process and better retain the model's performance. Therefore, combining these model compression approaches in a knowledge distillation training framework should be a promising line of future research.

2.4 Conclusion

Throughout this chapter, we have provided an insightful overview about Deep Learning, DNN accelerator platforms and hardware-algorithmic enablers for embedded DL inference. Section 2.1 briefly introduces the basic notions of Deep Learning, from the commonly used layers to the complex model architectures. As DNNs are generally over-parameterized, they usually exhibit overfitting. This indicates that there exists a room for improving the model's hardware efficiency while reducing the generalization gap.

Section 2.2 surveys current DNN accelerators which can be categorized into three classes based on their application-versatility. Although general-purpose platform like CPUs, GPUs and FPGAs are considered as the *de-facto* accelerator for executing DNNs thanks to their high level of maturity, their power inefficiency is a major downside. This has motivated the development of more efficient platforms dedicated only to DNN processing. In the most extreme case, when the application is well defined and the computational complexity could be upper-bounded, the DNN model could be designed by integrating some hardware implementation perspective. Indeed, this hardware-algorithm co-optimization is a promising research line in the aforementioned room of efficiency improvement, especially when it is empowered by the tremendous progress of DNN architecture designs and model compression techniques.

Section 2.3 provides an overview about the hardware-algorithms enablers towards efficient DNNs. It is worth mentioning that although these techniques are presented separately, they can be combined together to reduce furthermore the hardware implementation cost, as they are complementary to each other. Indeed, while model compression techniques like quantization, pruning or tensor decomposition are mainly used to compress existing large model architectures, the implementation of the resulting model is still questionable. Another direction consists of proposing a hardware-compliant DNN topology on which we apply model compression techniques to reduce furthermore the memory and computation costs. This thesis is placed in the same vein by leveraging the quantization as the pivotal approach throughout the conception of the DNN architectures and their later compression to meet the constrained resource budget of an ASIC design.

3

Histogram-Equalized Quantization (HEQ) for low-precision weighted networks

As mentioned in the previous chapter, model quantization is one of the most relevant approaches for designing compact DNNs enabling efficient hardware implementation and deployment. This chapter will start by reviewing the state-of-the-art quantization-aware training techniques allowing to train the DNNs with low-precision weights. In particular, we focus on the case of linear symmetric quantization due to its simplicity and hardware compatibility. Adjusting the quantization mapping according to the data or to the model loss seems mandatory to enable a high accuracy in the context of quantized neural networks. In the same vein, this chapter present Histogram-Equalized Quantization (HEQ), an adaptive framework which automatically adapts the quantization thresholds using a unique step size optimization. HEQ is then evaluated on multiple datasets and DNN architectures. We empirically show that HEQ may achieve state-of-the-art performances while being more hardware-compliant compared to previous works.

Contents

3.1 Context **52**
3.2 State of the art in linear symmetric quantization **53**
3.2.1 Quantization error minimization methods 53
3.2.2 Task loss gradient-based methods 54
3.2.3 Statistics-based methods 54
3.3 Linear symmetric quantization and the unbalanced histogram **55**
3.4 Histogram-Equalized Quantization (HEQ) **55**
3.5 Experiments **58**
3.6 Conclusion and Perspective **60**

3.1 Context

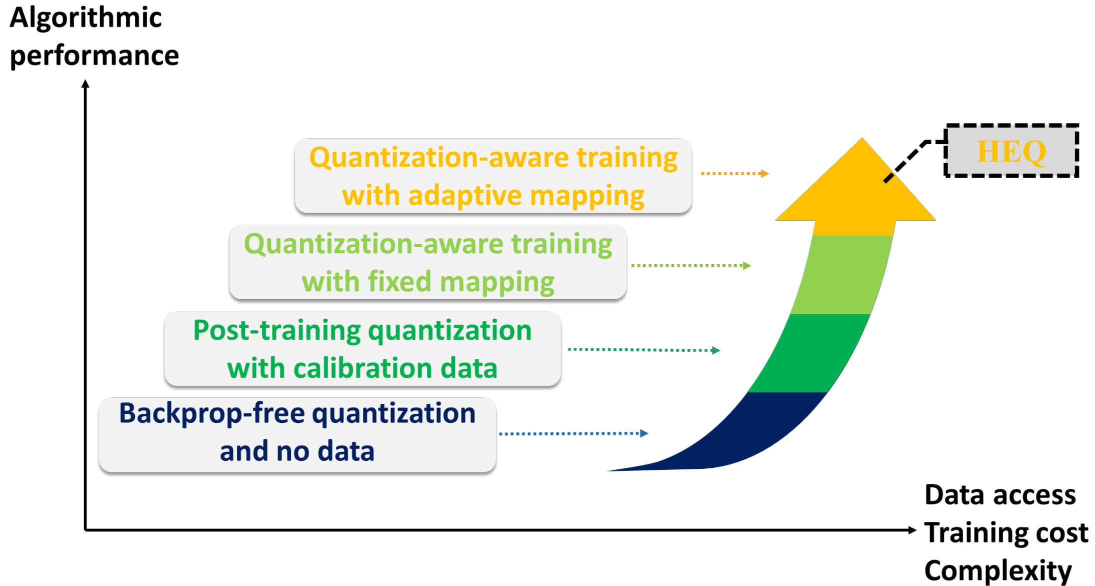


Figure 3.1: Quantization frameworks in increasing order of data availability, training cost and also algorithmic performance.

Designing low-precision networks [189] is a promising area of research aiming at reducing the bit width to represent weights and activations, thus reducing the overall computational complexity and memory-related costs, as well as easing the implementation on resource-constrained devices to perform inference at the edge. The advantages of quantization have been demonstrated on several resource-efficient low-precision CNN accelerators [121], [190], [191], [192], [193].

In pratice, the quantization of a model depends on several factors: the availability of data (*i.e.*, labeled and unlabeled data), the affordable training cost, the algorithmic complexity, and the desired level of performance. Based on these elements, we can loosely categorize existing quantization frameworks into four classes as depicted in Figure 3.1. In the most restricted case, the quantization can be done without back-propagation nor data access [194], however, this comes at the cost of significant performance loss while still maintaining a high-precision network (*e.g.* $\geq$ 6-bit) that limits its benefits compared to the full-precision network counterpart. Post-training quantization [146] is also a backpropagation-free approach, but it requires a small portion of data (*i.e.*, not necessarily labeled) to calibrate the quantization mapping for activations. These two approaches allow to accelerate the deployment while alleviating privacy concerns regarding the data. However, in the case where we have full access to the training data, Quantization-aware training (QAT) is needed if the algorithmic performance and the hardware efficiency are both crucial. Advantageously, QAT enables us to preserve the performance of quantized models and avoid unacceptable accuracy degradation due to the limited precision. QAT usually consists in using real-valued proxies of the model weights that are on-the-fly quantized during the forward pass while being updated during the backward pass [148]. As such, the full-precision weights of the model is learned to adapt the model behavior accordingly to the quantization process. While the baseline QAT approaches apply a fixed quantization mapping which remains unchanged during the training, more advanced methods argue that the quantization mapping should be also adjusted along with the model's parameters as this enables a better performance level. Nevertheless, these adaptive QAT methods may require a higher complexity in designing the quantization and the learning procedure.

Choosing the quantization mapping is another important question. Although several nonlinear quantization mappings [195], [142] and [196] have demonstrated remarkable algorithmic performances, they are not fully compliant with a simple hardware implementation. On the contrary, a linear symmetric mapping [25] naturally matches off-the-shell hardware, making it a more relevant and reasonable choice for model quantization. However, its lack of flexibility makes the adaptation of linear symmetric quantization an open question so far.

As mentioned before, this thesis targets the hardware-algorithm compromise of DNN accelerators that can be deployed for an long-term period, therefore we make use of QAT methods as the pivotal element for building compact DNNs with highly acceptable algorithmic performance. In this section, we present a novel QAT method which efficiently adjusts the linear symmetric mappings to obtain the quantized model with very low-precision weights.

3.2 State of the art in linear symmetric quantization

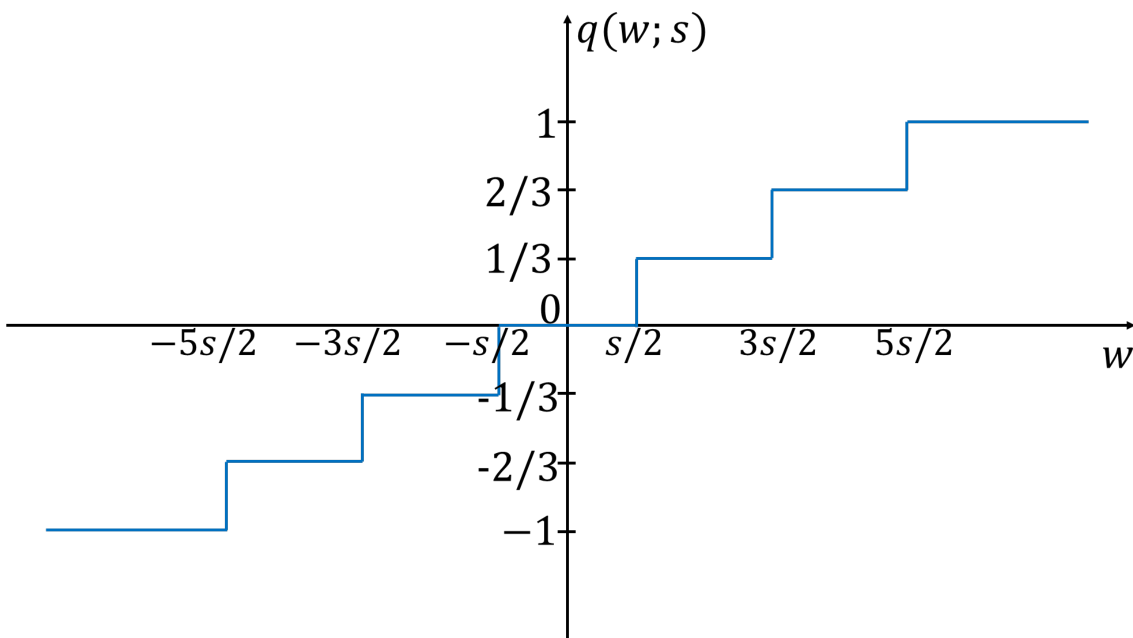


Figure 3.2: An example of the linear symmetric quantization where the step size s defines the quantization thresholds. Here the quantization outputs could be encoded by a 3-bit representation.

Figure 3.2 typically depicts an element-wise linear symmetric quantization, in which the thresholds are derived from a unique step size s. The calibration of this scaling factor plays a key role and we state that it is likely intractable to find the optimal a priori value, given that it deeply depends on the model topology, its initialization, the inference task and the training procedure. Therefore, using an adjustable scaling factor during the training has demonstrated to be more favorable because of taking into consideration the evolution of weight/activation layer-wise distributions. Existing methods on optimizing these parameters can be divided into three groups: quantization error minimization methods, task loss gradient-based methods, and statistics-based methods.

3.2.1 Quantization error minimization methods

Ternarization with only 3 output values can be considered as the extreme case of the linear symmetric quantization. Ternary Weight Network (TWN [197]) finds the ternary mapping that aims at

minimizing the mean squared error between the floating-point weights and their ternarization. This allows them to approximate the optimal threshold (*i.e.*, $\frac{s}{2}$) in the case of a symmetric ternarization:

$$\frac{s}{2} = 0.7\frac{\sum_{i=1}^{n}|W_i|}{n} \tag{3.1}$$

where the term $\frac{\sum_{i=1}^{n}|W_i|}{n}$ computes the mean absolute value of the full-precision proxy weights of the layer. Similarly, [198] seeks to optimize the step size s of a general k-bit quantization under the minimization of the quantization error and proposes an iterative approach to obtain the filter-wise step size. In the same vein, [25] generates a simulated gradient $\nabla_s = -\alpha^2 d$ to find the near-optimal s, where the descent direction d is determined by comparing the quantization errors at s (middle point, $d = 0$), $\frac{s}{2}$ (left point, $d = -1$) and $2s$ (right, $d = 1$). However, even when the quantization error is minimized, it is difficult to ensure that the output of a deep, complex model with cascade of quantized layers is still coherent with that of the full-precision original counterpart.

3.2.2 Task loss gradient-based methods

Learning the quantization mapping under the task loss minimization perspective is a recently promoted approach. Back to the case of ternarization, [199] presents the ternarization under a residual framework in which the proxy weights and the step size are jointly learned under the task loss backpropagation. Furthermore, in a general k-bit linear symmetric quantization, [200] derives an STE gradient of step size with respect to the loss. Based on this work, [201] and [202] take advantage of bitwidth-dependent regularizations to optimize the layer-wise bit allocation given a target model size or a computational budget. Although learning the step size to directly optimize the task loss seem to be the most reasonable choice, the coarse gradient obtained might likely result in a sub-optimal performance and also an unstable training if its magnitude is not carefully controlled.

3.2.3 Statistics-based methods

The case of TWN shows an example of using the statistics of the full-precision proxy weights (in this case is the mean absolute value) to determine the value of the step size. [203] reviews the ternary quantization under the learnable framework, and sets the threshold using the maximum absolute value of proxy weights, *i.e.*, $\frac{s}{2} = 0.05 \times \max(|W|_i)$. Similarly, DoReFa [145] makes use of the maximum absolute value to normalize the proxy weights before the quantization.

In this chapter, we also leverage the statistics of the proxy weights to adjust the quantization mapping during training. Concretely, we claim that –in most cases– a proper low-bit quantization scheme should cover all the available data representation space, somehow maximizing the entropy of the weights [204]. Based on this hypothesis, [205] presents a 2-bit quantization methods for recurrent models where the step size equals to a constant multiple of the mean value of the proxy weights. Similarly, [206] determines the thresholds of 3-value and 4-value quantizations according to the mean and the standard deviation of the proxy weights. However, these approaches are not generic and applied only to under 3-bit quantization. On the contrary, our proposed method –Histogram-Equalized Quantization (HEQ)– automatically adjusts the step size of an n-value quantization according to its n-quantiles such that the resulting quantized values are more balanced, without any further regularization. We empirically show that our method provides a better accuracy than previous methods on different topology variants, from the baseline plain model to residual networks.

3.3 Linear symmetric quantization and the unbalanced histogram

This chapter focuses on the linear symmetric quantization to a restricted range of odd $n > 2$ discrete values. We consider the mapping $g : \mathbb{R} \to [[-1,+1]]$ applied to the weight w as:

$$g(w;s) = \frac{2}{n-1}\mathrm{Clip}\left(\left\lceil \frac{w}{s} \right\rfloor, \frac{1-n}{2}, \frac{n-1}{2}\right), \tag{3.2}$$

where $[[-1,+1]]$ is discretized with an output step size of $\frac{2}{n-1}$, s is the input step size and $\mathrm{Clip}(x;a,b) = \min(\max(x,a),b)$ with $a < b$. While existing works usually keep the range of floating values by the factor s outside of the clipping function, here we use the $\frac{2}{n-1}$ scale factor so that all the quantized values are explicitely shrunk in the interval $[-1,+1]$. During backpropagation, we use the straight-through-estimated (STE [59]) gradient $\frac{\partial g}{\partial w} = 1_{\{|x|\leq 1\}}$ to update the proxy weights.

This formulation thus depends on the definition of s that deeply impacts on the model accuracy. Let us consider Ternary Weight Networks (TWN) [197] as the baseline where $s = 2\tau\frac{\sum_{i=1}^{n}|W_i|}{n}$, with a fixed norm factor $\tau = 0.7$. Although the optimal s may change depending on the data distribution, this method cannot be applied to higher precisions and the predefined τ limits the adaptability of the quantizer. Similarly, DoReFa [145] forces the real values into the range of $[-1,+1]$ by a mapping adapted to the data, but the thresholds remain fixed. Fig. 3.3 depicts the histogram of proxy weights (blue bars) and quantized weights (horizontal green lines) in the case of 3-value (Figs. 3.3c, 3.3e) and 5-value (Figs. 3.3d, 3.3f) quantization whose initialization (from full-precision model) is shown in Figs. 3.3a and 3.3b, respectively. We can observe that in both cases of TWN (Fig. 3.3c) and DoReFa (Fig. 3.3d), the proxy weights are mainly concentrated around zero and the distributions between thresholds (vertical green lines) are unbalanced. In particular, the quinary weights (3-bit) in Fig. 3.3d can be approximated by only 3 values (2-bit). Consequently, the quantized weights fail to exploit all available values which may cause the model to be sub-optimal. This motivates the use of a proper quantizer favoring the balance of quantized weights.

3.4 Histogram-Equalized Quantization (HEQ)

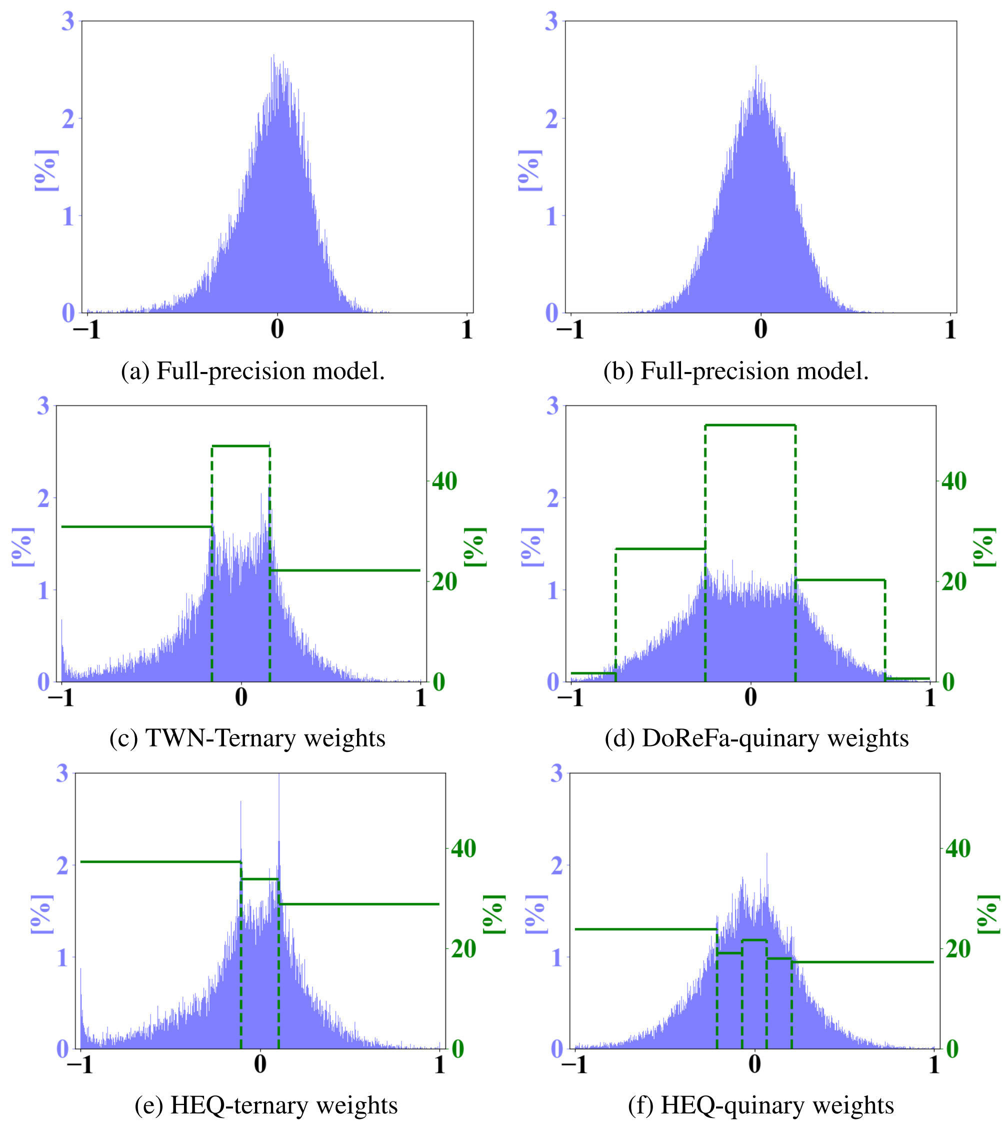
To resolve the aforementioned unbalance between quantized values, we propose HEQ to automatically adjust s during training. Assuming that a properly scaled quantizer should optimize the balanced use of available discrete values in the data representation space, we iteratively tune s based on the histogram of the proxy weights to equi-distribute quantized weights.

In Fig. 3.4 we denote $\{(q_i, q_{-i})\}_{i\in[[1,\frac{n-1}{2}]]}$ as $n-1$ points which divide the histogram of weights into n intervals with equiprobabilities (namely n-quantiles). Observing that the weights distribution may change during the training procedure but with a median value that usually stays around zero, we assume that these quantiles are symmetrically distributed around zero, *i.e.* $q_{-i} \approx -q_i$ with $q_i > 0$ or $|q_i| = q_i$.

In order to equalize the histogram bins of quantized values, we thus re-estimate and update s at the beginning of each epoch, such that the resulting thresholds used by the quantization function (see Fig. 3.4) are getting closer to these quantiles. Therefore, s can be approximated by a weighted sum of the quantiles such that q_i approximately coincides with $\frac{(2i-1)s}{2}$. Concretely, we assume that the sum of the absolute value of n-quantiles is approximately equal to that of the thresholds:

$$\sum_{i=1}^{\frac{n-1}{2}}(|q_{-i}| + q_i) = 2\sum_{i=1}^{\frac{n-1}{2}}\frac{(2i-1)s}{2}, \tag{3.3}$$

from which we can derive the following updating formula:



(a) Full-precision model.
(b) Full-precision model.
(c) TWN-Ternary weights
(d) DoReFa-quinary weights
(e) HEQ-ternary weights
(f) HEQ-quinary weights

Figure 3.3: Weight distributions of 2 layers (in 2 columns) after training of: full-precision model (1st line), existing ternary-weight and quinary-weight model (2nd line), and our proposed HEQ method (3rd line) along with quantization thresholds.

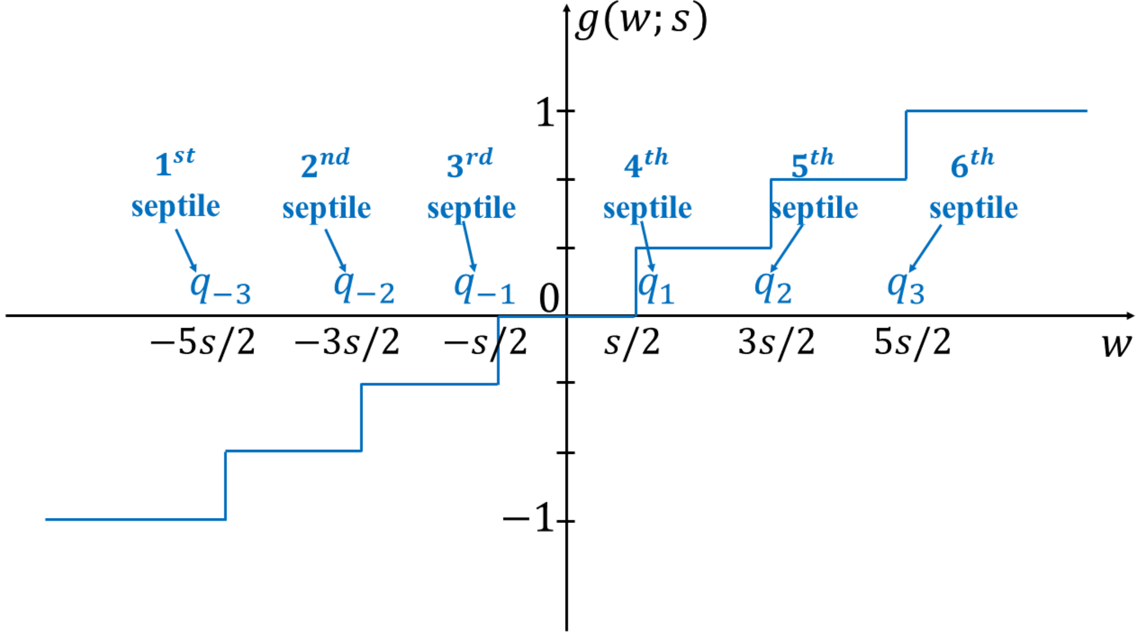


Figure 3.4: Symmetric linear quantization with histogram bin equalization when n-quantiles (q_{-i}, q_i) are symmetrical and coincide with the quantized thresholds.

$$s = \frac{4 \sum_{i=1}^{\frac{n-1}{2}} (|q_{-i}| + q_i)}{(n-1)^2}. \tag{3.4}$$

This approach has the great advantage of being generic, compatible with almost all use cases regardless the quantization level, the position of the layer and its type (with a possible extension to an even n). To maintain the stability of the model during optimization, we compute and update s only at the beginning of each epoch. The formal training procedure is detailed in Algorithm 1. The proxy weight distributions obtained after a training stage that are reported in Figs. 3.3e and 3.3f clearly demonstrate that HEQ provides a more balanced distribution of quantized weights.

Algorithm 1 Training QNN with Histogram-Equalized Quantization (HEQ)

Input: Initial proxy weights $\{\mathbf{W}_l\}_{l=1}^{L}$ and training dataset
Output: Optimized $\{\mathbf{W}_l\}_{l=1}^{L}$, $\{s_l\}_{l=1}^{L}$

// B, I, L: #batches, #epochs, #layers
// $\mathbf{W}_l$: full-precision proxy weights of the lth layer,
// s_l: quantization step size used at the lth layer.

```
1:  for i = 1 to I do
2:    for l = 1 to L do
3:      Find n-quantiles of layer l
4:      Compute and update s_l (Eq. 3.4)
5:    end for
6:    for b = 1 to B do
7:      Forward pass using {g(W_l; s_l)}_{l=1}^L (Eq. 3.2)
8:      Backward pass and update {W_l}_{l=1}^L
9:    end for
10: end for
```

3.5 Experiments

HEQ has been evaluated on the CIFAR-10 dataset [73] with 32×32 RGB images and using the VGG-Small model like in [196]. A combination of a scale-invariant random crop (performed on all-sided 4-pixel padded images) combined with a random horizontal flip is used for data augmentation. Initial proxy weights are from a pre-trained full-precision network. Motivated by Hardware considerations, a 2-bit activation scheme as detailed in DoReFa [145] has been used. Our model is trained during 100 epochs with a small batch size of 50 to favor exploration. The learning rate is set to 10^{-3} during the first 50 epochs, then exponentially rescaled by a factor of 0.9 at each epoch. Finally, a very last epoch with a larger batch size of 100 and a smaller learning rate of 10^{-5} is performed for fine-tuning. Fig. 3.5 allows a comparison between TWN [197] and our HEQ method with respect to the resulting weight distributions. While the zero values dominate all layers in the case of TWN, our method reduces the variance and limits the number of weights at 0 to nearly $1/3$ as shown in Fig. 3.5. The number of -1 values slightly dominates as more proxy weights are concentrated on the negative side. The effect of HEQ on increasing the information-carrying capacity of the quantized weights is clearly demonstrated in Figure 3.6, with nearly maximized entropy values in the case of HEQ-ternary and HEQ-quinary. On the other hand, quantized weights trained by TWN and especially DoReFa-quinary have significant lower entropy. Fig. 3.7 depicts the variation of s during training in both ternary and quinary cases. It shows that the evolution of s depends on the layer and has different convergence values.

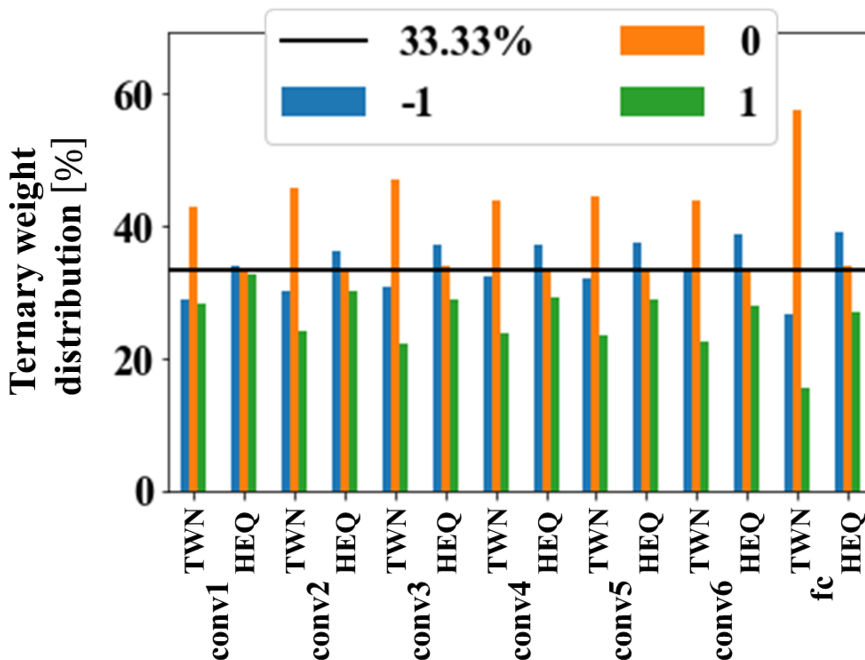
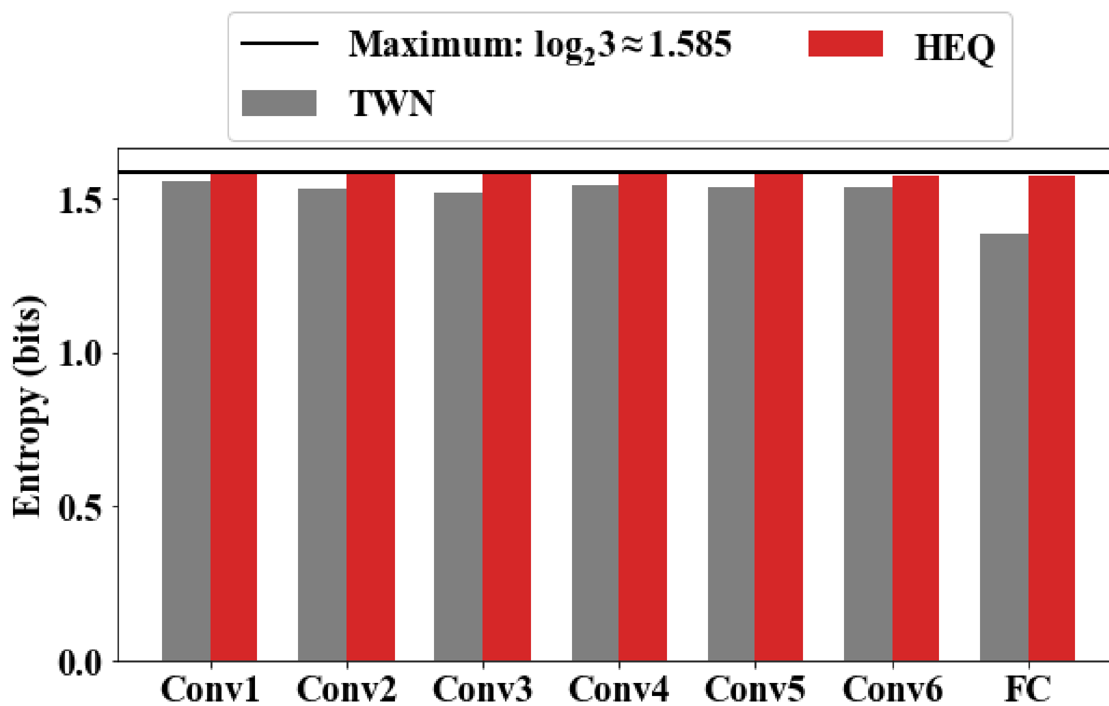
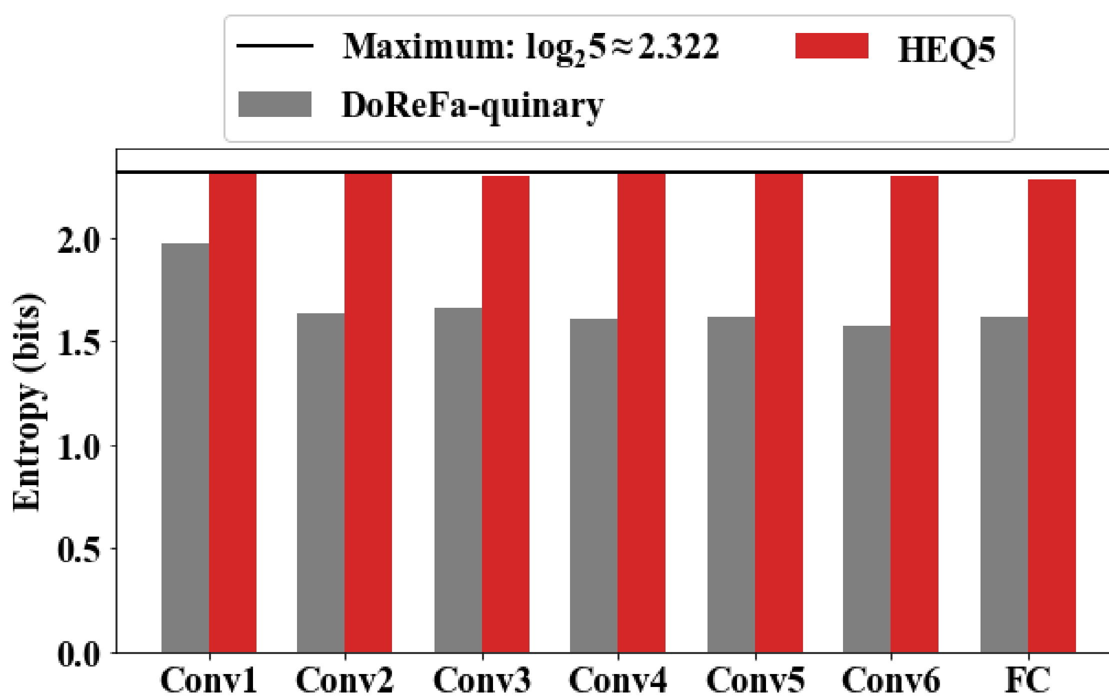


Figure 3.5: Comparison of the ternary-weight distribution using TWN and our HEQ method.

Table 3.1 reporting the average accuracy of each configuration over 5 realizations, demonstrates the competitiveness of HEQ compared to the state-of-the-art quantization methods. For instance, when quantizing both weights (W) and activations (A) into 2-bit, we obtain 93.51% accuracy while having only 3 values $\{-1, 0, +1\}$ over 4 values possible like LQ [196] and LLSQ [25]. Compared to the full-precision model, we observe mostly no degradation in the case of quinary weights ($n = 5$) and even a gain with septenary weights ($n = 7$). Moreover, while other works give rise to a full-precision scaling factor besides the integer weights which demands the fusion into Batch Normalization [207] (BN) for later hardware implementation, our models trained with HEQ-ternary and HEQ-quinary obtain directly integer values $0, \pm 1$ (logical operations) and ± 0.5 (bitshifts) which is already compatible for an easy hardware deployment.

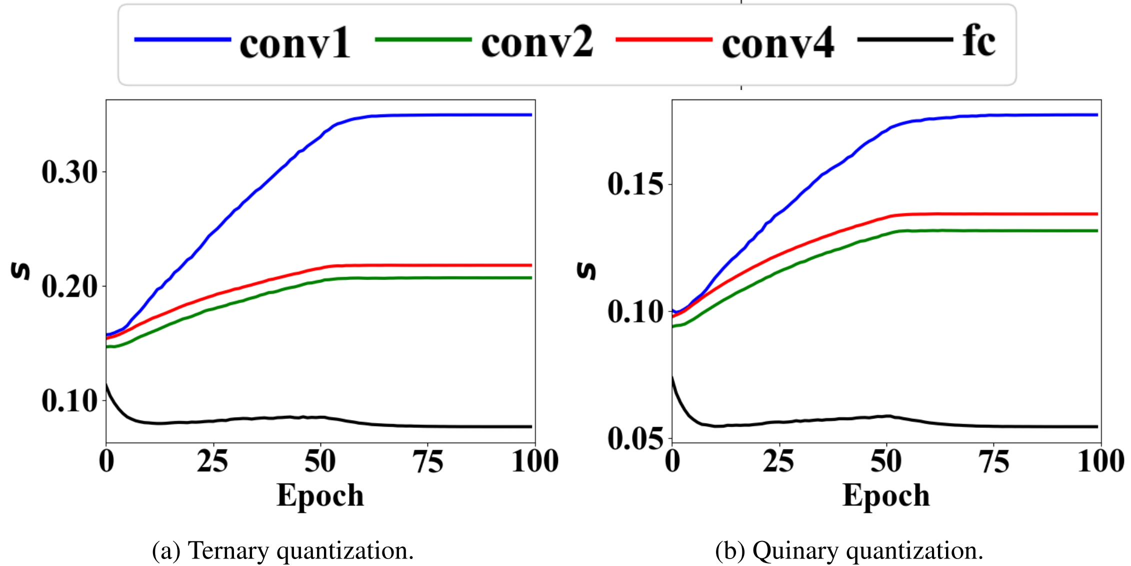


(a) Ternary quantization.



(b) Quinary quantization.

Figure 3.6: Information entropy of the quantized weights of different layers in VGG-Small on CIFAR-10. The entropy reaches its maximum value when all the quantized levels have equi-probabilities.



(a) Ternary quantization.

(b) Quinary quantization.

Figure 3.7: Evolution of the step size s during training.

Table 3.1: Comparison with the state-of-the-art low-precision quantization methods on CIFAR-10.

Method	HW-Compatibility	Bitwidth W/A	Accuracy(%)
TWN [197]	+	2/32	92.56
STTN [208]	+	2/2	92.93
TRQ [199]	+	2/2	91.2
LQ [196]	-	2/32	93.8
		2/2	93.50
LLSQ [25]	+	2/2	93.31
FP32 baseline		32/32	93.68
HEQ-ternary	++	2/2	93.51
HEQ-quinary	++	3/2	93.66
HEQ-septenary	+	3/2	93.75

3.6 Conclusion and Perspective

In this chapter, we presented a novel approach for quantizing weights in DNNs which is termed as HEQ. This is an adaptive QAT method which allows to adjust the linear symmetric quantization mappings according to the statistics of the proxy weights. During the training process, the step size is adjusted as a linear combination of the n-quantiles of the proxy weights, such that the quantized levels are approximately equalized. From an information theory point of view, HEQ seeks to maximize the information entropy of the quantized weights. The advantages of HEQ mainly rely on its simplicity and genericity which can be applied to different quantization levels. Besides, the process of adjusting the step size of HEQ is more training-efficient, as it can be done at the beginning of each epoch, rather then at every backpropagation iteration like existing works.

We empirically show that the models trained with our HEQ can achieve state-of-the-art accuracy on different datasets at low bitwidth, while being more hardware-compliant compared to previous methods. Moreover, throughout this thesis, we employ HEQ to train the quantized models with quinary and ternary weights in Chapter 4 and Chapter 5. The results obtained on these chapters also demonstrate the effectiveness of HEQ as a generic quantization approach that can be applied to different types of DNN architectures.

From a more general point of view, it is noteworthy mentioning that we can seek other distributions for the quantized values. In some cases, we can use another regularization to obtain non-uniform quantized distribution of a general (linear or nonlinear) quantization mapping. One straightforward example is the quantization of ReLU activations where the balanced distribution between activation values may harden the learning of most discriminant features. In that case, a long-tailed distribution dominated by zero values is more likely suitable. Back to the case of weight quantization, since the trained full-precision weights are usually assumed to follow gaussian distribution, we may also seek a Gaussian quantized histogram in the means of minimizing quantization error. Moreover, since most of the proxy weights have small magnitude, we can adopt a mixed-precision quantization scheme which coarsely quantizes near-zero values by low bitwidth, but finely represents high-magnitude values with more bits. This can be done by a learnable entropy coding, or a pre-determined low-bit representation for the sake of hardware compatibility. Those aforementioned extensions may be addressed in our future works.

4

Mixed-precision Deep Neural Networks for image classification and patch-based compression

Even if ASAs have proven to be a relevant choice for integrating inference at the edge, they are often limited in terms of versatility. In this chapter, we demonstrate that an application-specific NN accelerator dedicated to image processing can be applied to multiple tasks of different levels: image classification and compression, while requiring a very limited hardware. The key component is a reconfigurable, mixed-precision encoder that are properly designed leveraging hardware-compliant quantization and pruning techniques.

Contents

4.1 ASIC designs: energy efficiency versus flexibility 64
4.2 Related works 65
4.2.1 Model quantization 67
4.2.2 Alternatives to the Fully Connected layer (CNN bottleneck) 68
4.2.3 Autoencoder for patch-based image compression 68
4.3 Nonlinear quantized encoder 69
4.3.1 Half-Wave Most-Significant-Bit (HWMSB) activation 70
4.3.2 Layer-shared BitShift-based Normalization (BSN) 71
4.3.3 Pre-defined pruning with Group-wise Convolution (GC) 72
4.3.4 Compression of the bottleneck dense layer 73
4.4 Image reconstruction from patch-based quantized measurements with PURENET 73
4.5 Simulation results 75
4.5.1 Image classification on CIFAR-10 75
4.5.2 Comparison with prior works 76
4.5.3 Full-frame image compression 79
4.6 Conclusion and Perspective 81

4.1 ASIC designs: energy efficiency versus flexibility

In section 2.2 we have overviewed different classes of DNN accelerators: GPA, AIDA, and ASA. Indeed, the choice of DNN inference platform deeply depends on the targeted applications (level of versatility and programmability), the available power consumption of the system, the complexity of the inference task and its required quality of service (*e.g.* level of accuracy). While GPAs such as CPU, GPU or FPGA can serve for different type of computations, AIDA and ASA platforms are specifically designed for DNN processing and more related to ASIC design. However, these two classes still differ from each other in terms of design patterns, scalability and power consumption level.

AIDAs are relatively generic hardware platforms dedicated to multi-purpose DNNs with the highest TOPS/W as the main Figure of Merit. Recent works improve the performance and energy efficiency of AIDAs by focusing on three main key points: reconfigurability, sparsity and weights bitwidth. Within the scope of a reconfigurable platform, we can cite [38], a deep neural architecture enabling highly reconfigurable patterns while targeting a wide range of models with a limited power budget of 479mW. Some other works also introduce specific architectures that limit data movement in CNN. [6] proposed Eyeriss - a spatial architecture with row-stationary dataflow that takes advantage of local data reuse mechanism even exhibiting a lower overall chip power consumption of 278mW. On the other hand, [39] handles sparsity with binary maps and arrays for nonzero values, skipping null activations to reach a power consumption of only 155mW. Variable bitwidth computations allow good trade-offs between accuracy and memory/power budgets as depicted in [121] that presents a DNN accelerator supporting variable weights bitwidth precision from 1 to 16 bits, with a power consumption ranging from 3.2mW to 297mW (depending on the master clock frequency and power supply). [209] proposed a computing-in-memory neural network processor efficiently dealing with sparsity and weight storage on SRAM. [191] presents a resource-efficient architecture for BNN inference accelerator, processing blocks in an output-oriented manner while skipping redundant operations.

While AIDA platforms can be used to deployed different DNN architectures for different applications, ASAs are dedicated to only certain tasks, considering both the hardware specifications and the algorithm perspectives proposing co-optimized designs. This promotes compact network topology design with improved power consumption efficiency while capping algorithmic loss of accuracy. For instance, [210] proposed an accelerator optimized for binary-weights CNN with a power consumption of 895µW. Using quantization technique in both analog and digital domain, Kim et al. [126] proposed an always-on face recognition processor integrated with a CMOS image sensor (CIS) consuming about 620µJ per inference. [211] even proposed a sub-10µW QQVGA imager enabling both motion detection with background estimation and face recognition using XOR-based edge extraction combined with a SVM. [212] proposed a QQVGA imager which can operate on convolution mode with ternary-weighted filters and Haar-like detection mode, under less than 206µW. [27] presented a CNN-based accelerator that can serve for both face detection and facial landmarks localization. More recently, [190] presented a digital Binary Neural Network (BNN) chip achieving a power consumption of 5.6mW. Low-precision CNN processors were also applied to real-time object detection task in [213] and [214]. In particular, [214] adopted a mixed data flow for each layer along with an intra-layer mixed weights precision quantization, in which the weights were decomposed to a dense binary kernel and a sparse 8-bit kernel in order to improve the accuracy/compression ratio trade-offs.

Those aforementioned examples of AIDAs and ASAs has demonstrated the key difference between these two classes in terms of power consumption at the expense of application-versatility. By sacrificing the flexibility to target only specific tasks with fine-grained hardware-algoritm co-

design, ASAs can advantageously limit the power consumption in the order of 1mW or even below. However, the downside of ASAs is that they only support a certain task or some correlated tasks such as face detection and facial landmarks localization. If the applications require different levels of feature processing, *e.g.*, classification and compression, it is difficult to design a single model topology that can successfully handle both of them within resource-constrained budget.

Motivations:
Taking into consideration ASIC design issues, the work in this chapter has a dual purpose:

- increase the application-versatility of an Image and Signal Processor, dedicated to classification and compression;
- improve the hardware efficiency and algorithmic performances trade-off (*i.e.* inference and reconstruction accuracy).

To this end, we propose a hardware-compliant mixed-precision encoder and its decoder counterpart. The main advantage of the proposed encoder topology is that it can be declined for both HW-light embedded inference (Figure 4.1, a) path) and image compression tasks (Figure 4.1, b) path) with proper weights trained separately for each application. Indeed, for each task, the mixed-precision quantized encoder will load the corresponding weights (thanks to its **reconfigurability**) to process and output discriminant patterns as a latent binary representation of the data. From this binary coding, we can thus either use directly a classifier to perform the embedded inference or a remote decoder network (PURENET) for image recovery (see Figure 4.2).

Contributions:

1. We introduce a mixed-precision encoder design with reconfigurability that may serve for both image compression and classification. It is noteworthy to mention here the proposed HEQ framework (*c.f.*, chapter 3) is applied it to the quinary and ternary weights in the encoder. Besides, the Batch Normalization obstacle is successfully replaced by a layer-shared Bit-Shift operation. We also propose a Half-wave Most-Significant-Bit (HWMSB) function for 2-bit activation with favorable hardware compatibility.
2. We propose a novel decoder called PURENET that takes as input the patch-based binary measurements from the quantized encoder. To our knowledge, this work presents one of the first low-precision quantized encoder for patch-based image compression. Our experiments show that image decompression can be performed without block artifacts at low bitrate.

The rest of this chapter is divided as follows: Section 4.2 presents related works, Section 4.3 details our nonlinear quantized encoder design, including details of proposed algorithmic enablers. Section 4.4 describes the network topology of the Decoder for image reconstruction. Finally, Section 4.5 presents simulation results on both image classification and image compression tasks.

4.2 Related works

Weights and activations quantization, connectivity pruning and alternatives to the Fully Connected layer bottleneck are usually used as common techniques to facilitate the implementation of an Artificial Neural Network in terms of hardware mapping. For the sake of avoiding redundancy (*c.f.*, see 2.3), here we will only discuss about specific problematic regarding mixed-precision topologies, the crucial but hardware-unfriendly use of BN in QNNs, and the alternatives to the FC layer bottleneck. Apart from algorithmic enablers for inference, the last part of this section depicts previous works on image reconstruction from patch-based compression.

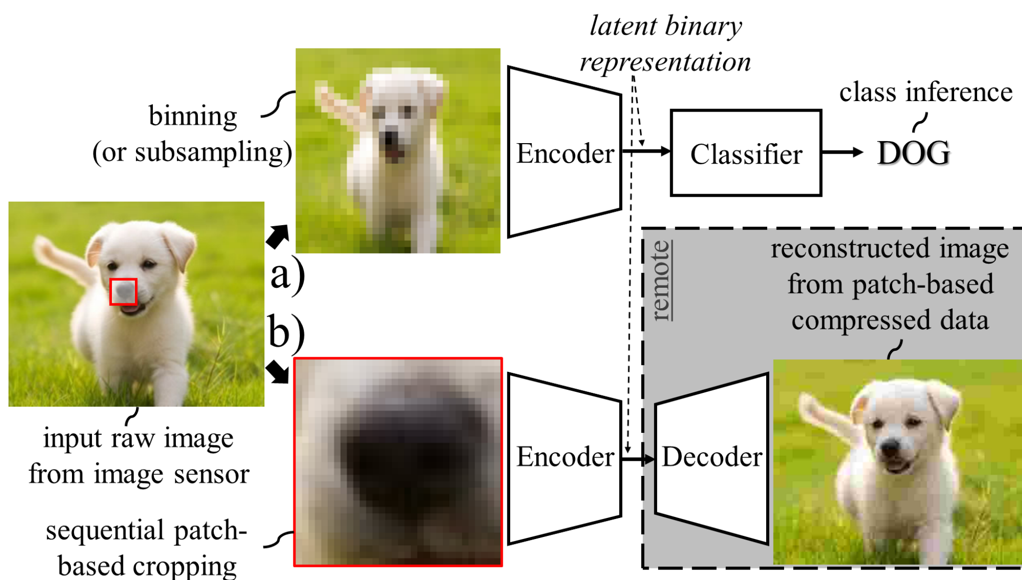


Figure 4.1: The joint framework for both image classification and image embedded compression and remote decompression.

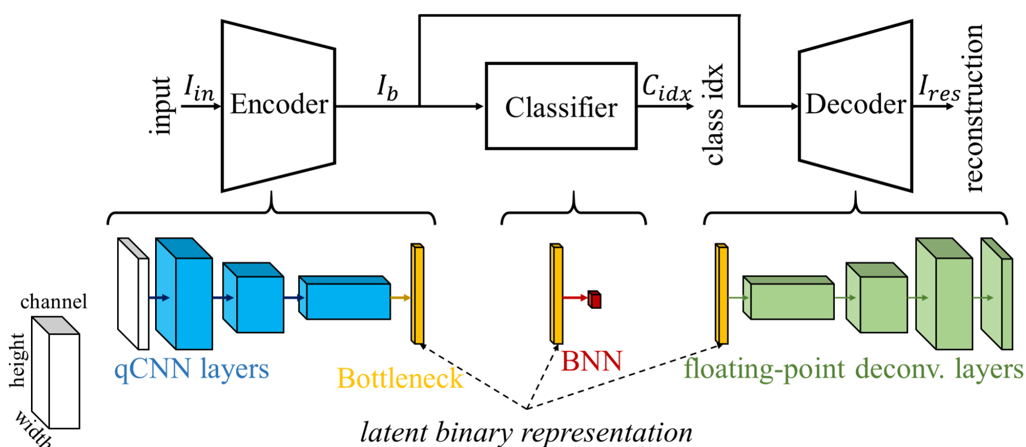


Figure 4.2: Schematic description of our framework involving neural network topology parts.

4.2.1 Model quantization

Mixed-precision DNNs

Existing works usually adopt a fixed-precision strategy to all layers in the DNNs, either to simplify the design space, or to easily evaluate the proposed quantization methods compared to state-of-the-art. However, the role of every layer in the model changes accordingly to their positions as well as the model architecture and the targeted task itself. Therefore, it is more reasonable to adopt a mixed-precision topologies to DNNs which allows to prioritize important layers at higher precision. This way, the design space of mixed-precision DNNs falls between full-precision 32-bit networks and binarized networks, where the optimal topology enables the best trade-offs between hardware efficiency and algorithmic performance. The mixed-precision topology has different levels of granularity: filter-wise [160], *i.e.*, each filter in a layer has a different bit-width, or layer-wise [215] where the precision changes from one layer to another.

Assigning the optimal bit-width for every components in DNN is basically a searching problem. As a result, the optimal mixed-precision strategy can be found under the framework of NAS based on Reinforcement Learning approach [215], [216], or DARTS approach [160], [201]. Besides, [217] propose a Hessian-based framework to measure the sensitivity of layers to the quantization, enabling the bit-width selection for each layer in DNNs. Although obtaining promising results, these methods introduces additional shortcomings in terms of training cost and computational overheads.

In our work, we also have a mixed-precision approach in order to improve the harware/algorithm trade-offs. However, for the sake of simplicity, the mixed-precision topology presented in this chapter still relies on a hand-crafted design which manually assigns higher precision to weights and the activations of the first layers, as they are crucial to extract meaningful information from the data.

Batch Normalization in Quantized Neural Networks (QNNs)

An important issue about QNNs is that they generally fail to properly converge without Batch Normalization (BN) [218]. However this property becomes an obstacle for efficiently deploying QNNs in embedded systems, because the affine transform of BN at inference stage is still an expensive operation from the hardware point of view. Due to the crucial role of BN during the training process, there is no surprise that previous works either kept BN in full precision or avoided to use it. [219] proposed to fuse the batch normalization parameters into the kernel and bias of BNNs and perform an Addition with the 9b equivalent bias. Riptide [220] even approximated the variance of the BN and the scale terms of XNOR-Nets by bitshift operations, and combined them to embed both the magnitudes of weights and normalizations. Another scheme is Sign Comparison which absorbs the BN into the Sign function by an equivalent comparison between the pre-activation and the bias obtained after training [221], [222]. However, both of these two approaches still kept the presence of the bias which surely introduced an additional module in terms of hardware implementation. Besides, these specific schemes can only be applied in the case of activation binarization, otherwise in the case where BN is followed by a multi-bit quantization, they are no more applicable.

In this work, we propose a BitShift-based Normalization (BSN) that approximates the affine transform of the whole BN layer at testing time by a single power-of-2 rescaling, without the introduction of additional biases. Due to the crucial role of BN in the training of QNNs, we still leverage BN to train the quantized model from scratch, then replace all BN layers by BSNs before fine-tuning the model to recover the performance degradation.

4.2.2 Alternatives to the Fully Connected layer (CNN bottleneck)

Traditional deep CNN architectures consist of a convolutional block followed by one or many Fully Connected (FC) layers to perform the classification. Although it has shown the success on various datasets, these FC layers, in particular the very first FC layer (CNN bottleneck), often hold a huge percentage of model's parameters. For resource-constrained applications, this becomes a main limitation for the CNN deployment. Besides, these FC layers are also a factor that is prone to overfitting. There exists several methods replacing the first FC layer to achieve a better efficiency. Global average pooling is first introduced in [223] that outputs only one value per feature map. In the same perspective, [224] proposed a 3D convolution in which only local feature maps are combined to get the prediction for each class. Both these two approaches dramatically reduces the number of parameters while achieving a competitive performance compared to FC-based classifier. They all demonstrate that the FC layers exhibit redundancy and can be replaced with an acceptable loss of accuracy and generalization of the model. Indeed, both global average pooling and 3D convolutions are just a formulation of linear transformation with reduced support size compared to a dense support of FC. However, while the difference between these classifiers and FC-based classifiers is not significant in the context of full-precision models, there may be a clear gap in the specific case of quantized neural networks.

Another direction to reduce the number of parameters of FC layers is to use a fixed transform whose weight parameters are predefined. The deployment of such a model with fixed parameters is more suitable for devices with limited resources. [225] shows that any fixed orthogonal matrix can be used to efficiently replace a learnable FC. Even if it does not reduce the total number of operations, they demonstrate that a Hadamard matrix can improve the efficiency in terms of hardware implementation. Besides, the use of pseudo-random deterministic projections can also be a good alternative because they preserve the linear separability while being hardware-implementable, this without additional memory needs [226].

In this work, we propose an alternative to the first FC layer with input data of shape $H \times W \times C$ and output of C hidden units. Our idea basically consists in using a depthwise convolution (DW-Conv) of kernel size $H \times W$ followed by an FC layer without activation in-between. Later experiments show that this approach achieves highly competitive results.

4.2.3 Autoencoder for patch-based image compression

Patch-based (or block-based) encoding is preferred to its full-resolution counterpart, as it reduces the processing complexity and storage-related costs. Besides standards such as JPEG [227], there exists a lot of other patch-based compression schemes, from dimensionality reduction using block-based Compressed Sensing (BCS) [228], [229] to deep learning-based approaches [230]. In the BCS framework, the high-dimensional image is divided into non-overlapping patches which are then projected separately onto a low-dimensional space via a common projection matrix, therefore we can consider it as a linear encoding scheme. The reconstruction of the entire image from these patch-based measurements can be done using iterative algorithms [231], [232] or recent learning-based methods [233], [234]. On the other hand, neural network-based algorithms have been used as an alternative approach for image compression. Toderici *et al.* firstly introduced a Long Short-Term Memory (LSTM)-based autoencoder [235] enabling a patch-based 32×32 thumbnails compression at variable compression rates, which encodes the residual error between the current reconstruction and the original image at each iteration. This framework was then developed and applied to full-resolution image compression [236] with the intensive use of an LSTM-based entropy coder to capture long-term dependencies of patches. Having the same perspective, [237] introduced a progressive encoding scheme exploiting dependencies between adjacent patches. Although achieving favorable results at shallow compression rates, these Recurrent Neural Network

(RNN)-based autoencoders are not suitable for being deployed in resource-constrained systems due to their complexity and hardware incompatibility. Also, these methods are not purely patch-based as they make use of spatial coherence between adjacent regions to –more efficiently– encode every patch and alleviate the block artifacts. [238] and [239] proposed deep learning image compression techniques while adapting the region-based bitrate accordingly to the local content of image. Note that even if such an adaptive bit allocation would improve the overall performances, it does not fit within our hardware-restricted and patch-by-patch compression framework. Ensemble learning is recently deployed in [240] which uses several networks of the same structure but different parameters, and a network is selected when encoding each image block before signaling to the decoder.

QNNs for image compression: Quantizing the weights and activations has been showing excellent results on semantic tasks such as image classification and object detection. However, representing the weights and activations using a low precision causes a considerable loss of information at pixel level, which prevents QNNs from achieving good performances for image compression or super resolution. This explains the absence of quantized models for these pixel-level tasks in the related state-of-the-art. We can only cite some works of [241] and [242] that are dedicated to super resolution. In the case of image compression, the role of pixel-level information is more crucial in encoding image data, hence there exists hardly no prior works on quantized models for this task. However, in this work, we use the same quantized encoder topology for patch-based image compression as the one designed for image classification. We show that even with such an encoder, the reconstruction is still guaranteed by the proposed PURENET decoder with almost no block artifacts at the equivalent bit-rate of 0.25 bits-per-pixel (bpp).

4.3 Nonlinear quantized encoder

Our proposed framework first targets inference for always-on embedded systems. To ease fair comparisons with state-of-the-art approaches, we use the basic but reproducible CIFAR-10 image classification dataset [73]. Even though a practical application may differ for final sizing of the topology, CIFAR-10 is generally deployed to benchmark both algorithm and hardware designs [219], [221], [191], [222], [243]. Based on literature reviews related to network quantization and the VGG model [81], we took the VGG-7 model for the sake of simplicity as the pivotal element for constructing our topology variants. Note that all the reported contributions here are compatible with other kinds of neural networks architectures such as ResNet [24], MobileNet [8], ...

Table 4.1: NQE topology with details on layer inputs and kernels precision.

<table>
<tr><th colspan="2">Layer</th><th>Input shape</th><th>Weight shape</th><th>Input precision</th><th>Weight precision</th></tr>
<tr><td colspan="2">Conv layer</td><td>$32\times32\times3$</td><td>$3\times3\times3\times F$</td><td>8</td><td>3</td></tr>
<tr><td colspan="2">Conv layer 2</td><td>$32\times32\times F$</td><td>$3\times3\times F\times F$</td><td>1</td><td>3</td></tr>
<tr><td colspan="2">Conv layer 3</td><td>$16\times16\times F$</td><td>$3\times3\times F\times2F$</td><td>2</td><td>2</td></tr>
<tr><td colspan="2">Conv layer 4</td><td>$16\times16\times2F$</td><td>$3\times3\times2F\times2F$</td><td>1</td><td>2</td></tr>
<tr><td colspan="2">Conv layer 5</td><td>$8\times8\times2F$</td><td>$3\times3\times2F\times4F$</td><td>2</td><td>1</td></tr>
<tr><td colspan="2">Group Conv ($G=4$)</td><td>$8\times8\times4F$</td><td>$3\times3\times F\times4F$</td><td>1</td><td>1</td></tr>
<tr><td rowspan="2">Bottleneck</td><td>Depthwise Conv</td><td>$4\times4\times4F$</td><td>$4\times4\times1\times4F$</td><td rowspan="2">1</td><td rowspan="2">1</td></tr>
<tr><td>FC</td><td>$1\times1\times4F$</td><td>$4F\times4F$</td></tr>
<tr><td colspan="2">FC</td><td>4F</td><td>$4F\times10$</td><td>1</td><td>1</td></tr>
</table>

Figure 4.3 depicts our hand-crafted mixed-precision network topology integrating all proposed algorithmic enablers for the image classification task. The model is divided into 5 main modules,

with 3 Convolution blocks, 1 Bottleneck layer and 1 output Classifier. In this specific setting (*i.e.* the baseline model), a multi-bit quantization scheme for both activations and weights is already applied to the first two convolutional modules, *i.e.* Quinary weights Quantization (QQ) for the first, Ternary weights Quantization (TQ) for the second and Binary weights Quantization (BQ) [12] for all the rest. Since the first conv layers are crucial to extract meaningful features and preserve core information from input data, therefore more bits for them leads the network to better performance trade-offs. We notice that the QQ and TQ are obtained using the proposed HEQ framework (*c.f.*, chapter 3) with $n = 5$ and $n = 3$, respectively. Note that the very first conv layer additionally embeds channel-wise biases to properly enable image dynamics feature extraction. We also put HWMSB activations (*c.f.*, Section 4.3.1) at the end of the first two convolutional blocks and the heaviside activation at the end of the third block, as having zero values –instead of signed ones– helps learning to discriminate better data features. The first non-convolutional layer is the bottleneck which holds the half of model's parameters in its baseline format (*i.e.* a FC layer). This first FC is thus replaced by a depthwise convolution followed by a far smaller FC (*c.f.*, Section 4.3.4). These first fours blocks constitute together what is denoted a Nonlinear Quantized Encoder (NQE). The final block as for it, is a classifier composed of one single FC. From a top-level system view, the NQE will perform the image compression over non-overlapped 32×32 patches while in classification mode it is combined with the last-stage classifier. We also want to stress that applying 2×2 MaxPooling (MP2) on a binarized tensor results in a tensor with almost all ones, confusing the training procedure when choosing the argmax positions during backpropagation. However, applying MP2 over quantized values is more hardware-friendly than over full-precision values. In Figure 4.3, MP2 are thus put after HWMSB but before Heaviside keeping in mind that from the hardware point of view (*i.e.*only for feedforward pass) this last order can be reversed. Note that details of the layer weights and input configuration are described in Table 4.1.

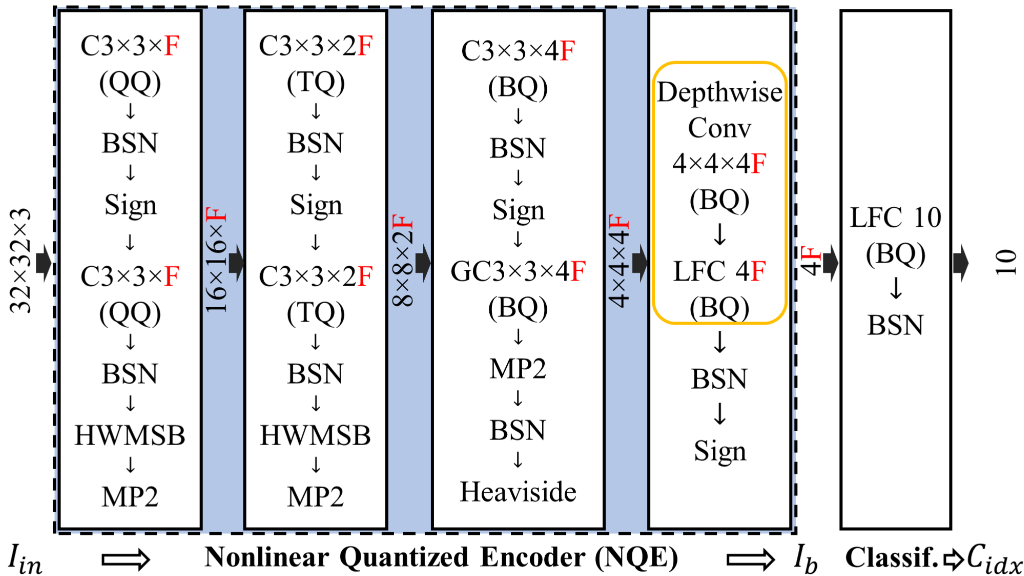


Figure 4.3: Topology of the Nonlinear Quantized Encoder (NQE) + Classifier. F in red stands for the hyperparameter corresponding to the size scale of the feature map (*i.e.* the number of the feature map of the first convolution module). GC stands for Group-wise Convolution of 4 groups.

4.3.1 Half-Wave Most-Significant-Bit (HWMSB) activation

To compensate additional hardware needs when using more bits for intermediate values, we propose to simply extract the position of the most significant bit of the income value. This operation advantageously embeds two wanted features, namely log_2 dynamic range compression and intrin-

sic requantization. Its original real-valued function mapped in the $[-1,+1]$ range can be defined as:

$$f(x) = \begin{cases} \text{sign}(x)\ \min\left(\frac{4+\log_2(|x|)}{3}, 1\right) & \text{if } x \geq \frac{1}{8}, \\ \frac{8x}{3} & \text{otherwise.} \end{cases} \quad (4.1)$$

The MSB quantization function and its Straight-Through-Estimated (STE [59]) gradient are then described as follow:

$$q(x) = \begin{cases} \text{sign}(x)\ \min\left(\frac{\lfloor 4+\log_2(|x|)\rfloor}{3}, 1\right) & \text{if } |x| \geq \frac{1}{8}, \\ 0 & \text{otherwise.} \end{cases} \quad (4.2)$$

$$\frac{\partial q(x)}{\partial x} = \begin{cases} \frac{1}{3|x|\ log2} & \text{if } \frac{1}{8} \leq |x| \leq 1, \\ \frac{8}{3} & \text{if } |x| < \frac{1}{8}. \end{cases} \quad (4.3)$$

When this MSB is followed by a ReLU activation, only positive values can be passed and negative values are zeroed out. We call this combination as Half-wave Most-Significant-Bit (HWMSB) activation. Unlike the MSB function, the HWMSB activation has only 4 possible output values $\{0, 1/3, 2/3, 1\}$, therefore it needs only 2 bits to represents the output data. Another advantage of HWMSB compared to MSB is that attenuating negative values improves the learning as ReLU does in several topologies. Table 4.2 reports the value mapping between the decimal, naive binary representations and the outputs. The first significant bit is assigned to the third bit on the right of the point. We call this the reference position, which is determined by the integer bias of 4. If the input is multiplied by a power-of-2 factor before the MSB, we can absorb this multiplication into the MSB by just shifting the reference position accordingly. Note that this quantization scheme differs from existing logarithmic quantizations such as [244] on these two points, first it has the integer bias of 4 and it also zeroes out negative values. Note that the normalization factor (here, 3 to keep the dynamic in the wanted range) can be assigned independently to this scheme.

Table 4.2: 2-bit HWMSB input-output mapping with naive binary representation (sign+magnitude)

Input		**Output**	
Decimal	***Binary***	***Decimal***	***Binary***
$x < 0.125$	x.000xx/ 1.xxxxx	0	0.00
$0.125 \leq x < 0.25$	0.001xx	1/3	0.01
$0.25 \leq x < 0.5$	0.01xxx	2/3	0.10
$x \geq 0.5$	0.1xxxx	3/3	0.11

4.3.2 Layer-shared BitShift-based Normalization (BSN)

BN keeps a crucial role to QNNs, especially BNNs, as these networks fail to well converge without a proper rescaling. However, BN is not tractable for Deep Learning in highly constrained embedded systems, since at inference time, it consists of one full-precision addition and multiplication per scalar, which is a computational-demanding operation. A quantized BN from scratch is definitely not as robust as the standard BN as it is difficult to estimate the appropriate scaling factor, and unfortunately involves a significantly lower network's accuracy (*e.g.* much larger than 1%).

BN replacement by a single BitShift: The straightforward option considered here is we still employ the standard BN to train the model from scratch and then simplify all BN affine transforms by a single BitShift approximation, with later retraining in order to update the weights accordingly. This two-step training procedure allows preserving its accuracy performances with a high simplification of the final hardware implementation. We thus approximate the scale constants of BN layers obtained after first training in a power-of-2 fashion, that advantageously corresponds to the bitshift operation. After the training stage, BN layer has properly estimated batch statistics μ and σ^2, respectively representing the moving mean and the moving variance. Let us recall that at the inference stage, the BN consists in processing the input x to provide the output y as follows:

$$y = \gamma \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}} + \beta \equiv \hat{\gamma}x + \hat{\beta} \tag{4.4}$$

where –using the same notations as in [218]– $\hat{\gamma} = \frac{\gamma}{\sqrt{\sigma^2+\varepsilon}}$ and $\hat{\beta} = \beta - \frac{\gamma\mu}{\sqrt{\sigma^2+\varepsilon}}$ are equivalent to a scale and an offset (*i.e.* channel-shared additional weight and bias if applied after 2D convolution, and unit-shared if applied after 1D Dense layer, therefore we have a vector of different $\hat{\gamma}$ for each BN). Concretely, in our framework, we choose the 0.9-quantile value from these scales at each layer, denoted as $\tilde{\gamma}$, to serve as the unique scale for all the BSNs, approximating it as follows:

$$y = 2^{\lfloor log_2|\tilde{\gamma}|\rfloor}x \tag{4.5}$$

Note that the experimentally chosen 0.9-quantile is considered as it is a good trade-off between the maximum value that may explode the dynamic range along with the gradient, and the minimum value that may slow down the gradient update of previous layers. After replacing all BN layers of the model by this single BSN transform, we train again the network one more time. Note that for all the BSNs, we can get rid of the bias $\hat{\beta}$ as it does not improve the inference in general, at the expense of additional computations for hardware mapping.

Hardware implementation of BSN: Generally, having only one bitshift-based scale replacing the whole BN layer considerably reduces the computational complexity in all cases, regardless its following quantization. While needing only 4 bits to store the bitshift scale, all input data will share a common bitshift operation, that is much cheaper than affine transforms of different scales and biases, even if theses parameters are quantized. Moreover, in the context of our model topology where we have either HWMSB, Sign or Heaviside quantization after the BSN, this becomes more advantageous. Clearly, the single bitshift keeps the sign of data unchanged, therefore, it has no impact to the results of the later Sign or Heaviside quantization. Consequently, the BSNs followed by Sign or Heaviside quantization do not need to be explicitly implemented. Similarly, we can also get rid of the final BSN as it does not change the order of the output layer's logit prediction. In addition, the bitshift scale of BSN can be intrinsically fused into the HWMSB by just shifting the reference position of the HWMSB accordingly.

4.3.3 Pre-defined pruning with Group-wise Convolution (GC)

Inspired by ShuffleNet [9] and justified by the advantages of a structurally pre-defined pruning, we propose to perform convolutions of CNN in a Group-wise manner instead of an all-channel-fully-connected conventional topology (see Figure 4.4). This group-wise pruning is only applied to the last convolutional layer of the NQE, as it contains most of the parameters. A Group-Convolution reduces the parameters and the number of MACs by a factor equal to the number of groups. Consequently, it also reduces the memory needed for intermediate values. Compared to unstructural pruning scheme, a predefined structural pruning like the Group Convolution may be embedded directly to the hardware platform, without the need of additional memory to save the connection positions. In our model, the number of groups is set to 4.

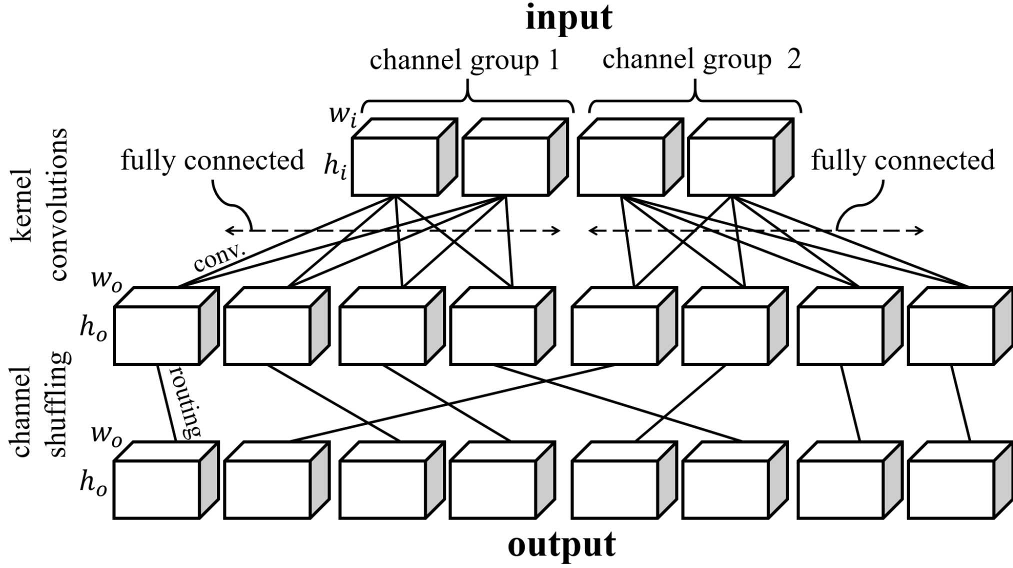


Figure 4.4: Example of Group-wise convolution with a 4-channel input divided into 2 groups and 8-channel output. The intermediate values are also divided into two groups, and each convolution is performed with a kernel that takes only two input channels from the corresponding group. These output channels are then structurally shuffled.

4.3.4 Compression of the bottleneck dense layer

In our NQE, the first affine transform performed after conv modules has an input of shape $4 \times 4 \times 4F$ and outputs $4F$ hidden units. In the case of a dense layer, it will contain $256F^2$, *i.e.* almost 50% of the model's parameters. Even though the weights are binary, it still requires a large percentage of the total memory needs. Therefore, replacing this bottleneck layer while preserving the model's performance is crucial for improving the overall efficiency. In this work, we propose to firstly use a depthwise convolution of kernel size 4×4, to transform the 3D tensor into a vector of length $4F$. Formally, this depthwise convolution is equivalent to a block-diagonally connected layer, with only $4 \times 4 \times 4F = 64F$ learnable parameters. Then we have a square learnable FC layer with size $4F$, therefore holding only $16F^2$ parameters. Since there are no activation between these two sub-layers, this approach is equivalent to decomposing the fully dense matrix into two sub-matrices, one for only spatial operation (depthwise convolution), the other for the combination of channels (dense layer). Consequently, the total number of parameters is $16F^2 + 64F$, which is very small compared to the initial $256F^2$ parameters of the FC version of the bottleneck. In Figure 4.3, these two layers are surrounded by a yellow rounded rectangle, denoting that they form together an alternative to the bottleneck dense layer.

4.4 Image reconstruction from patch-based quantized measurements with PURENET

Figure 4.5 shows the proposed combination of an image patch-based encoder with a full-frame decoder. The large image is first divided into small patches of size 32×32 and then processed by the NQE to obtain a binary representation for each patch independently. The Patch-based Upsampling (PU) module will learn to increase the spatial resolution of these codes from 1×1 (vector) to 16×16 (*i.e.* half of the final full-resolution). These patches are then aggregated together to form proxy feature maps at $\frac{1}{2} \times \frac{1}{2}$ full-resolution, which are then processed by the Refinement module to obtain the final reconstruction. The term PURENET hereafter denotes for the Patch-based Up-

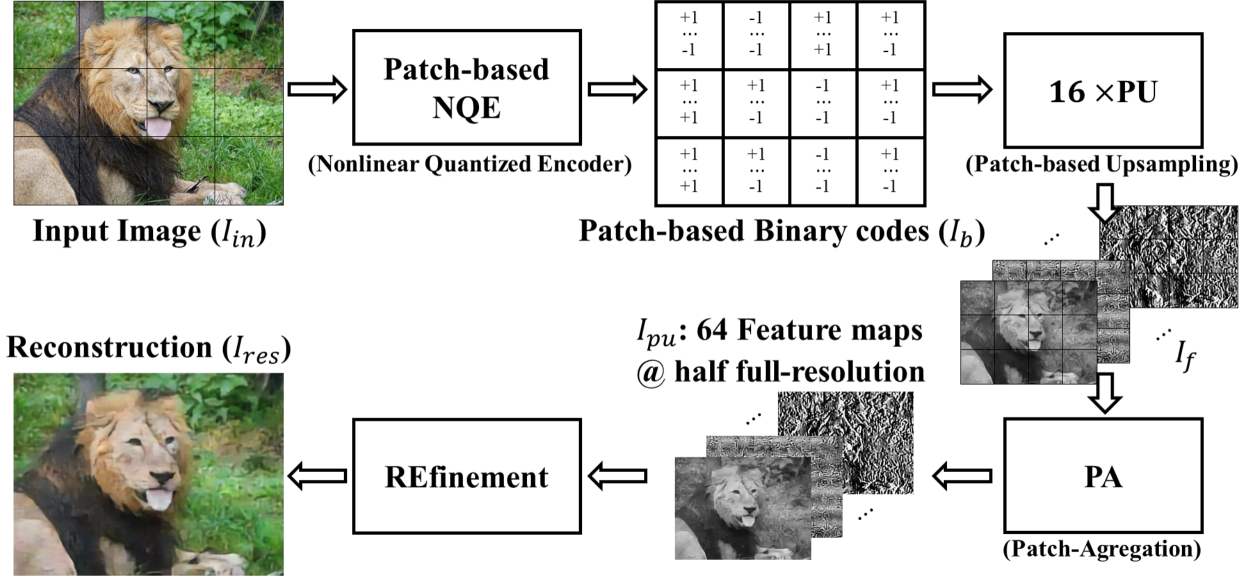


Figure 4.5: The patch-based encoding with Full-Resolution PURENET decoder. Compressed binary codes are vector of length 256, corresponding to the bitrate of 0.25bpp. Note that without the Patches Aggregation (PA), this topology may still be applied to the decoding of patches independently. This variant without PA is denoted as patch-independent PURENET (PI-PURENET).

sampling and REfinement NETwork.

Upsampling module: In details, the Upsampling model contains 4 blocks of Transpose Convolution + BN + ReLU (CBR). Each Transpose Convolution (ConvT) is of kernel size 3×3 and strides of 2. Figure 4.6 depicts the topology of this module. In PURENET, the Upsampling module is independently applied to every patch, so that the information of each patch is properly preserved and do not mix with the neighborhood. We observe that aggregating the patches before the final 2×2 upsampling is an appropriate trade-off between alleviating the block artifacts and limiting the color errors due to the mixture of patches.

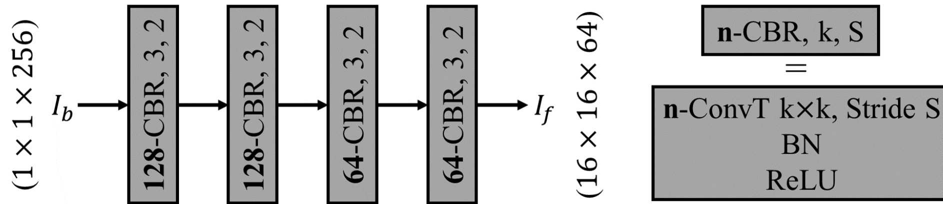


Figure 4.6: PU: The Patch-based Upsampling performed using 4 CBRs (ConvT + BN + ReLU), all with a kernel size of 3×3 and a stride of 2.

Refinement module: After aggregating all patches, we obtain a tensor of half resolution of the original image. The Refinement stage (Figure 4.7) then allows smoothing the image, hence removing unwanted block-artifacts. Each patch is now reconstructed by using not only its own information but also its neighbors. In particular, this Refinement model makes use of several Residual-Concatenation (RC) blocks before the final upsampling in order to reach the original resolution. Each RC block consists of 2 CBRs, with feature maps fusion inserted in-between that concatenates the RC input with the output of the first CBR and an add-skip connection at the output. Once the last upsampling is performed, the image feature maps pass through a last RC

block followed by a self-attention like mechanism. In this sub-module, each of the two branches contains a CBR with pointwise (1×1) convolution, and one branch has a softmax activation at the end to normalize the response of the pixels across the channel dimension. The two branches are then combined together by an element-wise multiplication, before outputting an RGB image for the final reconstruction. To demonstrate the impact of the Refinement module, we denote also the Block-Based Decoder (BBD) which contains the PU and an additional CBR with a stride of 2 for a neighborhood-independent reconstruction of patches.

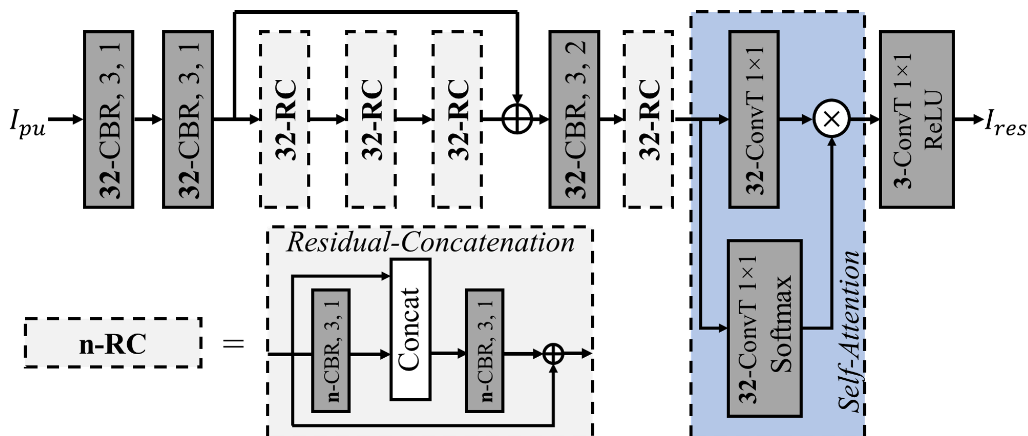


Figure 4.7: RE: Refinement model architecture with Residual and Concatenation (RC) Block. The parameter $n = 32$ denotes the number of feature maps.

PURENET training: We adapt a two-stage training procedure for PURENET. Firstly, we train the NQE with the patch-independent PURENET (*i.e.* PURENET without the Patches Aggregation, PI-PURENET) so that the NQE learns to compress 32×32 patches. After this stage, we obtain patch-based binary codes for each image which are then used as the input of PURENET in the second training stage. Finally, the pre-trained PI-PURENET is used to initialize the weights of corresponding modules (Patch-based Upsampling and Refinement) in the full-resolution mode (PURENET).

4.5 Simulation results

In this section, we target a configuration such that the memory budget for embedded classification is only approximately of 1Mb (with naive weights encoding), by choosing $F = 64$. Note that if we encode the quinary (2.32b) and ternary (1.58b) weights properly with an entropy coder, the on-chip memory may even be reduced to a sub-1Mb budget (which is not dealt in the scope of this work). All the software implementations of this work have been done in Python using a Tensorflow2 backend.

4.5.1 Image classification on CIFAR-10

In order to train our models, we apply the same data augmentation scheme as proposed in [245], *i.e.* a 4-sided 4-pixel padding followed by a random crop with random horizontal flip in order to provide 32×32 images. For classification task, the models are trained with a batch size of 50 using a squared hinge loss and the Adam optimizer [71]. 100 epochs are performed for the first training with standard BN and then 120 epochs for the fine-tuning stage, with BSN. The learning rate is initialized at 10^{-3} and decreased with an exponential decay with the rate of 0.8. The reported average accuracies are over 3 realizations for each point.

Effect of the bottleneck alternative

To highlight the efficiency of the proposed alternative to the dense layer, we conduct a study on the impact of different types of bottleneck. We compare our proposition with two other settings. The first option is a canonical Fully Connected denoted as **LFC**, *i.e.* a Learnable dense layer with binary weights ($256F^2$ bits). The second option is the same FC layer but with fixed weights (*i.e.* not trained) following a zero-mean Rademacher distribution (RCS), followed by a small FC layer of binary weights from $4F$ to $4F$ ($16F^2$ bits), denoted as **(RCS)+FC**. This second option needs 16 less bits to store the weights compared to the LFC, since the Rademacher matrix can be generated on-chip with a pseudo random generator. Table 4.3 reports our results, clearly demonstrating that a large LFC layer of $256F^2$ consumes more than 50% of the overall memory in all cases. When the model size is small, this parameter-heaviness is crucial to improve the model's performance, explaining why the gap between LFC and its two alternatives is more important at $F = 32$ (more than 1%). On the contrary, its two alternatives contribute only around 6% of the overall memory. The proposed DWConv+FC obtains much higher inference accuracy, while increasing by just a tiny amount the memory budget compared to RCS+FC (< 10kb).

Memory vs Accuracy curves

Apart from the 1Mb model, we also report the accuracy for different feature map sizes (Figure 4.8). It shows that our mixed-precision outperforms FPNN (VGG7 with full-precision) and BNN (VGG7 with binarized weights and input activations) at iso-memory. It even exhibits a large gap, as it provides an efficient design to capture enough discriminant information, compared to the loss due to either a full binarization (BNN) or a "too tiny size" (FPNN).

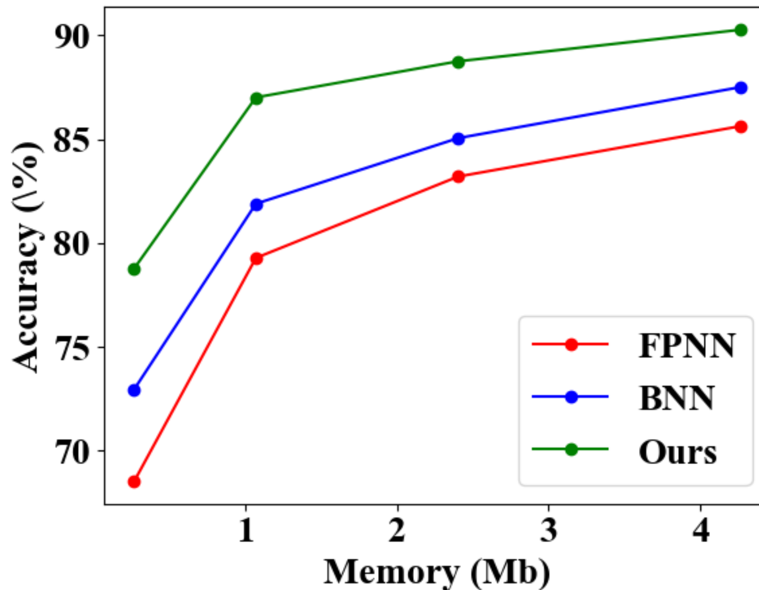


Figure 4.8: Memory-accuracy curves of different model's precision (floating-point, binary and ours mixed-precision).

4.5.2 Comparison with prior works

Table 4.4 provides a comparison with state-of-the-art CNN accelerators that have been recently demonstrated including both BNN and mixed-precision based designs. To measure the computational complexity, we make use of two bitwidth-aware metrics called MAC×bit [246] and Bit-Operations (BOPs [247]). For a relative complex dataset like CIFAR-10, a multi-bit quantization accelerator is proved to be more robust than a binarized accelerator. An important question has

Table 4.3: Effect of the bottleneck and its two alternatives to the overall memory and accuracy.

F	Model memory (Mb) without bottleneck	LFC		RCS + FC		**DWConv + FC (Ours)**	
		Memory (Mb)	Accuracy (%)	Memory (Mb)	Accuracy (%)	Memory (Mb)	Accuracy (%)
32	0.253	0.262	80.53	0.016	77.98	0.018	79.51
64	1.003	1.049	87.69	0.066	86.33	0.070	87.48
128	3.997	4.194	90.75	0.262	89.77	0.270	90.15

Table 4.4: Comparison of low-precision CNN processors (CIFAR-10 image classification task use case).

		Kim et al. [191] *TCS'21*	Valavi et al. [221] *JSSC'19*	Bankman et al. [219] *JSSC' 19*	Jia et al. [222] *JSSC' 20*	Cai et al. [243] *JSSC' 20*	***This work***	
							binary	mixed
Weight precision		1	1	1	4	2	1	1,2,3
Input Activation precision		1	1	1	4	4	1	1,2
Batch Norm	Share Level	channel/unit	channel/unit	channel/unit	channel/unit	channel/unit	layer	
	Bias-precision	6	6	9	6	10	0	
	Implementation	Sign Comp.	Sign Comp.	Addition	Sign Comp.	Fused into Kernels	Intrinsic	
On-chip Memory (Mb)		14.022	2.4	2.624	29.873	18.607	0.774	**1.073**
MAC×bit ($\times 10^9$)		0.617	0.158	1.007	1.847	1.253	0.125	**0.210**
BOPs ($\times 10^9$)		0.642	0.170	1.007	7.445	5.040	0.137	**0.287**
Accuracy (%)		88.80	84.37	86.05	92.70	90.03	82.40	**87.48**

thus arisen: how to design such a mixed-precision architecture with higher accuracy while lowering memory needs, computational complexity and simplifying hardware implementation? While the two mixed-precision designs in [222] and [243] obtain an accuracy larger than 90%, they also contain large on-chip memory of nearly 30 and 19Mb along with sub-1G MAC×bit and BOPs, which generally overpass the capacity of resource-constrained ASICs. On the other hand, BNN-based designs obtain lower accuracy at lower on-chip memory. For instance, the binary NQE (sharing the same model architecture but with all binary weights/activations) obtains only 82.40% accuracy with a tiny budget of both memory and computation, while our mixed-precision NQE topology achieves 87.48% accuracy in average, while requiring only 1Mb of weight parameters as well as under-0.3G MAC×bit and BOPs. This improvement of 5% demonstrates the significant contribution of the mixed-precision topology to the overall performance. Compared to the reported design with the nearest accuracy [219], our design requires 2.4× less on-chip memory and 3.5× less BOPs, with a 1.4% higher accuracy. Furthermore, the BSN offers a great relaxation for a future hardware implementation compared to the alternative approaches to handle normalization layers.

Table 4.5: VGA image compression comparison between different methods at the bitrate of 0.25bpp (0.2423bpp for CAEM-PSNR and 0.2459bpp for CAEM-SSIM).

Method \ Metrics	Block-based encoder	Block-based decoder	PSNR (dB)	MS-SSIM
JPEG	Yes	Yes	19.82	0.7610
JPEG2000	No	No	24.24	0.9057
WD-TV3D	Yes	No	19.67	0.7255
RCAE	No	No	22.45	0.8959
CAEM-PSNR	No	No	25.33	0.9437
CAEM- MS-SSIM	No	No	24.19	0.9640
2.5 BBD (Ours)	Yes	Yes	20.51	0.7900
PI-PURENET (Ours)	Yes	Yes	20.58	0.7964
PURENET (Ours)	Yes	No	20.76	0.8136

4.5.3 Full-frame image compression

In this section we only focus on the reconstruction of images with a VGA resolution (480×640). To this end, each image is divided into 15×20 non-overlapping patches of 32×32 pixels to apply the NQE compression scheme.

VGA images dataset

We employ the DIV2K dataset [75] which includes 800 images for training and 100 validation images for testing. In order to meet the target resolution and because the original dataset provides a large variety of image resolutions, the images are cropped accordingly to the target height/width ratio and then resized with a Lanczos kernel with radius of 3. Then we extract all non-overlapping patches of size 32×32, obtaining 240k patches for the training of NQE.

NQE training

The NQE is trained with the PI-PURENET decoder using a batch size of 100, during 60 epochs with the standard BN, and then with 30 epochs after replacing BN by BSN. To train this auto-encoder structure, we used the Mean Square Error (MSE) loss with the Adam optimizer.

PURENET training

The PURENET decoder is then trained (with a fixed NQE) in 30 epochs with a small batch size of 1 (mainly due to computational resources limitations). To obtain a proper convergence of the decoder, the learning rate is initialized at 0.001 and then rescaled by 0.95 at each epoch after the fifth epoch. We then freeze the Patch-based Upsampling module and fine-tune the Refinement with a batch size of 2 during 30 epochs. The learning rate is also initialized at 0.001 and then rescaled with the same factor 0.95 after the 10th epoch.

Comparison with state-of-the-art methods

We compare NQE - PURENET with different methods: JPEG [227] and JPEG2000 [248] which are patch-based standard image compression techniques (*i.e.* using an entropy coder after a sparsifying transform), one BCS encoding with regularization-based iterative method for decoding consisting in a L1-regularization on a 2D-Daubechies Wavelet Dictionary applied to 3D image gradients (denoted WD-TV3D, inspired from [232] and giving better results than a basic TV [249]); the end-to-end learning based methods using Recurrent Convolutional AutoEncoder [236] called RCAE and the Context-Adaptive Entropy Model ([250]) optimized with Peak-Signal-to-Noise Ratio (CAEM-PSNR) or with MultiScale Structural SIMilarity metric [251](CAEM- MS-SSIM), both dedicated to full-resolution image compression; the Block-Based Decoder (BBD); our NQE with PURENET for patches reconstruction, namely NQE- PI-PURENET. We notice that for WD-TV3D, we apply the same Rademacher matrix to all non-overlapping patches of size 32×32, every measurement is 5-bit uniformly quantized in the range of [-3σ, 3σ], where $\sigma = \frac{0.5}{\sqrt{1024}}$ is the estimated standard deviation of the measurements distribution. Besides, CAEM compression rate changes from one image to another, therefore we choose the quality level which allows the average bit-rate close to 0.25bpp.

Image quality evaluation

The average PSNR and MS-SSIM performances over the 100 test images are reported in Table 4.5 while the reconstructed images are displayed in Figures 4.9 to 4.14. It clearly shows that our NQE-PURENET framework delivers better compression quality compared to JPEG and WD-TV3D in both cases patch-based or full-resolution reconstruction, with higher average PSNR/MS-SSIM and better image quality. However, when comparing this method with JPEG2000 and end-to-end learning methods (RCAE, CAEM-PSNR and CAEM- MS-SSIM) which achieve higher PSNR/MS-SSIM thanks to their specific design for this task, we clearly observe the lack of finest details, for example the lion's eye and the castle in the third and the fourth columns. This lack can be easily explained as a direct consequence of the quantized nature of the NQE.

Effect of the Refinement module

Compared to the BBD and PI-PURENET, the PURENET slightly improves the PSNR but significantly the MS-SSIM, which is easily explainable with respect to the noticeable enhancement in terms of visual quality. We can see the block artifacts in BBD and PI-PURENET, as they do not take into account the surrounding context of each patch. On the contrary, the PURENET successfully renders the smoothness between patches thanks to several convolutional layers which will

broaden the information propagation from neighboring patches. This explains why it has MS-SSIM much higher (*i.e.* 0.0172 compared to PI-PURENET and 0.2136 compared to BBD).

4.6 Conclusion and Perspective

This work presents a mixed-precision CNN topology which is compliant with a ASIC design, enabling to perform both low-complexity image classification and embedded patch-based compression. The reported results demonstrate the possible degree of versatility in terms of application use cases for a specific neural network architecture targeting an ASIC design. Numerically speaking, our NQE exhibits a 87.48% accuracy for CIFAR-10 while requiring 1Mb of memory and whose weights and activations are quantized with a mixed-precision approach. In addition, Batch Normalization layers are replaced by layer-shared BitShift Normalisations, in order to further ease a possible hardware implementation. In terms of image compression, our PURENET architecture typically deals with patch-based binary coding to perform a collaborative reconstruction, providing images with a relatively high rendering quality at a bitrate of only 0.25bpp. Besides, PSNR and MS-SSIM metrics are better than relevant alternatives such as JPEG and BCS compression schemes. The proposed approach shows a good quality of service versus its computational complexity, especially for an embedded patch-based image compression. Aforementioned results all confirm the advantage of an algorithm/hardware co-design to reach the best trade-off between hardware implementation complexity and algorithmic accuracy. Additionally, it also demonstrate that the HEQ framework presented in the previous chapter can be used to efficiently train mixed-precision QNNs targeting different applications.

For future works, we may seek to find the mixed-precision topology using gradient-based neural architecture search or an input-driven dynamic approach. The model architecture itself can be derived from efficient building blocks of existing light-weight models, with residual connections for improving representation power. Regarding the compression task, a direct extension may involve the use of quantized RNN units to extend NQE compression and classification capabilities to frame sequences, not only still images. Another approach may also be using ensemble learning like [240] (thanks to the reconfigurability) to enhance the compression quality. In other respects and to improve the performances of the NQE, its topology would also benefit from the use of skip connections so as self-attention mechanisms. More concretely, since the current PURENET treats every pixel equally without considering whether the pixel is at the border of the patch, it would be more difficult to ensure both fine-grained details and block artifact removing at the same time. One possible solution for this is to introduce a spatial attention mechanism only focusing on patch border pixels.

Figure 4.9: Image compression results at 0.25bpp on the test image indexed 801 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image.

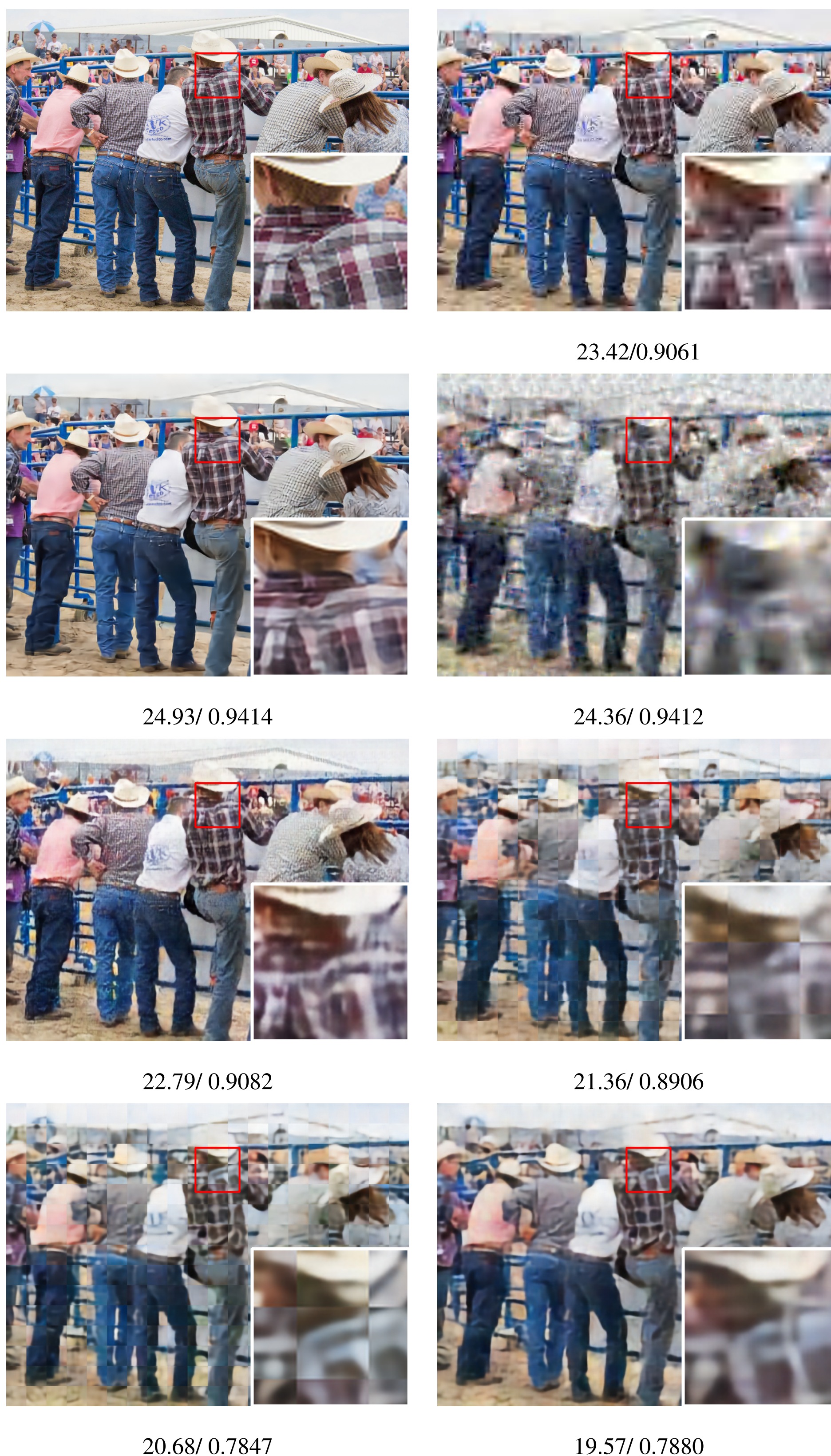


Figure 4.10: Image compression results at 0.25bpp on the test image indexed 804 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image.

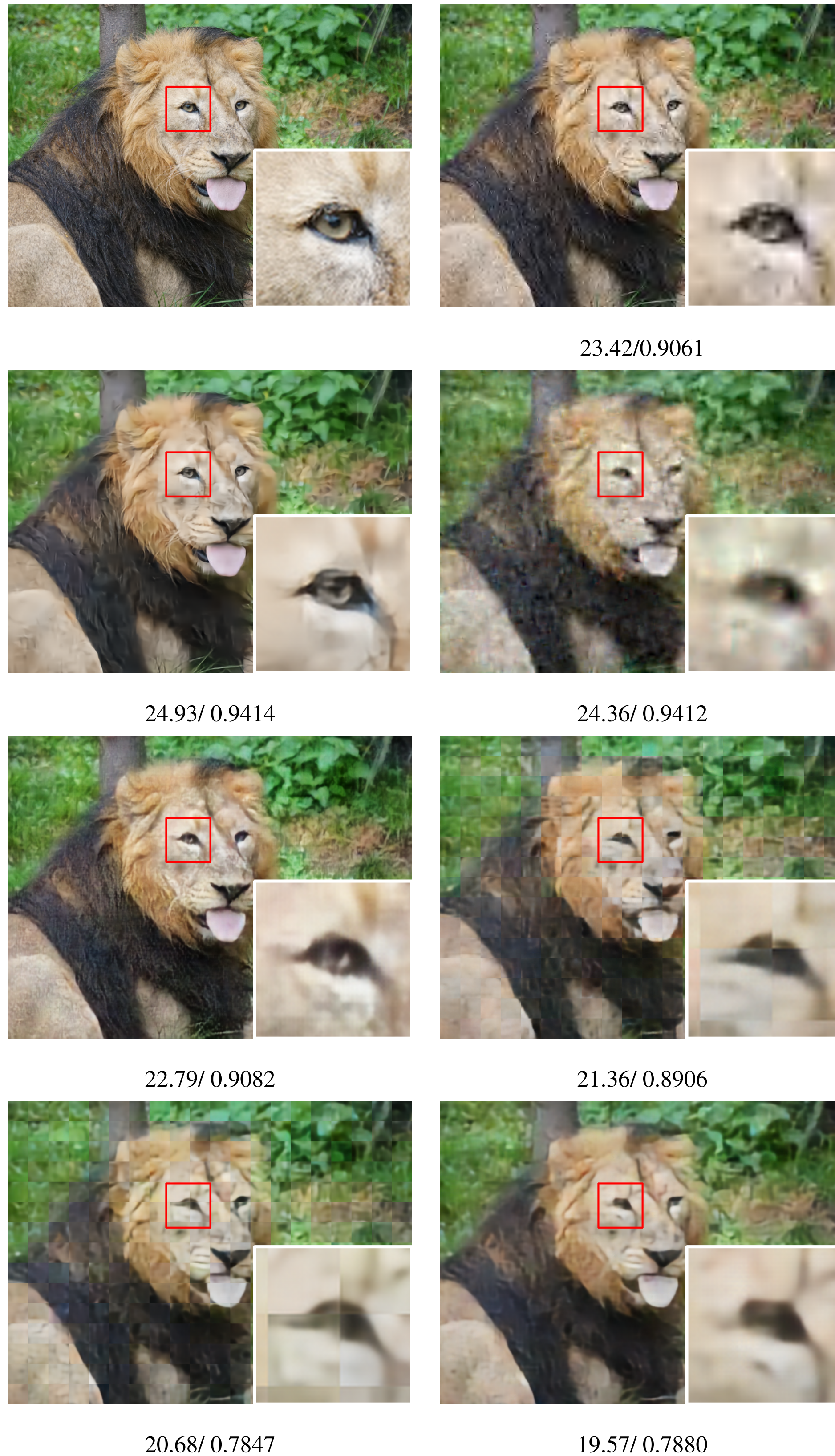


Figure 4.11: Image compression results at 0.25bpp on the test image indexed 809 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image.

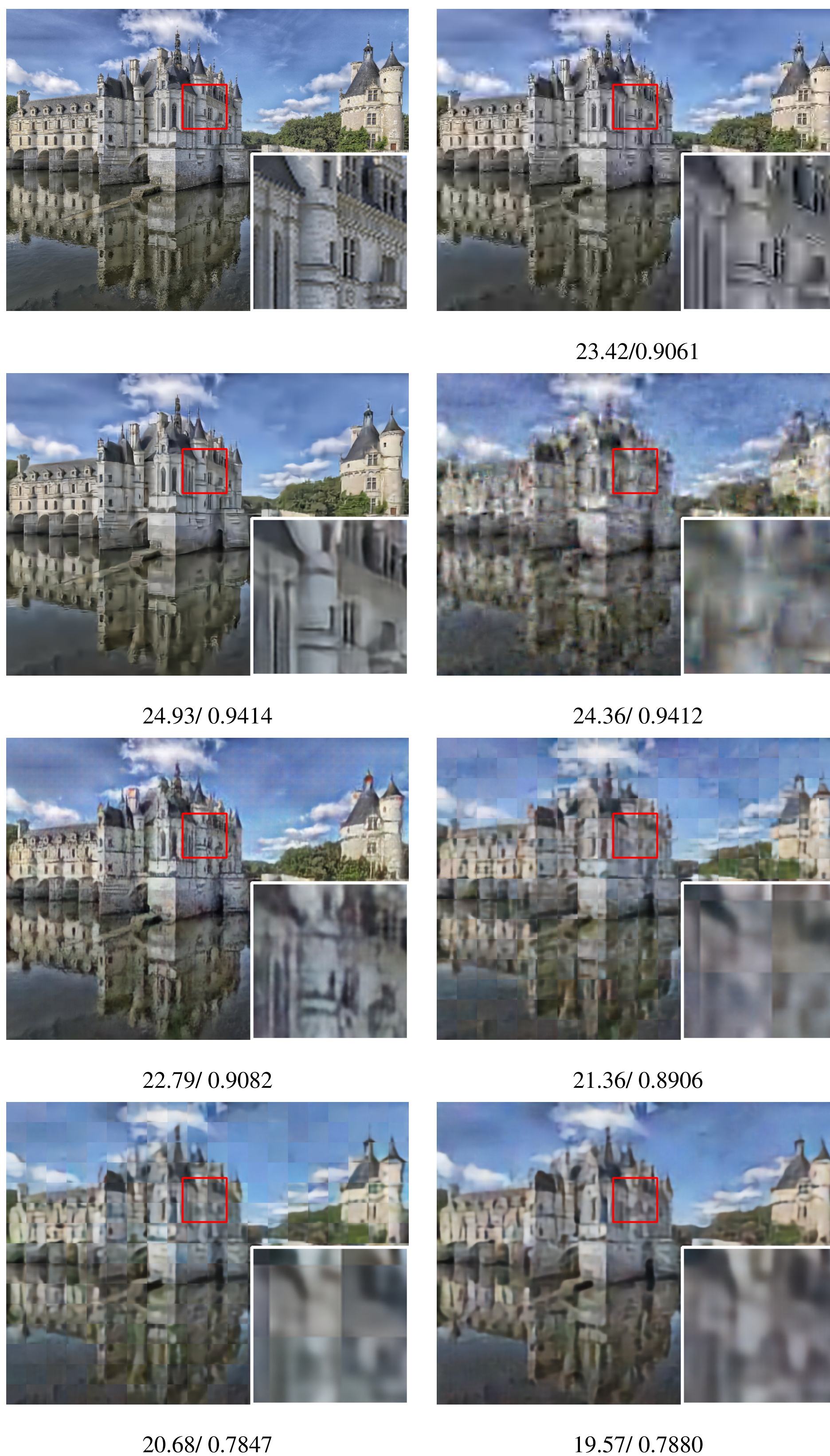


Figure 4.12: Image compression results at 0.25bpp on the test image indexed 865 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image.

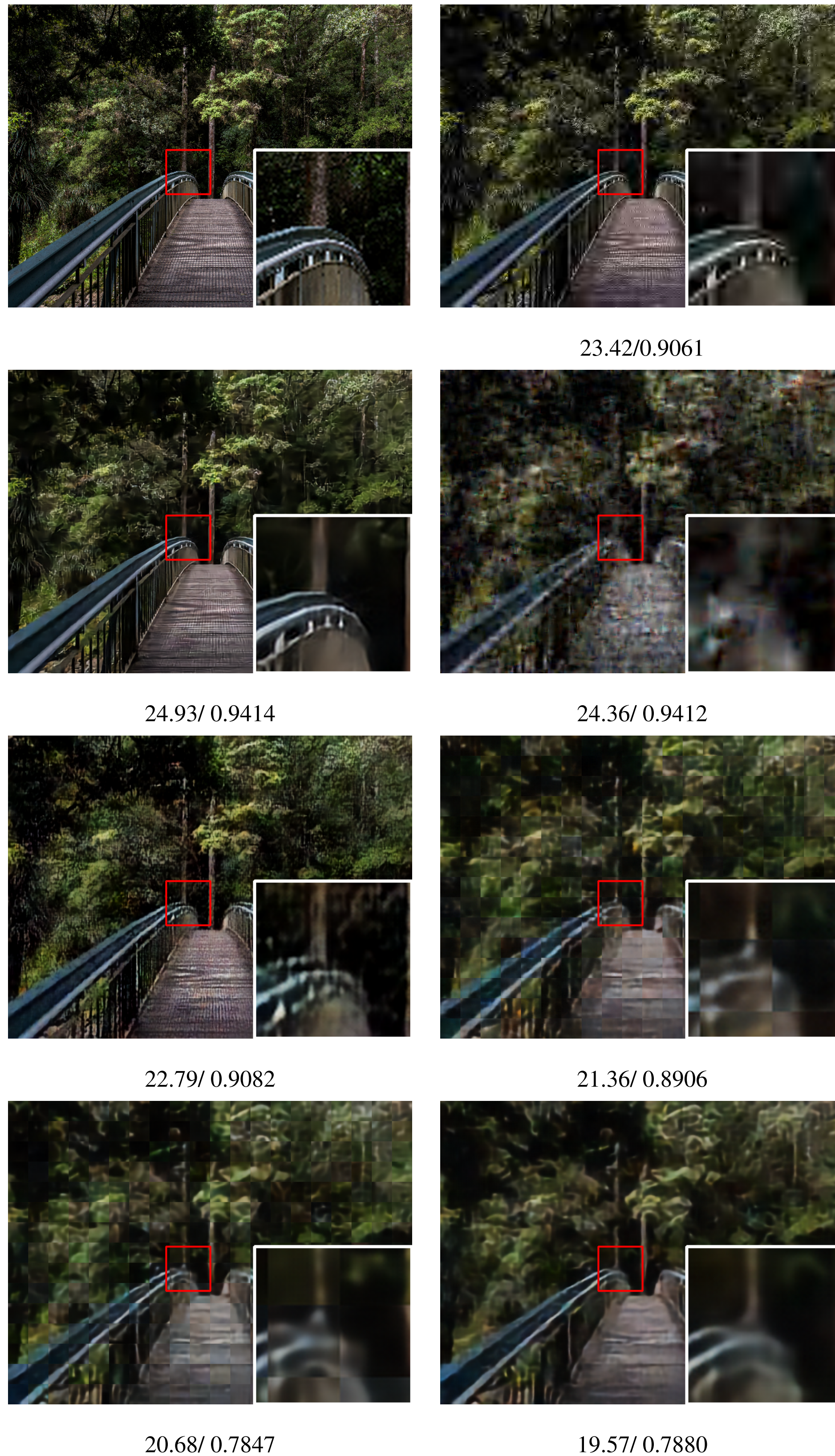


Figure 4.13: Image compression results at 0.25bpp on the test image indexed 876 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image.

Figure 4.14: Image compression results at 0.25bpp on the test image indexed 894 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image.

5

Hardware-aware Residual Networks with logic-gated skip connections and light-weight convolutional factorization

Applying model compression techniques to existing large DNNs may achieve remarkable performance, however, their hardware implementation is still questionable. In this chapter, we propose a more hardware-compliant approach for designing compact residual neural networks, by integrating the quantization perspective into the skip connections. We will start by presenting our proposed logic-gated residual building blocks, which allows to implement the skip connections with negligible hardware costs while obtaining higher accuracy than the plain model counterpart. To reduce furthermore the model size, we also propose a light-weight convolutional factorization leveraging on-line generated weights via a Cellular Automaton. The results show that our models obtain better hardware-algorithm trade-offs at low memory budget compared to state-of-the-art.

Contents

5.1 Context **90**

5.2 Logic-gated Residual Neural Networks **91**

5.2.1 Skip connections with OR and MUX gates 91

5.2.2 Experimental results 92

5.3 Compact CNN with light-weight convolutional factorization **93**

5.3.1 MUX-Residual Block (MRB) 95

5.3.2 Convolution factorization leveraging CA-generated weights 96

5.3.3 Experiments 97

5.4 Conclusion **100**

5.1 Context

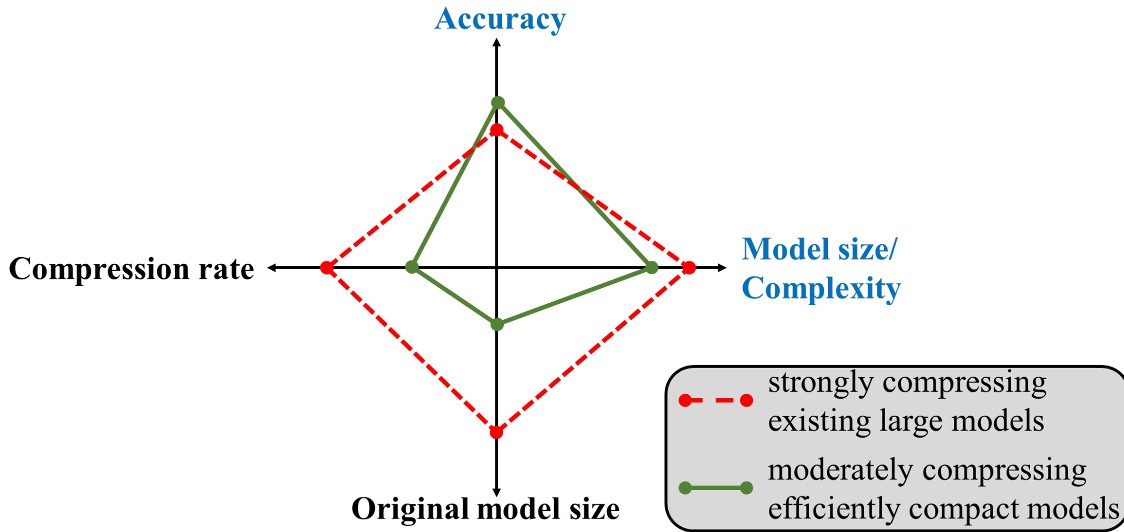


Figure 5.1: Two common strategy of compressing the models: obtain an extremely high compression rate on existing large model, and obtain a moderate compression rate on efficiently compact models. The latter results in compressed models with better hardware-algorithm trade-offs.

Model compression techniques have achieved tremendous progress in the last few years. Previous works mainly apply these techniques to existing large models and evaluate the results as a compromise between the algorithmic performance (*e.g.*, accuracy) and the compression rate. For instance, [252] reduces the size of VGG-16 up to 49× almost without no loss of accuracy. More recently, [253] even obtains a significant compression rate of 590× in VGG-16, with an accuracy loss of 3.4% on CIFAR-10. [254] then reduces the size of ResNet-18 model by 214.9× at the expense of 3.49% accuracy on CIFAR-10. Unfortunately, the implementation of these compressed models may still faces several challenging problems that degrade their hardware efficiency. One of the main issues relies on the hardware compatibility of these model compression methods. A typical example is the case of unstructured pruning which requires additional modules for handling the remaining weights of the model. Another example is the entropy coding in [253] or the Huffman coding in [252] that are not favorable from a hardware point of view.

The second issue that limits the efficiency of model compression techniques is the DNN model itself. Indeed, existing model architectures like VGG-16, ResNet-18 are typically over-parameterized to improve accuracy in different applications without considering the hardware context. This results in complex operations such as normalization, residual connections and attention mechanism while model compression methods mainly focus on the MAC operation. Moreover, these large models imply an extremely high compression ratio to obtain low-sized models, as a result, this strategy usually exhibits significant accuracy loss as mentioned above. The recent development of compact DNN designs (*e.g.*, MobileNet, ShuffleNet) and NAS-based models has given rise to a question: is it better to apply model compression methods to an efficiently-designed model? In the context of quantized neural networks, [255] has demonstrated that it is better to quantize a slim model (*i.e.*, less channels) with a moderately low bit-width, rather than quantizing a fat model with an extremely low precision. This result can be qualitatively illustrated with the line polar graphs in Figure 5.1, where the original model size and the compression rate determine the strategy of compressing DNN models, while the model size/complexity and the accuracy depicts the hardware-algorithm compromise of the compressed model. In a more general case, we argue that the original uncompressed model itself should be considered as a “hyper-parameter”

and jointly optimized with the model compression technique.

Although quantization methods have been mainly applied to a wide range of DNN topologies, their usage is mainly focused on reducing the weight and activation bit widths. On the other hand, the element-wise addition in the case of skip connections (ResNet [24]) is still performed using a full-precision like in [256], [257] and [258]. The reason is that apart from improving feature map reusability, these full-precision additions are mostly used to handle the gradient vanishing and mismatching issues, which seem to be even more crucial in the context of quantized models. However, it results in additional costs with respect to the corresponding hardware implementation.

In this section, we focus on the compression of those skip connections in QNNs, including the residual addition and the attention-like multiplication [259], such that these residual connections can be implemented by only integer operations and MUX gates rather than 32-bit arithmetic hardware. In details, section 5.2 investigates the possibility of using OR and MUX gates in the case of 1-bit Heaviside activations. This 1-bit MUX-OR residual mechanism is then generalized into the case of n-bit in section 5.3. To improve the efficiency of the model, we replace the costly regular Conv2D by a light-weight factorization including two pointwise layers and a grouped convolution inserted in-between. In particular, the weight matrix of the second pointwise convolution is generated on-the-fly via a Cellular Automaton, enabling reducing furthermore the on-chip memory.

5.2 Logic-gated Residual Neural Networks

5.2.1 Skip connections with OR and MUX gates

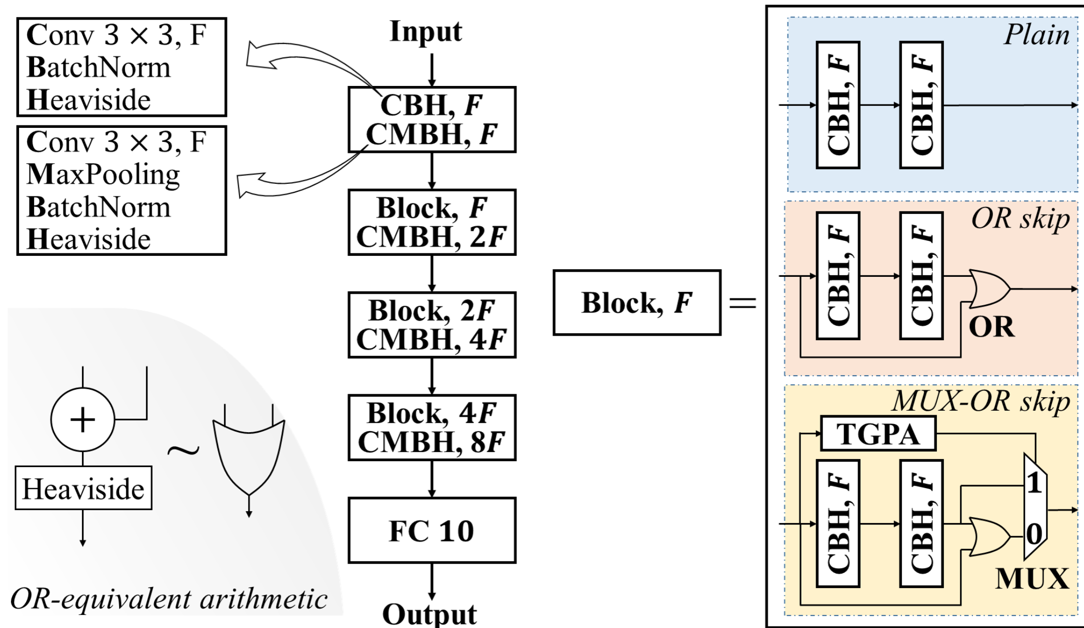


Figure 5.2: Models with the plain block (11-hidden layer VGG [81]-variant), OR-gated block and MUX-OR gated block.

Fig. 5.2 depicts the proposed model design with 3 consecutive convolution blocks with different variants: plain block (denoted as VGG-11), OR block (ORNet-11) and MUX-OR block (MUXORNet-11), where F denotes the basis number of convolutional output feature maps. All the activations are binarized using the Heaviside function $H(x) = 1_{\{x>0\}}$ where 1 is the indicator

Table 5.1: OR gate and its arithmetical operation

x_1	x_2	$x_1 \vee x_2$	$\text{Heaviside}(x_1 + x_2)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

function. The logical OR operation between two binary inputs is arithmetically performed as:

$$x_1 \vee x_2 = \text{Heaviside}(x_1 + x_2) \tag{5.1}$$

which is demonstrated in Table 5.1. The MUX-OR block additionally embeds an attention-alike branch (called MUX branch) along with the OR skip connection. This MUX branch is composed of a channel-wise Thresholded Global Average Pooling (TGPA) that corresponds to a Global Average Pooling (GPA) followed by the (re)binarization $T(x) = 1_{\{x>0.5m\}}$, where m is set to the maximum of the GPA's outputs in full-precision model and to 1 in quantized model. When deploying the quantized model, this operation can be basically implemented via a bit-count followed by a comparison with a threshold level equal to half the number of pixels. Concretely, the OR-skip connection will be performed for each input feature map channel that has more zeros than ones. Otherwise, the MUX will simply keep the straightforward output of the second Convolution-BatchNorm-Heaviside (CBH) module. One interesting aspect of such a MUX-skip connection is that it favors to balance the number of 1 with respect to the number of 0 throughout the networks, this without any other specific regularization. In terms of hardware deployment, while existing approaches with 32-bit additions and multiplications require hundreds of Xilinx FPGA slices [37], a 1-bit OR only costs a single slice and consumes much less energy [260].

5.2.2 Experimental results

In this section, we evaluate the aforementioned Neural Network topology variants using the proposed HEQ (*c.f.*, chapter 3) with ternary weights on STL-10 dataset [74] of 96×96 RGB images. To limit the overfitting, we used the following data augmentation scheme: random crop from all-sided 12-pixel padded images combined with random horizontal flips and cutouts of 32×32 pixel patches [261]. All the parameters of the quantized model are initialized from its pre-trained full-precision network counterpart, in which all Heaviside functions are replaced by ReLU. We set $F = 64$ instead of 128 in VGG-7, resulting in smaller-sized model.

Table 5.2: Comparison with the state-of-the-art low-precision quantization methods on STL-10 dataset.

Model	Training	Regularization	# params. (M)	Bitwidth W/A	Acc. (%)
VGG-7	LSQ [200]	#params×bit [201]	4.57	2.5/8	83.6
		#MACs×bit [202]		2.2/8	83.8
VGG-11	HEQ	None	3.14	2/1	83.34
ORNet-11					83.82
MUXORNet-11					**84.17**

Table 5.2 summarizes the performance of our proposed models compared to previous work [202] which uses the LSQ [200] method to jointly adapt the step size and the layer-wise bitwidths under a model size-based [201] or a MAC×bit-based regularization. Note that we only took into account the number of convolution parameters for the sake of a fair comparison. While the plain baseline obtains only 83.3% accuracy, ORNet-11 achieves 83.8% and the MUXORNet-11 even achieves up to 84.2%, *i.e.* a noticeable improvement without increasing the overall model size and with a negligible extra cost of 2-input MUX, OR gates and thresholded bitcounts.

Figure 5.3 shows the impact of the OR gate and the MUX-OR mechanism on the distribution of 0 and 1 intermediate values in 3 blocks of the proposed OR-Net and MUXOR-Net. At the output of the second Heaviside in each block, the activations are dominated by zero values. The OR connection increases the number of 1 values and thus makes the activation's distribution more balanced. On the other hand, the MUX-OR mechanism only performs the OR connection if the corresponding input channel of the block has more zeros than ones, therefore the distribution at the output of the MUX gate is slightly more unbalanced towards zero values. This is more favorable in terms of feature learning, since a too balanced activation's distribution may likely harden the extraction of meaningful features from the data.

In terms of model size, all our 3 model variants contain less parameters at lower precision compared to [202]. However, the proposed ORNet-11 optimized by HEQ already achieves the same level of accuracy while MUXORNet-11 even obtains a better accuracy (0.37%). These results demonstrate the effectiveness of HEQ on different DNN designs, from VGG-like to the proposed ORNet and MUXORNet. It also shows the possibility of compressing the skip connections via logic gates in order to significantly simplify the hardware mapping of more sophisticated ternarized neural network topologies than VGG-like. In particular, the proposed MUX-OR mechanism allows to advantageously increase the representation power and ease the learning of the model, while advantegeously keeping the dynamic range of the low-precision activations. Therefore, in the next section, we make use of this MUX-OR mechanism and propose a general formulation in the case of n-bit activations.

5.3 Compact CNN with light-weight convolutional factorization

Our goal here is to reduce the overall hardware needs required to run a model implemented in resource-constrained devices (*e.g.*, for ASIC design) while still ensuring an acceptable accuracy. Unlike several works focusing on large models to achieve extremely high compression rates [252], [262], we first propose a hardware-compliant model architecture to which we further apply efficient quantization methods.

In this section, we present the compact MOGNET model architecture which combines:

- quantized residual modules with a Multiplexer-based skip mechanism and,
- a custom factorization of convolution layers that uses on-line generated weights.

Indeed, a Cellular Automaton (CA) is used to automatically generate the weights of a pointwise convolution in each factorized-CNN block, thus reducing parameter-related storage requirements. Moreover, we introduce a novel training framework to obtain the ternary weights in our model which favors the balance between 3 discrete levels.

Figure 5.4 describe the MOGNET architecture that uses integer-only MACs and hardware-compliant operations such as 1-bit Bitshifts and 2-input multiplexers. The following description yet presents MOGNET from its algorithmic view point, *i.e.* with computations done in a real-valued domain but with relevant hardware-equivalent specializations. In this section, we first

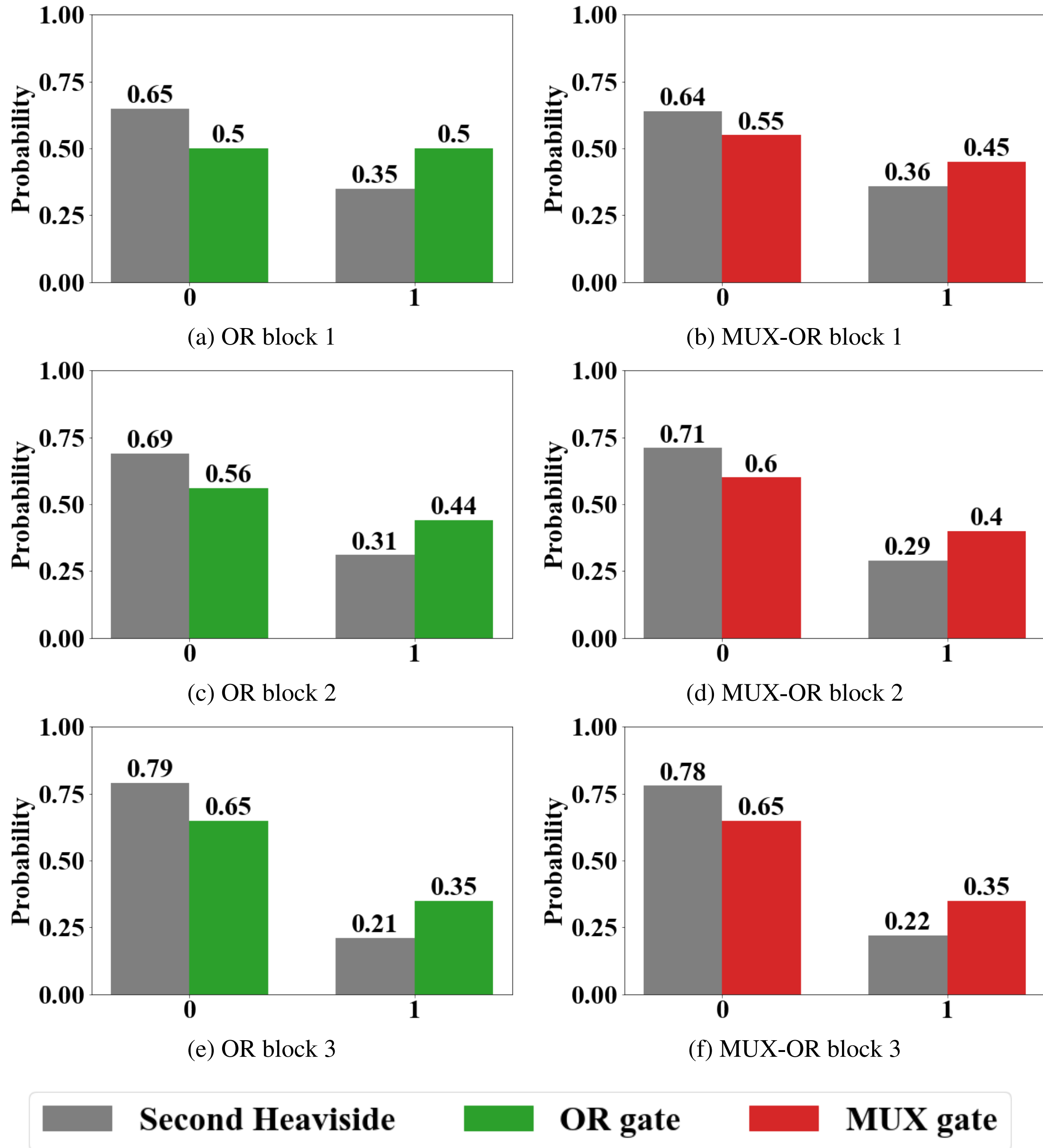


Figure 5.3: Impact of the OR gate and the MUX-OR mechanism on the distribution of 0 and 1 intermediate values in 3 blocks of the proposed OR-Net and MUXOR-Net.

focus on our custom MUX Residual Block (MRB), then on our convolution layer factorization (CFLOG).

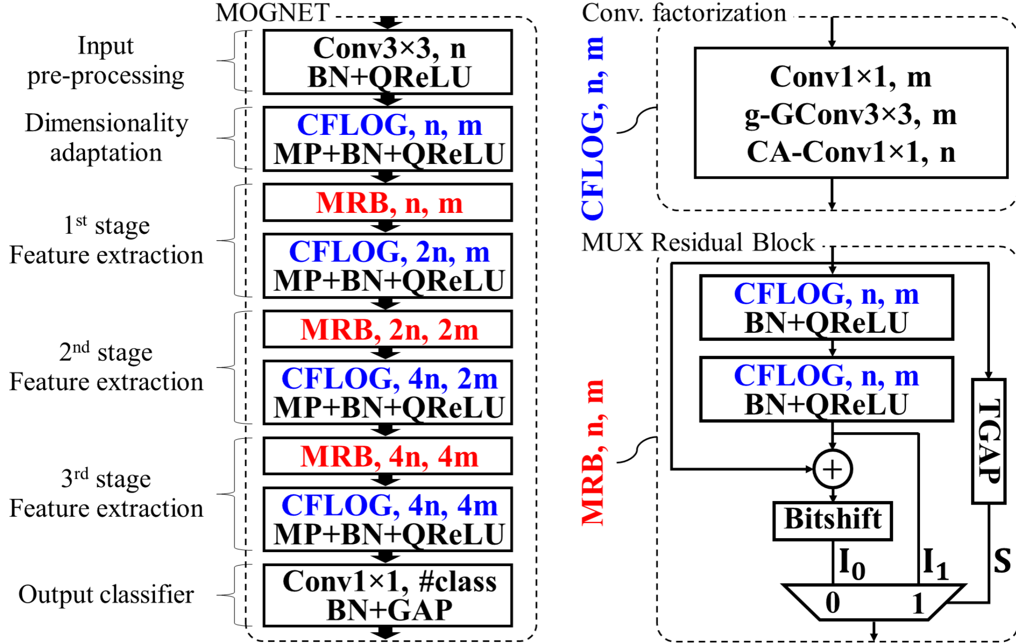


Figure 5.4: Top-level architecture description of MOGNET with Convolutional Factorization Leveraging On-line Generated weights (CFLOG) and MUX Residual Block (MRB). The final 1×1 convolution is followed by Batch Normalization (BN) prior to a Global Average Pooling (GAP). Here n, m are the parameters controlling the number of output feature maps and the latent dimension in CFLOG, MP stands for 2×2 Max Pooling and g-GConv is Grouped Convolution with g groups.

5.3.1 MUX-Residual Block (MRB)

We denote k as the quantization bitwidth of the activations throughout the network. Indeed, a k-bit Quantized Rectified Linear Unit (QReLU) is defined so that for any input, the outputs are in the set $\{0, \frac{1}{2^k-1}, \frac{2}{2^k-1}, \ldots 1\}$:

$$\text{QReLU}(x;k) = \begin{cases} \frac{1}{2^k-1}\lfloor(2^k-1)x\rceil & \text{if } k>1, \\ 1_{\{x>0\}} & \text{if } k=1. \end{cases} \tag{5.2}$$

For the backward pass, we compute the gradient using the Straight-Through-Estimator strategy (STE, [59]) $\frac{\partial \text{QReLU}}{\partial \boldsymbol{x}} = 1_{\{|\boldsymbol{x}|\leq 1\}}$. As the Addition $y = x_1 + x_2$ of two unsigned k-bit activations x_1, x_2 will increase the dynamic range by 1-bit, we make use of the following Bitshift software description inside the MRB, to keep y always at k-bit:

$$\text{Bitshift}(y;k) = \begin{cases} \frac{1}{2^k-1}\lfloor\frac{(2^k-1)y}{2}\rceil & \text{if } k>1, \\ \lceil\frac{y}{2}\rceil & \text{if } k=1. \end{cases} \tag{5.3}$$

Due to the specific use of this function for the add-type connection, we adopt a completely-passed-through gradient $\frac{\partial \text{Bitshift}}{\partial \boldsymbol{y}} = 1$. For $k>1$, this rescaling can be implemented by a 1-bit bit-shift, while in the specific case of $k=1$, the combination of the addition and the bitshift can be replaced by an appropriate single **OR** gate. Let us denote $\mathbf{I}_0, \mathbf{I}_1 \in \mathbb{R}^{h\times w\times n}$ as the output of the Bitshift operation and the second QReLU where h, w, n are the height, width and number of channels; $\mathbf{S} \in \{0,1\}^{1\times 1\times n}$ as the binary control signal. The MRB core element is **MUX** which can be mathematically described as:

$$\mathbf{MUX}(\mathbf{I}_0, \mathbf{I}_1; \mathbf{S}) = \mathbf{I}_1 \odot \mathbf{S} + \mathbf{I}_0 \odot (\mathbf{1} - \mathbf{S}) \quad (5.4)$$

where $\odot$ is a channel-wise multiplication. The control signal $\mathbf{S}$ embeds a parameter-free channel attention which consists in a Thresholded Global Average Pooling (TGAP). TGAP simply corresponds to a channel-wise Global Average Pooling (GAP) followed by a binarization $T(x) = 1_{\{x>0.5m\}}$, where m is set to the maximum of the GAP's outputs in the full-precision representation and to 1 in the quantized model which is the maximum possible value of quantized activations. For the hardware deployment, this last operation can be implemented via an integer accumulation followed by an integer-to-integer comparison. This way, the input of each MRB will automatically control the operation of the Multiplexer module in a channel-wise manner. Concretely, MRB will perform the Additional connection for each input feature map that is dominated by small values. Otherwise, the MRB will simply keep the straightforward output of the second QReLU. One interesting aspect of this MUX-skip connection is that it favors the balance between the number of large-valued data with respect to the number of small-valued data throughout the networks, this without any other regularization strategy.

5.3.2 Convolution factorization leveraging CA-generated weights

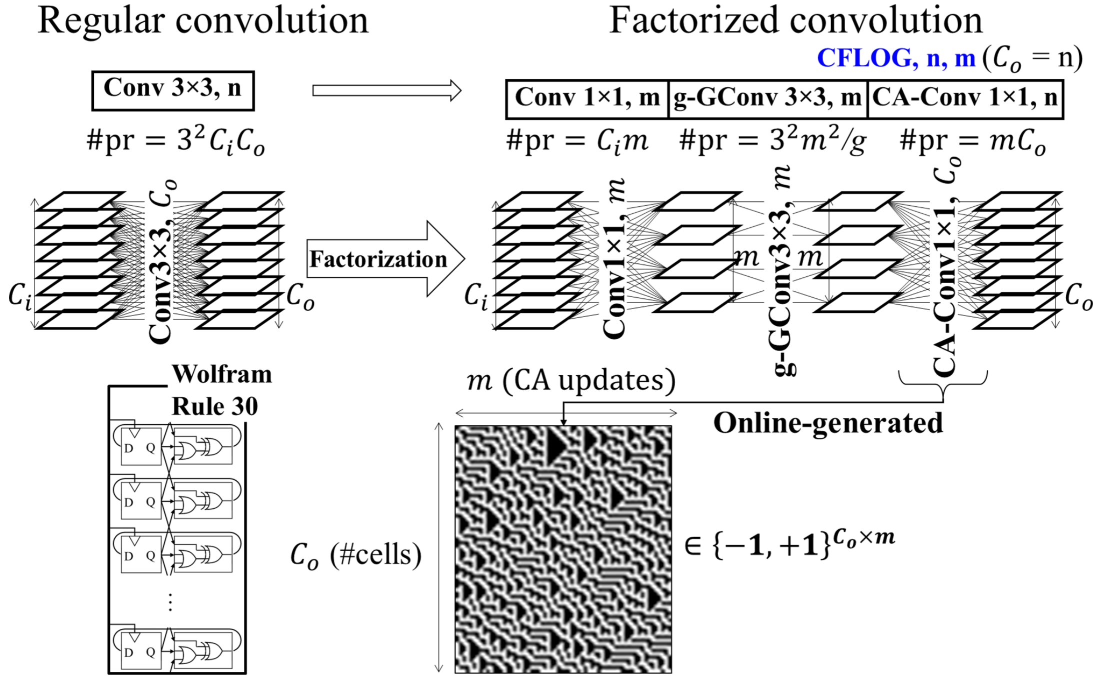


Figure 5.5: CFLOG description with CA-generated weights.

The use of PRNG in DNNs

As previously mentioned in section 2.1, DNN's parameters are randomly initialized before being adjusted during the learning process. However, it is not necessarily to learn all parameters to obtain a good performance. [263] indicates that in some classification tasks, it is possible to use neural networks with fixed random weights in the first layer and trainable weights in the next layers, as they are still able to preserve the distance between the latent representation of different classes. [264] then shows that a randomized NN can be used as a hand-crafted prior in some inverse problems. These theoretical results open a great possibility of alleviating the memory burden in DNN implementation, by leveraging PRNG such as LFSR or CA to generate on-the-fly the fixed random weights.

[265] enable the generation of random states given an initialization and update rule function. Hence, this repetitive structure is commonly used for generating on-the-fly sets of random vectors [266], or computing in random representations [267]. In MOGNET, we leverage CA to generate part of weight parameters of our model, consequently reducing the overall model memory-related footprint.

Light-weight convolution factorization

To further reduce the on-chip memory and the computational complexity of the model, we replace all regular convolutions (except the first and the last layers, cf. Fig. 5.4) by a light-weight factorization consisting of 2 pointwise layers and a grouped convolution (Fig. 5.5), namely CFLOG. Unlike the building block in ResNext [28], we do not use any nonlinearity (*e.g.* normalization, activation) between these layers. Moreover, to further reduce the model size, the last pointwise convolution's weights are fixed during training and generated in real-time by a CA given a certain seed. The first pointwise convolution embeds the input feature into low-dimension $m < C_i$, the number of input channels. The grouped conv layer performs g groups of convolutions and also outputs m feature maps. Finally, these feature maps are sequentially projected back to a high-dimensional space of C_o channels ($C_o = n$ in CFLOG, n, m) thanks to a CA-generated kernel. As depicted in Fig. 5.5, this kernel is formed by concatenating all states obtained when evolving a C_o-cell CA during m update states. In this work, we consider Wolfram's rule 30 for the local evolution function between states. We choose $m = \frac{C_i}{2}$ that gives the following compression rate (CR) between the number of trainable parameters (#pr) of CFLOG and that of the regular convolution:

$$\text{CR} = \frac{C_i m + \frac{3^2 m^2}{g}}{3^2 C_i C_o} = \frac{C_i}{C_o}\left(\frac{1}{18} + \frac{1}{4g}\right) \tag{5.5}$$

Table 5.3: Training and optimization settings

Dataset	CIFAR-10	CIFAR-100
Optimizer	Adam [71] ($\beta_1 = 0.9, \beta_2 = 0.999$)	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Initial learning rate (LR)	10^{-3}	10^{-3}
Batch size	50	50
First stage epoch	180	250
First stage LR schedule	Exponentially decay after 120-th epoch	Exponentially decay after 150-th epoch
Second stage epoch	150	250
Second stage LR schedule	Exponentially decay after 80-th epoch	Exponentially decay after 150-th epoch
Rate of LR decay	0.9	0.9

5.3.3 Experiments

Experimental settings

We implemented all the proposed elements using the TensorFlow library and CellPylib [268] package. The quantized models are first initialized from their full-precision counterparts being trained on CIFAR-10 and CIFAR-100 [73] datasets from scratch, where ReLU and Linear activations replace our QReLU and BitShift. Then, we train the quantized models through a 2-stage procedure:

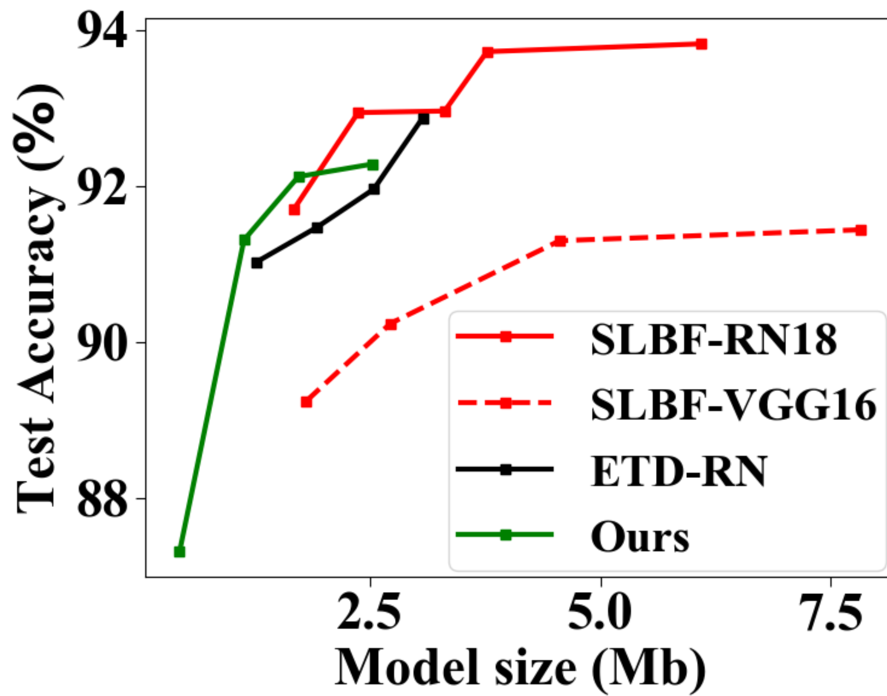
first train quantized weights with full-precision activations and second, fine-tune the quantized weights with all quantized activations. We apply the simple data augmentation scheme for training: random crop from all-sided 4-pixel padded images combined with random horizontal flips. Table 5.3 details the training and optimization setting used to derive our experimental results.

Experimental results

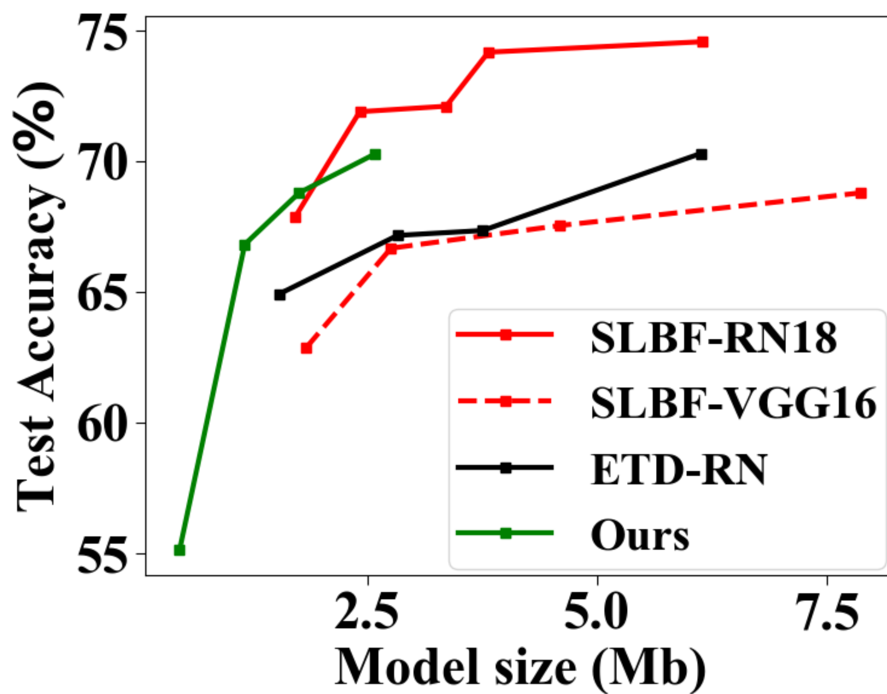
We evaluate the performance of our models in comparison with recent state-of-the-art model compression techniques: Stacking Low-dimensional Binary Filters (SLBF [254]) on ResNet (RN)-18 and VGG-16 [81]; Efficient Tensor Decomposition (ETD [179]) on RN-20 and RN-32. Table 5.4 reports the model size, the activation precision as well as the accuracy of different methods and models. It demonstrates that for $n = 128$ and $g = 4$, MOGNET achieves the highest accuracy level on CIFAR-10, while having lower model size and 3-bit only activations. Moreover, at the same configuration, MOGNET outperforms other methods on CIFAR-100 with a clear gap of nearly 1% (67.89 → 68.80%) at the similar weight-related memory. We can mention the impact of the hyperparameter k on the model performance, with a significant degradation when decreasing the activation precision to 1-bit or 2-bit. Figure 5.6 additionally reports accuracy versus model size curves of various considered models/hyperparameters. On both two datasets, MOGNET stays in the optimal top-left zone implying low on-chip memory requirements with high accuracy. However, when increasing the model size ($n > 128$), MOGNET curves fall under that of SLBF-RN18. This last result means that MOGNET, with its limited depth, is more relevant to target extremely low-sized (< 2Mb) models.

Table 5.4: Comparison of different network compression methods on CIFAR-10 and CIFAR-100.

Method Model	Activation Bitwidth (k)	CIFAR-10		CIFAR-100	
		Model size (Mb)	Acc (%)	Model size (Mb)	Acc (%)
SLBF [254]					
RN-18	32	1.67	91.70	1.72	67.89
VGG-16	32	1.84	89.24	1.89	62.88
ETD [179]					
RN-20	32	1.94	91.47	3.80	67.36
RN-32	32	2.54	91.96	2.83	67.17
Ours (n=128, g=8)	1	1.13	87.60	1.22	59.30
	2		90.81		65.88
	3		91.31		66.83
Ours (n=128, g=4)	1	**1.72**	88.99	**1.76**	61.27
	2		91.16		66.55
	3		**92.12**		**68.80**



(a) CIFAR-10.



(b) CIFAR-100.

Figure 5.6: Test accuracy of different compression method-model couplings. Our models are with 3-b activations.

5.4 Conclusion

In this chapter, we discussed about the advantage of a co-design between the DNN architecture and the model compression method. In particular, we focus on the hardware implementation of residual connections in the context of QNNs, and figure out that these operations should be also taken into consideration to fully exploit the benefits of low-precision networks. We started by 1-bit skip connections with the hardware-friendly OR and MUX gates, which may obtain a clear gain over the plain architecture counterpart in terms of accuracy. These building blocks are then generalized into a case of n-bit activation in the proposed MOGNET. To boost the efficiency of the MOGNET, we also make use of a streamlined convolutional factorization which leverages CA-generated weights. Experimental results show that given tiny memory budget (*e.g.*, sub-2Mb), MOGNET can achieve higher accuracy with a clear gap up to 1%, at a similar or even lower model size compared to recent state-of-the-art methods. Additional, MOGNET also limits the digital dynamic range using 3-bit (or lower) quantized activations for integer-only MACs. This result thus demonstrates the benefit of hardware-algorithm co-optimization which efficiently integrates the implementation perspective into the design of model architectures.

It is noteworthy pointing out that the architecture of MOGNET is not optimized, with manually settings for the depth, the width, the type of building blocks and the bitwidth of weights/activations. A possible extension is to search for a more efficient architecture under the NAS framework. On the other hand, we may seek a dynamic convolutional building block which adaptively process the input channels on a group manner based on the channel-wise statistics.

6

From image to video: Binarized Conv3D-LSTM model for efficient video inference

Long Short-Term Memory (LSTM) and 3D convolution (Conv3D) show impressive results for many video-based applications but require large memory and intensive computing. Motivated by recent works on hardware-algorithmic co-design towards efficient inference, we propose a compact binarized Conv3D-LSTM architecture called BILLNET, compatible with a highly resource-constrained hardware. Firstly, BILLNET proposes to factorize the costly standard Conv3D by two pointwise convolutions with a grouped convolution in-between. Secondly, BILLNET enables binarized weights and activations via a 3D MUX-OR-gated residual architecture similar to the architecture proposed in Chapter 5. Finally, to efficiently train BILLNET, we propose a multi-stage training strategy enabling to fully quantize LSTM layers. Experiments show that BILLNET can obtain high accuracy with extremely low memory and computational budgets compared to existing efficient Conv3D models.

Contents

6.1 Context **102**
6.2 Related works **102**
6.3 BILLNET **103**
6.3.1 Conv 3D Factorization 103
6.3.2 3D MUX-OR Residual (MOR) block 105
6.3.3 Fully-quantized LSTM 105
6.4 Multi-stage quantization training algorithm **107**
6.5 Experiments **108**
6.5.1 Settings 108
6.5.2 Results 109
6.6 Conclusion and Perspectives **113**

6.1 Context

Video recognition has recently drawn substantial attention due to the success of several Deep Neural Networks (DNNs) [269], [270], [271] and the increasing number of large-scale video datasets [272], [273], [76]. Compared to other tasks like image classification that only relies on spatial data, video recognition is much more complex since it also requires extracting the underlying temporal features in the time direction. Among existing model architectures for spatio-temporal pattern recognition, Conv3D [49] and Recurrent Neural Networks (RNNs), *e.g.* LSTM [86], have demonstrated to be relevant for learning latent spatio-temporal representations. However, these model components exhibit hardware-related drawbacks such as large memory requirements so as a high computational complexity. For instance, Conv3D expands the convolution kernel to the time direction for capturing local temporal features, therefore increasing both the local memory and computational needs by an order of magnitude compared to Conv2D. On the other hands, LSTM is computationally expensive because of its stateful nature, *i.e.*, computing the current features taking into account previous states. Consequently, designing a hardware-compliant Conv3D-LSTM model for embedded inference applications remains a significant challenge.

Motivated by the need for efficient video inference, recent works have been focusing on the design of light-weight architectures [274], [275] or hardware-aware network pruning [276]. Another approach to accelerate the computation during inference and further reduce the hardware-related costs consists in lowering the bitwidth of model's weights and activations [150]. Even though significant works on Quantization Aware Training (QAT) [12], [26] have been done in recent years, they are mostly focused on the quantization of feed-forward Convolutional Neural Networks (CNNs). On the contrary, training fully quantized models that embed recurrent layers like LSTMs remains an important issue. Indeed, quantizing the hidden states of LSTM involves a quantization error that is accumulated throughout the data sequence due to its recurrent nature, hence implying an overall accuracy degradation. It may explain why existing approaches [277], [278] are limited to the use of quantization regarding the LSTM weights while keeping activations in a floating-point representation.

This work thus aims at demonstrating that a fully-quantized model can also be deployed for video-based inference. To this end, we propose a hardware-compliant Conv3D-LSTM architecture called BILLNET on which binarization techniques are applied to further reduce the model size as well as the computational costs. Our main contributions are then:

- A compact Binarized Conv3D-LSTM model architecture with a MUX-OR skip connection mechanism,
- A multi-stage training procedure that provides a fully quantized BILLNET with bitshift normalization (removing additional biases related to batch normalization).

6.2 Related works

Residual connections for video inference

Residual learning such as element-wise addition [24] and attention mechanism [83], [84] is firstly introduced in 2D CNNs in order to increase network expressivity, favor feature reuse while easing the back-propagation for deeper models. Since Conv3D has become a more preferable option for video recognition than its 2D counterpart, it is straightforward that 3D CNNs should adopt residual architecture paradigms like element-wise addition [279], [280] and attention mechanisms [281] to improve model performance. However, it is noteworthy mentioning that these skip connection operations are mostly performed using full-precision arithmetic, which results in additional hardware-related costs, especially in the context of fully-quantized models (targeting a dedicated hardware mapping). In our work, we thus introduce a 3D quantized MUX layer with an OR-gated

connection that allows integrating both an element-wise additional connection and a channel-wise attention-like mechanism, while keeping a binarized data representation. This is an extension of the proposed 2D MUX-OR mechanism in Chapter 5.

Efficient 3D CNN architecture

Several works have recently proposed alternative architectures to alleviate the parameter-heaviness of Conv3D. [282], [283] partly replace Conv3D by 2D convolutions. [284] proposes different variants for the 3D residual block by separating $3\times3\times3$ kernels with $1\times3\times3$ spatial convolutions and $3\times1\times1$ temporal convolutions. On the other hand, [285] processes the temporal features without parameters and multiply-accumulate (MAC) operations by shifting part of the channels along the time dimension. Finally, [274] converts various well-known resource-efficient 2D CNNs such as MobileNet [8], ShuffleNet [9] to 3D CNNs. In this Chapter, BILLNET uses a 3D version of the factorization previously proposed in Chapter 5, including 2 pointwise layers and a grouped Conv3D, without nonlinearity (*i.e.*, activation) inserted in-between. Note that, all layers in this factorization are learnable.

Network quantization

Network quantization [150] reduces the bitwidth of weights and/or activations. In the most extreme case, Binarized Neural Networks (BNNs) [12] restrict both weights and activations to a 1-bit representation using Sign function, this ways reducing the costly full-precision MACs to bitwise operations (*i.e.*, using XNOR gates). Despite tremendous progress during the last few years, there still exists a lack of efforts on model quantization for video inference. Recently, [286] adaptively selects the per-frame optimal bitwidth, conditioned on input data. [287] proposes a binary 3D CNN constraining weight and activation values to 0 or 1. Besides, existing approaches [288], [278], [289], [89] mostly focus on the compression of LSTM for language or speech models only. To the best of our knowledge, there is no prior works on fully-quantized LSTM in the context of video inference. Our work tries to fill in this gap by proposing a multi-stage training algorithm to provide a fully-quantized Conv3D-LSTM model.

6.3 BILLNET

Figure 6.1 depicts the top-level view of BILLNET that involves integer-only MACs and bitwise operations such as 2-input multiplexers and OR gates. This model takes as input a sequence of 16 frames with a spatial resolution of 96×128. BILLNET contains a spatio-temporal feature extractor with a Conv3D part to extract spatio-temporal features between adjacent frames, and a LSTM part to keep track longer-term temporal dependencies. In this section, we first focus on the 3D Convolution Factorization (CF), then on the custom 3D MUX-OR Residual (MOR) block, finally on the LSTM weights and activations quantization.

6.3.1 Conv 3D Factorization

The core building block of BILLNET is a light-weight factorization, namely CF, consisting of 2 pointwise layers (filter size: $1\times1\times1$) and a g-grouped convolution (filter size: $3\times3\times3$). Unlike the building block of 3D ResNext in [279], there is no nonlinearity (*e.g.*, normalization, activation) between these layers. The number of output channels C_o of each CF is defined by the parameter n (*i.e.*, $C_o \in \{n, 2n, 4n\}$). Note that, the number of channels in low dimension of every CF is set to $C_i/2$.

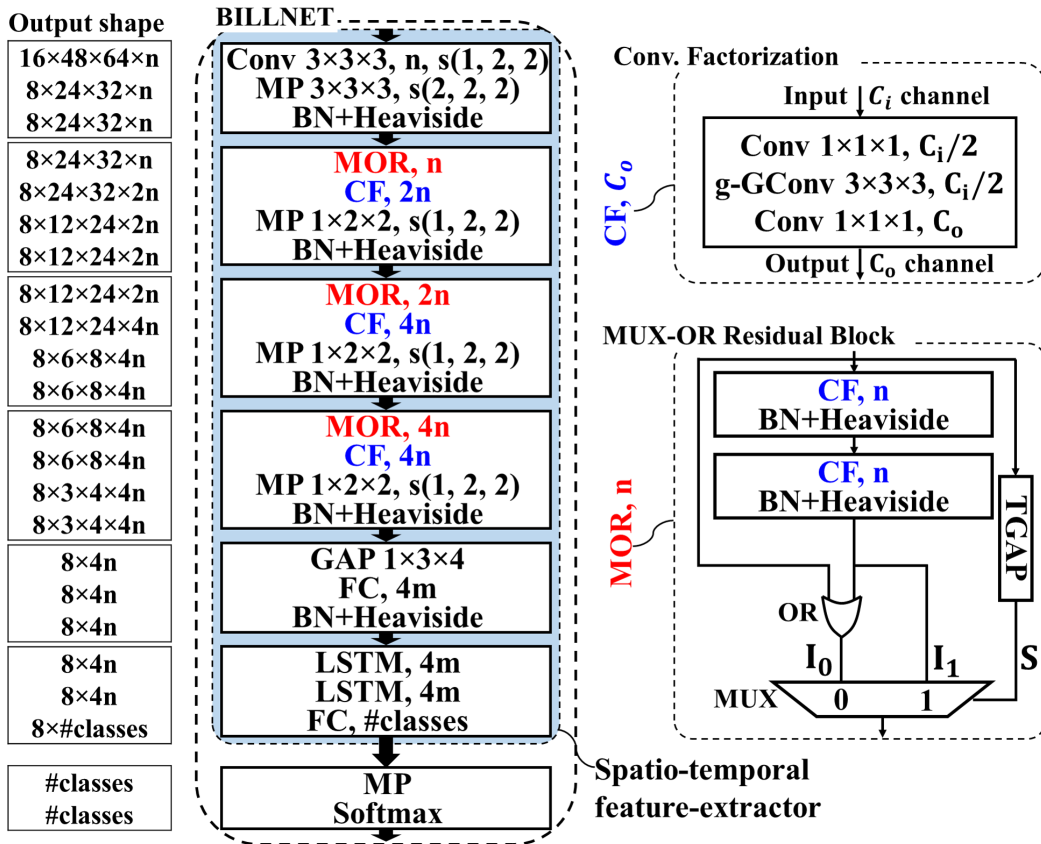


Figure 6.1: Top-level architecture description of BILLNET with Convolutional Factorization (CF) and MUX-OR Residual (MOR) Block. Here n is the parameters controlling the number of output feature maps, g-GConv is Grouped Convolution with g groups, MP and GAP stand for Max Pooling and Global Average Pooling.

6.3.2 3D MUX-OR Residual (MOR) block

In BILLNET, we binarize all the activations using the Heaviside function $H(x) = 1_{\{x>0\}}$, where 1 is the indicator function. During the backward pass, the Straight-Through-Estimated gradient (STE [290]) $\frac{\partial H}{\partial x} = 1_{\{|x|\leq 1\}}$ is used. For the sake of genericity, we then define the Clipped Identity:

$$\mathrm{Clip}(y) = \max\left(-1, \min\left(1, y\right)\right) \tag{6.1}$$

with STE gradient $\frac{\partial \mathrm{Clip}}{\partial y} = 1$. Concretely, applying this function to the sum of x_1, x_2 will obtain the same binary output $\{0,1\}$ as performing the logical **OR** operation, *i.e.*, $\mathrm{Clip}(x_1 + x_2) = x_1 \vee x_2$. Therefore, we employ the Clipped Identity to keep the data in binary representation. Let us denote $\mathbf{I}_0, \mathbf{I}_1 \in \mathbb{R}^{T\times h\times w\times n}$ as the output of this OR operation and the second Heaviside where T, h, w, n are the time steps, height, width and number of channels; $\mathbf{S} \in \{0,1\}^{T\times 1\times 1\times n}$ as the binary control signal. The 2-MUX layer can be described as:

$$\mathbf{MUX}(\mathbf{I}_0, \mathbf{I}_1; \mathbf{S}) = \mathbf{I}_1 \odot \mathbf{S} + \mathbf{I}_0 \odot (\mathbf{1} - \mathbf{S}) \tag{6.2}$$

where $\odot$ is a channel-wise multiplication. The control signal $\mathbf{S}$ embeds a parameter-free channel attention through a Thresholded Global Average Pooling (TGAP). TGAP simply consists of a channel-wise Average Pooling (AP) with filter size and strides of $1 \times h \times w$, followed by a binarization $T(x) = 1_{\{x>0.5m\}}$ where m is first set to the maximum of the layer-wise AP's tensor outputs for the full-precision mode, then being replaced by 1 for the final quantized model version (see section 6.4). When deploying the quantized model, this operation can be implemented via a bit-count followed by an integer-to-integer comparison. This architecture allows the input of each MOR to control the operation of the MUX gate in a channel-wise manner. In details, if the input feature map is dominated by zero values, the OR skip-connection will be performed (Fig. 6.2a). Otherwise, the MUX gate will simply keep the output of the second Heaviside (Fig. 6.2b). Consequently, this mechanism intrinsically balances zero and one latent values throughout the network, this without any additional regularization.

6.3.3 Fully-quantized LSTM

LSTM [86] is commonly used because of its capability to capture long-term dependencies within sequences. The basic structure of a cell in LSTM can be described as follows:

$$i_t = \mathrm{sigmoid}\left(W^i.[x_t, h_{t-1}] + b^i\right) \tag{6.3}$$

$$f_t = \mathrm{sigmoid}\left(W^f.[x_t, h_{t-1}] + b^f\right) \tag{6.4}$$

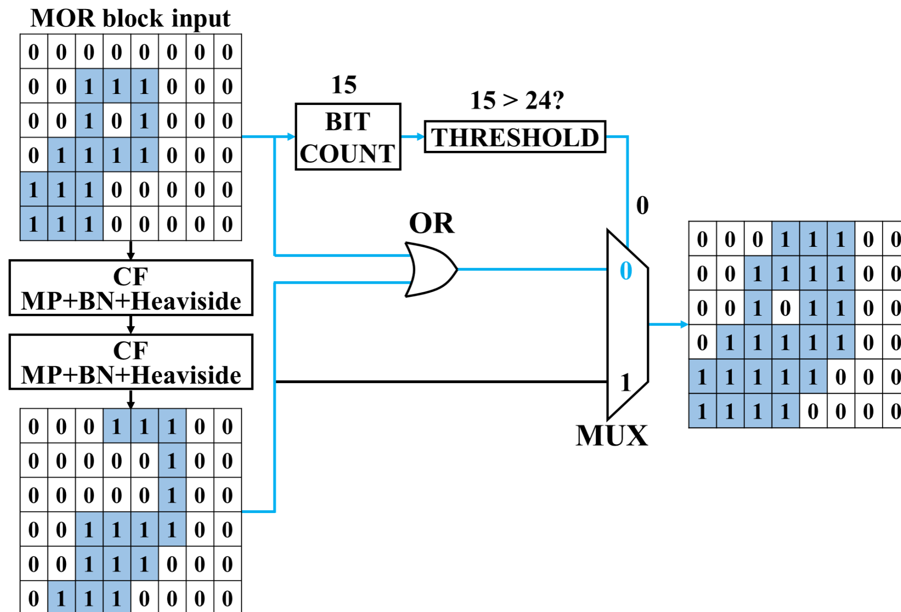
$$o_t = \mathrm{sigmoid}\left(W^o.[x_t, h_{t-1}] + b^o\right) \tag{6.5}$$

$$\tilde{c}_t = \tanh\left(W^c.[x_t, h_{t-1}] + b^c\right) \tag{6.6}$$

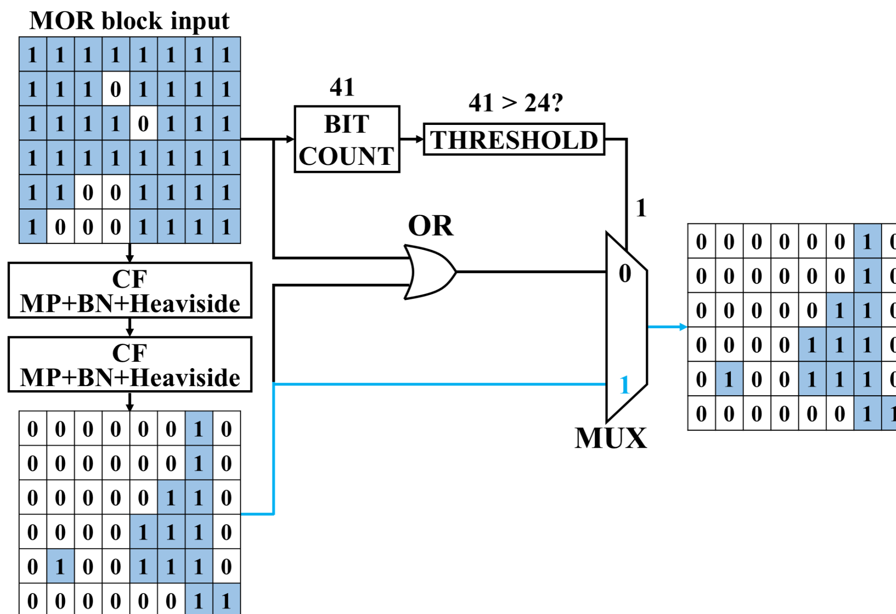
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{6.7}$$

$$h_t = o_t \odot \tanh(c_t) \tag{6.8}$$

Eqs. 6.3- 6.6 defines the input gate, forget gate, output gate and candidate memory, respectively. Temporal information is transferred along time steps via c_t and h_t (Eqs. 6.7- 6.8).



(a) Case of an input dominated by zeros.



(b) Case of an input dominated by ones.

Figure 6.2: The operation of the channel-wise MUX gate with feature maps extracted during inference of a test sample. The TGAP is implemented by a bitcount followed by an integer-to-integer comparison, where the threshold is equal to one half of the spatial resolution ($\frac{6 \times 8}{2} = 24$).

LSTM weight binarization

Let us denote n_i, n_o as the dimension of input and output sequences, therefore we have $[x_t, h_{t-1}] \in \mathbb{R}^{n_i+n_o}$. In order to simplify the hardware mapping, all biases are removed. Since the projections will increase the dynamic range of the data, the weight binarization of LSTMs are done with a scaling factor as follows:

$$\text{SSign}(w) = \frac{3}{\sqrt{n_i + n_o}}\text{Sign}(w) \tag{6.9}$$

The scaling factor $\frac{3}{\sqrt{n_i+n_o}}$ is chosen as a compromise between scaling the propagated gradients of the activation functions and matching the bipolar distributions of the later quantized **sign** and **heaviside** activations. During backward pass, we still employ the same STE gradient as [12]. This scheme is applied to all 4 kernels of the LSTM layers.

LSTM activation quantization

Whereas quantizing weights is almost straightforward, it becomes much more complex in the case of activations in LSTM due to its internal structure. We replace all **sigmoid** activations in Eqs. 6.3-6.5 by the Heaviside function like in 6.3.2 and **tanh** in Eq. 6.6 by a strict Sign. Since the addition in Eq. 6.7 will increase the dynamic range of data, we will keep c_t values within $\{-1,0,+1\}$ by using the already introduced Clipped Idendity (Eq. 6.1). Consequently, the **tanh** activation applied to the ternary c_t in Eq. 6.8 is simply removed, which allows obtaining the output h_t in a ternary representation $\{-1,0,1\}$. Figure 6.3 depicts the computational graph of the proposed Quantized LSTM (QLSTM) according to the aforementioned scheme. To better visualize the quantization aspect, we also display the dynamic of internal intermediate values along the connection lines.

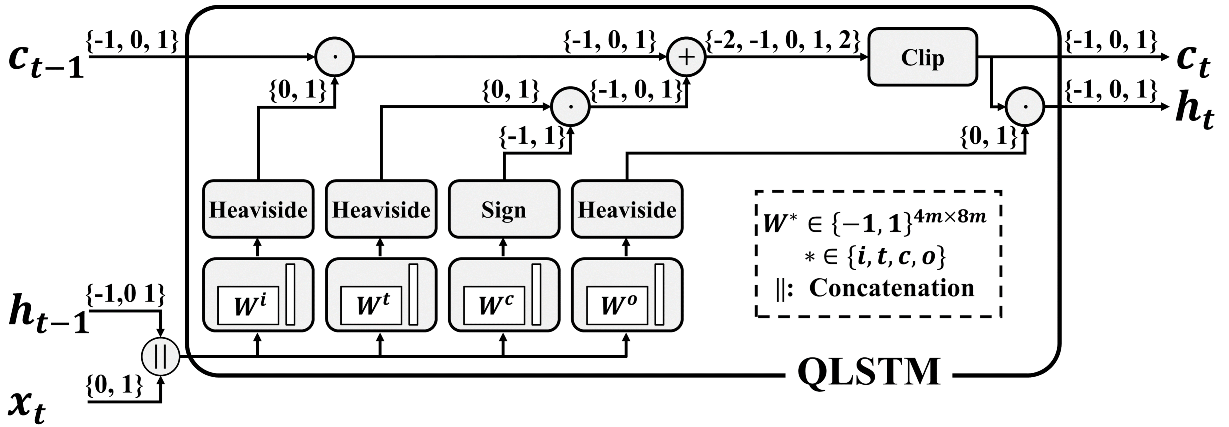


Figure 6.3: Computational graph of the proposed Quantized LSTM.

6.4 Multi-stage quantization training algorithm

This work targets a fully-quantized model, including weights, quantizations, hidden states of LSTM and even the Batch Normalizations (BN [23]). In order to limit the model performance degradation, we apply a multi-stage training procedure in which, we iteratively replace elements of BILLNET by its corresponding quantized version, intrinsically fine-tuning the model to retain the accuracy.

1st **stage: Training full-precision model.** We firstly train the 32-bit model with ReLU activations in Conv3D part and use it as a proper model initialization.

2nd **stage: Quantizing all weights.** We keep the full-precision activations and binarize weights using Sign [12] for the Conv3D part and SSign for the LSTMs (*c.f.* 6.3.3). Similarly, for the last

Dense layer which reduces the dimension of data from $4m$ to #classes, we also apply a Scaled Ternarization (STern) to its weights:

$$\text{STern}(w) = \frac{1}{\sqrt{4m}}\text{Tern}(w) \tag{6.10}$$

where the Tern function is originally introduced in [13]. It is worth mentioning that in hardware implementation, we can simply get rid of these scaling factors, since they do not affect the results of the later Sign/ Heaviside activations and the Argmax operations.

3rd **stage: Quantizing Conv3D activations.** In this stage, we replace all ReLU by the Heaviside activations while keeping the LSTM activations at full-precision.

4th **stage: Removing BNs.** The full-precision affine transform of BN remains an obstacle for model hardware deployment, in particular for 3D CNNs where the data is 4D tensors with an additional temporal dimension. Therefore, we approximate the scaling factors of BN layers in a power-of-2 fashion, which advantageously corresponds to the bitshift operation. Denoting μ, σ^2 as the moving mean and the moving variance of BN after the second stage, at inference time, the BN processes the input x to provide the output y as follows:

$$y = \gamma\frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}} + \beta \equiv \hat{\gamma}x + \hat{\beta} \tag{6.11}$$

where $\hat{\gamma} = \frac{\gamma}{\sqrt{\sigma^2+\varepsilon}}$ and $\hat{\beta} = \beta - \frac{\gamma\mu}{\sqrt{\sigma^2+\varepsilon}}$ are equivalent to the scale and the offset vectors. We replace all BN layers by the following offset-free BitShift Normalization (BSN):

$$y = 2^{\lfloor log_2|\hat{\gamma}|\rceil}x \tag{6.12}$$

Note that in BILLNET, each BN layer is followed by a Heaviside activation. When replacing the BN by BSN, since the equivalent scaling factors are always positive, they will simply keep the sign of data unchanged, therefore, they have no impact to the outcomes of the later Heaviside function. Consequently, all BSNs in BILLNET do not need to be explicitly implemented. Unlike the BSN in Chapter 4, we keep different scaling factors for different channels instead of using a unique scaling. This allows to retain more representation power for the quantized model and thus reduce the accuracy degradation.

5th **stage: Quantizing LSTM activations.** Finally, we replace the **sigmoid** and **tanh** used in the LSTM layers (as discussed in subsection 6.3.3). The model is now fully quantized with mostly all weights and activations are binarized, except for the ternary output of the QLSTMs and the ternary weights of the last Dense layer.

6.5 Experiments

6.5.1 Settings

Data pre-processing: We consider 16-frame sequences with a resolution of 96×128 for training and testing. The Jester Dataset V1 [76] is a large-scale hand gesture recognition dataset composed of video clips with a variable number of frames (from 12 to 70). In particular, most of the sequences have between 30-40 frames. Therefore, in order to properly fit the target temporal dimension of 16, we first apply a $2\times$ temporal down-sampling for sequences of more than 24 frames. This allows us to capture all the hand gestures from end to end. In addition, if the resulting video contains less than 16 frames, we symmetrically repeat the first and the last frames, otherwise, we randomly select the initial time index for the first frame.

Training stages: To train the models, we employ the Adam optimizer [291] and the standard Categorical Cross-Entropy (CCE) loss, with a mini-batch size of 40. For each stage, the learning rate firstly remains unchanged, before being exponentially decayed during the last 50 epochs, with a fixed decay rate of 0.85. Table 6.1 shows more details about the training procedure and the settings for each training stage.

Table 6.1: Training and optimization settings of BILLNET

Taining stage	Initialized learning rate	Epochs	Exponentially decay	Decay rate
First stage	5×10^{-4}	100	After 50-th epoch	0.85
Second stage	3×10^{-4}	80	After 30-th epoch	0.85
Third stage	3×10^{-4}	80	After 30-th epoch	0.85
Fourth stage	2×10^{-4}	80	After 30-th epoch	0.85
Fifth stage	1×10^{-6}	80	After 30-th epoch	0.85

Hardware-related metrics: We measure the model hardware efficiency in terms of the memory cost using weight-related memory (model size), and the computational complexity using Bit-OPerations (BOPs [247]). This allows us to assess the number of parameters and MACs along with the precision of weights and activations. Conventionally, we assume that each full-precision weight and activation requires 32 bits.

6.5.2 Results

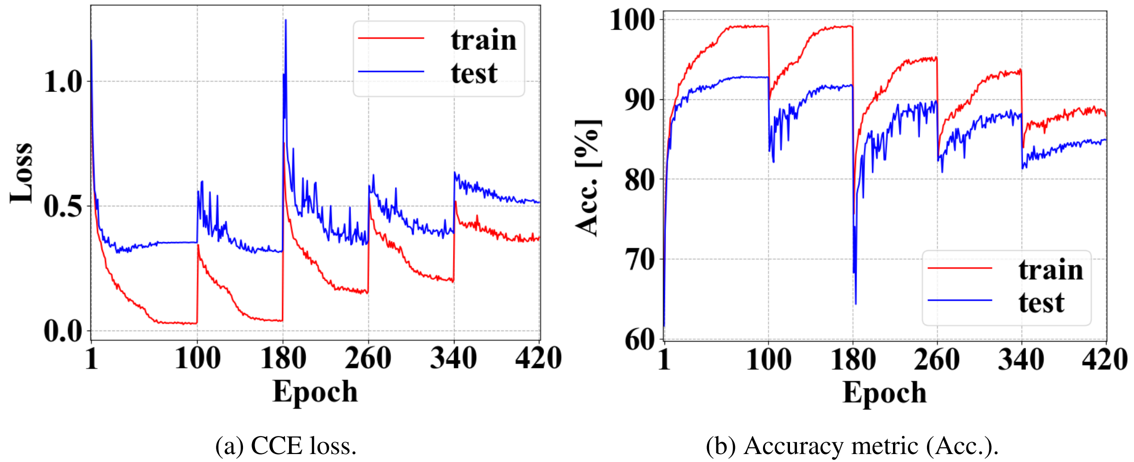
We denote the proposed model with $n = 32k$ as BILLNET $k\times$. Figure 6.4 reports the evolution of the accuracy and train/test losses. Each training stage (denoted from S1 to S5) allows retaining the accuracy despite introducing quantization effects, even if the remaining gap is significant when quantizing the LSTM activations (S5). Figure 6.5 depicts the accuracy/efficiency compromise of BILLNET and 3D efficient models from [274]. Since [274] does not show the model size and the number of GBOPs for Jester dataset, we compute these values based on their trained models and code (publicly available [1]).

It is clear that all quantized versions (S2 to S5) of BILLNET stay on the optimal top-left corner, enabling various types of hardware/accuracy compromises. Table 6.2 reports the performance of BILLNET 2× with the specific configuration of g=4 and m=$n/2$ compared to other resource-efficient models. Please note that since BILLNET does not reduce the temporal dimension (except for the first convolution) in order to cap the inference output latency, the full-precision (S1) model involves a higher computational cost than MobileNet and ShuffleNet versions. However, when quantizing the weights and activations, we can advantageously reduce the weight-related memory and computational complexity with at least one order of magnitude. In particular, compared to a 3D-MobileNet V1, the weight-quantized BILLNET 2× (S2) provides a higher accuracy (+0.8%) with a significantly smaller model size (1%) and lower computation needs in terms of #GBOPs (17%). Besides, GBOP reduction between S3 and S4 (8.53 → 6.39 GBOPs) shows that it is crucial to replace the BNs by bitshifts, to fully benefit from a hardware simplification in practice. The quantization of LSTM activations has limited impacts on the total number of GBOPs while significantly decreasing the performance from S4 to S5 (3.78% loss). However, it is still highly relevant

[1] https://github.com/okankop/Efficient-3DCNNs

considering a dedicated hardware mapping, designed to handle only bit-wise and bit-count operations.

Figure 6.6 exhibits a class-temporal response of size 8×27 for a *Pulling Two Fingers In* gesture example. It can be easily observed that the most highlighted zones in this map highly correlate to the temporal positions of the most informative frames of the input video. Besides, similar classes (along the vertical axis) such as *Pulling Hand In* and *Sliding Two Fingers In* also exhibit high values at the same columns. This demonstrates the learning capability of our spatio-temporal feature extractor although being fully quantized at binary/ternary precision.



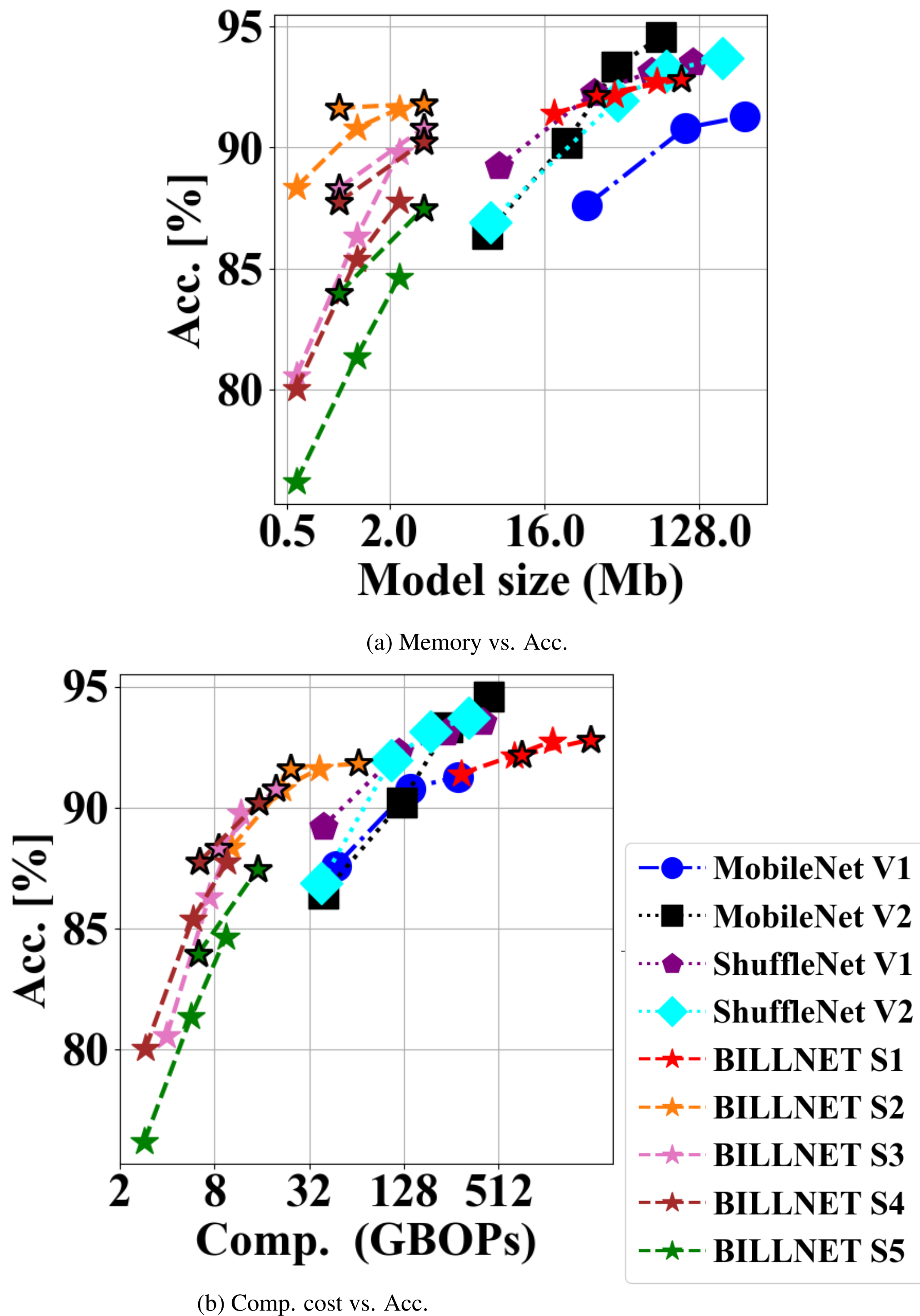
(a) CCE loss.

(b) Accuracy metric (Acc.).

Figure 6.4: Training curves (CCE loss and accuracy) of BILLNET 2× throughout all 5 training stages.

Table 6.2: Comparison of resource-efficient models on Jester hand gesture dataset, weight-related memory (model size), and bitwidth-aware computational complexity (GBOPs). Results reported here are for BILLNET with g=4, n=64 and m=32.

Model	Model size (Mb)	Comp. (GBOP)	Acc. (%)
3D-ShuffleNet V1 [274]	31.04	119.25	92.27
3D-ShuffleNet V2 [274]	42.56	106.09	91.96
3D-MobileNet V1 [274]	106.56	141.02	90.81
3D-MobileNet V2 [274]	42.24	243.18	93.34
BILLNET 2× -S1	32.23	718.89	92.18
BILLNET 2× -S2	1.01	24.54	91.64
BILLNET 2× -S3	1.01	8.53	88.33
BILLNET 2× -S4	1.01	6.39	87.75
BILLNET 2× -S5	1.01	6.34	83.97



(a) Memory vs. Acc.

(b) Comp. cost vs. Acc.

Figure 6.5: Weight memory (Mb) and computational costs (GBOPs ~ 10^9 BOPs [247]) versus Top-1 Accuracy. Edge-less stars are for BILLNET with g=2 and m=n, edged stars are for m=$n/2$ and $g = \frac{n}{16}$ (with $n \in \{64, 128\}$).

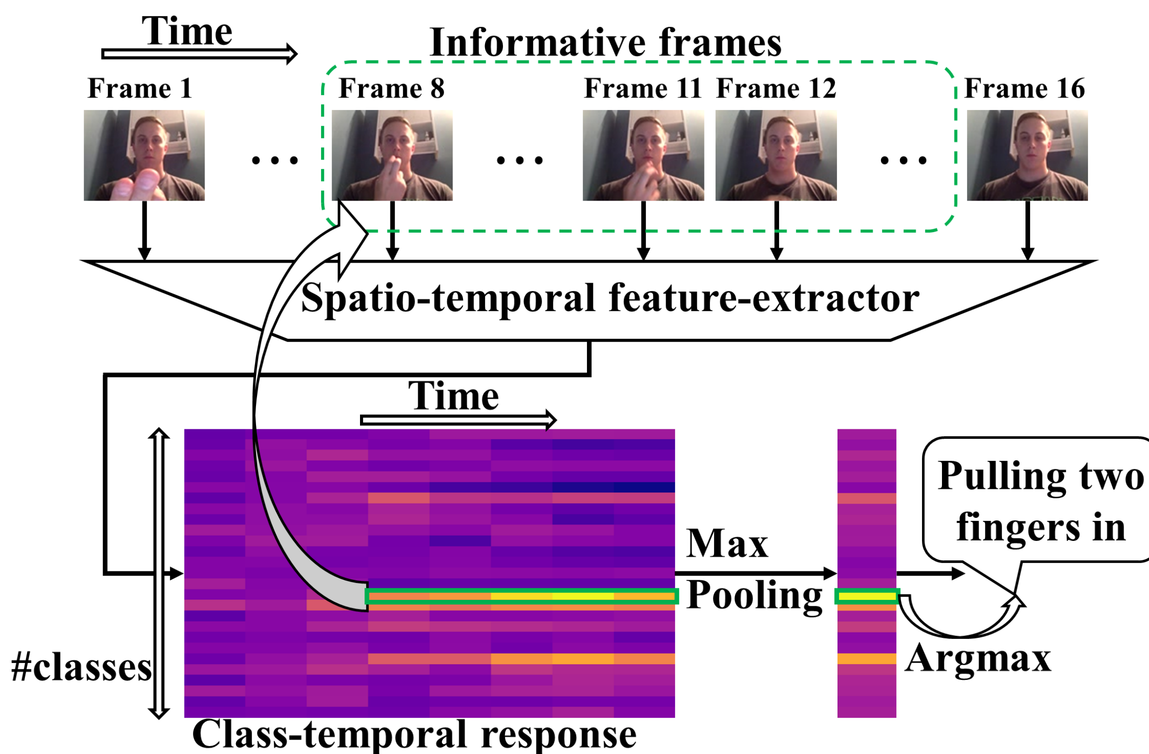


Figure 6.6: Class-temporal BILLNET output responses for a Jester test sample labeled as a *Pulling Two Fingers In* gesture. Highlighted time/class positions with maximum values correlate to the most informative frames of the input video. Besides, similar classes (*Pulling Hand In*, *Sliding Two Fingers In*) also exhibit high values at the same columns.

6.6 Conclusion and Perspectives

In this chapter, we introduced a hardware-tiny model for video inference called BILLNET, which involves binary/ternary weights and activations. BILLNET integrates 3D MUX-OR skip connections and a Conv3D factorization to limit the memory and computation needs. Thanks to a 5-stage training procedure, BILLNET offers different hardware-algorithmic trade-offs with significantly reduced model size and #GBOPs (together with its inherent arithmetic simplifications), while providing an on-par accuracy compared to previously published compact models. More importantly, we aim at designing a hardware-compliant network suitable for later implementation on FPGA or ASIC-based platforms. For this purpose, the fully-quantized BILLNET (S5) can fit a hardware supporting only bit-wise and bit-count operations.

Our future works is to revise the last training stage approach to reduce the accuracy loss due to LSTM activations quantization. This can be done by gradually adjusting the sigmoid and the tanh activation functions inside the LSTM layers, such that these functions become more sharpen during the training and at the end they converge to the Heaviside and the Sign, respectively. This may likely ease the training while reducing the mismatching effect due to the use of STE gradient. Another option consists in using the knowledge distillation training to force the training of quantized LSTM to emulate its full-precision version.

7

Model compression via weight generation network and PRNG

Unlike previous chapters which deeply relate to the quantization as the main compression technique, in this chapter, we mainly aim at reducing memory needs via the use of weight generation network and PRNG. It may enable reducing as well the number of memory accesses and the overall computational complexity but to a lesser extent. This approach is a more exploratory work compared to previous chapters. We propose to take advantages of a PRNG in order to indirectly produce Primary Model (PM) weights –in an on-line fashion– when shaped through the weight generation network. Besides, the possible interest of hardware implementations of such a topology component highly relates to the complexity of involved generative models and the type of PRNG.

Contents

7.1 Hypernetworks: a bridge between random input and DNN weights 116

7.2 Practical Weight generation network designs 118

7.2.1 Weight generation network for FC layer 118

7.2.2 Conv2D layer with weight generation network 120

7.3 Experiments 123

7.3.1 MLP on MNIST 123

7.3.2 VGG-7 on CIFAR-10 124

7.4 Discussion 124

7.4.1 Configurations of the PRNG 124

7.4.2 Weight generation network architecture 126

7.4.3 Application 126

7.5 Conclusion and Perspective 127

7.1 Hypernetworks: a bridge between random input and DNN weights

In section 2.3.2 we have reviewed different model compression techniques, including weight generation networks, also termed as *hypernetworks* [21]. These are auxiliary model which aims at generating the weights of layers in the primary/main network rather than directly outputting the prediction for a specific task. The weight generation process in a layer may be done in a dynamic fashion, *i.e.* taking into account the information of the input data of that layer [22], [181]. On the other hand, static weight generation [21], [292] consists in applying a transform mapping the layer's low-dimensional embedding to the weight's high-dimensional space. For instance, [21] makes use of learnable embedding which are optimized along with other parameters of the hypernetwork. On the contrary, [292] draws the embedding from a uniform/gaussian distribution.

In this chapter, we also present some preliminary studies on the static weight generation with pseudo-randomly generated embeddings. Generally speaking, this proposes to use a PRNG (*e.g.*, Cellular Automaton) combined with a hypernetwork to reduce the overall memory footprint of the primary model (PM). Figure 7.1a illustrates a standard layer with input data $\boldsymbol{X}$, output data $\boldsymbol{Y}$ and layer weights $\boldsymbol{W}$. The following equation describes the linear operation of this layer:

$$\boldsymbol{Y} = h(\boldsymbol{X};\boldsymbol{W}) \tag{7.1}$$

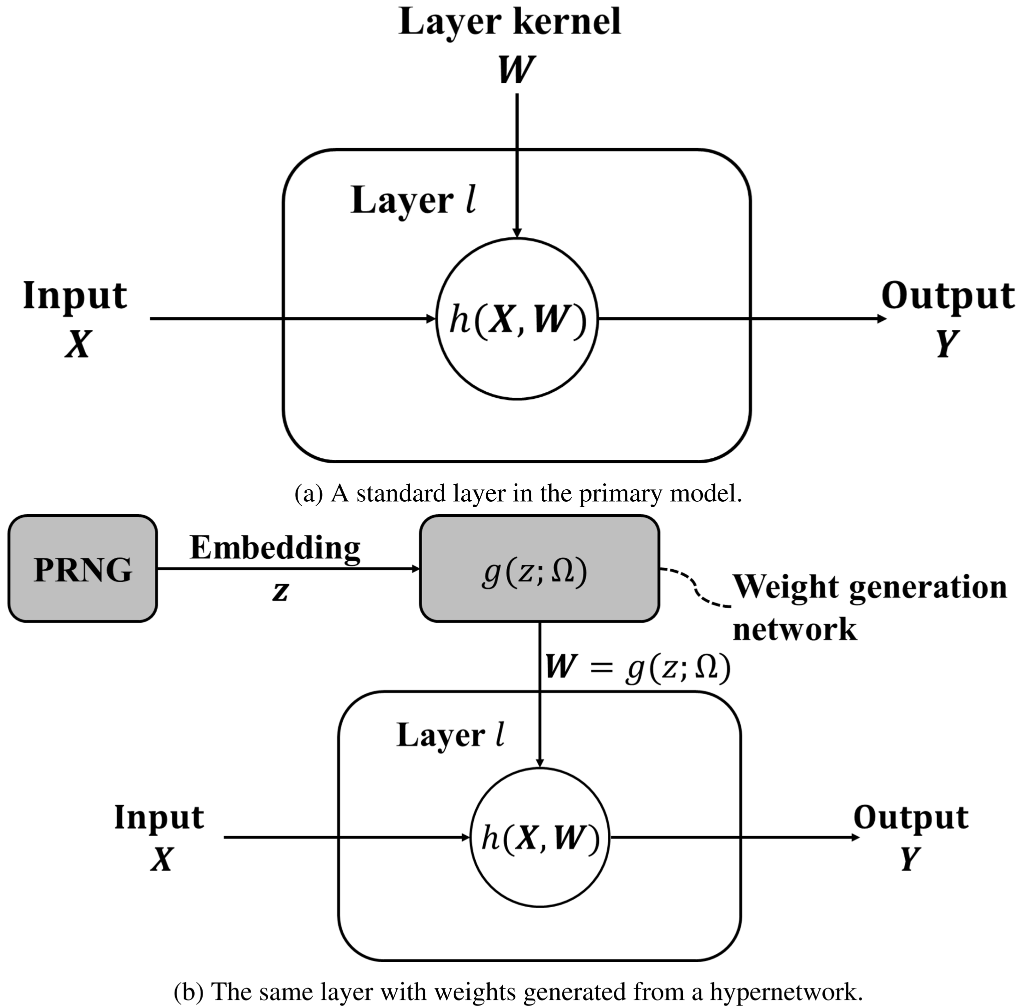
The operation $h()$ of this layer may be either of fully-connected, convolution or any kind of layers and the PM layer weights $\boldsymbol{W}$ have to be saved in memory. Figure 7.1b presents the static weight generation mechanism where layer weights $\boldsymbol{W}$ are not stored in-situ but generated from a hypernetwork. The PRNG generates on-the-fly the sequences of embedding $\boldsymbol{z}$ of low dimensionality which are fed into the hypernetwork which contains learnable parameters $\boldsymbol{\Omega}$. This weight generation network (or weight net for short) is therefore learned (through $\boldsymbol{\Omega}$, depending on $\boldsymbol{z}$) to produce proper weights $\boldsymbol{W}$ used by the primary model for the targeted task. It is required that the size of $\boldsymbol{\Omega}$ is much smaller than that of $\boldsymbol{W}$ in order to efficiently compress the overall model and alleviate the memory bottleneck. Another specification is that the additional operations related to the hypernetwork $g(\boldsymbol{\Omega},\boldsymbol{Z})$ should be of low computational complexity as well, so that it does not increase much top-level hardware implementation costs. The output of this layer is described as follows

$$\boldsymbol{Y} = h\left(g(\boldsymbol{\Omega},\boldsymbol{z}),\boldsymbol{X}\right) \tag{7.2}$$

Assuming that $\boldsymbol{z} \in \mathbb{R}^{d_z}$ and $\boldsymbol{X} \in \mathbb{R}^{d_W}$ (when the weight $\boldsymbol{W}$ is vectorized), where $d_z \ll d_W$. It is easily to see that a direct linear projection from $\mathbb{R}^{d_z}$ to $\mathbb{R}^{w}$ requires $d_z d_W$ parameters, which is d_z times higher than the size of $\boldsymbol{W}$. Therefore, it is necessary that the PM weights $\boldsymbol{W}$ are partitioned into several sub-weights $\boldsymbol{W}_i$, each of them are generated from the hypernetwork $g(\boldsymbol{W}_i,\boldsymbol{z}_i)$ given a embedding $\boldsymbol{z}_i$ in a sequential fashion, hence easing the hardware implementation. This also implies a dependence between these sub-weights as they share a single hypernetwork. Concretely, in the case of FC layer, the 2D weight matrix $\boldsymbol{W}$ is partitioned into several 1D projection vector, wheareas for Conv2D, the 4D weight tensor may be partitioned into a collection of 3D filters. It is noteworthy pointing out that the weight net can be learned to fulfill various requirements, being adapted to the layer type and the input/output shapes. It is widely compatible from basic Dense layers, Convolutional layers to RNNs, making the approach highly generic compared to other state-of-the-art of model compression approaches.

Using weight generation network may introduces several advantages:

- Conventional models require the storage of all weights in memory and usually represents the main bottleneck preventing DNNs from being deployed in tiny low-memory systems. Besides this, when model size is large, one usually needs additional DRAM to save memory but being costly in terms of power consumption related to weights loading.



(a) A standard layer in the primary model.

(b) The same layer with weights generated from a hypernetwork.

Figure 7.1: A standard layer where the weights $\boldsymbol{W}$ is stored in memory, and the corresponding layer with weight generation network.

- This power consumption reduction is expected to cover the additional costs introduced by generating and performing $g()$. In particular, when the operations of $f()$ and $g()$ are both linear, it enables to change their order of execution, *i.e.*, performing $f()$ in the low-dimensional latent space. Consequently, this results in a reduction of the overall number of operations, the power consumption and the latency as well.

- Furthermore, on top of this, we can still apply other model compression techniques such as quantization, pruning or dynamic networks to the hypernetwork which allows us to reduce more drastically hardware latency during the processing/inference.

In this chapter, we present some propositions for the hypernetwork to generate the weights of FC and Conv2D layers in an on-line fashion. We also discuss about the choices of the PRNG and the configurations of the hypernetworks.

7.2 Practical Weight generation network designs

7.2.1 Weight generation network for FC layer

In Figure 7.2, we have the description of one possible proposition applied to a simple PM Dense layer in combination with a "Vanilla" weight generation network, namely *WeightGenNet FC*. A typical Dense layer is characterized by n_l input neurons and n_{l+1} output neurons, meaning that it receives input $\boldsymbol{y} \in \mathbb{R}^{n_l}$ and outputs $\boldsymbol{y} \in \mathbb{R}^{n_{l+1}}$. Here for convenience, we get rid of bias parameters knowing that they can be easily fused into the kernel matrix $\boldsymbol{W}$. The layer operation $h(.,.)$ is a matrix-to-vector multiplication as follows:

$$\boldsymbol{y} = h(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{W}^T \boldsymbol{x} \tag{7.3}$$

where $\boldsymbol{W} \in \mathbb{R}^{n_l \times n_{l+1}}$. The number of parameters in this case is $n_l n_{l+1}$. We can say that the kernel matrix stands for n_{l+1} projections with the support size of n_l. If both input and output are in a high-dimensional space, we thus need a large memory to store these weight parameters.

WeightGenNet FC proposes a basic solution to reduce memory needed for storing the parameters of FC layers. The PRNG will generate on-the-fly n_{l+1} pseudo-randomly generated vectors $\boldsymbol{z}_i$, $i = 1 \ldots n_{l+1}$, each has length of $m \ll n_{l+1}$. These values may be floating points or fixed points (even binary values). Each vector $\boldsymbol{z}_i$ is then fed into a hypernetwork whose operation $g(.,.)$ is also equivalent to a Dense layer with m input neurons and n_l output neurons, with a nonlinear operation $\sigma()$ (*e.g.*, normalization and/or nonlinear activation). This model, with the kernel matrix $\boldsymbol{\Omega} \in \mathbb{R}^{n_i \times m}$, is learned to convert each random vector $\boldsymbol{z}_i$ into a projection of the main kernel matrix $\boldsymbol{W}$. The whole n_{l+1} projections will be performed in the same manner from different embeddings. This way, the main kernel matrix $\boldsymbol{W}$ is on-line generated from a much smaller kernel matrix $\boldsymbol{\Omega}$ with only mn_l learnable parameters. Denoting each 1D projection vector as $\tilde{\boldsymbol{w}}_i \in \mathbb{R}^{n_i}$, the following equation describes how these vector are generated from the hypernetwork

$$\tilde{\boldsymbol{w}}_i = g(\boldsymbol{\Omega}, _i) = \sigma(\boldsymbol{\Omega} \boldsymbol{z}_i) \tag{7.4}$$

The whole main layer's kernel matrix W can be represented as follows:

$$\boldsymbol{W} = \begin{bmatrix} \tilde{\boldsymbol{w}}_1^T \\ \tilde{\boldsymbol{w}}_2^T \\ \ldots \\ \tilde{\boldsymbol{w}}_{n_{l+1}}^T \end{bmatrix} \tag{7.5}$$

This ways, the output can be generated on-the-fly with the term $\tilde{\boldsymbol{w}}_i^T \boldsymbol{x}$. Each element of y is computed from each embedding $\boldsymbol{z}_i$ generated from the PRNG. This on-line processing strategy is depicted in Figure 7.2, where the elements in red stand for the current "iteration" including random embedding generation – projection weights computation – output computation. The benefit of this realization is that we no further need to store all $n_l n_{l+1}$ parameters of the main layer weights $\boldsymbol{W}$. Instead, we only need to keep in memory the smaller sized matrix $\boldsymbol{\Omega}$ containing mn_l parameters. The ratio between the number of learnable parameters in WeightGenNet FC and regular FC is

$$\text{CR}_{params} = \frac{mn_l}{n_l n_{l+1}} = \frac{m}{n_{l+1}} \tag{7.6}$$

Choosing a small number of $m \ll n_{l+1}$ results in a low compression rate or the reduction in high memory needs. The number of MACs of a standard Dense layer is $n_{l+1} n_l$ and that of this "Vanilla" hypernetwork Dense layer is $n_l m n_{l+1} + n_l n_{l+1}$, where the additional term $n_l m n_i$ is due to n_{l+1} times reuse of the projection matrix $\boldsymbol{\Omega}$. The MAC ratio is thus:

$$\mathrm{CR}_{MAC} = \frac{n_l m n_{l+1} + n_l n_{l+1}}{n_l n_{l+1}} = 1 + m \tag{7.7}$$

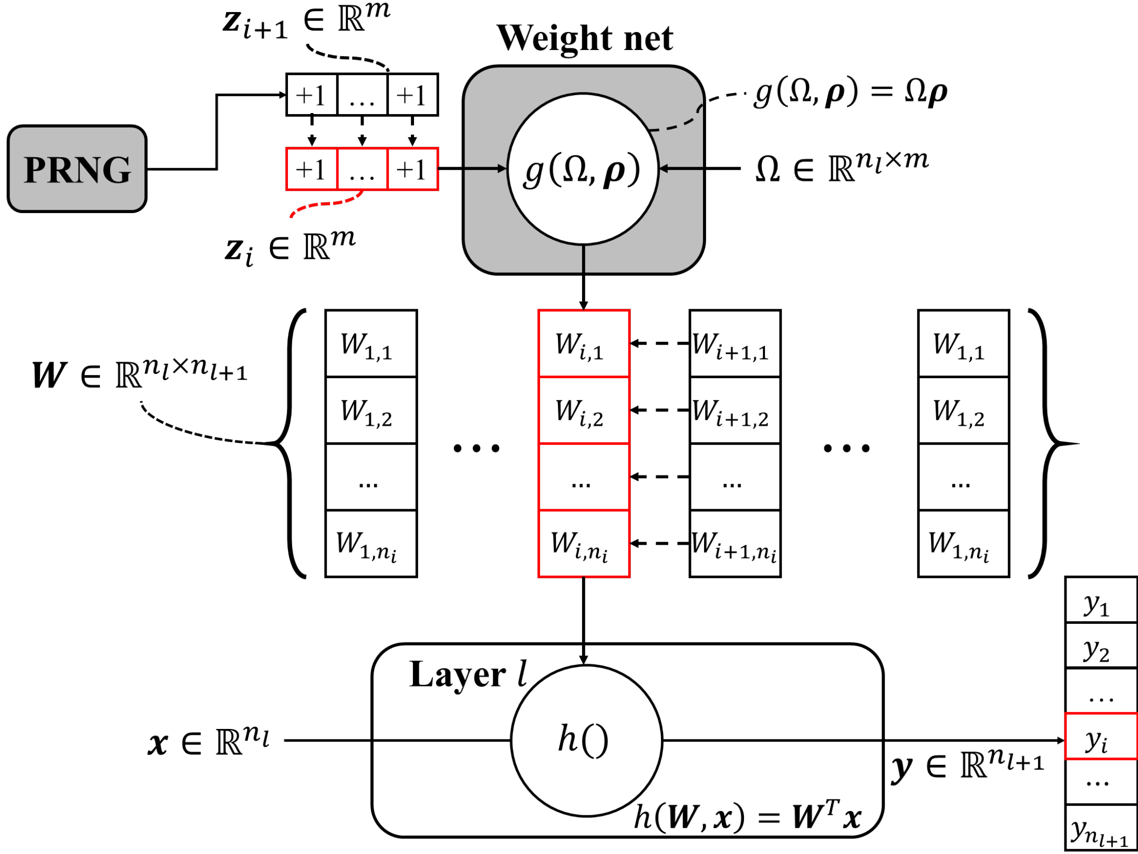


Figure 7.2: Fully-connected layer with weight generated network.

This ratio is unfortunately high, even with a small number of m. This, in terms of computational complexity becomes indeed the main obstacle of our realization. However, as mentioned in the second benefits of our invention, when both main operation $f()$ and auxiliary operation $g()$ are linear operators (*i.e.* not involving activation functions), we have the possibility of changing their order of execution, so that the main operation $f()$ can be performed in the latent space of low dimension. In details, from the equation above, if $\sigma(x) = x$, the output of this layer can be computed as follows:

$$y = \begin{bmatrix} \tilde{w}_1^T \\ \tilde{w}_2^T \\ \dots \\ \tilde{w}_{n_{l+1}}^T \end{bmatrix} x = \begin{bmatrix} (\Omega z_1)^T x \\ (\Omega z_2)^T x \\ \dots \\ (\Omega z_{l+1})^T x \end{bmatrix} = \begin{bmatrix} z_1^T \\ z_2^T \\ \dots \\ z_{l+1}^T \end{bmatrix} \Omega^T x \tag{7.8}$$

In this formulation, the input data x is firstly projected into a latent space of dimension m by the function $\tilde{f}(\Omega, x) = \Omega^T x = \tilde{y}$. Then the result $\tilde{y}$ is sequentially projected by n_{l+1} vectors z_i that are generated on-the-fly, to obtain the final output y. Figure 7.3 describes how this formulation can be implemented in practice. This way, the number of MACs is reduced to $mn_l + mn_{l+1}$. In other words, this linear mode is equivalent to a learned Dense layer of m output units (*i.e.* function $\tilde{f}$ in Fig. 7.3) followed by another Dense layer composed of the concatenation of n_{l+1} embedding vectors generated by the PRNG (*i.e.*, function $\tilde{g}$ in Fig. 7.3).

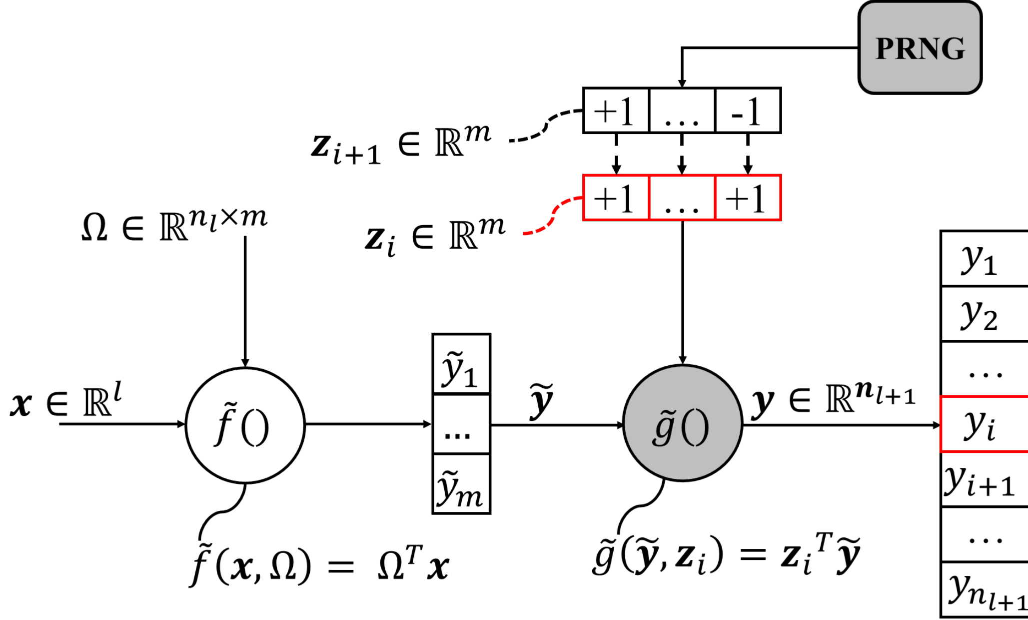


Figure 7.3: The equivalent processing of the proposed FC layer with weight generation network, where $\sigma(x) = x$.

The ratio between the number of MACs of this processing mode of WeightGenNet FC and that of the standard FC layer is:

$$\mathrm{CR}_{MAC} = \frac{mn_l + mn_{l+1}}{n_l n_{l+1}} = \frac{m}{n_l} + \frac{m}{n_{l+1}} \tag{7.9}$$

When $m \ll n_i$ and $m \ll n_o$ which is the common settings, we may likely obtain a ratio smaller than 1, *i.e.* we even reduce the number of MACs. This illustrates the benefits of constructing simple linear operations $g()$ inside the auxiliary generative model, so that we can reduce both the memory needs and the complexity at the same time. We also notice that this mode is able to be established if and only if there is no nonlinearity in the weight generation network. It somehow corresponds to a layer factorization with a part of the weights being fixed and on-line generated.

7.2.2 Conv2D layer with weight generation network

A conventional Conv2D layer indexed l takes input $\boldsymbol{X} \in \mathbb{R}^{H_l \times V_l \times C_l}$ and computes output $Y \in \mathbb{R}^{H_{l+1} \times V_{l+1} \times C_{l+1}}$. Here we also get rid of the bias parameters for convenience. The PM weight tensor is $W \in \mathbb{R}^{h \times v \times C_l \times C_{l+1}}$, meaning that we have C_{l+1} filters $\boldsymbol{W}^i$, $i = 1...c_o$, each kernel has c_i filters $\boldsymbol{W}_{[i]} \in \mathbb{R}^{h \times v \times C_l}$, $i = 1 \dots C_{l+1}$. The ith output channel is computed as follows:

$$\boldsymbol{Y}_{[i]} = \boldsymbol{X} * \boldsymbol{W}_{[i]} \tag{7.10}$$

The number of parameters of a conventional 2D Convolutional layer is hvC_lC_{l+1} and the number of MACs is $H_{l+1}V_{l+1}C_{l+1}hvC_l$. When the number of input channels C_l and output channels C_{l+1} are high, both memory needs and computation complexity become the main bottleneck for the deployment of this layer.

In Figure 7.4 we present a hypernetwork for generating on-the-fly the weight filters of the Conv2D layer, namely *WeightGenNet Conv2D*. Each random vector ρ_i is fed into a weight generation network that is bounded by the gray box. This weight net includes m basis filters $\boldsymbol{F}_{[k]} \in \mathbb{R}^{h \times v \times m}$, $k = 1...m$, and a dense matrix $\boldsymbol{D} \in \mathbb{R}^{C_l \times m}$. The process of generating on-line a filter $\boldsymbol{W}_{[i]}$ is as follows:

1. First, the current embedding z_i is convolved with m basis filters $\boldsymbol{F}_{[k]}$, $k = 1...m$. This can be equivalently represented by a pointwise convolution (PWConv) where the basis filter $\boldsymbol{F}_{[k]}$ plays the role of the "image" to be convolved. Each PWConv outputs a single intermediate filter $\tilde{\boldsymbol{F}}_k \in \mathbb{R}^{h\times v}$ as follows
$$\tilde{\boldsymbol{F}}_{[k]} = \sum_{h=1}^{m} \boldsymbol{F}_{[k,h]} z_{[i,h]} \tag{7.11}$$

2. Second, we concatenate all the intermediate filters together to form a 3D tensor, then apply a nonlinear operation $\sigma_1()$ to this tensor to obtain $\tilde{F}^i \in \mathbb{R}^{h\times v\times m}$.

3. Finally, we apply a PWConv with the weight matrix $\boldsymbol{D} \in \mathbb{R}^{C_l\times m}$ to $\tilde{\boldsymbol{F}}$. The result is then passed through a nonlinear operation $\sigma_2()$ to obtain the filter $\boldsymbol{W}_{[i]}$ of the PM Conv2D layer. The kernel j in the filter i, *i.e.*, $\boldsymbol{W}_{[i,j]}$, is obtained as follows
$$\boldsymbol{W}_{[i,j]} = \sigma_2\left(\sum_{k=1}^{m} D_{k,j}\tilde{\boldsymbol{F}}_k\right) \tag{7.12}$$

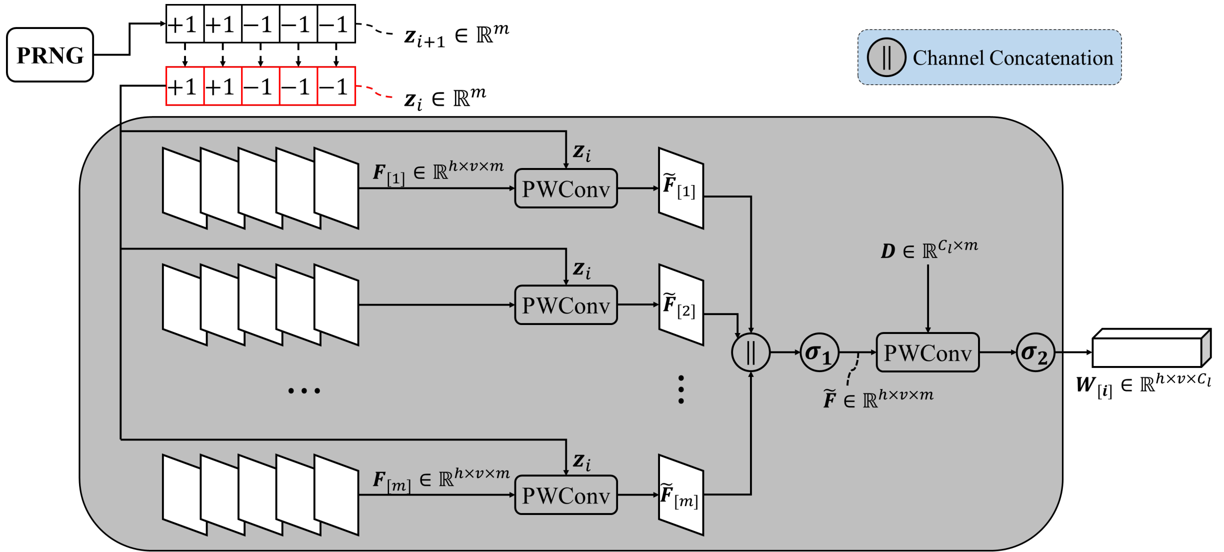


Figure 7.4: WeightGenNet Conv2D and the processing of on-line generating a convolutional filters for the PM.

All C_{l+1} filters are produced on-the-fly from C_{l+1} random embeddings z_i, in the same manner described above. This way, all the main layer weights $\boldsymbol{W}$ are generated on-line from the hypernetwork of $m^2hv + mC_l$ parameters, which is much smaller than the size of conventional 2D Convolutional layer. The number of MAC operations according to this process is $(hvm^2 + hvC_lm + H_{l+1}V_{l+1}hvC_l)C_{l+1}$, which includes an additional cost of generation weights $(uvm^2 + uvc_im)c_o$. The ratio between the number of MACs operation of this specific implementation and that of the conventional 2D Convolutional layer is:

$$\mathrm{CR}_{MAC} = \frac{(hvm^2 + hvC_lm + H_{l+1}V_{l+1}hvC_l)C_{l+1}}{H_{l+1}V_{l+1}hvC_lC_{l+1}} = 1 + \frac{m^2}{H_{l+1}V_{l+1}C_l} + \frac{m}{H_{l+1}V_{l+1}} \tag{7.13}$$

In general, we have $m < C_l$, therefore this ratio mainly depends on the spatial size $H_{l+1} \times V_{l+1}$ of the output. When the spatial size of data is high, this ratio becomes smaller and the additional operation cost becomes negligible.

There are several options for the nonlinear operations $\sigma_1()$ and $\sigma_2()$. We can choose nonlinear activation functions like softsign, tanh, or a normalization. In later experiment, we let $\sigma_1(x) = x$ (linear activation) and $\sigma_2()$ be the layer normalization [52].

Similarly to the case of weight generation network in FC layer, when both $\sigma_1(x) = x$ and $\sigma_2(x) = x$, we can exhibit a linear processing mode that is more compact in terms of complexity. Indeed, the whole process to compute on-line an output map $\boldsymbol{Y}_{[i]}$ can be described as follows:

$$\boldsymbol{Y}_{[i]} = \sum_{j=1}^{C_l} \boldsymbol{X}_{[j]} * \boldsymbol{W}_{[i,j]} = \sum_{j=1}^{C_l} \boldsymbol{X}_{[j]} * \sum_{k=1}^{m} D_{k,j} \tilde{\boldsymbol{F}}_k = \sum_{j=1}^{C_l} \sum_{k=1}^{m} D_{k,j} \boldsymbol{X}_{[j]} * \tilde{\boldsymbol{F}}_k \tag{7.14}$$

As the term $\tilde{\boldsymbol{F}}_k$ does not depend on the index j, we can permute the order of summations

$$\boldsymbol{Y}_{[i]} = \sum_{k=1}^{m} \tilde{\boldsymbol{F}}_k * \left(\sum_{j=1}^{C_l} D_{k,j} \boldsymbol{X}_{[j]} \right) = \sum_{k=1}^{m} \sum_{h=1}^{m} \boldsymbol{F}_{[k,h]} z_{[i,h]} * \left(\sum_{j=1}^{C_l} D_{k,j} \boldsymbol{X}_{[j]} \right) \tag{7.15}$$

Similarly, as the term $_{i,h}$ does not depend on the index k, we can exchange the order of summations

$$\boldsymbol{Y}_{[i]} = \sum_{h=1}^{m} \left\{ z_{[i,h]} \left[\sum_{k=1}^{m} \boldsymbol{F}_{[k,h]} * \left(\sum_{j=1}^{C_l} D_{k,j} \boldsymbol{X}_{[j]} \right) \right] \right\} \tag{7.16}$$

The operation $\sum_{j=1}^{C_l} D_{k,j} \boldsymbol{X}_{[j]}$ corresponds to a PWConv with C_l input channels and m output channels. The operation $\sum_{k=1}^{m} \boldsymbol{F}_{[k,h]} *$ stands for applying a $h \times v$ Conv2D with m input channels and also m output channels to the output of the previous PWConv, where each convolution filter is the collection of all 2D tensor $F_{k,h}$ across the column h. Finally, the operation $\{z_{i,h}[.]\}$ is equivalent to another PWConv layer with m input channels and 1 output channels, but this time the parameters of the PWConv $z_{i,h}$ is fixed and provided by the PRNG. The whole equivalent processing is depicted in Figure 7.5, which is similar to the proposed factorization in Chapter 5, but with a regular Conv2D instead of a GConv2D. This way, we can alleviate the increasing number of operations due to the conversion of random embeddings into convolution filters. Moreover, the number of MACs is even reduced compared to the conventional 2D Convolutional layer. This again demonstrates the benefits of designing simply linear generative model $g()$. Indeed, the total number of MACs of this linear mode is $H_{l+1}V_{l+1}mC_l + H_{l+1}V_{l+1}m^2hv + H_{l+1}V_{l+1}C_{l+1}m$. The ratio between the number of MACs of this process and that of the regular Conv2D layer is

$$\text{CR}_{MAC} = \frac{H_{l+1}V_{l+1}mC_l + H_{l+1}V_{l+1}m^2hv + H_{l+1}V_{l+1}C_{l+1}m}{H_{l+1}V_{l+1}hvC_lC_{l+1}} = \frac{m}{hvC_{l+1}} + \frac{m^2}{C_lC_{l+1}} + \frac{m}{hvC_l}. \tag{7.17}$$

The ratio between the number of learnable parameters of the proposed hypernetwork and that of the conventional layer is:

$$\text{CR}_{params} = \frac{m^2hv + mC_l}{hvC_lC_{l+1}} = \frac{m^2}{C_lC_{l+1}} + \frac{m}{hvC_{l+1}}. \tag{7.18}$$

In practice, we select $m \ll C_l$ and $m \ll C_{l+1}$ to obtain a high compression rate in terms of both parameters and MACs. More specifically, both the filters $\boldsymbol{F}_{[k]}$ and the dense coefficients $\boldsymbol{D}$ can be quantized, or pruned, enabling further memory needs and computation complexity reduction.

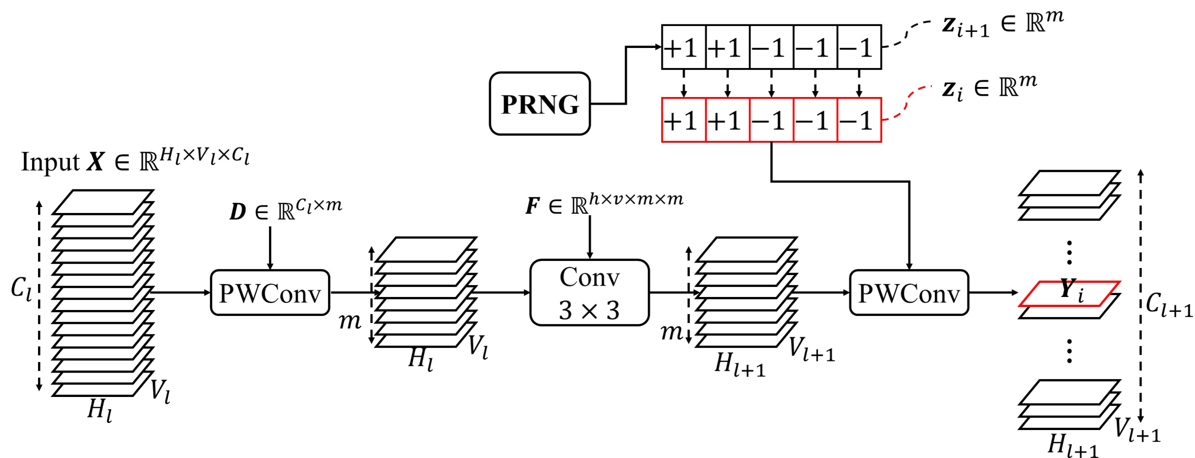


Figure 7.5: The equivalent processing of the proposed WeightGenNet Conv2D layer when $\sigma_1(x) = x$ and $\sigma_2(x) = x$. It is similar to the convolutional factorization in Chapter 5, except for the regular Conv2D instead of the GConv2D.

7.3 Experiments

7.3.1 MLP on MNIST

Setting

We reshape 28×28 gray-scale images in MNIST to 784-dimensional vectors. The MLP architecture consists of 3 hidden FC layers of n units and a Softmax output layer. Each FC layers is followed by a BN and a ReLU activation. The PM model complexity thus depends on the hyperparameter n_{reg}. Throughout this experiment, we term the MLP with regular FC layers of n_{reg} hidden units as **Regular MLP** - n_{reg}.

To evaluate the proposed FC layer with weight net, we replace all 3 hidden FC layers with the WeightGenNet FC layers of n_{wn} output units and m-dimensional embeddings. This model is termed as **WeightGenNet MLP** - n_{wn}-m. The PRNG generates 32-b random embeddings of gaussian distribution and all other parameters are full-precision values. Here we choose Softsign function as the linear operation of the WeightGenNet FC, *i.e.*, $\sigma(x) = \text{softsign}(x) = \frac{x}{|x|+1}$. Here we choose $n_{\text{wn}} = 256$ and $m = 16$.

All models are trained during 100 epochs with a batch size of 100, using Adam optimizer over the binary cross-entropy loss function. The parameter n_{reg} can be set to have the same PM architecture, *i.e.*, $n_{\text{reg}} = 256$, or to have the same number of learnable parameters as WeightGenNet FC, which gives rise to $n_{\text{reg}} = 29$. We also note that the number of MACs in the case of WeightGenNet MLP is computed based on the linear processing mode (*i.e.*, Fig. 7.3). Each result have been obtained after 5 realizations and taking the average accuracy.

Results

From Table 7.1, we can see a marginal gain (0.16%) when introducing the Softsign nonlinearity into WeightGenNet MLP. This comes as a cost of increasing the #MACs from 0.036M to 5.643M. Compared to the Regular MLP - 29 having the same #params, WeightGenNet MLP with Softsign obtain an accuracy gain of 0.44%. However, at the same PM architecture, Regular MLP - 256 has a 0.87% higher accuracy while consuming more than 13.5× #params. The huge computational cost of WeightGenNet MLP points out that the proposed hypernetwork is not efficiently designed, and more compact architecture can be proposed to reduce the additional #MACs.

Table 7.1: MNIST classification results with the proposed WeightGenNet FC layers.

Model	Regular MLP ($n_{\text{reg}} = 29$)	Regular MLP ($n_{\text{reg}} = 256$)	WeightGenNet MLP	WeightGenNet MLP + Softsign
Accuracy (%)	97.27	98.58	97.55	97.71
#params ($\times 10^6$)	0.025	0.336	0.025	0.025
#MACs ($\times 10^6$)	0.025	0.335	0.036	5.643

7.3.2 VGG-7 on CIFAR-10

Settings

The proposed WeightGenNet Conv2D is benchmarked on CIFAR-10 dataset using a VGG-7 architecture, by "$2 \times (128 - \text{C3}) + MP2 + 2 \times (256 - \text{C3}) + 2 \times (512 - \text{C3}) + MP2 + 512 - \text{FC} + 10 - \text{FC} +$ Softmax", where $128 - \text{C3}$ denotes for the 3×3 Conv2D layer with 128 output feature maps. We term this model as **Regular VGG-7**. To evaluate the impact of the proposed hypernetwork on 2D convolution, we replace all the regular Conv2D layers (except for the first one) with the WeightGenNet Conv2D, namely **WeightGenNet VGG-7**, where the dimension of the embeddings are also scaled with the number of output feature maps in the PM, *i.e.*, $32, 64, 128$. Similarly to the case of on-line generated weights in Section 5.3.2, the PRNG is a Cellular Automaton following Wolfram rule 30 as the evolution rule, with a random initialization. The length of each sequence is m, and the number of time steps is C_{l+1}. Note that all the basis filters $\boldsymbol{F}$ and dense matrices $\boldsymbol{D}$ are binarized using the Sign function. To add a nonlinearity into the hypernetwork, we employ the Layer Normalization (LN) [52] as the $\sigma_2()$ function while keeping $\sigma_1(x) = x$. Note that the on-chip memory and #MACs are only computed over convolution layers. We use the following data augmentation scheme: add 4 pixels each side, randomly crop a 32×32 image and apply horizontal flip. Models are trained using Adam optimizer during 150 epochs with batch size of 50. The learning rate is initialized and kept at 10^{-3} during the first 50 epochs, then divided by 10 after every 25 epochs.

Results

As reported in Table 7.2, the Regular VGG-7 obtains 93.12% accuracy, but contains up to 146 Mb of on-chip memory, and 0.608 Giga-MAC (GMAC) operations. Having the same PM architecture, the WeightGenNet VGG-7 achieves 91.15% accuracy, *i.e.*, a degradation of 1.97%. This comes at the benefit of using only 0.37Mb on-chip memory and 0.099 GMACs, corresponding to a reduction of 197.8× and 6.14× in terms of on-chip memory and #MACs. When using Layer Normalization, WeightGenNet VGG-7 performs slightly better, with a small gain of 0.11% at the cost of doubling the #MACs compared to that of the Regular VGG-7. However, these operations are with binary weights, so it consumes much less energy and is much simpler to be implemented compared to the case of 32b-representation. These results demonstrate the effectiveness of our realization compared to state-of-the-art conventional layer.

7.4 Discussion

7.4.1 Configurations of the PRNG

The role of PRNG in our framework is to supply embeddings to the weight generation networks that further convert them to PM layer weights. This component relies on both the algorithmic and hardware aspect of the proposed weight net. For example, both WeightGenNet FC and Weight-

Table 7.2: CIFAR-10 classification results with the proposed WeightGenNet Conv2D.

Model	Regular VGG-7	WeightGenNet VGG-7	WeightGenNet VGG-7 + Layer Normalization
Accuracy (%)	93.12	91.15	91.26
On-chip Memory (Mb)	146	0.37	0.37
#MACs ($\times 10^9$)	0.608	0.099	1.299

GenNet Conv2D in the previous section have the equivalent linear processing including a linear combination with the embeddings issued from PRNG. In this specific case, these vectors are preferred to be uncorrelated to avoid redundant information in the hypernetwork, hence improving the primary model's performance. In terms of hardware implementation, we would like to use simple PRNG whose computation cost is negligible. There are following options for PRNGs:

- Type of generators: Pseudo random generators of type LFSR or Cellular Automata (CA); bit sequence distribution, sequence pseudo-random shuffling. . .
- Sequence settings: prefixed initialization or time-stamped pseudo-random initialization; structural and/or not structural; 1D (coefficients) or 2D (maps, cf. 2D CA); 1-bit (binary) or multiple-bit (combining bits to equivalent data representation, e.g., integers, floating points. . .).
- Enabled topology learning: the rule of sequence generation (e.g., when using CA) may be learned during the learning procedure of the main task to both define best initializations and internal logical functions. This possible learning can be considered in the framework of Neural Architecture Search (Figure 7.6).

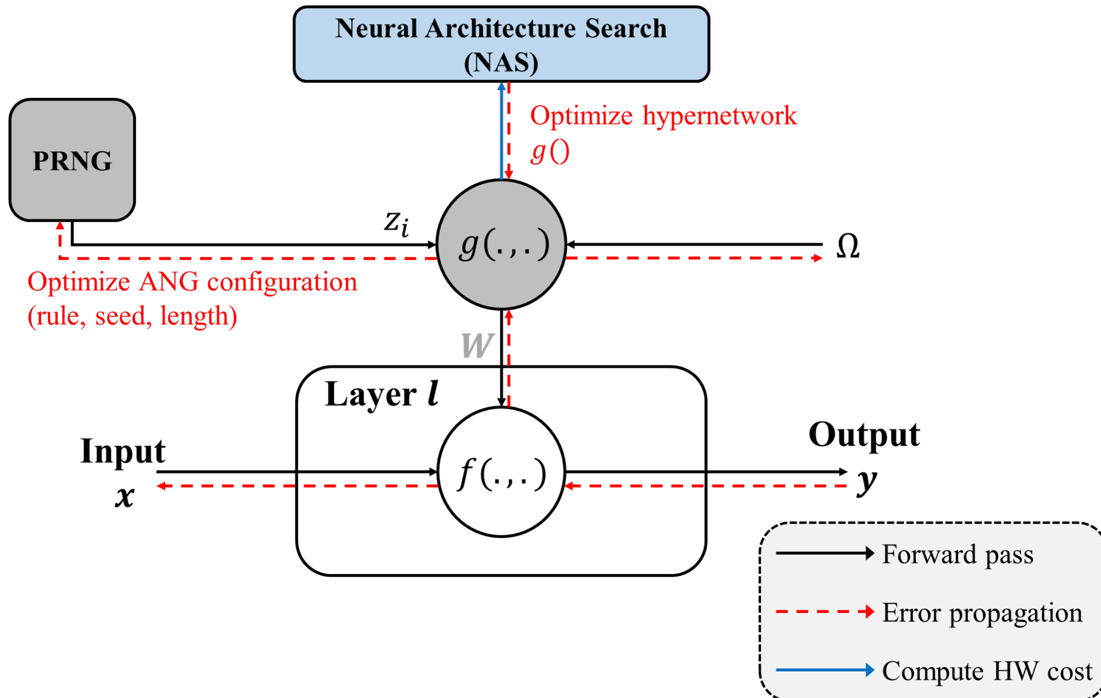


Figure 7.6: Jointly optimizing the hypernetwork $g()$ and its parameters $\mathbf{\Omega}$ and the configuration of the PRNG (*e.g.*, generation rule, random seed, dimension of embeddings).

7.4.2 Weight generation network architecture

The role of weight net in our framework is to convert sequence of embeddings into layer weights. The design requirements are: minimizing hardware implementation complexity, versatility and high memory efficiency (*i.e.*, upscaling factor denoted CR in previous sections). This can be obtained by increasing furthermore the level of factorization and the parameter-sharing inside the hypernetwork to reduce the number of operations of the weight generation network.

7.4.3 Application

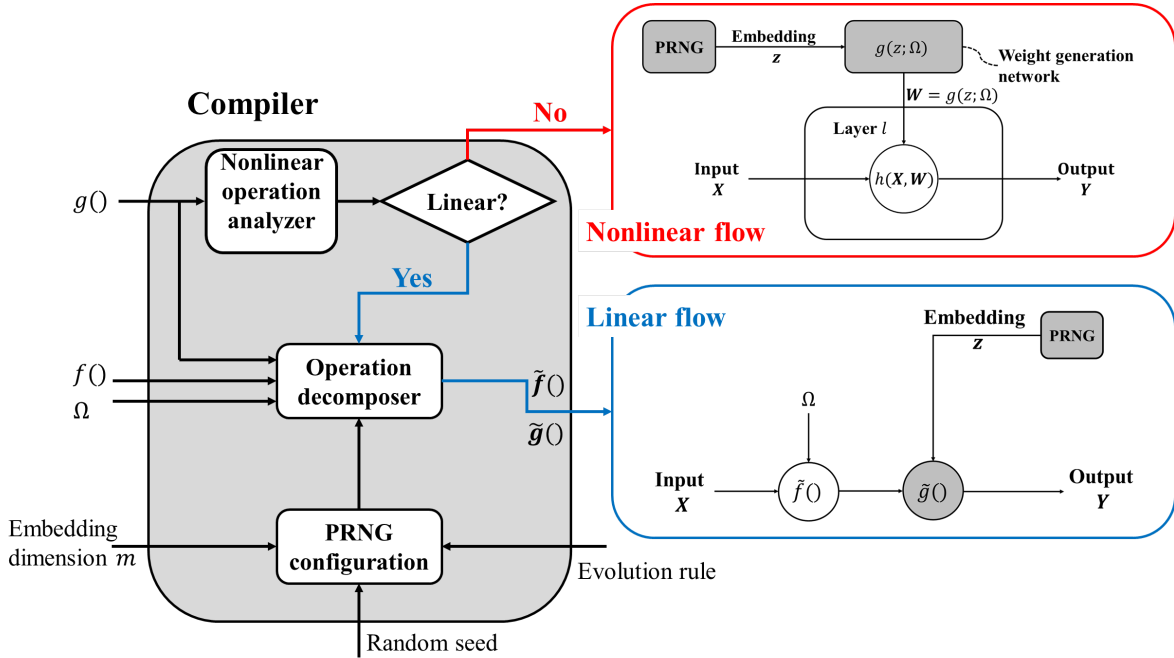


Figure 7.7: Compiler for generic DNN accelerators to handle two functional modes (linear versus nonlinear) of the weight generation networks with PRNG.

The use of weight net is applicable to various layer types and PM architectures. It proposes an alternative way to reduce memory needs related to model weights while limiting algorithmic degradation of the main inference task. Designing smart devices with highly accurate inference capabilities while being Hardware-frugal is now a trending topic. The proposed WeightGen-Net framework definitively has the advantage of being highly generic in that context because of enabling a novel approach to compress DNN models using hypernetworks with PRNG embeddings. As the hypernetwork is learned within a unified framework with the inference task, this can be applied to a wide range of applications, from classification, regression to data compression/reconstruction. Regarding hardware implementation, it maps well for AIDA and ASA platforms.

When mapping the proposed WeightGenNet with PRNG to AIDA platforms, the implementation requires a model architecture description interpreter, a compiler (Figure 7.7) and a specific PRNG. Firstly, the model architecture description and interpreter enables a compilation stage to target multiple types of PM's layer operation $f()$ and weight generation architectures $g()$. Next, the compiler receives description of the weight generation network $g()$, the PM's layer operation description $f()$, the learnable weights description $\mathbf{\Omega}$, and the PRNG's configuration (*e.g.*, sequence length m, random seed, evolution rule...). It embeds a Nonlinear Operation Analyzer that analyzes the linearity characteristics of the hypernetwork from the description of $g()$. If $g()$

contains nonlinearity, it will drive the computation flow in a standard linear flow mode (in blue). Otherwise, an Operation Decomposer will analyze the given information of $f()$ and $g()$ to determine the latent operation $\tilde{f}()$ and the PRNG operation $\tilde{g}()$. Concretely, $\tilde{f}()$ corresponds to the first FC layer in the linear processing mode of WeightGenNet FC in Fig. 7.3. On the other hand, $\tilde{f}()$ consists of the first two layers in the linear mode of WeightGenNet Conv2D in Fig. 7.5. In general, this latent operation aims at mapping the data dimension to the latent dimension m. On the other hand, the PRNG operation $\tilde{g}()$ corresponds to the last FC with online-generated weights which maps the data from the latent dimension to the desired output dimension. The compiler then drives the computation flow in this linearly optimized mode. Finally, the generation of pseudo random sequences $\boldsymbol{z}$ with PRNG can be performed on the targeted execution chip in an optimized fashion thanks to the compiler and given programmable configurations (for example using a reconfigurable CA that is directly implemented in a specific hardware circuit, e.g., using In-Memory Computing). This last consideration is compliant with both nonlinear flow and linearly-optimized flow.

7.5 Conclusion and Perspective

In this chapter, we address an emerging model compression technique using weight generation networks, which mainly aim at alleviating the parameter-heaviness of DNNs. We present a static framework for generating on-the-fly the PM's weights with the random embeddings issued from the PRNG such as Cellular Automaton. Concretely, we propose two weight generation network architectures for the FC and the Conv2D layers, namely WeightGenNet FC and WeightGenNet Conv2D, along with the corresponding linear processing mode when the weight net does not contains any nonlinear operation. Preliminary results show that the models containing the proposed WeightGenNet layers can obtain a significant compression rate compared to the model with regular layers. However, the additional computational cost is a major downside of this framework. To address this issue, it is crucial to embeds other compression techniques such as quantization and dynamic pruning to the weight generation network. We also discuss about some possible optimizations for the PRNG's configuration and the weight net architecture under the NAS perspective. It is noteworthy pointing out that this static weight generation framework can be properly mapped to common DNN accelerators such as AIDAs or ASAs, to serve for different applications.

For future works, we have identified some promising dynamic weight generation architectures which feature the weight-sharing [31] between the hypernetworks of different layers. This can drastically reduce the on-chip memory needed for storing the whole model. Moreover, the generation of the layer's embeddings can also be done in a dynamically manner by using the input data itself. A further step is to adopt a dynamic pruning strategy to the weight net, which adaptively provides the pruning patterns for each filters in the Conv2D layer. Another approach is to search for the optimal architecture of the weight net given some hardware-related budgets such as model size or #MACs.

8

Thesis Conclusion & Perspectives

8.1 Thesis summary

In this thesis, we have addressed the problem of designing deep neural network algorithms enabling efficient on-chip image/video processing in the context of ASIC design. We have reviewed current deep learning accelerators and the popular approaches for building compact DNNs such as efficient architecture design, quantization, pruning, tensor decomposition and weight generation network. We have next introduced novel hardware-compliant methods for quantizing different components in DNNs involving not only the weight and the activations but also the batch normalization and the skip connections. These quantization methods, when combined with a light-weight convolutional factorization leveraging on-line generated weights, allow us to drastically reduce memory needs and computational costs to perform DNN inference while facilitating hardware implementation. Finally, we have also presented some preliminary studies on the weight generation network with random layer embedding issued from PRNGs, which enables us to alleviate the parameter-heaviness of DNNs.

8.2 Summary of the Contributions

All the contributions presented throughout this thesis mainly rely on the co-design of DNN architecture and model compression approaches, with a hardware implementation perspective for compact ASIC accelerators towards efficient embedded image/video inference. In details, we have focused on:

- **Low-precision weighted networks (Chapter 3):** among the state-of-the-art DNN compression techniques, model quantization has emerged as one of the most relevant approaches allowing to reduce not only the feature-related and on-chip memory, but also the computation cost and hardware complexity. Unlike previous weight quantization methods that mainly rely on the minimization of the quantization error or the task loss, we have proposed a novel adaptive QAT framework, namely HEQ, using statistics (*i.e.*, n-quantiles) of the proxy weights. Specifically, in the context of linear symmetric quantization, HEQ automatically adjusts the step size parameter to equalize the histogram of the quantized weights, and thus, maximizing their information entropy. Experimental results have demonstrated that HEQ can achieve state-of-the-art results on low-precision DNNs, while offering a better hardware and training compatibility compared to previous works. Furthermore, HEQ

can also be considered as a regularization approach which constraints the quantized weights distribution to be uniform, while easing the optimization process of QNNs. Note that, any other distributions could also be used instead in our framework. The effectiveness and versatility of HEQ have also been demonstrated through different model architectures in the next chapters.

- **Resource-constrained, mixed-precision encoder for image classification and patch-based compression (Chapter 4):** the downside of resource-constrained, application-specific accelerators are the low level of flexibility and application-versatility. In Chapter 4, we have demonstrated that it is possible to design an ASIC NN accelerator that can be applied to two uncorrelated tasks such as image classification and compression. In particular, we have presented a light-weight reconfigurable quantized encoder featuring fine-grained design, with a hand-crafted mixed-precision topology, the HWMSB activation quantization, and the hardware-compliant BSN. Besides, we have also demonstrated that the proposed quantized encoder can be used to compress image patch-by-patch while the reconstruction can be performed remotely, by a dedicated full-frame decoder namely PURENET. Through the experimental results, we aim at highlighting the important role of a fine-grained hardware-algorithmic co-design in improving the efficiency and the application- versatility of resource-constrained ASIC platforms.

- **Hardware-aware Residual Networks with logic-gated skip connections and light-weight convolutional factorization (Chapter 5):** in Chapter 5 we have addressed a scientific question about the efficient way to apply model compression techniques, in particular quantization. Although achieving remarkable compression rates, the hardware implementation of some previous quantization techniques is still questionable. On the other hand, while quantized networks have low-precision weights/activations, somehow it is still ignoring other components such as the BN or the skip connections, resulting in additional costs with respect to the targeted low bit-width hardware implementation. Based on these statements, novel logic-gated residual building blocks (*e.g.*, OR block, MUX-OR block and MRB) have been presented, allowing to perform the skip connection with the streamlined AND, OR, MUX gates and bitshift operation. In addition, we have also proposed a light-weight convolutional factorization leveraging Cellular Automaton-generated weights, to reduce furthermore the hardware requirements for DNN inference. Experimental results show the great interest of the proposed methods in terms of hardware-algorithmic trade-offs (on-chip memory needs and implementation complexity/compatibility), especially in the context of DNN accelerators with extremely low resource budget and integer-only computation support.

- **Efficient video inference with Binarized Conv3D-LSTM model (Chapter 6):** despite the tremendous progress of model compression techniques in the recent years, they are mainly applied to image-related applications. In Chapter 6, we have addressed the need for efficient video inference, by proposing a fully-binarized Conv3D-LSTM model containing only bit-wise and integer computations. Besides the Conv3D part which presents a 3D version of the proposed MUX-OR block and light-weight factorization in Chapter 5, we also introduce a multi-stage training scheme to fully quantize the LSTM layers with only 1-bit weights and binary/ternay hidden units. Experimental results have hightlighted the advantages of the proposed quantized model in terms of both on-chip memory and computational cost. From a implementation point of view, a fully binarized Conv3D-LSTM model can greatly fit low-power ASIC accelerators supporting only bit-wise and integer operations.

- **Model compression via weight generation network and PRNG (Chapter 7)** apart from previous chapters which consider quantization as the pivotal model compression technique, in Chapter 7, we have investigated the possibility of circumventing the parameter-heaviness of DNNs using weight generation network and PRNG. In particular, two different weight

generation networks along with their linear processing mode have been proposed to statically transforming random embeddings from PRNGs into the weights of FC and the Conv2D layers in an on-line fashion. Preliminary results have demonstrated the promising results of this weight generation framework, especially when combined with other model compression methods such as quantization or dynamic pruning.

8.3 Perspectives

As mentioned at the end of the previous chapters, the different contributions discussed in this manuscript can be extended in various ways, and used as a starting point to explore other hardware-algorithmic co-optimization methods, in combination with neural architecture search, training trick such as knowledge distillation, and other model compression frameworks such as pruning and dynamic networks. This section will present our perspectives in a more synthesized manner regarding the presented hardware-algorithm DNN enablers.

Quantization is the main model compression used throughout this thesis. Our work and state-of-the-art methods has demonstrated that using the statistics of the proxy weights is an effective way to adjust the quantization mapping during training in a versatile manner. Although HEQ has shown its capability in the means of maximizing the information entropy, we believe that another non-uniform distribution can also be adopted to adjust the quantizers, in particular when quantizing the activations. Besides, a promising approach is to target an intra-layer mixed-precision strategy for layer's weights. This idea consists of quantizing the dominant near-zero proxy weights with low bit-width, while dedicating more precision for the minority long-tailed values [214]. In this case, the bit-width may be learned with an entropy coding, or manually designed. Moreover, the additional bit-width for long-tailed distribution can be represented as a residual term [199], which can be dynamically assigned to only "difficult" input data. Apart from the design of adaptive quantization mapping, another research line that should be explored to enhance the performance of extremely low-precision DNNs, especially BNNs, is to introduce high-precision shortcuts to alleviate disastrous information loss [293], [29]. Although significantly reducing the performance degradation between the full-precision and the low-precision models, this approach introduces additional full-precision operations which may be the major bottleneck if the implementation platform only supports low-precision computations. This opens a room for improvements, mostly regarding the design of shortcut branches and the hardware-compliant skip connections that are inference-efficient.

Neural Architecture Search is currently one of the most active research area in deep learning which allows to automatically design DNN architectures. Recent advances in NAS [30] allow to perform the search in a more acceptable training cost, thus making it practical to be widely adopted in a resource-constrained context. Compact model design, including quantized neural networks, can be naturally plugged into the NAS frameworks to find the optimal model architecture within a given hardware budget. Since all the proposed model architectures in this thesis are manually designed, a direct extension is to use learning-based or an input-driven dynamic approach to improve the efficiency of these hand-crafted models, *e.g.*, searching for the optimal width/depth [139] or bit-width for each layer [215]. Concretely, the mixed-precision topology as well as the topology in Chapter 4 can be optimally found given on-chip memory/BOP budget. On the other hand, we can also search for different variants of the logic-gated residual blocks proposed in Chapter 5 and Chapter 6. This search may cover different components of the models: type of logic gates (*e.g.*, OR, AND, MUX), type of main branch transformation (*e.g.*, Conv2D, DWConv, PWConv, efficient convolution cell), architecture of the attention branch, the width or the depth of the model. It may also be adopted to find the efficient architecture of the weight generation network in Chapter 7. If the cost of performing NAS is considerable, one option is to consider architectures found by hardware-aware NAS as a starting point for the baseline model, before modifying its structures

regarding specifications in hardware implementation.

Weight-sharing [31], [32] is another algorithmic enabler to circumvent the parameter redundancy of DNNs. Considering the case of 2D convolution for images, a highly abstract feature is learned through a cascade of lower-level features which may be simply the result of some basis type of filtering such as edge extraction or average. Based on this observation, it is able to favor a commonly-shared filter base in CNNs as demonstrated in existing approaches. This weight-sharing may be established at different levels: intra-model or inter-model. In our thesis, a possible extension regarding the first level consists of using a unique filter base for all weight generation networks of different convolutional layers. On the other hand, an inter-model weight-sharing may be advantageously introduced in a reconfigurable multi-task ASIC accelerator (eg Chapter 4), allowing us to store only a small portion of weights instead of the whole set of parameters for each application.

From a hardware point of view, the proposed algorithmic enablers throughout this thesis will certainly have impacts on at least one of the following resource constraints: memory, computational capability, energy consumption, latency and footprint. Although we can explicitly quantify the memory needs and the computational complexity of the models, it is still not clear what is the quantitative impact of those propositions in terms of energy or latency, since it also depends on the deployment platform. This suggests us an important question about the target hardware implementation, whether it is a conventional ASIC design, or an emerging compute-in-memory architecture (e.g. SRAM, RRAM). At least, we can leverage some prior modelling (e.g. [294]) or a simulation framework (e.g. NeuroSim [295]) to benchmark and evaluate DNN-based accelerators on proposed compact models, hence obtaining a more complete picture of the hardware-related benefits.

Finally, based on our work and the current trend in developing advanced model compression algorithms, we believe that a combination approach is promising to boost the efficiency of DNNs. This involves not only the co-design of a unified model compression framework (*e.g.*, pruning-quantization-hypernetwork) and the baseline model architectures, but also the learning procedure, where training trick (*e.g.*, knowledge distillation) can be applied to ease the learning and reduce the generalization gap.

List of Figures

1 Compromise matérielle-algorithmique de différentes plateformes d'accélération de DNNs. (∗): Die micrographe de l'accélérateur générique Eyeriss [6]. (∗∗): Die micrographe d'un chip de 4.53µW dédié pour la vérification de voix et le keyword spotting [7]. 2

2 Quantification linéaire symmétrique avec une distribution uniforme lorsque n-quantiles (q_{-i}, q_i) sont symmétriques and coincident avec les seuils de quantification. 4

3 Description schématique de l'encodeur multi-tâche à précision mixte, du classificateur binarisé et du décodeur flottant pour la reconstruction d'image entière à partir des vecteurs de compression par blocs. 5

4 Le bloc résiduel élémentaire avec des portes logiques OU, MUX et l'opération bitshift. 6

5 Génération des poids d'une couche à l'aide d'un modèle auxiliaire et générateur automatique de nombres pseudo-aléatoire (PRNG). 7

1.1 Typical hardware-algorithmic trade-offs of different DNN processing platforms. (∗): Die micrograph of Eyeriss DNN accelerator [6]. (∗∗): Layout of the 4.53µW accelerator enabling both speaker verification and keyword spotting [7]. 11

2.1 Example of a simple neural networks with 2 hidden layers. Neuron's computation involves applying a nonlinear activation function f to the weighted sum of the input values. 16

2.2 Computational relationship between input and output channel maps in the case of a 2D convolution with 8 input channels and 4 output channels. a) regular convolution with dense connection; b) structurally grouped convolution with sparser connections. The number of connections is reduced by a factor equal to the number of groups ($g = 2$). 18

2.3 Sign function and its approximated derivative using STE technique. a) the clipped identity is considered as a continuous approximation of the sign function. b) STE derivative of the sign function allowing the gradient to pass through within the clipping range $[-1, 1]$. 23

2.4 Heaviside function and its approximated derivative using STE technique. a) the hard sigmoid is considered as a continuous approximation of the Heaviside function. b) STE derivative of the Heaviside function allowing the gradient to pass through within the clipping range $[-1, 1]$. 24

2.5 Three stages of a typical gradient descent iteration. 26

2.6 AlexNet architecture. Note that this figure is originally introduced in [1]. 30

2.7 VGG16 [81] architecture. 30

2.8 A general residual block. 30

2.9 Building block of ResNext [82]. 31

2.10 Fire module in the SqueezeNet [82]. 31

2.11 Residual Attention block [83]. 32

2.12 Squeeze-and-Excitation (SE) block [84]. Note that this figure is originally introduced in [84]. 32

2.13 An example of recurrent neural network. This model receives the sequential input data $\boldsymbol{x}$ and combines it with the hidden state $\boldsymbol{h}$ which is passed through the time, to obtain the output $\boldsymbol{y}$. *Left*: the folded graph represents the whole neural network. *Right*: the unrolled graph depicts the whole computation graph, allowing us to see the information flow in the forward pass and also the gradient propagation in backward pass. 33
2.14 Computational graph of the LSTM cell. 34
2.15 Computation graph of a typical quantized layer in Larq. In details, the quantization is done through the *input_quantizer* and *kernel_quantizer* arguments. 35
2.16 Power versus computational speed of different DNN accelerator platforms. Note that this figure is originally introduced in [129]. 37
2.17 The total power consumption of different platforms as a function of #TOPs. . . . 38
2.18 The building residual block of the MobileNet V2 [134]. The parameter t is the expansion factor for increasing the number of intermediate channels. 40
2.19 Building blocks of ShuffleNet V1. 40
2.20 Building blocks of ShuffleNet V2. 41
2.21 Optimal cells learned by DARTS on CIFAR-10. Note that the operations are characterized by colors. $c_{\{k-1\}}$ and $c_{\{k-2\}}$ are outputs of the two previous cells. 42
2.22 Taxonomy of model compression approaches. The central operation causing the hardware-related bottleneck of DNNs is the linear operation $\boldsymbol{Z} = h(\boldsymbol{W}, \boldsymbol{X})$ (*e.g.* matrix multiplication or convolution). Compression techniques consists of applying additional operations to the operand $\boldsymbol{X}$ and $\boldsymbol{W}$ in order to lighten this operation or simply reduce the model size. 43
2.23 Three stages of a STE-based gradient descent iteration of a quantized neural network. 45
2.24 Power versus Computation Speed of different DNN accelerator platforms based on precision. Note that this figure is originally introduced in [129]. 46

3.1 Quantization frameworks in increasing order of data availability, training cost and also algorithmic performance. 52
3.2 An example of the linear symmetric quantization where the step size s defines the quantization thresholds. Here the quantization outputs could be encoded by a 3-bit representation. 53
3.3 Weight distributions of 2 layers (in 2 columns) after training of: full-precision model (1st line), existing ternary-weight and quinary-weight model (2nd line), and our proposed HEQ method (3rd line) along with quantization thresholds. 56
3.4 Symmetric linear quantization with histogram bin equalization when n-quantiles (q_{-i}, q_i) are symmetrical and coincide with the quantized thresholds. 57
3.5 Comparison of the ternary-weight distribution using TWN and our HEQ method. 58
3.6 Information entropy of the quantized weights of different layers in VGG-Small on CIFAR-10. The entropy reaches its maximum value when all the quantized levels have equi-probabilities. 59
3.7 Evolution of the step size s during training. 60

4.1 The joint framework for both image classification and image embedded compression and remote decompression. 66
4.2 Schematic description of our framework involving neural network topology parts. 66
4.3 Topology of the Nonlinear Quantized Encoder (NQE) + Classifier. F in red stands for the hyperparameter corresponding to the size scale of the feature map (*i.e.* the number of the feature map of the first convolution module). GC stands for Groupwise Convolution of 4 groups. 70

4.4 Example of Group-wise convolution with a 4-channel input divided into 2 groups and 8-channel output. The intermediate values are also divided into two groups, and each convolution is performed with a kernel that takes only two input channels from the corresponding group. These output channels are then structurally shuffled. 73

4.5 The patch-based encoding with Full-Resolution PURENET decoder. Compressed binary codes are vector of length 256, corresponding to the bitrate of 0.25bpp. Note that without the Patches Aggregation (PA), this topology may still be applied to the decoding of patches independently. This variant without PA is denoted as patch-independent PURENET (PI-PURENET). 74

4.6 PU: The Patch-based Upsampling performed using 4 CBRs (ConvT + BN + ReLU), all with a kernel size of 3×3 and a stride of 2. 74

4.7 RE: Refinement model architecture with Residual and Concatenation (RC) Block. The parameter $n = 32$ denotes the number of feature maps. 75

4.8 Memory-accuracy curves of different model's precision (floating-point, binary and ours mixed-precision). 76

4.9 Image compression results at 0.25bpp on the test image indexed 801 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image. 82

4.10 Image compression results at 0.25bpp on the test image indexed 804 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image. 83

4.11 Image compression results at 0.25bpp on the test image indexed 809 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image. 84

4.12 Image compression results at 0.25bpp on the test image indexed 865 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image. 85

4.13 Image compression results at 0.25bpp on the test image indexed 876 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image. 86

4.14 Image compression results at 0.25bpp on the test image indexed 894 of the DIV2K validation dataset. From top to bottom: Original image, JPEG2000, CAE-PSNR, WD-TV3D, CAE, our BBD, NQE- PI-PURENET and NQE- Full-Resolution PURENET along with the corresponding **PSNR/ MS-SSIM** values under each image. 87

5.1 Two common strategy of compressing the models: obtain an extremely high compression rate on existing large model, and obtain a moderate compression rate on efficiently compact models. The latter results in compressed models with better hardware-algorithm trade-offs. 90

5.2 Models with the plain block (11-hidden layer VGG [81]-variant), OR-gated block and MUX-OR gated block. 91

5.3 Impact of the OR gate and the MUX-OR mechanism on the distribution of 0 and 1 intermediate values in 3 blocks of the proposed OR-Net and MUXOR-Net. . . . 94

5.4 Top-level architecture description of MOGNET with Convolutional Factorization Leveraging On-line Generated weights (CFLOG) and MUX Residual Block (MRB). The final 1×1 convolution is followed by Batch Normalization (BN) prior to a Global Average Pooling (GAP). Here n,m are the parameters controlling the number of output feature maps and the latent dimension in CFLOG, MP stands for 2×2 Max Pooling and g-GConv is Grouped Convolution with g groups. 95
5.5 CFLOG description with CA-generated weights. 96
5.6 Test accuracy of different compression method-model couplings. Our models are with 3-b activations. 99

6.1 Top-level architecture description of BILLNET with Convolutional Factorization (CF) and MUX-OR Residual (MOR) Block. Here n is the parameters controlling the number of output feature maps, g-GConv is Grouped Convolution with g groups, MP and GAP stand for Max Pooling and Global Average Pooling. 104
6.2 The operation of the channel-wise MUX gate with feature maps extracted during inference of a test sample. The TGAP is implemented by a bitcount followed by an integer-to-integer comparison, where the threshold is equal to one half of the spatial resolution ($\frac{6\times8}{2}=24$). 106
6.3 Computational graph of the proposed Quantized LSTM. 107
6.4 Training curves (CCE loss and accuracy) of BILLNET 2× throughout all 5 training stages. 110
6.5 Weight memory (Mb) and computational costs (GBOPs ~ 10^9 BOPs [247]) versus Top-1 Accuracy. Edge-less stars are for BILLNET with g=2 and m=n, edged stars are for m=$n/2$ and $g=\frac{n}{16}$ (with $n\in\{64,128\}$). 111
6.6 Class-temporal BILLNET output responses for a Jester test sample labeled as a *Pulling Two Fingers In* gesture. Highlighted time/class positions with maximum values correlate to the most informative frames of the input video. Besides, similar classes (*Pulling Hand In*, *Sliding Two Fingers In*) also exhibit high values at the same columns. 112

7.1 A standard layer where the weights $\boldsymbol{W}$ is stored in memory, and the corresponding layer with weight generation network. 117
7.2 Fully-connected layer with weight generated network. 119
7.3 The equivalent processing of the proposed FC layer with weight generation network, where $\sigma(x)=x$. 120
7.4 WeightGenNet Conv2D and the processing of on-line generating a convolutional filters for the PM. 121
7.5 The equivalent processing of the proposed WeightGenNet Conv2D layer when $\sigma_1(x)=x$ and $\sigma_2(x)=x$. It is similar to the convolutional factorization in Chapter 5, except for the regular Conv2D instead of the GConv2D. 123
7.6 Jointly optimizing the hypernetwork $g()$ and its parameters $\boldsymbol{\Omega}$ and the configuration of the PRNG (*e.g.*, generation rule, random seed, dimension of embeddings). 125
7.7 Compiler for generic DNN accelerators to handle two functional modes (linear versus nonlinear) of the weight generation networks with PRNG. 126

List of Tables

2.1 Qualitative comparison of the impact of hardware-algorithmic enablers to memory and computation costs during DL inference. ☺: reduced; ☹: increased; −: unchanged. 49

3.1 Comparison with the state-of-the-art low-precision quantization methods on CIFAR-10. 60

4.1 NQE topology with details on layer inputs and kernels precision. 69
4.2 2-bit HWMSB input-output mapping with naive binary representation (sign+magnitude) 71
4.3 Effect of the bottleneck and its two alternatives to the overall memory and accuracy. 77
4.4 Comparison of low-precision CNN processors (CIFAR-10 image classification task use case). 78
4.5 VGA image compression comparison between different methods at the bitrate of 0.25bpp (0.2423bpp for CAEM-PSNR and 0.2459bpp for CAEM-SSIM). 79

5.1 OR gate and its arithmetical operation 92
5.2 Comparison with the state-of-the-art low-precision quantization methods on STL-10 dataset. 92
5.3 Training and optimization settings 97
5.4 Comparison of different network compression methods on CIFAR-10 and CIFAR-100. 98

6.1 Training and optimization settings of BILLNET 109
6.2 Comparison of resource-efficient models on Jester hand gesture dataset, weight-related memory (model size), and bitwidth-aware computational complexity (GBOPs). Results reported here are for BILLNET with g=4, n=64 and m=32. 110

7.1 MNIST classification results with the proposed WeightGenNet FC layers. 124
7.2 CIFAR-10 classification results with the proposed WeightGenNet Conv2D. 125

Publications & Patent

Journals

- **V. T. Nguyen**, W. Guicquero and G. Sicard, "A 1Mb Mixed-Precision Quantized Encoder for Image Classification and Patch-Based Compression," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 8, pp. 5581-5594, Aug. 2022.
- V. Poisson, **V. T. Nguyen**, W. Guicquero and G. Sicard, "Luminance-Depth Reconstruction From Compressed Time-of-Flight Histograms," in *IEEE Transactions on Computational Imaging*, vol. 8, pp. 148-161, 2022.

International conferences

- **V. T. Nguyen**, W. Guicquero and G. Sicard, "MOGNET: A Mux-residual quantized Network leveraging Online-Generated weights," *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Songdo Convensia, Incheon, South Korea, 2022, pp. 90-93.
- **V. T. Nguyen**, W. Guicquero and G. Sicard, "Histogram-Equalized Quantization for logic-gated Residual Neural Networks," *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, Texas, USA, 2022, pp. 1289-1293.
- **V. T. Nguyen**, W. Guicquero and G. Sicard, "BILLNET: A Binarized Conv3D-LSTM Network with Logic-gated residual architecture for hardware-efficient video inference," *2022 IEEE 36th International Workshop on Signal Processing Systems (SiPS)*, Rennes, France, 2022, pp. 1-6.

Patent

- V. T. Nguyen and W. Guicquero, "Réseau de neurones avec génération à la volée des paramètres du réseau," *Pending*.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[3] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *EMNLP*, 2015.

[4] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.

[5] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, p. 1527–1533.

[6] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2016, pp. 367–379.

[7] B. Liu, H. Cai, X. Zhang, H. Wu, A. Xue, Z. Zhang, Z. Wang, and J. Yang, "A target-separable bwn inspired speech recognition processor with low-power precision-adaptive approximate computing," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 196–201.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[9] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *CoRR*, vol. abs/1611.01578, 2016. [Online]. Available: http://arxiv.org/abs/1611.01578

[11] J. Faraone, N. Fraser, M. Blott, and P. H. Leong, "Syq: Learning symmetric quantization for efficient deep neural networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4300–4309.

[12] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 4107–4115.

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

[14] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France*, 2017.

[15] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019.

[16] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot Network pruning based on connection sensitivity," in *International Conference on Learning Representations*, 2019.

[17] W. Hua, Y. Zhou, C. De Sa, Z. Zhang, and G. E. Suh, *Channel Gating Neural Networks*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[18] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, "Dynamic slimmable network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 8607–8617.

[19] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2014.

[20] X. Ruan, Y. Liu, C. Yuan, B. Li, W. Hu, Y. Li, and S. Maybank, "EDP: An Efficient Decomposition and Pruning Scheme for Convolutional Neural Network Compression," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4499–4513, 2021.

[21] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *International Conference on Learning Representations*, 2017.

[22] N. Ma, X. Zhang, J. Huang, and J. Sun, "Weightnet: Revisiting the design space of weight networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[25] X. Zhao, Y. Wang, X. Cai, C. Liu, and L. Zhang, "Linear symmetric quantization of neural networks for low-precision integer hardware," in *ICLR*, 2020.

[26] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[27] H. Mo, L. Liu, W. Zhu, Q. Li, H. Liu, S. Yin, and S. Wei, "A multi-task hardwired accelerator for face detection and alignment," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 11, pp. 4284–4298, 2020.

[28] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

[29] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, “Training binary neural networks with real-to-binary convolutions,” in *International Conference on Learning Representations*, 2020.

[30] Z. Yang, Y. Wang, X. Chen, J. Guo, W. Zhang, C. Xu, C. Xu, D. Tao, and C. Xu, “Hournas: Extremely fast neural architecture search through an hourglass lens,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 10 891–10 901.

[31] Y. Li, S. Gu, L. Van Gool, and R. Timofte, “Learning filter basis for convolutional neural network compression,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 5622–5631.

[32] Y. Yang, J. Yu, N. Jojic, J. Huan, and T. S. Huang, “Fsnet: Compression of deep convolutional neural networks by filter summary,” in *International Conference on Learning Representations*, 2020.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[34] Z. Dai, H. Liu, Q. V. Le, and M. Tan, “CoAtNet: Marrying Convolution and Attention for All Data Sizes,” in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 3965–3977.

[35] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[36] M. Kraus, S. Feuerriegel, and A. Oztekin, “Deep learning in business analytics and operations research: Models, applications and managerial implications,” *European Journal of Operational Research*, vol. 281, no. 3, pp. 628–641, 2020.

[37] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, “Embedded floating-point units in fpgas,” in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, 2006, p. 12–20.

[38] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, “Deep Convolutional Neural Network Architecture With Reconfigurable Computation Patterns,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[39] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbruck, “NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, Mar. 2019.

[40] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, “Scalable methods for 8-bit training of neural networks,” in *NeurIPS*, 2018.

[41] A. Abdolrashidi, L. Wang, S. Agrawal, J. Malmaud, O. Rybakov, C. Leichner, and L. Lew, “Pareto-optimal quantized resnet is mostly 4-bit,” *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3085–3093, 2021.

[42] S.-F. Hsiao, K.-C. Chen, C.-C. Lin, H.-J. Chang, and B.-C. Tsai, "Design of a sparsity-aware reconfigurable deep learning accelerator supporting various types of operations," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 376–387, 2020.

[43] S. Sen, S. Jain, S. Venkataramani, and A. Raghunathan, "Sparce: Sparsity aware general-purpose core extensions to accelerate deep neural networks," *IEEE Transactions on Computers*, vol. 68, no. 6, pp. 912–925, 2019.

[44] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *ECCV*, 2018.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research (PMLR), vol. 9, 2010.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[47] G. Huang, S. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2752–2761.

[48] X. Wang, M. Kan, S. Shan, and X. Chen, "Fully learnable group convolution for acceleration of deep neural networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9041–9050.

[49] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3D cnns retrace the history of 2D cnns and imagenet?" in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6546–6555.

[50] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[51] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *NeurIPS*, 2018.

[52] J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *ArXiv*, vol. abs/1607.06450, 2016.

[53] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *ArXiv*, vol. abs/1607.08022, 2016.

[54] Y. Wu and K. He, "Group normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[55] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[56] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[57] J. Tee and D. P. Taylor, "Is information in the brain represented in continuous or discrete form?" *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 6, no. 3, pp. 199–209, 2020.

[58] R. VanRullen and C. Koch, "Is perception discrete or continuous?" *Trends in Cognitive Sciences*, vol. 7, no. 5, pp. 207–213, 2003.

[59] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *arXiv:1308.3432 [cs]*, Aug. 2013.

[60] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[61] Ü. Doğan, T. Glasmachers, and C. Igel, "A unified view on multi-class support vector classification," *Journal of Machine Learning Research*, vol. 17, no. 45, pp. 1–32, 2016.

[62] S. Yang, W.-T. Xiao, M. Zhang, S. Guo, J. Zhao, and S. Furao, "Image data augmentation for deep learning: A survey," *ArXiv*, vol. abs/2204.08610, 2022.

[63] 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, no. 56, pp. 1929–1958, 2014.

[64] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 648–656.

[65] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[66] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, 1987, pp. 318–362.

[67] L. Bottou, "On-line learning and stochastic approximations," in *In On-line Learning in Neural Networks*. Cambridge University Press, 1998, pp. 9–42.

[68] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*. PMLR, 17–19 Jun 2013, pp. 1139–1147.

[69] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 2121–2159, 2011.

[70] T. Tieleman, G. Hinton *et al.*, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *Coursera: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[71] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017.

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

[73] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," p. 60, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[74] A. Coates, A. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *AISTATS*, 2011.

[75] E. Agustsson and R. Timofte, “Ntire 2017 challenge on single image super-resolution: Dataset and study,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[76] J. Materzynska, G. Berger, I. Bax, and R. Memisevic, “The Jester dataset: A large-scale video dataset of human gestures,” in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 2874–2882.

[77] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian, “Sample-efficient neural architecture search by learning actions for monte carlo tree search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5503–5515, 2022.

[78] S. Zhao, L. Zhou, W. Wang, D. Cai, T. L. Lam, and Y. Xu, “Splitnet: Divide and co-training,” *ArXiv*, vol. abs/2011.14660, 2020.

[79] X. Wang, D. Kihara, J. Luo, and G.-J. Qi, “EnAET: A Self-Trained Framework for Semi-Supervised and Supervised Learning With Ensemble Transformations,” *IEEE Transactions on Image Processing*, vol. 30, pp. 1639–1647, 2021.

[80] R. e. a. Timofte, “Ntire 2017 challenge on single image super-resolution: Methods and results,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 1110–1121.

[81] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015*, Y. Bengio and Y. LeCun, Eds., 2015.

[82] 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. [Online]. Available: http://arxiv.org/abs/1602.07360

[83] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, “Residual attention network for image classification,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6450–6458.

[84] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-excitation networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 8, pp. 2011–2023, 2020.

[85] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[86] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997.

[87] M. O. Turkoglu, S. D’Aronco, J. D. Wegner, and K. Schindler, “Gating revisited: Deep multi-layer rnns that can be trained,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4081–4092, 2022.

[88] W. Wen, Y. He, S. Rajbhandari, M. Zhang, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li, “Learning intrinsic sparse structures within long short-term memory,” in *International Conference on Learning Representations*, 2018.

[89] G. Nan, C. Wang, W. Liu, and F. Lombardi, "DC-LSTM: Deep compressed LSTM with low bit-width and structured matrices," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[90] A. Tjandra, S. Sakti, and S. Nakamura, "Tensor decomposition for compressing recurrent neural network," *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.

[91] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*, 2015.

[92] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.

[93] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[94] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[95] L. Geiger and P. Team, "Larq: An open-source library for training binarized neural networks," *Journal of Open Source Software*, vol. 5, no. 45, p. 1746, 2020. [Online]. Available: https://doi.org/10.21105/joss.01746

[96] R. Andres, L. Wei, D. Jason, Z. Frank, G. Jiong, and Y. Chong. (2017) Intel Processors for Deep Learning Training. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-processors-for-deep-learning-training.html

[97] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing cnn model inference on cpus," in *USENIX Annual Technical Conference*, 2019.

[98] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen, and K. Olukolun, "The stanford hydra cmp," *IEEE Micro*, vol. 20, no. 2, pp. 71–84, 2000.

[99] L. Tang, Y. Wang, T. L. Willke, and K. Li, "Scheduling Computation Graphs of Deep Learning Models on Manycore CPUs," *arXiv e-prints*, p. arXiv:1807.09667, Jul. 2018.

[100] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, *Intel Math Kernel Library*. Cham: Springer International Publishing, 2014, pp. 167–188.

[101] E. Georganas, S. Avancha, K. Banerjee, D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on simd architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. IEEE Press, 2018.

[102] A. Zlateski, K. Lee, and H. S. Seung, "Znn – a fast and scalable algorithm for training 3d convolutional networks on multi-core and many-core shared memory machines," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 801–811.

[103] P. Dempsey, "Reviews consumer technology: The teardown: Apple iphone pro 13 smartphone," *Engineering Technology*, vol. 16, no. 11, pp. 68–69, 2021.

[104] Samsung Tech Blog. (2022) All About Exynos: Meet the GPU-ISP Development Leaders. [Online]. Available: https://semiconductor.samsung.com/newsroom/tech-blog/all-about-exynos-meet-the-gpu-isp-development-leaders/

[105] F. Dustin. (2019) Jetson Nano Brings AI Computing to Everyone. [Online]. Available: https://developer.nvidia.com/blog/jetson-nano-ai-computing/

[106] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on fpga," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[107] C. Wu, J. Zhuang, K. Wang, and L. He, "MP-OPU: A Mixed Precision FPGA-based Overlay Processor for Convolutional Neural Networks," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 33–37.

[108] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, p. 369–381.

[109] E. Wu, X. Zhang, D. Berman, and I. Cho, "A high-throughput reconfigurable processing array for neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[110] X. Lin, S. Yin, F. Tu, L. Liu, X. Li, and S. Wei, "Lcp: a layer clusters paralleling mapping method for accelerating inception and residual networks on fpga," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[111] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[112] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[113] F. e. a. Akopyan, "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[114] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 240–241.

[115] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, "A multi-functional in-memory inference processor using a standard 6t sram array," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 2, pp. 642–655, 2018.

[116] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "Diannao family: Energy-efficient hardware accelerators for machine learning," *Communications of the ACM*, vol. 59, no. 11, p. 105–112, oct 2016.

[117] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti, and G. S. Sachdev, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[118] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten lessons from three generations shaped google's tpuv4i : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.

[119] J. Yue, Y. Liu, R. Liu, W. Sun, Z. Yuan, Y.-N. Tu, Y.-J. Chen, A. Ren, Y. Wang, M.-F. Chang, X. Li, and H. Yang, "STICKER-T: An Energy-Efficient Neural Network Processor Using Block-Circulant Algorithm and Unified Frequency-Domain Acceleration," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 6, pp. 1936–1948, 2021.

[120] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, X. Ma, Y. Zhang, J. Tang, Q. Qiu, X. Lin, and B. Yuan, "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 395–408.

[121] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.

[122] X. Si, J.-J. Chen, Y.-N. Tu, W.-H. Huang, J.-H. Wang, Y.-C. Chiu, W.-C. Wei, S.-Y. Wu, X. Sun, R. Liu, S. Yu, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, Q. Li, and M.-F. Chang, "A Twin-8T SRAM Computation-in-Memory Unit-Macro for Multibit CNN-Based AI Edge Processors," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 189–202, 2020.

[123] Y.-D. e. a. Chih, "16.4 An 89TOPS/W and 16.3TOPS/mm2 All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 252–254.

[124] J.-S. Park, J.-W. Jang, H. Lee, D. Lee, S. Lee, H. Jung, S. Lee, S. Kwon, K. Jeong, J.-H. Song, S. Lim, and I. Kang, "9.5 a 6k-mac feature-map-sparsity-aware neural processing unit in 5nm flagship mobile soc," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 152–154.

[125] C.-H. Lin, C.-C. Cheng, Y.-M. Tsai, S.-J. Hung, Y.-T. Kuo, P. H. Wang, P.-K. Tsung, J.-Y. Hsu, W.-C. Lai, C.-H. Liu, S.-Y. Wang, C.-H. Kuo, C.-Y. Chang, M.-H. Lee, T.-Y. Lin, and C.-C. Chen, "7.1 a 3.4-to-13.3tops/w 3.6tops dual-core deep-learning accelerator for versatile ai applications in 7nm 5g smartphone soc," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 134–136.

[126] J.-H. Kim, C. Kim, K. Kim, and H.-J. Yoo, "An Ultra-Low-Power Analog-Digital Hybrid CNN Face Recognition Processor Integrated with a CIS for Always-on Mobile Devices," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.

[127] Y. Lu, V. L. Le, and T. T.-H. Kim, "9.7 A 184 μ W Real-Time Hand-Gesture Recognition System with Hybrid Tiny Classifiers for Smart Wearable Devices," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 156–158.

[128] F. Chen, K.-F. Un, W.-H. Yu, P.-I. Mak, and R. P. Martins, "A 108-nW 0.8-mm ^{2} Analog Voice Activity Detector Featuring a Time-Domain CNN With Sparsity-Aware Computation and Sparsified Quantization in 28-nm CMOS," *IEEE Journal of Solid-State Circuits*, pp. 1–10, 2022.

[129] K. Guo, W. Li, K. Zhong, Z. Zhu, S. Zeng, S. Han, Y. Xie, P. Debacker, M. Verhelst, and Y. Wang, "Neural network accelerator comparison," Tech. Rep. [Online]. Available: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/

[130] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up Convolutional Neural Networks with Low Rank Expansions," in *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[131] X. Ding, Y. Guo, G. Ding, and J. Han, "ACNet: Strengthening the Kernel Skeletons for Powerful CNN via Asymmetric Convolution Blocks," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1911–1920.

[132] D. Haase and M. Amthor, "Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets," in *CVPR*, 2020.

[133] Z. Su, L. Fang, W. Kang, D. Hu, M. Pietikäinen, and L. Liu, "Dynamic group convolution for accelerating convolutional neural networks," in *ECCV*, 2020.

[134] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[135] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *International Conference on Learning Representations*, 2019.

[136] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1294–1303.

[137] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2815–2823, 2019.

[138] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[139] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114.

[140] C. Hsu, S. Chang, D. Juan, J. Pan, Y. Chen, W. Wei, and S. Chang, "MONAS: multi-objective neural architecture search using reinforcement learning," *CoRR*, vol. abs/1806.10332, 2018. [Online]. Available: http://arxiv.org/abs/1806.10332

[141] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and Y. Shi, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[142] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *ArXiv*, vol. abs/1603.01025, 2016.

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

[144] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4345–4354.

[145] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *ArXiv*, vol. abs/1606.06160, 2016.

[146] 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*, vol. 32, 2019.

[147] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "ZeroQ: A Novel Zero Shot Quantization Framework," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 13 166–13 175.

[148] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv: Learning*, 2016.

[149] 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 28*, 2015, pp. 3123–3131.

[150] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 6869–6898, jan 2017.

[151] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Towards effective low-bitwidth convolutional neural networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7920–7928.

[152] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Networks*, vol. 100, pp. 49–58, 2018.

[153] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 4851–4860.

[154] Z.-G. Liu and M. Mattina, "Learning low-precision neural networks without straight-through estimator (ste)," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, ser. IJCAI'19. AAAI Press, 2019, p. 3066–3072.

[155] Z. Yang, Y. Wang, K. Han, C. XU, C. Xu, D. Tao, and C. Xu, "Searching for low-bit weights in quantized neural networks," in *Advances in Neural Information Processing Systems*, 2020.

[156] Y. Wang, Y. Yang, F. Sun, and A. Yao, "Sub-bit neural networks: Learning to compress and accelerate binary neural networks," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 5340–5349.

[157] H. Liu, S. Elkerdawy, N. Ray, and M. Elhoushi, "Layer importance estimation with imprinting for neural network quantization," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 2408–2417.

[158] M. van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling, "Bayesian bits: Unifying quantization and pruning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[159] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," *CoRR*, vol. abs/1812.00090, 2018. [Online]. Available: http://arxiv.org/abs/1812.00090

[160] Z. Cai and N. Vasconcelos, "Rethinking differentiable search for mixed-precision neural networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 2346–2355.

[161] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *Advances in Neural Information Processing Systems 28*, 2015.

[162] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2017.

[163] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 2234–2240.

[164] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations*, 2017.

[165] Y. Zhang, Y. Yuan, and Q. Wang, "Acp: Adaptive channel pruning for efficient neural networks," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 4488–4492.

[166] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016.

[167] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1398–1406.

[168] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5068–5076.

[169] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2755–2763.

[170] A. Murata, V. Gallese, G. Luppino, M. Kaseda, and H. Sakata, "Selectivity for the shape, size, and orientation of objects for grasping in neurons of monkey parietal area aip." *Journal of neurophysiology*, vol. 83 5, pp. 2580–601, 2000.

[171] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic convolution: Attention over convolution kernels," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 027–11 036.

[172] Y. Wang, K. Lv, R. Huang, S. Song, L. Yang, and G. Huang, "Glance and focus: a dynamic approach to reducing spatial redundancy in image classification," in *Advances in Neural Information Processing Systems*, 2020.

[173] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, “Multi-scale dense networks for resource efficient image classification,” in *International Conference on Learning Representations*, 2018.

[174] Z. Chen, Y. Li, S. Bengio, and S. Si, “You look twice: Gaternet for dynamic filter selection in cnns,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9164–9172.

[175] X. Wang, F. Yu, Z. Dou, T. Darrell, and J. E. Gonzalez, “Skipnet: Learning dynamic routing in convolutional networks,” in *Computer Vision - ECCV 2018 - 15th European Conference*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., pp. 420–436.

[176] L. F. Brillet, S. Mancini, S. Cleyet-Merle, and M. Nicolas, “Tunable cnn compression through dimensionality reduction,” in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 3851–3855.

[177] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, “Coordinating filters for faster deep neural networks,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 658–666.

[178] Y. Li, S. Gu, C. Mayer, L. Van Gool, and R. Timofte, “Group sparsity: The hinge between filter pruning and decomposition for network compression,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 8015–8024.

[179] M. Yin, Y. Sui, S. Liao, and B. Yuan, “Towards efficient tensor decomposition-based DNN model compression with optimization framework,” in *CVPR*, 2021.

[180] V. Aggarwal, W. Wang, B. Eriksson, Y. Sun, and W. Wang, “Wide compression: Tensor ring nets,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9329–9338.

[181] X. Jia, B. D. Brabandere, T. Tuytelaars, and L. V. Gool, “Dynamic filter networks,” *ArXiv*, vol. abs/1605.09673, 2016.

[182] J. Zhou, V. Jampani, Z. Pi, Q. Liu, and M.-H. Yang, “Decoupled dynamic filter networks,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 6643–6652.

[183] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “Fitnets: Hints for thin deep nets,” in *3rd International Conference on Learning Representations, ICLR*, Y. Bengio and Y. LeCun, Eds., 2015.

[184] Z. Xu, Y.-C. Hsu, and J. Huang, “Learning loss for knowledge distillation with conditional adversarial networks,” *ArXiv*, vol. abs/1709.00513, 2017.

[185] T. Li, J. Li, Z. Liu, and C. Zhang, “Few sample knowledge distillation for efficient network compression,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14 627–14 635.

[186] J. Gu and V. Tresp, “Search for better students to learn distilled knowledge,” in *ECAI*, 2020.

[187] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, “Improved knowledge distillation via teacher assistant,” in *AAAI*, 2020.

[188] J. Liu, D. Wen, H. Gao, W. Tao, T.-W. Chen, K. Osa, and M. Kato, “Knowledge representing: Efficient, sparse representation of prior knowledge for knowledge distillation,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019, pp. 638–646.

[189] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, 2017.

[190] P. C. Knag, G. K. Chen, H. E. Sumbul, R. Kumar, M. A. Anders, H. Kaul, S. K. Hsu, A. Agarwal, M. Kar, S. Kim, and R. K. Krishnamurthy, "A 617 TOPS/W All Digital Binary Neural Network Accelerator in 10nm FinFET CMOS," in *2020 IEEE Symposium on VLSI Circuits*, Jun. 2020, pp. 1–2, iSSN: 2158-5636.

[191] T.-H. Kim and J. Shin, "A Resource-Efficient Inference Accelerator for Binary Convolutional Neural Networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 1, pp. 451–455, Jan. 2021, conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs.

[192] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "Chewbaccann: A flexible 223 tops/w bnn accelerator," *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021.

[193] X. Zhou, L. Zhang, C. Guo, X. Yin, and C. Zhuo, "A convolutional neural network accelerator architecture with fine-granular mixed precision configurability," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[194] M. Nagel, M. V. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1325–1334.

[195] 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.

[196] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," *ArXiv*, vol. abs/1807.10029, 2018.

[197] F. Li and B. Liu, "Ternary weight networks," *ArXiv*, vol. abs/1605.04711, 2016.

[198] 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)*, 2019, pp. 3009–3018.

[199] Y. Li, W. Ding, C. Liu, B. Zhang, and G. Guo, "TRQ: Ternary Neural Networks With Residual Quantization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, pp. 8538–8546, May 2021, number: 10.

[200] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *8th International Conference on Learning Representations, ICLR*, 2020.

[201] S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshiyama, J. A. García, S. Tiedemann, T. Kemp, and A. Nakamura, "Mixed precision DNNs: All you need is a good parametrization," in *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30, 2020*.

[202] K. Nakata, D. Miyashita, J. Deguchi, and R. Fujimoto, "Adaptive Quantization Method for CNN with Computational-Complexity-Aware Regularization," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2021, pp. 1–5, iSSN: 2158-1525.

[203] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[204] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[205] Q. He, H. Wen, S. Zhou, Y. Wu, C. Yao, X. Zhou, and Y. Zou, "Effective quantization methods for recurrent neural networks," *ArXiv*, vol. abs/1611.10176, 2016.

[206] M. Z. Alom, A. T. Moody, N. Maruyama, B. C. V. Essen, and T. M. Taha, "Effective quantization approaches for recurrent neural networks," *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.

[207] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015.

[208] W. Xu, X. He, T. Zhao, Q. Hu, P. Wang, and J. Cheng, "Soft threshold ternary networks," in *IJCAI*, 2020.

[209] J. Yue, X. Feng, Y. He, Y. Huang, Y. Wang, Z. Yuan, M. Zhan, J. Liu, J.-W. Su, Y.-L. Chung, P.-C. Wu, L.-Y. Hung, M.-F. Chang, N. Sun, X. Li, H. Yang, and Y. Liu, "15.2 A 2.75-to-75.9TOPS/W Computing-in-Memory NN Processor Supporting Set-Associate Block-Wise Zero Skipping and Ping-Pong CIM with Simultaneous Computation and Weight Updating," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, Feb. 2021, pp. 238–240.

[210] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights," in *2016 IEEE Computer Society Annual Symposium on VLSI*, Jul. 2016, pp. 236–241.

[211] A. Verdant, W. Guicquero, N. Royer, G. Moritz, S. Martin, F. Lepin, S. Choisnet, F. Guellec, B. Deschamps, S. Clerc, and J. Chossat, "A 3.0µW@5fps QQVGA Self-Controlled Wake-Up Imager with On-Chip Motion Detection, Auto-Exposure and Object Recognition," in *2020 IEEE Symposium on VLSI Circuits*, Jun. 2020, pp. 1–2, iSSN: 2158-5636.

[212] M. Lefebvre, L. Moreau, R. Dekimpe, and D. Bol, "7.7 A 0.2-to-3.6TOPS/W Programmable Convolutional Imager SoC with In-Sensor Current-Domain Ternary-Weighted MAC Operations for Feature Extraction and Region-of-Interest Detection," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, Feb. 2021, pp. 118–120.

[213] X. Chen, J. Xu, and Z. Yu, "A 68-mw 2.2 tops/w low bit width and multiplierless dcnn object detection processor for visually impaired people," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 11, pp. 3444–3453, 2019.

[214] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in fpga design for cnn-based object detectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 6, pp. 2450–2464, 2021.

[215] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8604–8612.

[216] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "Releq : A reinforcement learning approach for automatic deep quantization of neural networks," *IEEE Micro*, vol. 40, no. 5, pp. 37–45, 2020.

[217] Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks," in *Advances in Neural Information Processing Systems*, 2020.

[218] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167 [cs]*, Mar. 2015.

[219] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An Always-On 3.8 μ J/86% CIFAR-10 Mixed-Signal Binary CNN Processor With All Memory on Chip in 28-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.

[220] J. Fromm, M. Cowan, M. Philipose, L. Ceze, and S. Patel, "Riptide: Fast end-to-end binarized neural networks," in *Proceedings of Machine Learning and Systems*, 2020.

[221] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-Tile 2.4-Mb In-Memory-Computing CNN Accelerator Employing Charge-Domain Compute," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, Jun. 2019.

[222] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, 2020.

[223] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2014.

[224] H. Gao, Z. Wang, and S. Ji, "Channelnets: Compact and efficient convolutional neural networks via channel-wise convolutions," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[225] E. Hoffer, I. Hubara, and D. Soudry, "Fix your classifier: the marginal value of training the last weight layer," *ICLR*, 2018.

[226] W. Benjilali, W. Guicquero, L. Jacques, and G. Sicard, "Hardware-Friendly Compressive Imaging Based on Random Modulations Permutations for Image Acquisition and Classification," in *2019 IEEE International Conference on Image Processing (ICIP)*, Sep. 2019, pp. 2085–2089.

[227] G. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[228] J. E. Fowler, S. Mun, and E. W. Tramel, 2012.

[229] Y. Oike and A. El Gamal, "CMOS Image Sensor With Per-Column $\sigma\delta$ ADC and Programmable Compressed Sensing," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 318–328, Jan. 2013.

[230] Y. Wu, X. Li, Z. Zhang, X. Jin, and Z. Chen, "Learned block-based hybrid image compression," *ArXiv*, vol. abs/2012.09550, 2020.

[231] A. Beck and M. Teboulle, "Fast Gradient-Based Algorithms for Constrained Total Variation Image Denoising and Deblurring Problems," *IEEE Transactions on Image Processing*, vol. 18, no. 11, pp. 2419–2434, Nov. 2009.

[232] W. Guicquero, A. Verdant, and A. Dupret, "High-order incremental sigma–delta for compressive sensing and its application to image sensors," *Electronics Letters*, vol. 51, no. 19, pp. 1492–1494, 2015.

[233] K. Kulkarni, S. Lohit, P. Turaga, R. Kerviche, and A. Ashok, "ReconNet: Non-Iterative Reconstruction of Images from Compressively Sensed Measurements," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[234] H. Yao, F. Dai, D. Zhang, Y. Ma, S. Zhang, and Y. Zhang, "DR^2-Net: Deep Residual Reconstruction Network for Image Compressive Sensing," *CoRR*, vol. abs/1702.05743, 2017. [Online]. Available: http://arxiv.org/abs/1702.05743

[235] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. C. Minnen, S. Baluja, M. Covell, and R. Sukthankar, "Variable rate image compression with recurrent neural network," in *ICLR*, 2016.

[236] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full Resolution Image Compression with Recurrent Neural Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jul. 2017.

[237] M. H. Baig, V. Koltun, and L. Torresani, "Learning to inpaint for image compression," in *NIPS*, 2017.

[238] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang, "Learning convolutional networks for content-weighted image compression," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3214–3223.

[239] C. Cai, L. Chen, X. Zhang, and Z. Gao, "Efficient variable rate image compression with multi-scale decomposition network," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 12, pp. 3687–3700, 2019.

[240] Y. Wang, D. Liu, S. Ma, F. Wu, and W. Gao, "Ensemble learning-based rate-distortion optimization for end-to-end image compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 3, pp. 1193–1207, 2021.

[241] Y. Ma, H. Xiong, Z. Hu, and L. Ma, "Efficient super resolution using binarized neural network," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 694–703, 2019.

[242] J. Xin, N. Wang, X. Jiang, J. Li, H. Huang, and X. Gao, "Binarized Neural Network for Single Image Super Resolution," in *Computer Vision – ECCV 2020*. Cham: Springer International Publishing, 2020.

[243] Y. Cai, T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Low Bit-Width Convolutional Neural Network on RRAM," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1414–1427, 2020.

[244] D. Miyashita, E. G. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *ArXiv*, vol. abs/1603.01025, 2016.

[245] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-Supervised Nets," in *Artificial Intelligence and Statistics*. Proceedings of Machine Learning Research (PMLR), Feb. 2015.

[246] K. Nakata, D. Miyashita, J. Deguchi, and R. Fujimoto, "Adaptive quantization method for cnn with computational-complexity-aware regularization," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[247] Y. Wang, Y. Lu, and T. Blankevoort, "Differentiable joint pruning and quantization for hardware efficiency," in *European Conference on Computer Vision (ECCV)*, 2020.

[248] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[249] Y. Xiao, J. Yang, X. Yuan, ,Institute of Applied Mathematics, Henan University, Kaifeng 475004, ,Department of Mathematics, Nanjing University, Nanjing, 210093, and ,Department of Mathematics, Hong Kong Baptist University, Hong Kong, "Alternating algorithms for total variation image reconstructionfrom random projections," *Inverse Problems & Imaging*, vol. 6, no. 3, pp. 547–563, 2012.

[250] J. Lee, S. Cho, and S. Beack, "Context-adaptive entropy model for end-to-end optimized image compression," in *ICLR*, 2019.

[251] Z. Wang, E. P. Simoncelli, and A. Bovik, "Multi-scale structural similarity for image quality assessment," in *In Signals, Systems and Computers 2004. Conference Record of the Thirty-Seventh Asilomar Conference*, 2003.

[252] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[253] D. Oktay, J. Ballé, S. Singh, and A. Shrivastava, "Scalable model compression by entropy penalized reparameterization," in *International Conference on Learning Representations*, 2020.

[254] W. Lan and L. Lan, "Compressing deep convolutional neural networks by stacking low-dimensional binary convolution filters," in *AAAI*, 2021.

[255] K. Nakata, D. Miyashita, A. Maki, F. Tachibana, S. Sasaki, J. Deguchi, and R. Fujimoto, "Quantization strategy for pareto-optimally low-cost and accurate cnn," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[256] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K. Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *ECCV*, 2018.

[257] H. T. Phan, D. Huynh, Y. He, M. Savvides, and Z. Shen, "Mobinet: A mobile binary network for image classification," *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 3442–3451, 2020.

[258] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. W. Mahoney, and K. Keutzer, "Hawqv3: Dyadic neural network quantization," in *ICML*, 2021.

[259] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6450–6458, 2017.

[260] M. Horowitz, "Computing's energy problem (and what we can do about it)," *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014.

[261] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *ArXiv*, vol. abs/1708.04552, 2017.

[262] D. Oktay, J. Ballé, S. Singh, and A. Shrivastava, "Scalable model compression by entropy penalized reparameterization," in *ICLR*, 2020.

[263] R. Giryes, G. Sapiro, and A. M. Bronstein, "Deep neural networks with random gaussian weights: A universal classification strategy?" *IEEE Transactions on Signal Processing*, vol. 64, no. 13, pp. 3444–3457, 2016.

[264] V. Lempitsky, A. Vedaldi, and D. Ulyanov, "Deep image prior," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9446–9454.

[265] S. Wolfram, *A New Kind of Science*. Champaign, Ilinois, USA: Wolfram Media Inc., 2002.

[266] J. Liu and Q. Sun, "Chaotic cellular automaton for generating measurement matrix used in cs coding," *IET Signal Process.*, vol. 11, 2017.

[267] Ö. Yılmaz, "Reservoir computing using cellular automata," *ArXiv*, vol. abs/1410.0162, 2014.

[268] L. M. Antunes, "Cellpylib: A python library for working with cellular automata," *Journal of Open Source Software*, vol. 6, no. 67, 2021.

[269] S. Sudhakaran, S. Escalera, and O. Lanz, "Gate-shift networks for video action recognition," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1099–1108.

[270] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool, "Temporal segment networks for action recognition in videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 11, pp. 2740–2755, 2019.

[271] M. O. Turkoglu, S. D'Aronco, J. Wegner, and K. Schindler, "Gating revisited: Deep multi-layer RNNs that can be trained," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[272] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the Kinetics dataset," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4724–4733.

[273] S. Abu-El-Haija, N. Kothari, J. Lee, A. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "YouTube-8M: A large-scale video classification benchmark," *ArXiv*, vol. abs/1609.08675, 2016.

[274] O. Köpüklü, N. Kose, A. Gunduz, and G. Rigoll, "Resource efficient 3D convolutional neural networks," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 1910–1919.

[275] A. J. Piergiovanni, A. Angelova, and M. S. Ryoo, "Tiny video networks," *ArXiv*, vol. abs/1910.06961, 2019.

[276] M. Sun, P. Zhao, M. Gungor, M. Pedram, M. Leeser, and X. Lin, "3D CNN acceleration on FPGA using hardware-aware pruning," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[277] J. Ott, Z. Lin, Y. Zhang, S.-C. Liu, and Y. Bengio, "Recurrent neural networks with limited numerical precision," *ArXiv*, vol. abs/1611.07065, 2016.

[278] M. Z. Alom, A. T. Moody, N. Maruyama, B. C. Van Essen, and T. M. Taha, "Effective quantization approaches for recurrent neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[279] K. Hara, H. Kataoka, and Y. Satoh, “Learning spatio-temporal features with 3D residual networks for action recognition,” in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 3154–3160.

[280] X. Du, Y. Li, Y. Cui, R. Qian, J. Li, and I. Bello, “Revisiting 3D resnets for video recognition,” *ArXiv*, vol. abs/2109.01696, 2021.

[281] J. Li, X. Liu, W. Zhang, M. Zhang, J. Song, and N. Sebe, “Spatio-temporal attention networks for action recognition and detection,” *IEEE Transactions on Multimedia*, vol. 22, no. 11, pp. 2990–3001, 2020.

[282] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, “A closer look at spatiotemporal convolutions for action recognition,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6450–6459.

[283] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, “Rethinking spatio-temporal feature learning: Speed-accuracy trade-offs in video classification,” in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XV*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., pp. 318–335.

[284] Z. Qiu, T. Yao, and T. Mei, “Learning spatio-temporal representation with pseudo-3D residual networks,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5534–5542.

[285] J. Lin, C. Gan, and S. Han, “TSM: Temporal shift module for efficient video understanding,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 7082–7092.

[286] X. Sun, R. Panda, C.-F. R. Chen, A. Oliva, R. Feris, and K. Saenko, “Dynamic network quantization for efficient video inference,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 7375–7385.

[287] G. Li, M. Zhang, Q. Zhang, and Z. Lin, “Efficient binary 3D convolutional neural network and hardware accelerator,” *J. Real Time Image Process.*, vol. 19, no. 1, pp. 61–71, 2022.

[288] A. Fasoli, C. Chen, M. J. Serrano, X. Sun, N. Wang, S. Venkataramani, G. Saon, X. Cui, B. Kingsbury, W. Zhang, Z. Tüske, and K. Gopalakrishnan, “4-bit quantization of LSTM-based speech recognition models,” *CoRR*, vol. abs/2108.12074, 2021.

[289] P. Wang, X. Xie, L. Deng, G. Li, D. Wang, and Y. Xie, “HitNet: Hybrid ternary recurrent neural network,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[290] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv:1308.3432 [cs]*, Aug. 2013.

[291] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[292] L. Deutsch, E. Nijkamp, and Y. Yang, “A generative model for sampling high-performance and diverse weights for neural networks,” *ArXiv*, vol. abs/1905.02898, 2019.

[293] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, “Reactnet: Towards precise binary neural network with generalized activation functions,” in *European Conference on Computer Vision (ECCV)*, 2020.

[294] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, “Minimum energy quantized neural networks,” in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 1921–1925.

[295] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, “Dnn+neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies,” in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 32.5.1–32.5.4.

Deep Neural Networks hardware-algorithmic enablers for compact ASIC design towards embedded image/video processing

Conception de réseaux de neurones profonds pour le traitement embarqué d'image/vidéo, compatible avec une architecture micro-électronique frugale

Résumé

Les réseaux de neurones profonds (DNNs) sont devenus la solution de référence pour diverses applications requérant de l'inférence sur des données. Porter ce type d'algorithmes au sein de systèmes embarqués, pour des applications dites d'intelligence artificielle, constitue un défi pour des raisons liées à l'efficacité énergétique, la préservation de la vie privée, la latence de calcul et des problématiques de bande passante. Or les performances exceptionnelles de DNNs sont principalement liées au fait du haut niveau de complexité des architectures de calcul, exploitant un grand nombre de paramètres, mais limitant de facto leur potentiel déploiement au sein des systèmes fortement contraints. Le développement des DNNs nécessitent également des plateformes de calculs dédiées pour cela, lors des phases d'entrainement et d'inférence. Le choix de cet accélérateur matériel dépend par conséquent de la tâche ciblée et du budget disponible, surtout concernant la consommation. Dans un contexte où le cas d'usage est bien défini pour une application à basse consommation, la conception d'un dispositif spécifique dédié à une application précise est un choix pertinent, permettant d'atteindre de meilleurs compromis entre la performance algorithmique et l'efficacité énergétique. La clé de ce type d'accélérateur réside dans une conception conjointe matérielle-algorithmique. Cette thèse aborde donc le problème de conception des DNNs compacts pour le traitement embarqué d'image/vidéo, compatible avec une architecture micro-électronique frugale. Ce manuscrit se focalise sur la généralisation des techniques de quantification comme approche centrale pour compresser des réseaux exploitant les dernières avancées en termes de conception de DNNs (mécanismes d'attention, connexions résiduelles, réseaux récurrents et poids non appris), permettant de réduire considérablement le besoin de mémoire et de calcul, tout en facilitant son implémentation matérielle. En outre, cette thèse relate également une étude exploratoire portant sur la génération des poids de DNNs via des réseaux annexes pour réduire le nombre de paramètres utiles au niveau système.

Mots-clés : Apprentissage profonds, Quantification des DNNs, Réseaux récurrents, Réseaux auxiliaires, Classification d'image, Compression d'image.

Abstract

Deep Neural Networks (DNNs) have become the state-of-the-art solution for diverse applications in the last few years. As a result, pushing DNN inference to the edge devices for smart applications is increasingly demanded, for the reason of energy-efficiency, privacy, latency and limited bandwidth. Unfortunately, their excellent performances are mainly due to complex computational architectures involving a huge number of parameters and operations, causing several hardware-related overheads when deployed in resource-constrained systems. The choice of DNN accelerator deeply depends on the targeted task and the available resource budget, especially power consumption. In the context where the targeted task is well-defined for a low-power application, an application-specific accelerator (ASIC) is the most relevant choice in the means of achieving the best compromise between algorithmic performance and hardware efficiency, by featuring fine-grained hardware-algorithmic co-design in response to the complexity of the task and the available hardware budget. In this thesis, we address the problem of designing hardware-compliant DNNs for compact application-specific accelerators towards embedded image/video processing. We mainly focus on the generalization of model quantization as the pivotal approach to compress DNN with advanced features such as attention-like mechanisms, residual connections, recurrent layers and fixed weights. It enables to drastically reduce the memory needs and the computational costs, while easing hardware implementation. Besides, we also present some exploratory studies on weight generation networks to alleviate the parameter-heaviness of DNNs.

Keywords : Deep Learning, Quantization-Aware Training, Recurrent Neural Networks, Mixed-Precision, Image/Video classification, Image compression, ASIC.