TU Delft

Delft University of Technology

Algorithms for Efficient Inference in Convolutional Neural Networks

Zhu, B.

DOI
10.4233/uuid:0943e030-7486-4ee6-8e7e-b35d02d528b0

Publication date
2021

Document Version
Final published version

Citation (APA)
Zhu, B. (2021). *Algorithms for Efficient Inference in Convolutional Neural Networks*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:0943e030-7486-4ee6-8e7e-b35d02d528b0

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

ALGORITHMS FOR EFFICIENT INFERENCE IN CONVOLUTIONAL NEURAL NETWORKS

ALGORITHMS FOR EFFICIENT INFERENCE IN CONVOLUTIONAL NEURAL NETWORKS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus Prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Thursday 16 September 2021 at 12:30 o'clock

by

Baozhou ZHU

Master of Engineering in Electronic Science and Technology
National University of Defense Technology, China
born in Heze, China

This dissertation has been approved by the promotors:

promotor: Dr.ir. Z. Al-Ars
promotor: Prof.dr. H.P. Hofstee

Composition of the doctoral committee:

Rector Magnificus	chairman
Dr.ir. Z. Al-Ars	Delft University of Technology, promotor
Prof.dr. H.P. Hofstee	Delft University of Technology, promotor
Prof.dr.ir. R.L. Langendijk	Delft University of Technology
Prof.dr. T.M. Heskes	Radboud University Nijmegen, The Netherlands
Prof.dr. J. Lee	Yonsei University, South Korea
Prof. C. Wu	National University of Defense Technology, China
Dr. W. Pan	Delft University of Technology
Prof.dr.ir. A. Bozzon	Delft University of Technology, reserve member

This research was financially supported by China Scholarship Council (CSC)

Keywords: Convolution, inference, efficiency, acceleration, approximation, architecture design, search, attention, feature reuse, reconstruction
Printed by: Ipskamp Printing, the Netherlands
Front & Back: Designed by Baozhou Zhu.

Copyright © 2021 by Baozhou Zhu

ISBN 000-00-0000-000-0

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Dedicated to:

my parents and friends for supporting me in this journey.

CONTENTS

SUMMARY

In recent years, the accuracy of Deep Neural Networks (DNNs) has improved significantly because of three main factors: the availability of massive amounts training data, the introduction of powerful low-cost computational resources, and the development of complex deep learning models. The cloud can provide powerful computational resources to calculate DNNs but limits their deployment due to data communication and privacy issues. Thus, computing DNNs at the edge is becoming an important alternative to calculating these models in a centralized service. However, there is a mismatch between the resource-constrained devices at the edge and the models with increased computational complexity. To alleviate this mismatch, both the algorithms and hardware need to be explored to improve the efficiency of training various feedforward and recurrent neural networks and inferring using a DNN.

In this thesis, our focus is on the algorithms used for efficient inference in Convolutional Neural Networks (CNNs) and this thesis mainly covers acceleration, quantization, and efficient neural network architecture design, as discussed in the following five contributions. First, the proposed Diminished-1 Fermat Number Transform (DFNT) can accelerate integer CNNs without introducing any round-off error. The diminished-1 point-wise products between DFNT transformed feature maps are used to calculate convolution and are reused multiple times to reduce the number of multiplications and additions. To compute integer convolution with typical parameter configurations, the DFNT achieves a speedup of $2-3x$ compared with the direct method. Second, the proposed piecewise approximation scheme reduces accuracy drop for multiple binary CNNs. The proposed scheme makes full use of the pre-trained full-precision model since the distribution of the full-precision values is similar to that of the approximated values. With the proposed piecewise approximation, both the Top-1 and Top-5 accuracy drop of ResNet on the ImageNet is only approximately 1.0%. Third, neural architecture search and effective depth reduction techniques are investigated to design neural network architectures for binary CNNs. When adapting neural network search, the discovered neural network architectures achieve a better trade-off between accuracy and efficiency than current binary CNNs. With the negligible overhead increase, the Top-1 accuracy of binary CNNs discovered by the NASB strategy surpasses the existing single and multiple binary CNNs by 4.0% and 1.0%, respectively. When unifying the fractal architecture and shortcut effective depth reduction techniques, the developed unified architectures achieve better accuracy than Bi-Real ResNet and BinaryDenseNet. With almost the same computational complexity cost, the Top-1 accuracy of UA-ResNet37(41) and UA-DenseNet51(53) on ImageNet is 3.29% and 1.41% better than Bi-Real ResNet18(64) and BinaryDenseNet51(32), respectively. Fourth, the feature reuse reduction and attention mechanism are used to design efficient neural network architectures for full-precision CNNs. The proposed scheme reduces the feature reuse within the convolution to make channels accurate and outperforms the standard convolution with full feature

reuse. Under the same computational budget, the Top-1 accuracy of REAF-ResNet50 and REAF+-MobileNetV2 on ImageNet is 0.37% and 0.69% better than their standard counterparts, respectively. The proposed attention module solves the approximation problem and the insufficient capacity problem of the attention maps, and it can boost the accuracy of attention-based models further with proper integration. Integrating with the proposed attention model, the current attentional activations-based models can increase their Top-1 accuracy on ImageNet classification by 0.57% and COCO-style Average Precision on the COCO object detection by 0.45. Last, the proposed reconstruction method can automatically discover an optimized generator to compute the reconstructed training dataset when the original training dataset is not available for model compression. The accuracy of current data-free compression improves when replacing the human-designed generator with the searched generator. Using ResNet50 as the pre-trained model, 5-bit width quantization, and an optimized generator, our data-free compression on ImageNet outperforms the Top-1 accuracy of the GDFQ method by 8.43%.

These solutions proposed in this thesis can improve inference efficiency in CNNs and make it possible for complex CNNs to be deployed at resource-limited edge devices.

SAMENVATTING

In de afgelopen jaren is de nauwkeurigheid van Deep Neural Networks (DNNs) aanzienlijk verbeterd vanwege drie belangrijke factoren: de beschikbaarheid van enorme hoeveelheden trainingsgegevens, de introductie van krachtige, goedkope computermiddelen en de ontwikkeling van complexe deep learning-modellen. De cloud biedt toegang tot krachtige computers om DNN berekeningen uit te voeren, maar kan beperkt worden ingezet vanwege problemen met datacommunicatie en privacy. DNN berekeningen in de edge worden daarom een belangrijk alternatief voor het berekenen van deze modellen in een datacentrum. Er is echter een discrepantie tussen de middelen die computers in de edge tot hun beschikking hebben en de complexiteit van de modellen. Om deze discrepantie te verminderen, moeten zowel de algoritmen als de hardware worden onderzocht om de efficiëntie van het trainen van verschillende feedforward en recurrent neural networks en inferencing met behulp van een DNN te verbeteren.

In dit proefschrift ligt onze focus op de algoritmen die worden gebruikt voor efficiënte inferencing voor convolutional neural networks (CNNs). Dit proefschrift behandelt voornamelijk het versnellen, kwantiseren en ontwerpen van efficiënte architecturen voor neurale netwerken, middels de volgende vijf bijdragen. Ten eerste kan de voorgestelde Diminished-1 Fermat Number Transform (DFNT) integer CNNs versnellen zonder enige afrondingsfouten te introduceren. De diminished-1 point-wise products tussen de DFNT-getransformeerde feature maps worden gebruikt om convolutie te berekenen en worden meerdere keren hergebruikt om het aantal vermenigvuldigingen en optellingen te verminderen. Om convolutie van gehele getallen te berekenen met typische parameterconfiguraties, behaalt de DFNT-methode een versnelling van $2-3x$ vergeleken met de directe methode. Ten tweede vermindert het voorgestelde piecewise approximation scheme de nauwkeurigheidsafname voor meerdere binaire CNNs. Het voorgestelde schema maakt volledig gebruik van het vooraf getrainde model met volledige precisie, aangezien de verdeling van de waarden met volledige precisie vergelijkbaar is met die van de benaderde waarden. Met de voorgestelde piecewise approximation is zowel de top-1- als de top-5-nauwkeurigheidsdaling van ResNet op ImageNet slechts ongeveer 1.0%. Ten derde worden neural architecture search en effectieve dieptereductietechnieken onderzocht om architecturen voor binaire CNNs te ontwerpen. Bij het aanpassen van neural network search bereiken de ontdekte neurale netwerkarchitecturen een betere afweging tussen nauwkeurigheid en efficiëntie dan de huidige binaire CNNs. Met een verwaarloosbare overhead overtreft de Top-1-nauwkeurigheid van binaire CNNs die via de NASB-strategie zijn ontdekt de bestaande enkelvoudige en meervoudige binaire CNNs met respectievelijk 4.0% en 1.0%. Bij het verenigen van de fractal-architectuur en effectieve technieken voor diepte-reductie, bereiken de ontwikkelde verenigde architecturen een betere nauwkeurigheid dan Bi-Real ResNet en BinaryDenseNet. Met bijna dezelfde complexiteit is de Top-1 nauwkeurigheid van UA-ResNet37 (41) en UA-DenseNet51 (53) op ImageNet

3.29% en 1.41% beter dan respectievelijk Bi-Real ResNet18 (64) en BinaryDenseNet51 (32). Ten vierde worden het feature reuse reduction and attention mechanism gebruikt om efficiënte neurale netwerkarchitecturen te ontwerpen voor CNNs met volledige precisie. De voorgestelde aanpak vermindert het hergebruik van features binnen de convolutie om kanalen nauwkeurig te maken en overtreft de standaardconvolutie met volledig hergebruik van features. Met hetzelfde rekenbudget is de Top-1 nauwkeurigheid van REAF-ResNet50 en REAF+-MobileNetV2 op ImageNet respectievelijk 0.37% en 0.69% beter dan die van hun standaard tegenhangers. De voorgestelde attention module lost het benaderingsprobleem en het probleem van onvoldoende capaciteit van de attention maps op, en kan de nauwkeurigheid van op attention-gebaseerde modellen verder vergroten met de juiste integratie. Dankzij integratie met het voorgestelde attention model, kunnen de huidige op attention-activation gebaseerde modellen hun Top-1 nauwkeurigheid op ImageNet-classificatie verhogen met 0.57% en COCO-style Average Precision op de COCO-objectdetectie met 0.45. Ten slotte kan de voorgestelde reconstructiemethode automatisch een geoptimaliseerde generator ontdekken om de gereconstrueerde trainingsdataset te berekenen wanneer de originele trainingsdataset niet beschikbaar is voor modelcompressie. De nauwkeurigheid van de huidige datavrije compressie verbetert wanneer de door mensen ontworpen generator wordt vervangen door de gezochte generator. Bij het gebruik van ResNet50 met 5-bit kwantisering en een geoptimaliseerde generator als het vooraf getrainde model, overtreft onze gegevensvrije compressie op ImageNet de Top-1-nauwkeurigheid van de GDFQ-methode met 8.43%.

De oplossingen die in dit proefschrift worden voorgesteld kunnen zowel de inference-efficiëntie in CNNs verbeteren alsmede het mogelijk maken van implementaties van complexe CNNs op edge-apparaten met beperkte capaciteit.

ACKNOWLEDGEMENTS

Time flies. It has been four years since I started my Ph.D. position. It is an unforgettable experience to carry out my research with my professors, colleagues, and friends in a city like Delft. I am going to graduate soon and will attempt to express my appreciation for these wonderful people.

I would like to express my deepest gratitude to Zaid Al-Ars. It is one of the luckiest things to get in touch and interact with Zaid in the past four years. As my supervisor, Zaid instructs me on how to explore a research opportunity, write a scientific paper, present at a conference, and collaborate in a research team. As my mentor, Zaid exercises considerable influence over my philosophy, sense of worth, and worldview. As my friend, Zaid brings lots of happiness at labs, bars, barbecues, games, etc. Zaid, thanks a lot for your outstanding sense of responsibility, availability, optimism, patience, goodness, and wisdom.

I would like to say a very big thank you to Peter Hofstee. I learn gradually about the research methodology from you, which helps me a lot to figure out the direction during my exploration time. Besides research, I learn more from you how to have a warm, modest, understanding, and tolerant personality, which will be beneficial throughout my life. In one word, I can see a much broader open space of both the research and life when I discuss with you.

I would like to express my appreciation to Prof. Koen Berterls and Cees Timmers for putting their trust in me, interviewing me and providing me with the invitation letter to come to Delft.

I would like to especially thank Dr. Lee Jinho for his contributions to the brainstorm sessions every Friday. These sessions are very helpful and turn out to be fruitful collaboration opportunities. I would like to thank Dr. Pan Wei for engaging with me in valuable discussion. I would like to appreciate Dr. Zhuang Bohan, Dr. Wu Bichen, Dr. Gong Xinyu, Dr. Jiang Zixuan, and Dr. Jaemin Yoo for our email interactions and their discussions that sped up our research.

I would like to thank Lidwina, Joyce, Laura, Trisha, and Paul for managing paperwork, administration matters, and other secretary-related tasks. I am thankful to Erik, Robbert, and Vali for providing technical support related to cluster usage. Erik, thank you for all your technical help, and for fixing my computers when they broke. Without you, this research would've not been nearly as possible.

I would like to express my appreciation to my friends in the accelerated big data systems group. They are Johan Peltenburg, Nauman Ahmed, Robin Hes, Dorus Leliveld, Matthijs Brobbel, Jeroen van Straten, Tanveer Ahmad, Joost Hoozemans, Jakoba Petri-Koenig, Ji Mengfei, and Helmi Helmiriawan. Thank you for the valuable discussion and pleasant working environment. Thank you for the beer, coffee, and cheerful chats.

I would like to express my gratitude to Prof. Peng Yuanxi and Dr. Lei Yuanwu for encouraging me. I would like to thank my friends in Changsha for helping me dealing with

personal affairs. They are Zou Tong, Liu Lu, Zhou Shijie, Song Minghui, Zhao Liyuan, Hu Xiang, Wu Luting, Yang Shuo, and Li Fu.

I would like to extend my thanks to my Chinese Ph.D. friends in Delft. They are Wang Senlei, Zhang jian, Xie Lei, Yu Jintao, Ren Shanshan, Fang Jian, Fu Xiang, Lao LingLing, Jiang Yande, Na Chen, Wang He, Wu Lizhou, Zhang Yu, Yang Yazhou, Zhao Yue, Zhu Hai, Qiu Sihang, Wang Xiaohui, Zhang Meng, Jiang Min, and Zhang Xunhui. It is my pleasure to have a good time with you. Thank you for the beer, games, and traveling.

Last but not least, I would like to express my deepest thanks to my family for supporting me, in particular my father, mother, and little sister. A man who does not spend time with his family can never be a real man. I will spend more time with you and shoulder more responsibilities after returning to China.

1

INTRODUCTION

The recent success of Deep Neural Networks (DNNs) can be attributed to three main factors: 1. the increase of the volume of training data, 2. the increase of complexity of models, and 3. the increase of low cost computational power. Relying on cloud services to compute DNNs has a number of major challenges both in terms of communication cost and privacy issues. This resulted in a fast growing interest in computing DNNs at the edge. However, lightweight devices used at the edge cannot provide enough resources to compute DNNs with large computational complexity. Various algorithms and hardware are designed to enable computing DNNs at the edge. In particular, this thesis focuses on proposing new and innovative algorithms for efficient inference in Convolutional Neural Networks (CNNs).

This chapter introduces the motivation behind efficient processing of DNNs in Section 1.1 and discusses the opportunities and challenges related to both algorithm and hardware in Section 1.2. Then, it describes the research questions and the corresponding contributions in Section 1.3 and Section 1.4, respectively. Finally, it presents the thesis organization in Section 1.5.

1

1.1. MOTIVATION

Today, research on DNNs has achieved remarkable progress in various fields, including computer vision [1], natural language processing [2], healthcare [3], robotics [4], reinforcement learning [5], and so on. For example, the Top-5 error rate of ResNet ensembles [1] on the ImageNet classification dataset is 3.57% and is better than the human-level accuracy of 5.10% in 2015. In 2018, Microsoft's machine translation system [2] on the WMT 2017 Chinese to English news task has reached parity with professional human translations and significantly exceeded the quality of crowd-sourced non-professional translations.

Reasons for the success of DNNs: The success of DNNs is attributed mainly to three factors: an increase in the amount of available training data, an increase in neural network architecture complexity, and an increase in low cost computation power.

- **Data**—Large-scale labeled training data has recently become available for DNNs. For example, the ImageNet classification dataset contains 1.3 million images. The training data, on which GPT-2 [6] is pre-trained, consists of over 8 million documents for a total of 40GB of text. It has been reported in [7] that the performance on vision tasks increases logarithmically with the increase in training data volume.

- **Model**—It has been observed in [1, 8] that an increase in neural network architecture complexity can lead to an accuracy improvement. The neural network architecture complexity can be measured by the number of parameters and the number of floating-point operations.

- **Compute**—The increase in computation capacity is an essential prerequisite for benefiting from the increase in both training data and neural network architecture complexity. For example, the compute capacity of GPUs has been increasing significantly in the past decade, reaching 5.6 TFLOPs per second for the K80 GPU in 2014, up to 130 TFLOPs per second for V100 in 2017.

Drawbacks of computing DNNs in a cloud: DNNs rely on cloud services to provide a large volume of training data and powerful computation capacity, which impedes the deployment for mainly two reasons: communication cost and privacy concerns.

- Many applications cannot afford the cost of sending data to the cloud for processing. For example, autonomous driving [9] cannot tolerate the latency cost of transmitting data to the cloud since real-time computation is one of the most crucial factors to interact with a complex real-world environment.

- Many applications do not wish to send their data to the cloud to avoid information leakage. For example, privacy regulation prevents biometric identification data and other personal data [10] from being sent to the cloud.

As a result, there is a huge interest in deploying DNNs at the edge, where lightweight devices can deliver computation locally.

Characteristics of DNNs and lightweight devices: To deploy DNNs on lightweight devices, we need to consider the characteristics of DNNs and lightweight devices as follows.

- Accuracy improvement of DNNs comes with a significant increase in computational complexity. For example, the Top-1 accuracy of ResNet152 [1] on the ImageNet classification dataset is 23.00% and is 7.24% better than that of ResNet18. However, the number of FLOPs and the number of parameters to compute ResNet152 is 11.3×10^9 and 60.2×10^6, respectively, while ResNet18 needs 1.8×10^9 FLOPs and 11.7×10^6 parameters to calculate, respectively.

- Lightweight devices provide constrained computation capacity in terms of FLOPs, energy consumption, and on-chip storage size. For example, the latest mobile product MediaTek Helio P90 consists of two Arm Cortex-A75 processors, and each processor delivers 2.2×10^9 half-precision floating-point operations per second[1]. Typically, the number of parameters and the number of FLOPs for mobile models [8] are required to be less than 600×10^6 and 5×10^6, respectively.

Challenges of computing DNNs at the edge: The mismatch between the need of DNNs to have an increased computation complexity on the one hand, and the resource-constrained edge devices on the other hand, will lead to the following challenges.

- It is challenging for lightweight devices to provide low-latency and real-time computation for DNNs. More arithmetic operations, more computation cycles, and more memory reference cycles are needed to compute a larger model. Besides, it is slower to access off-chip DRAM rather than on-chip SRAM.

- It is challenging for battery-constrained devices to provide enough energy to calculate DNNs. A larger model, which cannot fit in on-chip storage, consumes more energy to fetch weights from off-chip DRAM. Energy consumption for an off-chip memory access is two orders of magnitude larger than that for an arithmetic operation [11].

1.2. OPPORTUNITIES AND CHALLENGES

In this section, we present a comparison between training and inference. Then, we compare feedforward and recurrent neural networks. In particular, we illustrate CNNs. Finally, we describe the algorithms and hardware for efficient inference in CNNs.

1.2.1. TRAINING VERSUS INFERENCE

We compare the training and inference processes and their corresponding computational platform, and we will focus on inference rather than training in this thesis below.

Training: Training is an iterative process to determine the weights and bias of DNNs with training data by minimizing the loss. There are three steps of the training process:

[1] https://www.mediatek.com/products/smartphones/mediatek-helio-p90

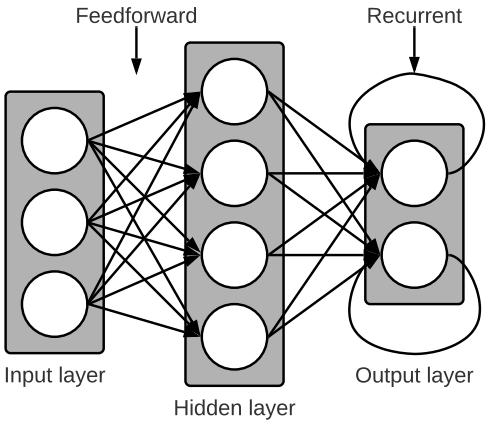


Figure 1.1: Feedforward versus recurrent networks.

forward propagation, backward propagation, and parameter update. The forward propagation computes the output of DNNs, with which loss is defined. Then, the partial derivatives of the gradient are derived from the chain rule in the backward propagation. Finally, the parameters are updated based on the partial derivatives of the gradient and the learning rate. Training has demanding computation and storage needs, and it is typically performed in a cloud. This iterative process and the three steps needed during training require significant computational resources. In addition, the training data and the intermediate outputs of DNNs for backward propagation need a large storage volume.

Inference: Inference is a process to compute the output of the DNNs with the determined weights and bias in the training process. In the inference process, there is only one step needed: the forward propagation step. Inference is much less computationally intensive than the training process, and it can happen either in the cloud or at the edge.

1.2.2. FEEDFORWARD VERSUS RECURRENT NEURAL NETWORKS

We present a comparison between feedforward and recurrent neural networks as shown in Figure 1.1. Our thesis is about CNNs, one of the most widely used forms of feedforward neural networks.

Feedforward neural networks: In a feedforward neural network, the input of each layer is the output of the previous layer, and the output of the last layer is the output of the feedforward neural network. The output of the feedforward neural network is independent of the inputs previously fed into the feedforward neural network since there is no internal memory.

Recurrent neural networks: In a recurrent neural network, the intermediate values are stored internally and will be used along with the later input of the recurrent neural network. The output of the recurrent neural network is computed based on the current input of the recurrent neural network and the intermediate values stored in the internal memory. Thus, the recurrent neural network can learn long-term dependencies in the input.

1.2.3. CONVOLUTIONAL NEURAL NETWORKS

In this section, we briefly give an overview of CNN applications, neural network architectures, and a description of various layers for CNNs.

Applications: CNNs have been explored for various application domains such as natural language processing [12], health care [13], time series prediction [14], and visual recognition [1]. In particular, CNNs are most commonly employed in computer vision systems and solve problems for a broad array of visual tasks [15], such as image classification [1], object detection [16], image segmentation [17], and face recognition [10]. Rather than using hand-crafted features, CNNs are trained in an end-to-end manner to generate high-level abstraction of the training data. This thesis focuses on tasks related to visual recognition, specifically image classification.

Neural network architectures: The main aspects of neural network architectures cover depth, width, groups, attention mechanism, and neural architecture search. Thus, various neural network architectures for CNNs have been designed to explore the influence of these aspects both on accuracy as well as efficiency. LeNet-5 [18], AlexNet [19], ZFNet [20], GoogLeNet [21], NiN [22], VGGNet [23], ResNet [1], DenseNet [24], SENet [25], and NASNet [26] are mainly designed for improving accuracy. ResNeXt [27], SqueezeNet [28], ShuffleNet [29], MobileNet [8], MnasNet [30], and GhostNet [31] are mainly proposed for improving efficiency. The neural network architecture of AlexNet is shown in Figure 1.2.

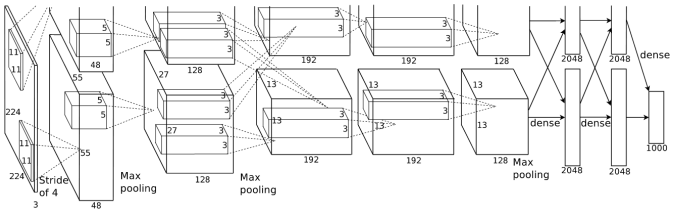


Figure 1.2: The neural network architecture of AlexNet. AlexNet is one of the most influential neural network architectures. It won the 1st place on the ImageNet challenge in 2012. This figure is from [19].

A description of CNN layers: A CNN is a stacking of various layers, including the convolutional layer, normalization layer, activation layer, and pooling layer.

- The convolutional layer is used to extract the features from the input data. The two main characteristics of the convolutional layer are local connections and weight sharing [18]. Each neuron of an output feature map has a receptive field and is connected to a neighborhood of neurons of the input feature maps by a set of weights. Also, different neurons of each output feature map use a set of weights.
- The normalization layer is used to speed up convergence during training and improve accuracy. The normalization layer adjusts the distribution of the training data across layers by the parameters of mean and standard deviation. There are local response normalization [19], batch normalization [32], instance normalization, group normalization, and layer normalization [33] for the normalization layer. Among them, batch normalization has become a standard choice in CNNs.
- The activation layer is used to introduce nonlinearity for CNNs. There are a number of functions used for the activation layer, such as sigmoid, hyperbolic tangent, and Rectified Linear Unit (ReLU) [34]. Currently, the ReLU is popular in CNNs since it can speed up the convergence during training.
- The pooling layer is used to reduce the spatial dimension and enable the spatial invariance to translation or distortion [18]. The popular pooling layers mainly include the max pooling layer and the average pooling layer, where the maximum value and average value within a receptive field propagate to the next layer, respectively.

1.2.4. ALGORITHMS FOR EFFICIENT INFERENCE

As shown in Figure 1.3, algorithms for efficient inference in DNNs mainly include acceleration [35–38], pruning [39–45], quantization [46–49], knowledge distillation [50–56], and efficient neural network architecture design [1, 21, 24, 27, 28, 57, 58].

Acceleration: Acceleration[2] means reducing the number of operations used for inference by using fast algorithms [35–38]. Fast algorithms for convolution mainly include Fast Fourier Transform (FFT) [35, 36], Strassen algorithm [37], and Winograd Transform [38]. Acceleration does not need to modify or fine-tune DNNs, which will maintain the accuracy of the network. However, given a fast algorithm for acceleration, the speedup is bounded by a theoretical maximum value and is closely related to the neural network architecture. For example, the speedup is larger when FFT is used to accelerate a convolution with a larger kernel size. [36] adopts FFT to accelerate a convolution with a kernel size of 3×3, where the maximum speedup is 1.84 than the standard algorithms used in cuDNN [59] even with the largest problem size.

Pruning: Pruning is a process to remove redundant connections or neurons, which does not contribute significantly to accuracy. Pruning can reduce both the number of parameters and the number of FLOPs required to compute DNNs. Criteria used to prune specific connections or neurons include the second derivative of the loss function (Hessian matrix) [39, 40], the magnitude of the weights [60], the magnitude of the activations [61, 62], and

[2] In this thesis below, we clarify that acceleration is achieved by algorithms rather than hardware.

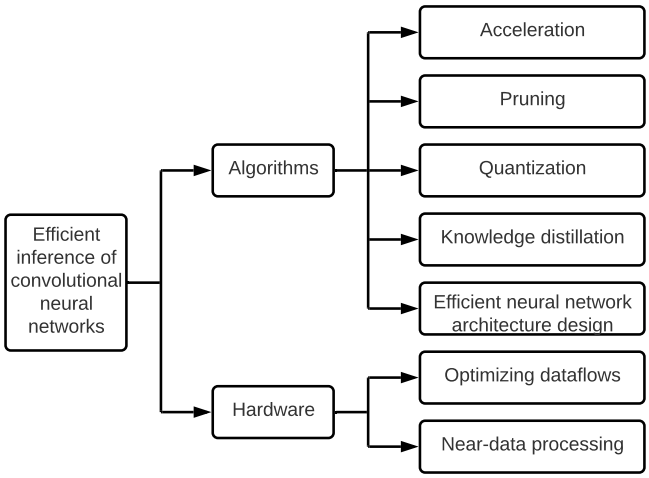


Figure 1.3: Algorithms and hardware for efficient inference in CNNs.

the LASSO penalty term [41–43]. Element-wise pruning, called unstructured pruning [60], has the smallest granularity of sparsity and is difficult to accelerate without specialized hardware support. Structured pruning [41, 43–45] can remove a group of connections or neurons to enable an efficient implementation. It is reasonable to expect that a higher pruning ratio will result in a larger accuracy degradation. However, it is worth exploring to limit the accuracy drop introduced by a high pruning ratio.

Quantization: Quantization means reducing the bit-width to represent the weights and activations. In the forward propagation, the quantization error is the distance between full-precision numbers and discrete values. There are mainly uniform quantization [46, 47] and non-uniform quantization [48, 49] to minimize the quantization error. In uniform quantization [46, 47], the full-precision numbers are quantized with a constant quantization step. In non-uniform quantization [48, 49], the quantization intervals and the parameters are jointly trained to minimize the quantization error. In the backward propagation, the gradient mismatch problem arises because of the non-differentiable quantizer. Different gradient approximations of the non-differentiable quantizer [63–65] are used to alleviate the gradient mismatch problem. Typically, there is a larger accuracy gap between the full-precision model and the quantized model with a shorter bit-width representation. Binary neural networks [66, 67] use binary values to replace full-precision counterparts, which leads to the largest accuracy drop among quantizations with different bit-widths. Thus, improving accuracy is needed for the deployment of binary neural networks in practical situations.

Knowledge distillation: Knowledge distillation transfers the knowledge from a teacher model to a student model. The accuracy of the student model improves without additional parameters or FLOPs needed to compute inference. The knowledge extracted from the

teacher model includes the output of the last layer of the teacher model [50, 51], the output of the intermediate layers [52–54], the relationship between feature maps of different layers [55, 56], and the relationship between different data samples [68, 69]. In offline distillation [50], the knowledge is transferred from a pre-trained teacher model to a student model. When the pre-trained teacher model is not available, online distillation [70, 71], such as self-distillation [72], can have both the teacher model and the student model updated simultaneously. Various distillation algorithms are used to improve the process of transferring knowledge, including adversarial distillation [73], multi-teacher distillation [74], cross-modal distillation [75], and graph-based distillation [76]. However, improving knowledge quality, distillation algorithm, and teacher and student models' architecture still deserves efforts.

Efficient neural network architecture design: Current network engineering considers both the accuracy and efficiency of neural networks. Given an accuracy constraint, designing efficient neural network architecture means reducing the number of parameters and the number of FLOPs. Feature reuse [1, 24] can not only alleviate the gradient vanishing problem [77] but also improve efficiency. In GoogLeNet [21] and SqueezeNet [28], convolutions with kernel size 1×1 are used to replace convolutions with kernel size 3×3. In ResNeXt [27], group convolution with a new cardinality is more effective than going deeper or wider. In Inception series [57], IGCV series [78], MobileNet series [79], ShuffleNet series [58], and Espnet series [80], depthwise separable convolution has been widely used to reduce computational complexity while increasing the representational efficiency. The attention mechanism is described as the allocation of limited cognitive resources [81] and adapted to develop attentional modules in [25, 82]. Neural architecture search can automatically discover efficient neural network architecture from a large search space, where the performance of FBNet [83] and MnasNet [30] surpasses the human-designed network. However, a better neural network architecture is still required to mitigate the contradiction between accuracy and efficiency.

1.2.5. HARDWARE FOR EFFICIENT INFERENCE

We will focus on algorithms rather than hardware for efficient inference in this thesis below. Thus, we briefly discuss the recent works about memory access when performing DNNs on hardware accelerators.

Optimizing dataflows: Given several levels of a memory hierarchy, optimized dataflows maximize data reuse from the low-cost memory hierarchy. In weight stationary dataflows [84, 85], the weights are reused as many times as possible while they are presented in the Register File (RF) at each Processing Element (PE). In output stationary dataflows [86, 87], the accumulation of the partial sums for each output activation value is processed in the RF at each PE. In no local reuse dataflows [88, 89], the input activations and the weights are read from the global buffer, and the partial sums are written back to the global buffer since there is no RF allocated in each PE. In row stationary dataflows [90, 91], the reuse of the weights, input activations, and partial sums are maximized in the RF at each

PE. However, we need to optimize the dataflows continuously with the evolution of neural network architectures.

Near-data processing: The goal is to move computing and data closer to each other in order to reduce data movement. To bring memory near the computation, embedded DRAM [89] integrates DRAM into the chip and 3-D memory [92] stacks DRAM on top of the chip. To bring the computation into the memory, the multiply and accumulate operation can be integrated into SRAM [93] and memristors [94, 95]. The multiplication and accumulation operations are completed in the analog domain [96, 97], which enables the computation close to the sensors. However, analog processing increases sensitivity to circuit and device nonidealities and introduces additional costs for analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC).

1.3. RESEARCH QUESTIONS

Many challenges have been discussed in the preceding section and need to be addressed. The thesis focuses on algorithms for efficient inference in CNNs. The research questions that we will shed light on are summarized as follows.

How to accelerate integer CNNs? This question belongs to acceleration and quantization. Current works have accelerated full-precision CNNs and have quantized full-precision CNNs. However, there are little efforts towards acceleration algorithms for quantized CNNs, specifically integer CNNs. The acceleration algorithm for integer CNNs is required to introduce zero round-off error since there is already an accuracy drop because of quantization.

How to improve multiple binary CNNs? This question belongs to quantization. Multiple binary CNNs achieve a better trade-off between accuracy and efficiency than single binary CNNs and fixed-point CNNs. However, the accuracy gap between full-precision CNNs and multiple binary CNNs is still considerable and needs to be reduced by a better approximation scheme. Regarding approximation, the distribution between the approximated values and the full-precision values is required to be similar, which takes full advantage of the pre-trained full-precision model.

How to design neural network architectures automatically for binary CNNs? This question belongs to quantization and efficient neural network architecture design. Among all the quantization methods with different bit-widths, the binarization error in the forward propagation is the largest, and the gradient mismatch problem is the severest in the backward propagation. Thus, the neural network architectures for full-precision models need to be re-designed after binarization. The search space of the automated neural network architectures is required to be carefully defined.

How to design neural network architectures manually for binary CNNs? This question belongs to quantization and efficient neural network architecture design. It is rea-

sonable to improve gradient backpropagation when the training difficulties increase. Current human-designed neural network architectures rely on enhancing the shortcut effective depth reduction techniques, i.e., increasing the number of shortcuts. However, increasing the number of shortcuts for Bi-Real ResNet and BinaryDenseNet will not improve the accuracy. Thus, developing better neural network architectures for binary CNNs requires thoroughly utilizing effective depth reduction techniques.

How to reduce feature reuse within the convolution for full-precision CNNs? This question belongs to efficient neural network architecture design. Current neural network architectures encourage feature reuse between convolutional layers and within the convolution. Feature reuse can help gradient backpropagation and improve efficiency. However, feature reuse within the convolution will result in the approximation problem of the channels. Thus, it is challenging to reduce feature reuse within the convolution to address the approximation problem.

How to improve the attention module for full-precision CNNs? This question belongs to efficient neural network architecture design. Attention-based models have achieved noticeable accuracy improvement without introducing significant computational complexity. In current attentional activations-based models, we identify the approximation and the insufficient capacity problem of the attention maps. Thus, an improved attention module is required to develop to address the above problem.

How to improve the reconstruction method for data-free compression? This question belongs to quantization, knowledge distillation, and efficient neural network architecture design. The reconstruction method addresses the new problem that the original training dataset is unavailable because of storage or privacy issues. However, the current reconstruction method is limited to extract prior information from the pre-trained model. Designing neural network architectures for the reconstruction method is worth exploring and is orthogonal to the current reconstruction methods.

1.4. CONTRIBUTIONS

The contributions of this thesis are directly associated with the research questions presented in the preceding section.

Propose Diminished-1 Fermat Number Transform to accelerate integer CNNs in Chapter 2: Integer data types are not only widely used in the research community [47, 98], but also supported by the recent commercial platforms [99, 100]. To accelerate the computation of integer convolution, we propose the Diminished-1 Fermat Number Transform (DFNT) after defining arithmetic operations using diminished-1. The proposed acceleration algorithm computes convolution as the diminished-1 point-wise products between DFNT transformed feature maps, which are reused multiple times. Besides, diminished-1 representation enables the computation of arithmetic operations with b bits for integers of $b+1$ bits. Thus, the proposed algorithm needs fewer multiplications and additions to

compute than the direct method. Experiments of integer convolution show that the proposed algorithm achieves faster speed and better parallelism than the direct method without introducing any round-off error.

Develop the piecewise approximation scheme for multiple binary CNNs in Chapter 3: Multiple binary CNNs achieve better accuracy than single binary CNNs and require less cost than fixed-point CNNs. Thus, we propose a piecewise approximation scheme to improve the performance of multiple binary CNNs. The scheme segments the whole range of weights and activations into many pieces and uses a scaling coefficient to approximate each piece. Using the scheme, the inference architecture of a full-precision convolution is implemented as a group of parallel bitwise convolutions. The distribution of the approximated values is similar to that of the pre-trained full-precision values, which indicates that the proposed scheme takes advantage of the pre-trained full-precision CNNs. Various experiments and computational complexity are analyzed for the proposed scheme and current quantization methods.

Develop the NASB strategy (i.e., automated neural network architectures) for binary CNNs in Chapter 4: It is challenging to figure an optimized neural network architecture for binary CNNs because of the large binarization error in the forward propagation and severe gradient mismatch problem in the backward propagation. Thus, we propose the NASB strategy, which leverages the neural architecture search (NAS) technique to design neural network architectures for binary CNNs. The NASB strategy consists of connections of a NASB-convolutional cell, operations of a NASB-convolutional cell, and a three-stage training algorithm. The connections of the NASB strategy are combined with a human-designed backbone and typical NAS-convolutional cells. The optimized neural network architectures from the NASB strategy are suitable for binarization, which achieves a better trade-off between accuracy and efficiency than human-designed binary CNNs.

Develop unified effective depth reduction techniques (i.e., human-designed neural network architectures) for binary CNNs in Chapter 5: Effective Depth Reduction (EDR) techniques for full-precision CNNs have been widely applied in network engineering to improve gradient backpropagation. When training difficulties increase, enhancing one effective depth reduction technique is proposed. In particular, enhancing the shortcut EDR technique has been used for binary CNNs. We identify the limitation of relying solely on enhancing the shortcut EDR technique since the accuracy of Bi-Real ResNet and BinaryDenseNet will not improve with the increase of the number of shortcuts. From a new perspective, we unified multiple EDR techniques into one model and develop unified models to improve gradient propagation. We unify the shortcut and fractal architecture EDR techniques and develop unified models for binary ResNet and DenseNet. Gradient analysis shows that the gradient paths of our unified models are better than that of Bi-Real ResNet and BinaryDenseNet. Experimental results demonstrate that both the shortcut and the fractal architecture EDR technique are necessary for our unified models.

Develop Reducing Approximation of channels by Reducing Feature reuse (REAF) scheme for full-precision CNNs in Chapter 6: Feature reuse is advantageous to effi-

ciency and gradient backpropagation. However, feature reuse will lead to an approximation problem of the channels within the convolution. In particular, the input feature maps to compute every channel of the output feature maps of the convolution are the same in a standard convolution. We propose the REAF scheme, where the input feature maps to compute every channel of the output feature maps of the convolution are customized and specialized. The proposed scheme with different configurations shows better accuracy than the standard convolution. Besides, we develop the REAF+ scheme, which adopts parameterized operations to enable the specialized input feature maps to compute every channel of the output feature maps of the convolution. The proposed scheme can have more merged channels than group convolution, which reduces feature reuse within the convolution. Finally, we analyze the class selectivity for the features that the proposed scheme and group convolution reduce the feature reuse compared with the standard convolution.

Develop an attention module for full-precision CNNs in Chapter 7: We analyze the problems of the current attentional activations-based models, including the approximation problem and insufficient capacity problem of the attention maps. To address the above problems, we propose an attention module, which mainly includes the Attentional Weight (AW)-convolution. In the AW-convolution, the shape of attention maps matches that of the weights rather than the activations. To reduce the computational cost, we refine the architecture of calculating the attention maps in the proposed attention module. Integrated with the proposed attention module, the accuracy of the models without the attention mechanism increases. More importantly, the accuracy of current attention-based models improves further when integrating with the proposed attention module. The proposed attention module is a complementary method to current attention-based models since it is designed to solve the problems of current attention-based models.

Propose the AutoReCon reconstruction method for data-free compression in Chapter 8: Computing the reconstructed training dataset is an essential step when the original training dataset is not available due to transmission or privacy concerns. From a new point of view, we consider network engineering of the reconstruction. Regarding the definition of the reconstruction method, we compute the reconstruction loss given the pre-trained model and the prior information exploited from the pre-trained model. Then, we propose the AutoReCon method, which applies neural architecture search for the current reconstruction method. The AutoReCon method includes a stochastic super net-based search space and a gradient-based search algorithm. Using the AutoReCon method for the data-free compression, there is an additional searching stage to discover an optimized generator. The accuracy of current data-free quantization and knowledge distillation will improve with the optimized generator because of the superiority of the optimized generator over the human-designed generator.

1.5. THESIS ORGANIZATION

The remainder of this thesis is organized as follows.

Chapter 2 describes an acceleration algorithm for integer CNNs. It presents the proposed Diminished-1 Fermat Number Transform to accelerate integer CNNs. Moreover, the ac-

curacy, speed, scalability of the Diminished-1 Fermat Number is compared with the direct baseline method.

Chapter 3 covers an approximation algorithm for multiple binary CNNs. It elaborates how the proposed piecewise approximation scheme works in forward and backward propagation. Also, it discusses the extensive experimental results and analyzes thoroughly efficiency.

Chapter 4 discusses an automated neural network architecture algorithm for binary CNNs. It describes the NASB strategy, including connections of a NASB-convolutional cell, the operations of a NASB-convolutional cell, and a three-stage training algorithm. Moreover, it shows diverse experiments to demonstrate the effectiveness of the NASB strategy.

Chapter 5 presents unified effective depth reduction techniques for binary CNNs. It describes the unified architectures, which unifies the shortcut and fractal architecture effective depth reduction techniques. It also analyzes the gradient path, computational complexity, and experimental results.

Chapter 6 is concerned with reducing feature reuse within the convolution for full-precision CNNs. It presents the proposed REAF and REAF+ schemes, which reduce feature reuse within the convolution to reduce the approximation of channels. Finally, it presents the comparisons with baseline models, which are composed of standard convolution and group convolution.

Chapter 7 introduces an attention module for full-precision CNNs. First, it describes the approximation problem and the insufficient capacity problem of the attention maps. Then, it discusses the proposed attention module, which mainly includes the AW-convolution. Finally, it presents experiments to demonstrate the benefit of integrating the proposed attention module.

Chapter 8 shows an automatic reconstruction method for data-free compression of CNNs. It describes the AutoReCon method, including the search space, search algorithm, and training process. In addition, it presents experiments of data-free compression to show the superiority of the proposed reconstruction method.

Chapter 9 concludes this thesis.

2

Acceleration with Diminished-1 Fermat Number Transform

Convolutional Neural Networks (CNNs) are a class of widely used deep artificial neural networks. However, training large CNNs to produce state-of-the-art results can take a long time. In addition, we need to reduce compute time of the inference stage for trained networks to make them accessible for real-time applications. To achieve this, integer number formats INT8 and INT16 with reduced precision are being used to create Integer Convolutional Neural Networks (ICNNs) to allow them to be deployed on mobile devices or embedded systems. In this chapter, Diminished-1 Fermat Number Transform (DFNT), which refers to Fermat Number Transform (FNT) with diminished-1 number representation, is proposed to accelerate ICNNs through algebraic properties of integer convolution. This is achieved by performing the convolution step as diminished-1 point-wise products between DFNT transformed feature maps, which can be reused multiple times in the calculation. DFNT adapts b bits to represent and compute all the integers in the ring of integers modulo Fermat number 2^b+1, which is more efficient than FNT, which uses $b+1$ bits to represent and perform arithmetic operations. In other words, DFNT computes arithmetic operations more exactly than FNT if the hardware platform supports arithmetic operations with a maximum of b bits. Using DFNT, integer convolution is implemented on a general-purpose processor, showing speedup of 2-3x with typical parameter configurations and better scalability without any round-off error compared to the baseline.

The content of this chapter is based on [1].

2.1. INTRODUCTION

Convolutional Neural Networks (CNNs) have emerged as one of the most influential neural network architectures to tackle large scale machine learning problems in image recognition, natural language processing, and audio analysis [2] [3]. Due to the high complexity of CNNs, orders of magnitude more data is required to take advantage of these powerful neural networks. For the training of CNNs on large datasets (such as ImageNet), several weeks of training time is usually needed, even using parallel computing environments. For the inference phase of trained CNNs, real-world applications with real-time requirements or for application in the embedded domain, requires very short inference time as well as using a low energy budget. Therefore, there is ongoing effort from the research community as well as industry to accelerate the training and inference of CNNs [4] [5].

To be deployed on embedded systems or mobile devices, CNNs with discrete parameters, i.e. Integer Convolutional Neural Networks (ICNNs), have been an active and promising research topic. Generally, there is better computing support for fixed-point as compared to floating-point operations. Currently, especially INT8 and INT16 are the most widely used data types for weight and input in the quantization of CNNs, and their feasibility has been demonstrated by many researchers [6] [7]. The benefits of these ICNNs include storage cost reduction and computation requirement reduction because of shorter bit-width of weight compared to single precision floating-point.

In this chapter we present a novel approach to speed up the training and inference of ICNNs using Diminished-1 Fermat Number Transform (DFNT), i.e, Fermat Number Transform (FNT) using diminished-1 number representation. This helps to reduce the complexity of integer convolutions through its algebraic properties. The innovative idea is based on performing the convolution as diminished-1 point-wise products between DFNT transformed feature maps, which can be reused multiple times. All input and output feature maps, and gradients of the loss with respect to these feature maps and weight kernels of ICNNs can be viewed as 2-D matrices. Therefore, most operations can be represented by convolutions between pairs of 2-D matrices. Since the convolutions are performed for all pairings between two sets of 2-D matrices, computing DFNT of all the matrices only once can complete all the convolutions as diminished-1 pairwise products. In addition, diminished-1 number representation is used for FNT to perform binary arithmetic with b bits for all the integers of $b+1$ bits in the ring of integers modulo Fermat number 2^b+1, which enables exact and efficient calculation.

The contributions of this chapter are as follows.

- This is the first work that optimizes the computation of ICNNs using the algebraic properties of integer convolution without introducing any round-off error.
- The chapter presents DFNT to speed up the training and inference of ICNNs, where diminished-1 number representation is used for FNT to enable exact and efficient calculation.
- The chapter shows a sustained speedup of 2-3x without any round-off error in computing the integer convolution with typical parameter configurations.

The rest of the chapter is organized as follows. Section 2.2 discusses related work mainly for the acceleration of ICNNs. Section 2.3 reviews the theory of CNNs. Then, Section 2.4

presents DFNT, including FNT and arithmetic operations using diminished-1 number representation. Compared with the direct method, arithmetic complexity is analyzed in Section 2.5 and experimental results are demonstrated in Section 2.6. Finally, we conclude this chapter in Section 2.7.

2.2. RELATED WORK

Fast Fourier Transform (FFT) was first used to reduce the computation complexity of CNNs by Mathieu et al. [2], refined by Vasilache et al. [3], and subsequently implemented in the NVIDIA cuDNN library [8]. However, this approach was applied only for floating-point calculations. The Strassen and Winograd algorithms for fast matrix multiplication were used by Cong et al. [4] and Lavin et al. [5] to reduce the number of multiplications in convolution. [9] has explored the feasibility of using FNT to accelerate CNNs. However, FNT requires $b+1$ bits to represent and perform all the integers in the ring of integers modulo Fermat number 2^b+1 since the representation of the quantity $2^b = -1 \bmod F_t$ requires the (b+1)th bit. Our proposal of DFNT adapts b bits to represent and compute all the integers in the ring of integers modulo Fermat number 2^b+1, which is more efficient than FNT, which uses $b+1$ bits to represent and perform arithmetic operations. These publications exploited the algebraic structure of convolution without influencing accuracy. Other approaches aim to reduce the complexity of the computation by computing convolution approximately [10], performing training with separable filters [11].

ICNNs are widely used in the training, fine-tuning and inference of CNNs. Shuchang et al. [6] used dynamic fixed-point format of INT8 in their proposed method DOREFA-NET to train and fine-tune CNNs. Philipp et al. [12] presented the Ristretto approximation framework, which can successfully condense CaffeNet and SqueezeNet to INT8. Benoit et al. [13] proposed a quantization scheme with INT8, which allows inference to be carried out using integer-only arithmetic and is adopted in Tensorflow Lite. Shuang et al. [14] developed a new method termed as "WAGE" to discretize both training and inference processes, where activations, gradients and errors among layers are shifted and linearly constrained to INT8. Suyog et al. [7] used only INT16 with stochastic rounding to train deep neural networks, and the demonstration on FPGA incurred little to no accuracy degradation. Dipankar et al. [15] used INT16 in dynamic fixed point data format to train state-of-the-art CNNs for ImageNet-1K dataset on general processor and achieved the highest reported accuracy using half precision representation.

Data types INT8 and INT16 are not only widely explored by the research community, but also have been supported in recent commercial platforms for deep neural network (DNN) processing. The Google's Tensor Processing Unit [16] announced in May 2016 was designed for 8-bit integer arithmetic. The Nvidia's PASCAL GPU [17] announced in April 2016 has 8-bit integer instructions for deep learning inference. Vector Neural Network Instructions Word variable precision (4VNNIW) is a set of vector instructions for INT16 newly introduced by Intel Xeon Phi Knights-Mill [18], which was announced in 2017. These use the AVX-512 SIMD instructions that are 512-bit extensions to 256-bit advanced vector extension instructions for x86 instruction set architecture. AVX-512 Vector Length Extensions is a set of vector instructions for INT8 supported by Skylake-SP and Skylake-X processors [18], which were announced in 2017. These commercial approaches can accel-

erate ICNNs through customizing hardware architecture design, which are considered as orthogonal and complementary to those that exploit algebraic structures.

Despite the fact that much research has been done to utilize INT8 and INT16 to reduce the complexity of CNNs, none of these effort have investigated the possibility to optimize the computation time of integer convolution using their algebraic properties. This chapter is the first to investigate possible speedup of the training as well as the inference of CNNs using algebraic transformations without introducing any round-off error.

2.3. CONVOLUTIONAL NEURAL NETWORKS

For a given convolution layer, notations are fixed below. There are a set of input feature maps x_f indexed by f, each one being a 2-D image of dimensions $n \times n$. The output of CNNs is recognized as a set of feature maps $y_{f'}$ indexed by f', each of which is also a 2-D image whose dimensions $n' \times n'$ depend on weight kernel and its stride. The trainable parameters of the convolution layer are a set of weight kernels $w_{f'f}$, each of which is a 2-D kernel of dimensions $k \times k$. Each input and output feature map is part of a batch size s.

In the forward pass, each output feature map is computed as a sum of the f input feature maps correlating with the f trainable weight kernels correspondingly, where $\bullet$ refers to cross-correlation.

$$y_{f'} = \sum_{f} x_f \bullet w_{f'f} \tag{2.1}$$

During the backward pass, the gradients of the loss C with respect to the input is computed as a sum of the f' gradients of the loss with respect to the output convolving with the f' trainable weight kernels, where $*$ represents convolution.

$$\frac{\partial C}{\partial x_f} = \sum_{f'} \frac{\partial C}{\partial y_{f'}} * w_{f'f} \tag{2.2}$$

Based on the gradients of the loss with respect to the input computed from equation (2), the gradients of the loss with respect to the weight is computed as a sum of the s gradients of the loss with respect to the output correlating with the s input feature maps.

$$\frac{\partial C}{\partial w_{f'f}} = \sum_{s} \frac{\partial C}{\partial y_{f'}} \bullet x_f \tag{2.3}$$

2.4. DIMINISHED-1 FERMAT NUMBER TRANSFORM

To represent all the integers in the ring of integers modulo F_t requires $b+1$ bits, where F_t is the t^{th} Fermat number. $F_t = 2^b + 1$, and $b = 2^t$. In order to avoid performing binary arithmetic for the additional bit in the $b+1$ representation, diminished-1 number representation is used for FNT. In this section, DFNT is illustrated as an exact and efficient approach, including FNT and the arithmetic operations with diminished-1 number representation to perform FNT.

Algorithm 1 Arithmetic Operations in Diminished-1

Function addition_d-1 (X, Y)
  if $X == 2^b$ **then**
    $Z = Y$
  else if $Y == 2^b$ **then**
    $Z = X$
  else
    $Z = X_b + Y_b + C_r$
  end if
  return Z
End Function
Function d-1_to_nb (X)
  if $X == 2^b$ **then**
    $x = 0$
  else
    x = addition_d-1 (X, 1)
  end if
  return x
End Function
Function nb_to_d-1 (x)
  X = addition_d-1 (x, 2^b - 1)
  return X
End Function
Function negation_d-1 (X)
  if $X == 2^b$ **then**
    $Z = 2^b$
  else
    $Z = 2^b - X_b$
  end if
  return Z
End Function
Function left shift_d-1 (X)
  if $X == 2^b$ **then**
    $Z = 2^b$
  else
    Z = left circular shift (X_b)
  end if
  return Z
End Function
Function multiplication_d-1 (X, Y)
  if $X == 2^b - 1$ **then**
    Z = negation_d-1 (Y)
  else if $Y==2^b - 1$ **then**
    X = negation_d-1 (X)
  else
    x = d-1_to_nb (X); y = d-1_to_nb (Y); $\{Z_h, Z_l\} = x_b \times y_b$; Z = subtraction_d-1 (Z_l, Z_h)
  end if
  return Z
End Function

2.4.1. FERMAT NUMBER TRANSFORM

FNT [19] is a kind of number theoretic transform, which is defined on a finite ring of integers carried out modulo Fermat numbers. FNT is exact, i.e. without round-off error. One dimensional FNT of sequence $x(l)$ with length L is defined as follows.

$$X(k) \equiv \sum_{l=0}^{L-1} (x(l) \times a^{lk}) \bmod F_t \tag{2.4}$$

where $k = 0, 1, \ldots, L-1$. a is the root of unity of order L, and generally selected as a power of 2 since it can be calculated through a shift operation. Especially when $a = 2$, the largest length of the transform will be $L = 2^{t+1}$. FNT is a FFT-like transform, which can be expressed using decimation in time structure and results in the following efficient implementation.

$$X(q) \equiv x_1(q) + x_2(q) \bmod F_t \tag{2.5}$$

$$X(q + L/2) \equiv x_1(q) - x_2(q) \bmod F_t \tag{2.6}$$

where $x_1(q)$ and $x_2(q)$ are calculated as follows.

$$x_1(q) = \sum_{p=0}^{L/2-1} (x(2p) \times 2^{2pq}) \tag{2.7}$$

$$x_2(q) = 2^q \sum_{p=0}^{L/2-1} (x(2p+1) \times 2^{2pq}) \tag{2.8}$$

The corresponding inverse FNT is expressed as below.

$$x(l) \equiv \frac{1}{L} \sum_{k=0}^{L-1} (X(k) \times a^{-lk}) \bmod F_t \tag{2.9}$$

Two dimensional FNT is used to process two dimensional feature maps of convolution layer, and defined as below. Two dimensional FNT is calculated based on one dimensional FNT. A one dimensional FNT is used to operate on each row of the feature map, and then it is used to operate on each column of the intermediate results, as follows.

$$X(p, k) \equiv \sum_{q=0}^{Q-1} \sum_{l=0}^{L-1} (x(q, l) \times a^{lk} \times a^{pq}) \bmod F_t \tag{2.10}$$

2.4.2. ARITHMETIC OPERATIONS USING DIMINISHED-1

To perform binary arithmetic operations with b bits for integers of $b+1$ bits, a modified binary number system, i.e. diminished-1 number representation [20], is adopted. In diminished-1 number representation, the numbers from 1 to 2^b are represented in order by the normal binary numbers from 0 to $2^b - 1$.

Using diminished-1 representation, pseudo code of binary arithmetic operations modulo F_t is defined as Algorithm 1. x in normal binary (nb) number representation corresponds

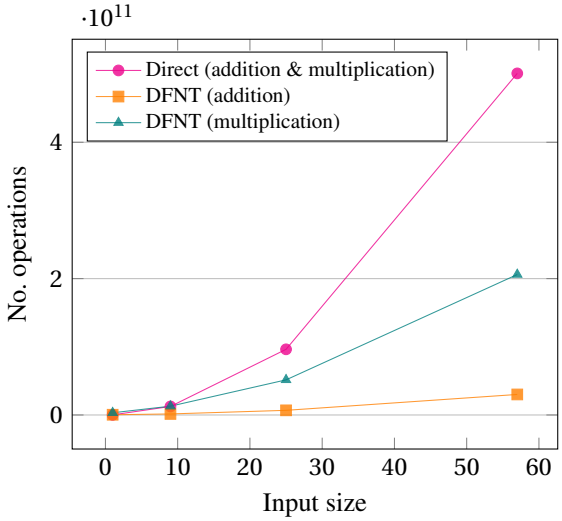


Figure 2.1: Number of operations of the forward pass for different input sizes with batch size = 128, input channel = 96, output channel = 256, kernel size = 7.

to X in diminished-1 (d-1) number representation. In d-1 number representation, X, Y and Z are of $b+1$ bits, and X_b and Y_b indicate the b Least Significant Bits (LSBs) of X and Y respectively. Z_h and Z_l are the b Most Significant Bits (MSBs) and the b LSBs, comprising result of $2b$-bit multiplication. In nb number representation, x_b and y_b are of b bits.

If operands are equal to special numbers, such as 2^b and 2^b-1, special results will be returned. Otherwise, b-bit operations in d-1 number representation will be performed. nb_to_d-1 is the translation from nb to d-1 number representation, and d-1_to_nb is the reverse process. If the operand X is not equal to 2^b, the complemented X_b is the result of its negation_d-1. If neither of the operands X and Y are equal to 2^b, adding the sum of X_b and Y_b with their complemented carry bit C_r gives the result of their addition_d-1. If the operand X is not equal to 2^b, the left circular shift of X_b with its complemented bits circulated into its LSBs performs its left shift_d-1. As for multiplication_d-1, if neither of the operands X and Y are equal to 2^b-1, they will be translated into nb number representation. Following a $2b$-bit binary multiplication, a subtraction_d-1 of the b MSBs of the product from the b LSBs is performed. Subtraction_d-1 is completed by addition_d-1 and negation_d-1, and right shift_d-1 can be calculated in a similar way of left shift_d-1.

2.5. ARITHMETIC COMPLEXITY ANALYSIS

In this chapter, we use "same" as the padding algorithm for the convolution, which means $n = n'$. The transform length $n+k-1$, which is not a power of 2, must be padded to the next highest power. This limitation is not intrinsic to DFNT and can be explored in the future. Since equations (1), (2) and (3) have exactly the same number of operations using either the direct method or DFNT, all complexity analysis and experiments in this chapter

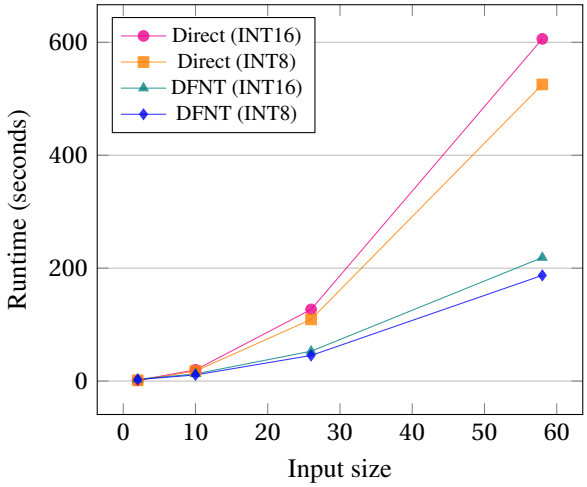


Figure 2.2: Time versus input size with batch size = 128, input channel = 96, output channel = 256, kernel size = 7

are illustrated only for equation (1).

Taking the forward pass, i.e, equation (1), with the input and weight of data type INT16 as an example, it accounts for a total of $s \times f' \times f \times n'^2 \times k^2$ 32-bit multiplications and 64-bit additions, as shown in Table 2.1.

DFNT has a fast transform structure similar to FFT, and all operations are performed in 64-bit data path. As shown in Table 2.1, it requires $2(n+k-1)^2 \times \log_2(n+k-1) \times (s \times f + f \times f')$ addition_d-1 and left shift_d-1 operations for the transform of the input feature maps and weight kernels, $2(n+k-1)^2 \times \log_2(n+k-1) \times (s \times f')$ addition_d-1 and left shift_d-1 operations for the inverse transform of the output feature maps, and $s \times f' \times f \times (n+k-1)^2$ multiplication_d-1 and addition_d-1 operations for the pointwise products. The translation between nb and d-1 number representation requires $n^2 \times (s \times f)$ nb_to_d-1 operations for the input, $k^2 \times (f \times f')$ nb_to_d-1 operations for the weight, and $n'^2 \times (s \times f')$ d-1_to_nb operations for the output.

Since $k < n$, this implies that $n+k-1 < 2n$. As a result (see Table 2.1), the total complexity of convolution of the direct method comes from the product of five terms, whereas DFNT has a sum of products with at most four terms.

Integer convolution is indeed composed of fixed-point multiply-and-accumulate operations, which means that the precision of the internal values should be higher than that of the weight and the input. To guarantee precision, the input and weight of t bits require t-bit $\times$ t-bit multiplication to generate $2t$ bits product, and the subsequent accumulation needs $2t+m$-bit addition, where m is determined based on the largest size of weight kernels. $m \geq \log_2(f \times k \times k)$, and it is typically in the range of 10 to 16 bits in modern popular ICNNs. To avoid overflow in convolution, for the input and weight of data type INT8, integer convolution refers to INT32+ = INT8 $\times$ INT8, and it requires INT64+ = INT16 $\times$ INT16 for data type INT16.

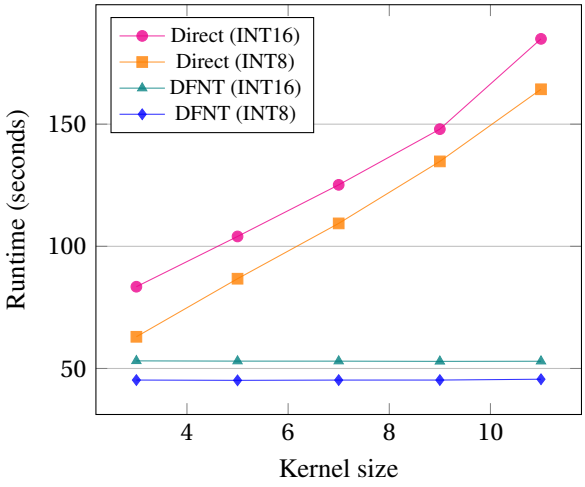


Figure 2.3: Time versus kernel size with batch size = 128, input channel = 96, output channel = 256, transform size = 32

Table 2.1: Complexity of the direct method and DFNT

Items	Number of operations
Direct (32-bit multiplication)	$s \times f' \times f \times n'^2 \times k^2$
Direct (64-bit addition)	$s \times f' \times f \times n'^2 \times k^2$
DFNT (d-1_to_nb)	$n'^2 \times (s \times f')$
DFNT (nb_to_d-1)	$n^2 \times (s \times f) + k^2 \times (f \times f')$
DFNT (multiplication_d-1)	$s \times f' \times f \times (n+k-1)^2$
DFNT (shift_d-1)	$2(n+k-1)^2 \times \log_2(n+k-1) \times (s \times f + f \times f' + s \times f')$
DFNT (addition_d-1)	$2(n+k-1)^2 \times \log_2(n+k-1) \times (s \times f + f \times f' + s \times f') + s \times f' \times f \times (n+k-1)^2$

Assume that four 32-bit multiplications are equal to one 64-bit multiplication, and two 32-bit additions are equal to one 64-bit addition. Fig. 2.1 shows the theoretical numbers of operations using the direct method and DFNT for various input sizes, where their complexities are unified as 64-bit addition and 32-bit multiplication.

2.6. EXPERIMENTAL RESULTS

The implementations were tested on a high-end machine consisting of 2.4 GHz Intel Xeon processors with a total of 28 two-way hyper-threaded cores and 192 GB RAM. Using the direct method, integer convolution is implemented as baseline to measure accuracy loss, complexity reduction and scalability of DFNT. For both the direct method and DFNT, the accuracy and speed of integer convolution with typical parameter configurations were measured for the weight and input of data types INT8 and INT16.

For the weight and input of data type INT8, 16-bit multiplication and 32-bit addition are

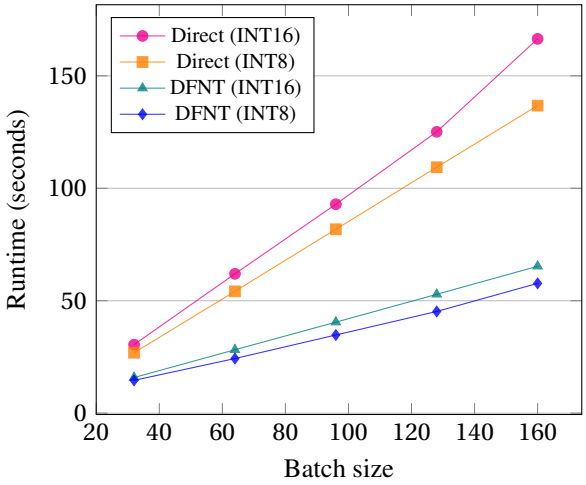


Figure 2.4: Time versus batch size with kernel size = 7, input channel = 96, output channel = 256, transform size = 32

Table 2.2: Accuracy comparisons of typical parameter configurations between the direct method and DFNT

$(k, n+k-1, f, f')$	(11, 32, 3, 96)	(7, 32, 96, 256)	(5, 16, 256, 384)	(5, 16, 384, 384)	(3, 16, 384, 384)
Mismatch/match (INT16)	0/100	0/100	0/100	0/100	0/100
Mismatch/match (INT8)	0/100	0/100	0/100	0/100	0/100

used to guarantee precision. Similarly, 32-bit multiplication and 64-bit addition are required for the weight and input of data type INT16. Using DFNT, integers in the ring of module F_5 and F_6 are used to calculate convolution for the weight and input of data types INT8 and INT16, respectively. In addition, diminished-1 arithmetic operations are performed in 32-bit and 64-bit data paths for the integers in the ring of module F_5 and F_6, respectively.

In terms of accuracy, DFNT has exactly the same precision as the direct method, since there is no round-off error for DFNT. We ran accuracy experiments with typical parameter configurations of convolution in ICNNs. In Table 2.2, tuple $(k, n+k-1, f, f')$ refers to the integer convolution layer with kernel size k, transform size $n+k-1$, input channel f and output channel f', and batch size is 128 for all configurations. For each of these typical parameter configurations, the input and weight were generated randomly, and all the numbers of the four dimensional output of the direct method and DFNT were compared. All the numbers during this comparison are the same, which we will refer to as "match". Otherwise, we will refer to it as "mismatch". As shown in Table 2.2, there is 0 "mismatch" associated with a round-off error out of the 100 comparisons performed compared with the direct method.

Similar to the overflow constraint in fixed-point precision, the peak output using DFNT should be bounded according to Equation 2.11. Under this condition, implementing convolution in the ring of integers modulo F_t can get the same result as that obtained with normal

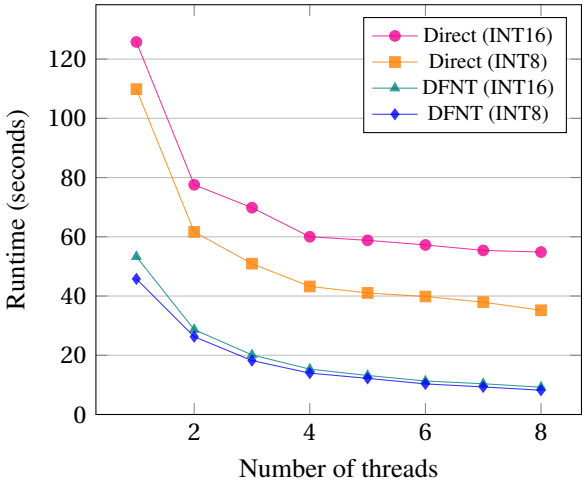


Figure 2.5: Time versus number of threads with batch size = 128, input channel = 96, output channel = 256, kernel size = 7, transform size = 32

Table 2.3: Runtime of typical parameter configurations and ICNNs

$(k, n+k-1, f, f')$	(11, 32, 3, 96)	(7, 32, 96, 256)	(5, 16, 256, 384)	(5, 16, 384, 384)	(3, 16, 384, 384)	Total
Direct (INT16)	2.48	126.40	83.45	130.03	97.52	488.21
DFNT (INT16)	2.73	52.73	47.82	70.49	70.58	274.55
Direct (INT8)	1.95	109.65	72.67	109.60	94.07	438.35
DFNT (INT8)	2.33	45.34	40.82	60.32	60.16	237.47

arithmetic.

$$\sum_{i=0}^{f-1}\sum_{j=0}^{k-1}\sum_{l=0}^{k-1} x_{[s,n,n,f]} \times w_{[k,k,f,f']} \leq F_t/2 \tag{2.11}$$

We limit the number of threads to 1 since we are measuring reduction of total workloads by runtime. The speed comparison for convolution between the direct method and DFNT with varying input sizes, kernel sizes, and batch sizes are demonstrated in Figs. 2.2, 2.3 and 2.4, respectively. As shown in these figures, with a larger input size, a larger kernel size and a larger batch size, a larger speedup of DFNT over the direct method can be expected for the weight and input of data types INT8 and INT16.

All the loops of the direct method and DFNT are parallelized using OpenMP, and their runtime with different number of threads is shown in Fig. 2.5. Compared with 1 thread, the speedup of DFNT with 4 threads is 3.50 and higher than that of the direct method at 2.10, which indicates a better scalability of DFNT over the direct method.

We measured execution time with typical parameter configurations of convolution in ICNNs, which are the same as in Table 2.2. As their runtime in seconds reported in Table 2.3 shows, DFNT performs faster than the direct method for all configurations. Using these typical parameter configurations of integer convolution layers, a complete network can be

obtained by inserting max-pooling and rectified linear unit layers and adding a fully connected layer for prediction with 1000 outputs. Its runtime is measured and reported in the last column, which accounts for the overall speedup of ICNNs.

2.7. CONCLUSION

We have introduced an DFNT method to accelerate integer convolution of ICNNs, which can be used for both the training and inference processes. Since integer convolution is performed as diminished-1 point-wise products between DFNT transformed feature maps, all the feature maps only need to be transformed once and reused multiple times. By exploiting diminished-1 number representation for FNT, the proposed DFNT can be calculated efficiently and exactly. Compared to the direct method, a faster speed and better parallelism without any round-off error have been shown according to the demonstration results on general processor.

3

PIECEWISE APPROXIMATION

Binary Convolutional Neural Networks (CNNs) can significantly reduce the number of arithmetic operations and the size of memory storage, which makes the deployment of CNNs on mobile or embedded systems more promising. However, the accuracy degradation of single and multiple binary CNNs is unacceptable for modern neural architectures and large scale datasets like ImageNet. In this chapter, we proposed a Piecewise Approximation (PA) scheme for multiple binary CNNs which lessens accuracy loss by approximating full precision weights and activations efficiently, and maintains parallelism of bitwise operations to guarantee efficiency. Unlike previous approaches, the proposed PA scheme segments piece-wisely the full precision weights and activations, and approximates each piece with a scaling coefficient. Our implementation on ResNet with different depths on ImageNet can reduce both Top-1 and Top-5 classification accuracy gap compared with full precision to approximately 1.0%. *Benefiting from the binarization of the downsampling layer, our proposed PA-ResNet50 requires less memory usage and two times fewer Flops than single binary CNNs with* 4 *weights and* 5 *activations bases. The PA scheme can also generalize to other architectures like DenseNet and MobileNet with similar approximation power as ResNet which is promising for other tasks using binary convolutions.*

The content of this chapter is based on [21].

3.1. INTRODUCTION

CNNs have emerged as one of the most influential neural network architectures to tackle large scale machine learning problems in image recognition, natural language processing, and audio analysis [22, 23]. At the same time, their deployment on mobile devices and embedded systems are gaining more and more attention due to the increasing interest from industry and academia [24, 25]. However, the limited storage and computation resources provided by these platforms are an obstacle that is being addressed by numerous researchers working to reduce the complexity of CNNs [5, 26–28]. Fixed-point CNNs [1, 6, 29–32] achieve even no accuracy loss with a suitable selection of bit-width, but the multiplication and the overflow processing of addition require considerable overhead. Binary CNNs have been demonstrated as a promising technique to make the deployment of CNNs feasible [33–36]. In single binary CNNs, full precision weights and activations are binarized into 1 bit, so the multiplication and addition of the convolution are transformed into simple bitwise operations, resulting in significant storage and computation requirements reduction [37]. The accuracy degradation of the recently enhanced single binary CNN [38] is still high (12.9% Top-1 and 9.7% Top-5 accuracy degradation for ResNet18 on ImageNet) since much information has been discarded during binarization. ABC-Net [39] is the first multiple binary CNN, which shows encouraging result (around 5% Top-1 and Top-5 accuracy degradation for ResNet on ImageNet). [40–43] calculate a series of binary values and their corresponding scaling coefficients through minimizing the residual error recursively, but they can not be paralleled. [44] propose Group-Net to explore structure approximation, and it is a complimentary approximation to value approximation. Multiple binary CNNs can be considered as a moderate way of quantization, that is much more accurate than single binary CNNs and more efficient than fix-point CNNs. But, there is still a considerable gap between full precision implementations and multiple binary CNNs, despite the fact that an unlimited number of weights and activation bases can be used.

To further reduce the gap between the full precision and multiple binary CNNs, we proposed Piece-wise Approximation (PA) scheme in this chapter. Our main contributions are summarized as follows.

- PA scheme segments the whole range of the full precision weights and activations into many pieces and uses a scaling coefficient to approximate each of them, which can maintain parallelism of bitwise operation and lessen accuracy loss.

- With less overhead, our scheme achieves much higher accuracy than ABC-Net, which indicates that it provides a better approximation for multiple binary CNNs. Benefited from the binarization of the downsampling layer, our proposed PAResNet50 requires less memory usage and two times Flops than Bi-Real Net with 4 weights and 5 activations bases, which shows its potential efficiency advantage over single binary CNNs with a deeper network.

- With the increase of the number of the weight and activation bases, our proposed PA scheme achieves the highest classification accuracy for ResNet on ImageNet among all state-of-the-art single and multiple binary CNNs.

3.2. RELATED WORK

In this Section, we describe the forward propagation and backpropagation of typical schemes to quantize CNNs. In addition, the advantages and disadvantages of these quantized CNNs are discussed concerning efficiency and accuracy.

3.2.1. SINGLE BINARY CONVOLUTIONAL NEURAL NETWORKS

In single binary convolutional neural networks [33–36, 45], weights and activations are constrained to a single value +1 or −1. The deterministic binarization function is described as follows.

$$x^b = \begin{cases} +1, x^r \geq 0 \\ -1, x^r < 0 \end{cases} \tag{3.1}$$

where x^b is the binarized variable, and x^r is the real-valued variable. During the backpropagation, the "Straight-Through Estimator" (STE) method [46] is adapted to calculate the derivatives of the binarization functions as follows, where C is the loss function.

$$\frac{\partial C}{\partial x^r} = \frac{\partial C}{\partial x^b} \tag{3.2}$$

Single binary CNNs is the most efficient quantization scheme among all the quantization schemes described in this chapter. But, its accuracy degradation is too high to be deployed in practice.

3.2.2. TERNARY CONVOLUTIONAL NEURAL NETWORKS

In ternary convolutional neural networks [47–50], ternary weights are used to reduce the accuracy loss of single binary CNNs by introducing 0 as the third quantized value, as follows.

$$x^t = \begin{cases} x^p : x^r > \Delta \\ 0 : |x^r| \leq \Delta \\ -x^n : x^r < -\Delta \end{cases} \tag{3.3}$$

where x^p and x^n are the positive and negative scaling coefficients, respectively, and Δ is a threshold to determine the ternarized variable x^t. During the backpropagation, the STE method is still applied.

Although the introduction of 0 improves the accuracy of single binary CNNs, it is still unacceptable to be deployed especially while training advanced CNNs on large scale dataset.

3.2.3. FIXED-POINT CONVOLUTIONAL NEURAL NETWORKS

In fixed-point convolutional neural network [6, 31, 32, 51], weights, activations, and gradients are quantized using fixed-point numbers of different bit-widths. Taking the weights as an example, the quantization works as follows.

$$x^f = 2 \cdot \texttt{quantize}_f\left(\frac{\tanh(x^r)}{2\max(|\tanh(x^r)|)} + \frac{1}{2}\right) - 1 \tag{3.4}$$

where $\texttt{quantize}_f$ function quantizes the real-valued number x^r to the f-bit fixed-point number x^f. During the backpropagation, the STE method still works.

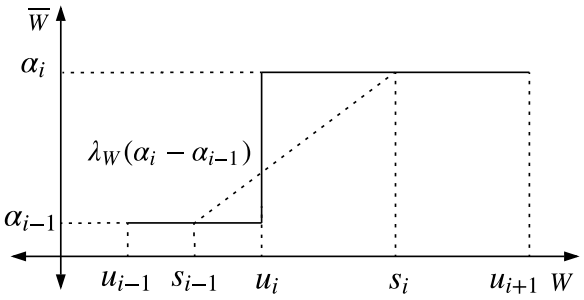


Figure 3.1: A sample of the forward propagation and backpropagation of weights approximation

With a configuration of different bit-widths for the weights, activations, and gradients, the accuracy degradation of DoReFa-Net can be preserved and controlled. But, fixed-point multipliers result in the most substantial overhead among that of all the quantization schemes in this chapter.

3.2.4. MULTIPLE BINARY CONVOLUTIONAL NEURAL NETWORKS

In multiple binary convolutional neural networks [39–44], a combination of multiple binary bases is adopted to approximate full precision weights and activations. Following is the weights approximation using linear combination.

$$x^r = \sum_{i=1}^{P} \varepsilon_i D_i \tag{3.5}$$

where ε_i is a trainable scaling coefficient and D_i is a binary (-1 and $+1$) weight base. During the backpropagation, STE method is still used.

The adoption of multiple binary bases in ABC-Net can lessen accuracy loss compared to single binary CNNs and maintain efficiency by using parallel bitwise operations compared to fix-point CNNs. Unfortunately, there is still a considerable gap between ABC-Net and full precision although as many as needed weight and activation binary bases can be used.

3.3. PIECEWISE APPROXIMATION SCHEME

In this section, the PA scheme for multiple binary CNNs is illustrated, including the approximations of weights and activations. Also, the training algorithm and the inference architecture of PA-Net are clarified.

3.3.1. WEIGHTS APPROXIMATION

Since approximating weights channel-wise needs much more computational resources during training, we approximated weights as a whole in this chapter.

Table 3.1: Endpoints of the weights with $M = 8$

Variables	u_1	u_2	u_3	u_4
Values ($\times\ std(W)$)	-1.5	-1.0	-0.5	-0.25
Variables	u_5	u_6	u_7	u_8
Values ($\times\ std(W)$)	0.25	0.5	1.0	1.5

The real-valued weights are $W \in R^{h\times w\times c_{in}\times c_{out}}$, where h, w, c_{in} and c_{out} represent the height and width of a filter, the number of input and output channels, respectively. In the forward propagation of PA scheme, these are estimated by $\overline{W}$, which is a piecewise function composed of the following $M+1$ pieces.

$$W \approx \overline{W} = \begin{cases} \alpha_1 \times B_W, W_j \in [-\infty, u_1] \\ \alpha_i \times B_W, W_j \in [u_{i-1}, u_i], i \in [2, \frac{M}{2}] \\ 0.0 \times B_W, W_j \in [u_{\frac{M}{2}}, u_{\frac{M}{2}+1}] \\ \alpha_i \times B_W, W_j \in [u_i, u_{i+1}], i \in [\frac{M}{2}+1, M-1] \\ \alpha_M \times B_W, W_j \in [u_M, +\infty] \end{cases} \tag{3.6}$$

where u_i and α_i are the endpoint and scaling coefficient of the pieces, respectively. W_j is a scalar and a single weight of the tensor W. $W_j \in [-\infty, u_1]$ refers to the j^{th} weight of the tensor W which is in the range of $[-\infty, u_1]$. B_W is a tensor with all the values equal to 1.0 and has the same shape as W. The distributions of the weights of spatial convolution layers and fully-connected layers are intending to follow Gaussian distribution, whose histogram is in bell-shape, owing to the regularization effect of L_2-norm weight penalty [50, 52, 53]. In particular, Gaussian distribution (i.e., $N(\mu, \sigma^2)$), where μ and σ are the calculated mean and standard deviation of the weight sample W. Since the distribution of the weights is close to Gaussian, all the endpoints of the weights are fixed using $mean(W)$ and $std(W)$, which refer to the mean and standard deviation of the full precision weights, respectively. The M endpoints are almost uniformly sampled from $-2.0\times std(W)$ to $2.0\times std(W)$ except those near 0.0. To set the endpoints of the weights properly, we attempted some different settings, where the performance difference is negligible. Taking $M = 8$ as an example, we directly recommend the endpoints set as listed in Table 3.1.

Except for the $(\frac{M}{2}+1)$-*th* piece, the mean of all the full precision weights of every piece serves as the optimal estimation of its scaling coefficient.

$$\begin{cases} \alpha_1 = \texttt{reduce\_mean}(W), W_j \in [-\infty, u_1] \\ \alpha_i = \texttt{reduce\_mean}(W), W_j \in [u_{i-1}, u_i], i \in [2, \frac{M}{2}] \\ \alpha_i = \texttt{reduce\_mean}(W), W_j \in [u_i, u_{i+1}], i \in [\frac{M}{2}+1, M-1] \\ \alpha_M = \texttt{reduce\_mean}(W), W_j \in [u_M, +\infty] \end{cases} \tag{3.7}$$

During the backpropagation, the relationship between $\overline{W}$ and W has to be established,

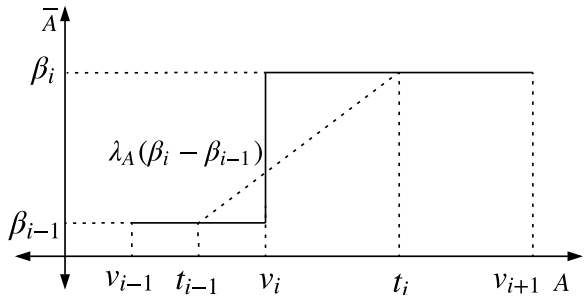


Figure 3.2: A sample of the forward propagation and backpropagation of activations approximation

and the whole range of the weights is segmented into M pieces.

$$\frac{\partial \overline{W}}{\partial W} = \begin{cases} \lambda_W(\alpha_2 - \alpha_1), W_j \in [-\infty, s_1] \\ \lambda_W(\alpha_{i+1} - \alpha_i), W_j \in [s_{i-1}, s_i], i \in [2, \frac{M}{2} - 1] \\ \lambda_W(0.0 - \alpha_{\frac{M}{2}}), W_j \in [s_{\frac{M}{2}-1}, s_{\frac{M}{2}}] \\ \lambda_W(\alpha_{\frac{M}{2}+1} - 0.0), W_j \in [s_{\frac{M}{2}}, s_{\frac{M}{2}+1}] \\ \lambda_W(\alpha_{i+1} - \alpha_i), W_j \in [s_i, s_{i+1}], i \in [\frac{M}{2} + 1, M - 2] \\ \lambda_W(\alpha_M - \alpha_{M-1}), W_j \in [s_{M-1}, +\infty] \end{cases} \tag{3.8}$$

where s_i is the endpoint of the pieces. λ_W is a hyper-parameter, which is different when a different number of weight pieces is used. The endpoint s_i can be determined simply as follows

$$s_i = (u_{i+1} + u_i)/2.0, i \in [1, M-1] \tag{3.9}$$

The forward propagation while $W_j \in [u_{i-1}, u_{i+1}]$ and backpropagation while $W_j \in [s_{i-1}, s_i]$ are presented in Figure 3.1, where a linear function with slope $\lambda_W(\alpha_i - \alpha_{i-1})$ is used to approximate the piecewise function during the backpropagation.

3.3.2. ACTIVATIONS APPROXIMATION

To utilize bitwise operation for convolution, activations should be binarized as well. However, the distribution of the activations will vary in the inference stage which motivates us to apply batch normalization [54]. Batch normalization is applied before the approximation of the activations to force them to have zero mean and unit standard deviation.

The real-valued input activations are $A \in R^{n \times h \times w \times c_{in}}$, where n, h, w and c_{in} refer to batch size, height, width and number of channels, respectively. In the forward propagation of PA scheme, these are estimated by $\overline{A}$, which is a piecewise function composed of the following $N+1$ pieces.

$$A \approx \overline{A} = \begin{cases} 0.0 \times B_A, A_j \in [-\infty, v_1] \\ \beta_i \times B_A, A_j \in [v_i, v_{i+1}], i \in [1, N-1] \\ \beta_N \times B_A, A_j \in [v_N, +\infty] \end{cases} \tag{3.10}$$

where v_i and β_i are the endpoint and scaling coefficient of the pieces, respectively. $A_j \in [v_i, v_{i+1}]$ refers to the activations of matrix A which are in the closed range of $[v_i, v_{i+1}]$. B_A is a tensor with all the values equal to 1.0 and has the same shape as A. Both the endpoint v_i and the scaling coefficient β_i are trainable to learn the statistical features of the full precision activations. The bounded activation function is omitted since the endpoints are initialized with positive values.

During the backpropagation, the relationship between $\overline{A}$ and A has to be established, and the whole range of the activations is segmented into $N+2$ pieces.

$$\frac{\partial \overline{A}}{\partial A} = \begin{cases} 0.0, A_j \in [-\infty, t_0] \\ \lambda_A \times (\beta_1 - 0.0), A_j \in [t_0, t_1] \\ \lambda_A \times (\beta_{i+1} - \beta_i), A_j \in [t_i, t_{i+1}], i = 1, ..., N-1 \\ 0.0, A_j \in [t_N, +\infty] \end{cases} \tag{3.11}$$

where t_i is the endpoint of the pieces. λ_A is a hyper-parameter, which is the same for all the layers in a given CNN and is different between different CNNs with different depths as used this chapter. The endpoint t_i can be determined as follows

$$\begin{cases} t_i = (v_i + v_{i+1})/2.0, i = 1, ..., N-1 \\ t_0 = 2.0 \times v_0 - s_1 \\ t_N = v_N + \lambda_\Delta \end{cases} \tag{3.12}$$

where λ_Δ is a hyper-parameter, which is the same for all the layers in a given CNN and is different between different CNNs in this chapter.

The forward propagation while $A_j \in [v_{i-1}, v_{i+1}]$ and backpropagation while $A_j \in [t_{i-1}, t_i]$ are presented in Figure 3.2, where a linear function with slope $\lambda_A(\beta_i - \beta_{i-1})$ is used to approximate the piecewise function during the backpropagation.

The scaling coefficient β_i is updated as follows

$$\frac{\partial C}{\partial \beta_i} = \frac{\partial C}{\partial \overline{A}} \frac{\partial \overline{A}}{\partial \beta_i} = \begin{cases} \texttt{reduce\_sum}(\frac{\partial C}{\partial \overline{A}}), A_j \in [v_i, v_{i+1}], i \in [1, N-1] \\ \\ \texttt{reduce\_sum}(\frac{\partial C}{\partial \overline{A}}), A_j \in [v_N, +\infty], i = N \end{cases} \tag{3.13}$$

Similarly, the endpoint v_i is updated as

$$\frac{\partial C}{\partial v_i} = \frac{\partial C}{\partial \overline{A}} \frac{\partial \overline{A}}{\partial v_i} = \begin{cases} \lambda_A(\beta_1 - 0.0) \times \texttt{reduce\_sum}(\frac{\partial C}{\partial \overline{A}}), A_j \in [t_0, t_1], i = 1 \\ \lambda_A(\beta_i - \beta_{i-1}) \times \texttt{reduce\_sum}(\frac{\partial C}{\partial \overline{A}}), A_j \in [t_{i-1}, t_i], i \in [2, N] \end{cases} \tag{3.14}$$

3.3.3. TRAINING ALGORITHM

A sample of the training algorithm of PA-Net is presented as Algorithm 2, where details like batch normalization and pooling layers are omitted. SGD with momentum or ADAM [55] optimizer can be used to update parameters. Since our PA scheme approximates full precision weights and activations, using pre-trained models serves as initialization.

Algorithm 2 Training a L-layer multiple binary CNN by PA scheme

Input: A mini-batch of inputs A_0 and targets A^*, weights W. Learning rate η, learning rate decay factor λ. The number of endpoints M, scaling coefficient α_i and endpoint u_i for weights, the number of endpoints N, scaling coefficient β_i and endpoint v_i for activations. PA is short for Piecewise Approximation scheme.

Output: Updated scaling coefficient β_i, endpoint v_i, weights W and learning rate η.

1. Computing the parameter gradients:

1.1. Forward path:

for $k = 1$ to L **do**
  $\overline{W} \leftarrow \mathrm{PA}(W, u_i, \alpha_i, M)$
  $A \leftarrow \mathrm{Conv}(\overline{A}, \overline{W})$
  if $k < L$ **then**
    $\overline{A} \leftarrow \mathrm{PA}(A, v_i, \beta_i, N)$
  end if
end for

1.2. Backward propagation:

for $k = L$ to 1 **do**
  if $k < L$ **then**
    $(g_A, g_{v_i}, g_{\beta_i}) \leftarrow \mathrm{Back\_PA}(g_{\overline{A}}, A, v_i, \beta_i, N)$
  end if
  $(g_{\overline{A}}, g_{\overline{W}}) \leftarrow \mathrm{Back\_Conv}(g_A, \overline{A}, \overline{W})$
  $g_W \leftarrow \mathrm{Back\_PA}(g_{\overline{W}}, W, u_i, \alpha_i, M)$
end for

2. Accumulating the parameter gradients:

for $k = 1$ to L **do**
  $\beta_i \leftarrow \mathrm{update}(\beta_i, \eta, g_{\beta_i})$
  $v_i \leftarrow \mathrm{update}(v_i, \eta, g_{v_i})$
  $W \leftarrow \mathrm{update}(W, \eta, g_W)$
  $\eta \leftarrow \lambda\eta$
end for

3.3.4. INFERENCE ARCHITECTURE

Regarding the inference implementation of PA-Net, the latency is one of the most important metrics to be considered. Fortunately, the piecewise approximated weights or activations can be viewed as a linear combination of multiple binary bases (+1 and 0), which indicates a parallel inference architecture.

In the forward propagation, the approximated weights are represented as follows.

$$\overline{W} = \sum_{i=1}^{M} \alpha_i T_i \tag{3.15}$$

where T_i is a binary weight base, given as

$$T_i = \begin{cases} \begin{cases} B_W, W_j \in [-\infty, u_1] \\ 0.0 \times B_W, W_j \notin [-\infty, u_1] \end{cases} & , i = 1 \\ \begin{cases} B_W, W_j \in [u_{i-1}, u_i] \\ 0.0 \times B_W, W_j \notin [u_{i-1}, u_i] \end{cases} & , i \in [2, \frac{M}{2}] \\ \begin{cases} B_W, W_j \in [u_i, u_{i+1}] \\ 0.0 \times B_W, W_j \notin [u_i, u_{i+1}] \end{cases} & , i \in [\frac{M}{2} + 1, M-1] \\ \begin{cases} B_W, W_j \in [u_M, +\infty] \\ 0.0 \times B_W, W_j \notin [u_M, +\infty] \end{cases} & , i = M \end{cases} \tag{3.16}$$

Similarly, the approximated activations in the forward propagation are expressed as follows.

$$\overline{A} = \sum_{i=1}^{N} \beta_i V_i \tag{3.17}$$

where V_i is a binary activation base, given as

$$V_i = \begin{cases} \begin{cases} B_A, A_j \in [v_i, v_{i+1}] \\ 0.0 \times B_A, A_j \notin [v_i, v_{i+1}] \end{cases} & , i = 1, ..., N-1 \\ \begin{cases} B_A, A_j \in [v_N, +\infty] \\ 0.0 \times B_A, A_j \notin [v_N, +\infty] \end{cases} & , i = N \end{cases} \tag{3.18}$$

Combined with the approximated weights, the forward propagation of the real-valued convolution can be approximated by computing $M \times N$ parallel bitwise convolutions. It is worth to notice that $\alpha_i \beta_j$ will be merged as one new scaling coefficient ϕ_k during the inference stage so that we omit their multiplication.

$$\begin{aligned} &Conv(W, A) \approx Conv(\overline{W}, \overline{A}) = Conv(\sum_{i=1}^{M} \alpha_i T_i, \sum_{j=1}^{N} \beta_j V_j) \\ &= \sum_{i=1}^{M} \sum_{j=1}^{N} \alpha_i \beta_j Conv(T_i, V_j) = \sum_{k=1}^{M \times N} \phi_k Conv(T_i, V_j) \end{aligned} \tag{3.19}$$

Taking $M = 3$ and $N = 3$ as an example, both the weights and activations use 3 bits to approximated their full precision counterpart. A full precision convolution can be computed with 9 parallel bitwise operations and 3 comparators, as shown in Figure 3.3, where the latency cost is as small as that of single binary CNNs. On the left is the structure of the activations approximation using binary activation bases V_1, V_2, and V_3. On the right is the structure of the weights approximation using binary weight bases T_1, T_2, and T_3. Thus, we implement the overall block structure of the convolution in the PA scheme with 9 parallel bitwise operations. It is worth to notice that computing the binary convolution blocks in this figure can be directly completed by AND and popcount operations, and the binary convolution blocks do not consist of Batch Normalization or Relu layer.

3.3.5. EFFICIENCY ANALYSIS OF DIFFERENT BINARY VALUES

To the best of our knowledge, this is the first time to use binary values +1 and 0 instead of binary values −1 and +1 for single or multiple binary CNNs, and we present their efficiency analysis in terms of costumed hardware FPGA/ASIC.

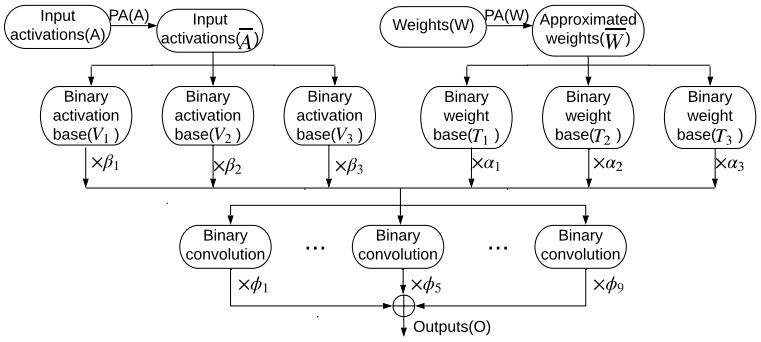


Figure 3.3: Parallel inference architecture of convolution in PA-Net

Table 3.2: 2-input 7-nm CMOS gates propagation delay, area, and power

Items	Propagation delay [ps]	Active area [nm^2]	Power [nW]
XNOR	10.87	2.90×10^3	1.23×10^3
AND	9.62	1.45×10^3	6.24×10^2

When binary convolutions are computed by bitwise operation with binary values 0 and +1, the dot product of two bit-vectors x and y is computed using bitwise operations as follows.

$$x \cdot y = \text{bitcount}(\text{AND}(x, y)), x_i, y_i \in \{0, +1\} \forall_i \tag{3.20}$$

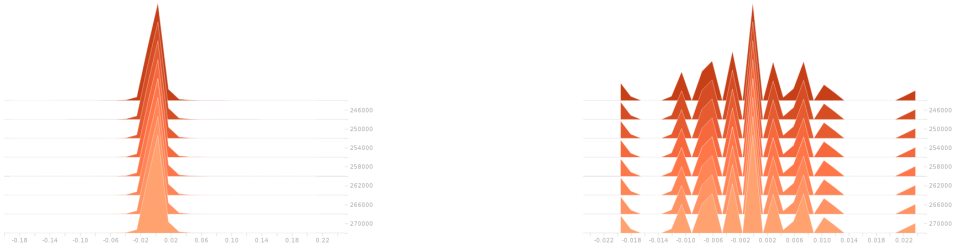
where $bitcount$ counts the number of bits in a bit-vector.

Similarly, when binary convolutions are computed by bitwise operation with binary values -1 and $+1$, the dot product of two bit-vectors x and y is computed using bitwise operations as follows.

$$x \cdot y = N - 2 \times \text{bitcount}(\text{XNOR}(x, y)), x_i, y_i \in \{-1, +1\} \forall_i \tag{3.21}$$

where N is the number of bits in a bit-vector.

In Table 3.2, we present the area footprint, the input to output propagation delay and the power consumption for 2-input Boolean gates using a commercial 7-nm FinFET technology (supply voltage $V_{DD} = 0.7V$). The active area and power consumption cost of an XNOR gate are two times as large as those of an AND gate, which indicates that the area and power consumption cost of a binary convolution with binary values -1 and $+1$ are two times as large as those of a binary convolution with binary values 0 and $+1$ (except for bitcount operation).



(a) Full precision weights (b) Approximated weights

Figure 3.4: Distribution of full precision and approximated weights.

3.4. EXPERIMENTAL RESULTS ON IMAGENET DATASET

We first trained and evaluated ResNet [22] using our proposed PA scheme on ImageNet ILSVRC2012 classification dataset [56]. Then we generalize our scheme to other CNN architectures such as DenseNet and MobileNet. Finally, the computational complexity of PA-Net is analyzed on CPUs and customized hardware.

We set the batch size of all our implementations to 64 due to the limit on available time and resources, which slightly limits the accuracy of the results. However, the accuracy is expected to increase with a larger batch size.

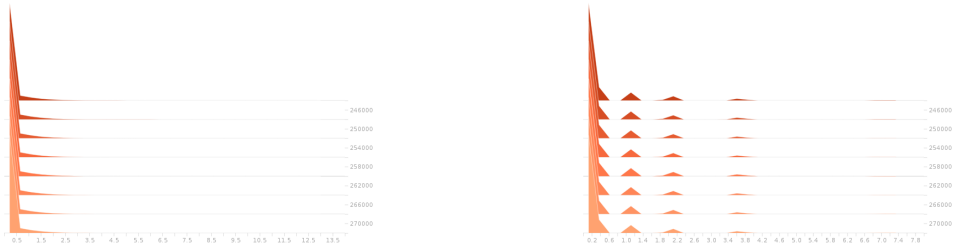
3.4.1. WEIGHTS AND ACTIVATIONS APPROXIMATIONS

Using the ResNet18/group2/block1/conv1 layer, we sampled full precision weights and their approximations with $M = 8$. Their histograms are shown in Figure 3.4a and 3.4b, respectively. Horizontal axis and longitudinal axis represent the values and the number of values of weights/activations, respectively. For the full-precision and approximated weights, their mode is around 0.0. The full-precision and approximated weights are in the range of -0.02 to 0.02. The distributions of the full-precision and approximated weights are symmetric and in bell shape. Similarly, the comparison of activation histograms are shown in Figure 3.5, which are acquired from the ResNet18/group2/block1/conv2 layer and include the full precision activations in Figure 3.5a and their approximation with $N = 5$ in Figure 3.5b. For the full-precision and approximated activations, their mode is near 0.0 and they are in the range of 0.0 to 4.0. The distribution of the full-precision and approximated activations is the second half of the bell curve. As the comparisons show, the distributions of the approximated weights and activations are similar to those of the full precision weights and activations, respectively, which means that PA scheme provides an accurate way for multiple binary bases to approximate the distribution of their full precision counterparts.

3.4.2. COMPARISON WITH ABC-NET

Both PA-Net and ABC-Net can utilize parallel bitwise operation and achieve higher accuracy than single binary CNNs, so the differences between them need to be analyzed. The accuracy comparisons between PA-Net and ABC-Net are shown in Table 3.3.

Table 3.3 shows that PA-Net achieves higher accuracy than ABC-Net while requiring less



(a) Full precision activations

(b) Approximated activations

Figure 3.5: Distribution of full precision and approximated activations.

Table 3.3: Comparison with ABC-Net using ResNet as backbones

Model	M	N	Top-1	Top-5	Top-1 gap	Top-5 gap
ABC-ResNet18	5	full precision	68.3%	87.9%	1.0%	1.3%
PA-ResNet18	4	full precision	68.4%	88.3%	0.9%	0.9%
PA-ResNet18	8	full precision	69.3%	88.9%	0.0%	0.3%
ABC-ResNet18	5	5	65.0%	85.9%	4.3%	3.3%
PA-ResNet18	4	5	66.6%	87.1%	2.7%	2.1%
PA-ResNet18	8	7	68.1%	88.1%	1.2%	1.1%
ResNet18	full precision	full precision	69.3%	89.2%	–	–
ABC-ResNet34	5	5	68.4%	88.2%	4.9%	3.1%
PA-ResNet34	4	5	70.1%	89.2%	3.2%	2.1%
PA-ResNet34	8	7	71.5%	90.0%	1.8%	1.3%
ResNet34	full precision	full precision	73.3%	91.3%	–	–
ABC-ResNet50	5	5	70.1%	89.7%	6.0%	3.1%
PA-ResNet50	4	5	73.0%	91.0%	3.1%	1.8%
PA-ResNet50	8	7	74.3%	91.9%	1.8%	0.9%
ResNet50	full precision	full precision	76.1%	92.8%	–	–

overhead, which strongly supports the idea that PA-Net provides a better approximation than ABC-Net for both the weights and activations. In addition, Table 3.3 shows the unique advantage of PA-Net over ABC-Net since PA-Net can give higher accuracy for multiple binary CNNs by increasing M and N. However, we also re-implemented ABC-Net and reproduced the results, which shows that its accuracy remains unchanged (or even becomes worse) as we keep increasing M and N more than 5.

For the weights approximation only (i.e., when N is full precision), PA-ResNet18 gives no Top-1 accuracy loss with $M = 8$. PA-ResNet achieves higher accuracy with $M = 4$ and $N = 5$ than ABC-ResNet with $M = 5$ and $N = 5$, which means that PA-ResNet provides better approximation with less overhead. PA-ResNet with $M = 8$ and $N = 7$ reduce the Top-5 accuracy gap around 1.0%. But the accuracy of ABC-Net remains unchanged or even becomes worse with the increase of M and N more than 5 based on our re-implementation. PA-Net is expected to reach no accuracy loss with the increase of M and N, which we have not attempted due to the limitations of computational resources, training time, and the slow increase trend of accuracy with the increase of M and N.

Table 3.4: Generalization to DenseNet and MobileNet.

Model	M	N	Top-1	Top-5	Top-1 gap	Top-5 gap
PA-DenseNet121	8	6	72.3%	90.8%	2.7%	1.5%
DenseNet121	full precision	full precision	75.0%	92.3%	–	–
PA-1.0 MobileNet-224	8	7	69.0%	88.4%	1.6%	1.5%
1.0 MobileNet-224	full precision	full precision	70.6%	89.9%	–	–

Table 3.5: Performances of the full-precision SSD300 network and its binary counterpart.

Detector	Backbone	Weights	Activations	mAP@0.5
SSD300	ResNet50	Full precision	Full precision	74.35
SSD300	ResNet50	$M = 4$	Full precision	72.53
SSD300	ResNet50	$M = 4$	$N = 5$	58.60

3.4.3. GENERALIZATION TO OTHER CNN ARCHITECTURES

To demonstrate the generalization of PA scheme, we applied it on 1.0 MobileNet-224 [24] and DenseNet121 [23]. The results are shown in Table 3.4. Due to memory limitation, we implemented PA-DenseNet121 with $N = 6$. Its Top-1 accuracy loss is 2.7%, which is expected to decrease further with increasing N. Top-1 accuracy loss of PA-1.0 MobileNet-224 achieves 1.6% with $N = 7$. Pointwise convolution is binarized while depthwise convolution is kept as full precision convolution since they do not need significant computational resources.

3.4.4. GENERALIZATION TO OBJECT DETECTION

We choose SSD300 with the backbone network of ResNet50 as our baseline. The training dataset is VOC2007 + 2012, while the testing dataset is VOC2007 [57]. In the SSD300 model, we use the layers from Conv1 to Conv5_x of the pre-trained ResNet50 as the backbone network, apply residual blocks as the extra layers, and keep the number of feature maps the same as the original implementation [58]. All the backbone layers except Conv1 are binarized, while all the convolutional layers of the head network remain in full precision. We train the full precision ResNet50 on the ImageNet classification dataset as the backbone network, and then the full precision object detector SSD300 using the pre-trained ResNet50. Finally, we binarize and finetune the pre-trained object detector SSD300 with the PA scheme.

Applying the PA scheme to the SSD300 network, we present the results in Table 3.5. When only weights are binarized using the PA scheme with $M = 4$, the binary SSD300 model achieves comparable accuracy by 1.82 mAP reduction compared with its full precision baseline networks. When applying the PA scheme with binary weights ($M = 4$) and binary activations ($N = 5$) for the SSD300 network, the binary SSD300 network shows an accuracy reduction in 15.75 mAP, which outperforms the real-time full-precision Fast YOLO [59] (52.7 mAP).

Table 3.6: Accuracy comparisons of ResNet18 with different quantized methods.

Model	W	A	$Top-1$	$Top-5$
Full Precision	32	32	69.3%	89.2%
BWN	1	32	60.8%	83.0%
XNOR-Net	1	1	51.2%	73.2%
Bi-Real Net	1	1	56.4%	79.5%
ABC-Net ($M=5$, $N=5$)	1	1	65.0%	85.9%
Group-Net (5 bases)	1	1	64.8%	85.7%
DoReFa-Net	2	2	62.6%	84.4%
SYQ	1	8	62.9%	84.6%
LQ-Net	2	2	64.9%	85.9%
PA-Net ($M=8$, $N=7$)	1	1	68.1%	88.0%
PA-Net ($M=8$)	1	32	69.3%	88.9%

Table 3.7: Memory usage and Flops calculation of Bi-Real Net, PA-Net, and full precision models.

Model	Memory usage	Memory saving	Flops	Speedup
Bi-Real ResNet18	33.6Mbit	11.14 ×	1.67×10^8	10.86 ×
ABC-ResNet18	77.1Mbit	4.85 ×	6.74×10^8	2.70 ×
PA-ResNet18	61.6Mbit	6.08 ×	6.74×10^8	2.70 ×
ResNet18	374.1Mbit	–	1.81×10^9	–
Bi-Real ResNet34	43.7Mbit	15.97 ×	1.81×10^8	18.99 ×
ABC-ResNet34	106.3Mbit	6.56 ×	1.27×10^9	2.88 ×
PA-ResNet34	85.0Mbit	8.20 ×	1.27×10^9	2.88 ×
ResNet34	697.3Mbit	–	3.66×10^9	–
Bi-Real ResNet50	176.8Mbit	4.62 ×	5.45×10^8	7.08 ×
ABC-ResNet50	201.6Mbit	4.06 ×	1.44×10^9	2.68 ×
PA-ResNet50	161.3Mbit	5.07 ×	1.44×10^9	2.68 ×
ResNet50	817.8Mbit	–	3.86×10^9	–

3.4.5. COMPARISONS WITH STATE-OF-THE-ART METHODS

The comparisons between PA-Net and recent developments are shown in Table 3.6, where PA-Net adopts the configuration of $M = 8$ and $N = 7$. Regarding single binary models BWN, XNOR-Net [37] and Bi-Real Net [38], and multiple parallel binary models ABC-Net[39] and Group-Net[44], PA-Net outperforms them by much higher accuracy. When it comes to the comparison with fix-point quantization DoreFa-Net [6, 31, 32], fixed-point CNNs can achieve the same or even higher performance with carefully customized bit-widths than PA-Net. But the advantage of PA-Net is the parallelism of inference architecture, which provides a much lower latency using bitwise operation than fixed-point CNNs.

Table 3.8: Latency cost of Bi-Real Net, PA-Net, and full precision models. T_{XNOR}, T_{pop}, T_{mul}, T_{AND}, T_{com}, T_{add} refer to the delay time of a XNOR, popcount, multiplication, AND, comparison, and addition operation, respectively.

Model	Latency cost	Speedup
Bi-Real Net	$c_{in}hw \times (T_{XNOR} + T_{pop}) + T_{mul}$	$\approx (T_{mul} + T_{add})/(T_{XNOR} + T_{pop})$
PA-Net	$c_{in}hw \times (T_{AND} + T_{pop}) + 5T_{mul} + 4T_{add} + T_{com}$	$\approx (T_{mul} + T_{add})/(T_{AND} + T_{pop})$
Full precision	$c_{in}hw \times T_{mul} + (c_{in}hw - 1) \times T_{add}$	–

3.4.6. COMPUTATIONAL COMPLEXITY ANALYSIS

In this part, we analyze and compare the computational complexity of Bi-Real Net (Liu et al. 2018), PA-Net, and full precision models on current CPUs in terms of computation and memory usage, and on customized hardware (i.e., FPGA/ASIC) in terms of latency. Bi-Real Net maintains high efficiency and achieves the state-of-the-art accuracy as a single binary CNN. During this analysis, PA scheme uses 4 bases for weights and 5 bases for activations approximation.

Computation and memory usage analysis: We analyze and compare the computational complexity of Bi-Real Net [38], PA-Net and full precision models, and their memory saving and speedup are shown in Table 3.7.

Unlike full precision models which require real-valued parameters and operations, PA-Net and Bi-Real Net have binary and real-valued parameters mixed, so their execution requires both bitwise and real-valued operations. To compute the memory usage of PA-Net and Bi-real Net, we use 32 bit times the number of real-valued parameters and 1 bit times the number of binary values, which are summed together to get their total number bit. We use Flops as the main metrics to measure the bitwise operations, the real-valued operations, and the speedup of implementation. Since the current generation of CPUs can compute bitwise AND and popcount operations in parallelism of 64, the Flops to compute PA-Net and Bi-Real Net is equal to the number of the real-valued multiplications, comparisons, and $1/64$ of the number of the bitwise operations.

We follow the suggestion from [37, 38] to keep the weights and activations of the first convolutional and the last fully connected layer as real-valued. It is worthy to notice that we binarize all the 1×1 downsampling layer in PA-Net to further reduce the computational complexity.

For ResNet18, ResNet34, and ResNet50, our PA scheme can reduce memory usage by more than 5 times and achieves a computation reduction of nearly 3 times, in comparison with the full precision counterpart. Compared with Bi-Real ResNet50, the computation reduction of our proposed PA-ResNet50 with 4 weights bases and 5 activations bases is only two times smaller, and it even requires less memory usage because of the binarization of the downsampling layer.

Combining Table 3 and Table 7, we can conclude that PA-Net can achieve better accuracy (1.6%, 1.7%, and 2.9% for ResNet18, ResNet34, and ResNet50) while consuming fewer parameters (15.4Mbit, 21.3Mbit, and 40.33Mbit for ResNet18, ResNet34, and ResNet50) and the same Flops compared to ABC-Net during the inference stage.

Latency analysis: To be implemented on customized hardware (i.e., FPGA/ASIC), latency cost is one of the most important metrics for real-time applications. As shown in Table 3.8, the latency cost of an individual convolution in Bi-Real Net, PA-Net, and full precision models is analyzed, where we assume that the convolution implementation is paralleled thoroughly. Compared with full precision models, the latency cost of PA-Net and Bi-Real Net is significantly reduced. T_{AND} is smaller than T_{XNOR}, and the latency cost of a convolution in PA-Net increased only by $4T_{mul} + 4T_{add} + T_{com}$ compared with that in Bi-Real Net.

3.5. CONCLUSIONS

In this chapter, we introduced the PA scheme for multiple binary CNNs, which adopts piecewise functions for both the forward propagation and backpropagation. Compared with state-of-the-art single and multiple binary CNNs, our scheme provides a better approximation for both full precision weights and activations. We implemented our scheme over several modern CNN architectures, such as ResNet, DenseNet, and MobileNet, and tested on classification task using ImageNet dataset. Results are competitive and almost close the accuracy gap compared with their full precision counterparts. Because of the binarization of downsampling layer, our proposed PA-ResNet50 requires less memory usage and only two times Flops than Bi-Real Net with 4 weights and 5 activations bases, which shows its potential efficiency advantage over single binary CNNs with a deeper network.

4

NEURAL ARCHITECTURE SEARCH

Binary Convolutional Neural Networks (CNNs) have significantly reduced the number of arithmetic operations and the size of memory storage needed for CNNs, which makes their deployment on mobile and embedded systems more feasible. However, after binarization, the CNN architecture has to be redesigned and refined significantly due to two reasons: 1. the large accumulation error of binarization in the forward propagation, and 2. the severe gradient mismatch problem of binarization in the backward propagation. Even though substantial effort has been invested in designing architectures for single and multiple binary CNNs, it is still difficult to find an optimized architecture for binary CNNs. In this chapter, we propose a strategy, named NASB, which adapts Neural Architecture Search (NAS) to find an optimized architecture for the binarization of CNNs. In the NASB strategy, the operations and their connections define a unique searching space and the training and binarization of the network progress in the three-stage training algorithm. Due to the flexibility of this automated strategy, the obtained architecture is not only suitable for binarization but also has low overhead, achieving a better trade-off between the accuracy and computational complexity compared to hand-optimized binary CNNs. The implementation of the NASB strategy is evaluated on the ImageNet dataset and demonstrated as a better solution compared to existing quantized CNNs. With insignificant overhead increase, NASB outperforms existing single and multiple binary CNNs by up to 4.0% *and* 1.0% *Top-1 accuracy respectively, bringing them closer to the precision of their full precision counterpart.*

The content of this chapter is based on [60].

4.1. INTRODUCTION

With the increasing depth and width of Convolutional Neural Networks (CNNs), these networks have demonstrated many breakthroughs in a wide range of applications, such as image classification, object detection, and semantic segmentation [22, 25, 61]. However, the large number of Flops and the storage associated with large numbers of parameters limits deployment on resource-constrained mobile and embedded platforms.

Numerous researchers have proposed a variety of approaches to address the efficiency problems associated with deploying CNNs, including low bit-width quantization [6, 62], network pruning [63], and efficient architecture design [25, 64]. Binarization [37, 38] is the most efficient quantization method among all those methods with reduced bit-widths, where a real-valued weight or activation is represented with a single bit and the multiplication and addition of a convolution can be implemented simply by XNOR and popcount bitwise operations, which are roughly 64 times faster to compute and require thirty two times less storage than their full precision counterparts. However, the extreme quantization method of single binary CNNs introduces the largest accumulation error in the forward propagation. In addition, during the backward propagation, its gradient flow is the most difficult to determine due to the high gradient mismatch problem [65] among all quantization methods with reduced bit-widths.

Existing published work focuses on improving the quantization quality mainly by using value approximation and structure approximation. These two approximations are complementary and could be exploited together. Value approximation seeks to find an optimized algorithm to quantize weights and activations while preserving the original network architecture. Knowledge distillation [29, 66] and loss-aware [67] objectives are introduced to find optimized local minima for quantized weights and activations. Advanced quantization functions [6, 62, 65] are proposed to minimize the quantization error between quantized values and their full-precision counterparts. Tight approximation of the derivative of the non-differentiable activation function [38, 68] is explored to alleviate the gradient mismatch problem. Unlike the above value approximation methods, structure approximation seeks to redesign the architecture of quantized CNNs to match the representational capacity of their original full-precision counterpart. Structure approximation is more important for binary CNNs than for other low bit-width CNNs because binarization introduces the largest accumulation error and the severest gradient mismatch problem among all quantization methods with reduced bit-widths. Bi-Real Net [38] and Group-Net [69] are the state-of-the-art structure approximation methods for single and multiple binary CNNs, respectively. However, designing architectures for quantized CNNs is highly non-trivial especially for binary CNNs.

In this chapter, the NASB strategy, a version of Neural Architecture Search (NAS) adapted for binarization, is proposed to automatically seek an optimized structure approximation for binary CNNs. After searching in a large space, the finalized CNN architecture is suitable for binarization, and the accuracy of the binarized version outperforms previous binary CNNs with insignificant computational complexity increase.

The main contributions of this chapter are:

- The NASB strategy, which adapts NAS to automatically find an optimized architecture for the binarization of CNNs. In the NASB strategy, the operations and their connections define a unique searching space and the training and binarization of the

network progress in the three-stage training algorithm.

- A comparison to the recent literature of binary CNNs. NASB achieves a sizable accuracy increase with negligible additional overhead, providing a better trade-off between accuracy and efficiency.
- An evaluation of the NASB strategy for ResNet on the ImageNet classification dataset, providing extensive experimental results to show its effectiveness.

4.2. RELATED WORK

In this section, recent network quantization methods and efficient architecture design developments of CNNs are described.

4.2.1. NETWORK QUANTIZATION

There is substantial interest in research and development of dedicated hardware for CNNs to be deployed on embedded systems and mobile devices, which motivates the study of network quantization. Low bit-width approaches [6, 32, 51, 62] use quantized weights and activations using fixed-point numbers, which reduces model size and compute time, but still requires multipliers to compute. Binary CNNs [33, 37, 45] are trained with weights and activations constrained to binary values $+1$ or -1, which can be categorized as single binary CNNs. The Ternary Weight Networks (TWN) [47] approach is proposed to reduce the loss of single binary CNNs by introducing 0 as the third quantized value, while Trained Ternary Quantization (TTQ) [48] enables the asymmetry and training of its scaling coefficients. However, the accuracy degradation of single binary and ternary CNNs is unacceptable for advanced CNNs like ResNet and large scale datasets like ImageNet. Multiple binary CNNs [39, 40, 69, 70] are promising attempts to reduce the accuracy gap between binary CNNs and their full precision counterpart. However, all the architectures of current single or multiple binary CNNs are human-designed. Further architecture optimization is possible using automated methods, such as [45], which encodes the number of channels in each layer, but does not change the operations and their connections in the model; changes that we do consider in our proposed NASB strategy.

4.2.2. EFFICIENT ARCHITECTURE DESIGN

Recently, more and more literature focuses on the efficient architecture design for the deployment of CNNs. Replacing 3×3 convolutional weights with 1×1 weights (in SqueezeNet [61] and GoogLeNet [71]) have been suggested to decrease the computational complexity. Moreover, separable convolutions are adopted in Inception series [72] and further generalized as depthwise separable convolutions in Xception [73], MobileNet [25] and ShuffleNet [64]. Group convolution has been used as an efficient way to enhance efficiency in [64, 74], where the input activations and convolutional kernels are factorized into groups and executed independently inside each group. The MobileNet [75] and ShuffleNet [76] series have been leveraging depthwise separable convolutions and shuffle operations to achieve a better trade-off between efficiency and accuracy. ESPNetv2 [77] uses group point-wise

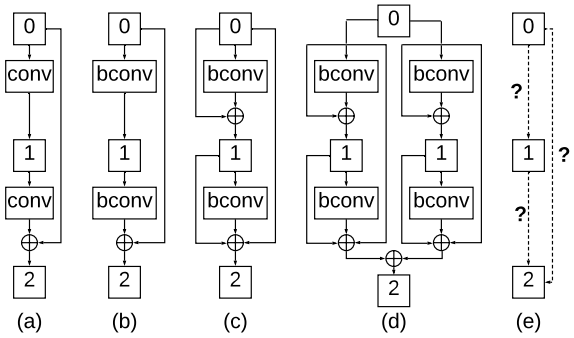


Figure 4.1: Human-designed architecture for single and multiple binary CNNs. conv and bconv refer to full precision and binary convolutional layer, respectively, while Batch Normalization and the Relu layers are omitted.

and depth-wise dilated separable convolutions to learn representations from a large effective receptive field, delivering state-of-the-art performance across different tasks. NAS [78–80] has demonstrated much success in automating network architecture design, achieving state-of-the-art efficiency [81, 82].

4.3. METHOD

In this section, the problem of finding an architecture for the binarization of CNNs is defined and presented. Then, we explain the NASB strategy, which adapts the NAS technique to find an optimized architecture for binarizing CNNs. Finally, variants of the NASB strategy are illustrated to enhance its efficiency.

4.3.1. PROBLEM DEFINITION

Given a full-precision convolutional cell, what is an optimized architecture to binarize it? The accumulation error in the forward propagation of binarization is the largest and the gradient flow in the backward propagation is the most difficult to take care of among all quantization methods with different bit-widths. As a result, it is non-trivial to find an optimized architecture for binarizing CNNs. For the purposes of this chapter, a convolutional cell can be a convolutional layer, block, group, and network.

There have been various attempts to answer the above question, as shown in Fig. 4.1. Fig. 4.1(a) is a full precision convolutional block. Fig. 4.1(b), (c), and (d) describe proposed architectures in the literature representing XNOR [37], Bi-Real [38], and Group-Net [69], respectively, where the scaling coefficients have been omitted. Although a lot of effort has been dedicated to manually designing an architecture for single and multiple binary CNNs, using an automated approach to explore an optimized convolutional cell architecture as represented by the question marks in Fig. 1(e) remains a fertile area for research.

The question can be expressed as a directed acyclic graph in Fig. 4.1(e), which represents an ordered sequence of three nodes and three edges with one operation for each edge. The

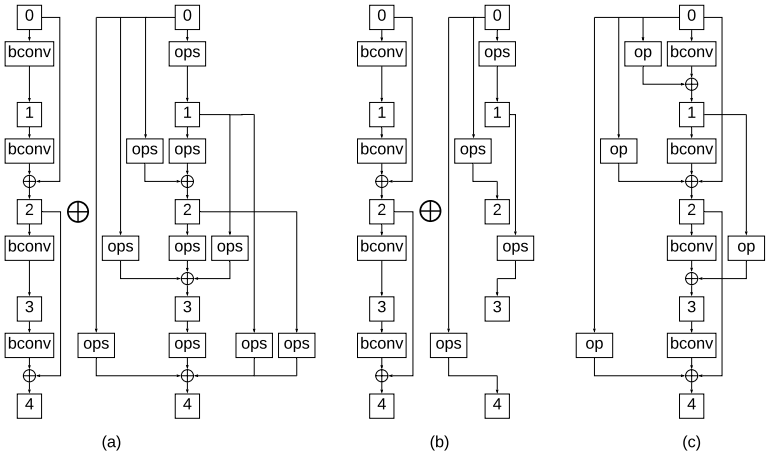


Figure 4.2: Exploring connections of a NASB-convolutional cell at the searching stage. conv and bconv refer to full precision and binary convolutional layer, respectively. ops refers to a set of operations as shown in Fig. 4.3, among which one operation is active during the training of the searching stage. $\oplus$ refers to the element-wise addition between the tensors of the two nodes with the same number.

number of nodes, edges, and operations for each edge can be freely selected. Each node x^i represents a feature map and each edge (i, j) is associated with several operations $o^{i,j}$ to transform x^i. Here the convolutional cell has one input and output node, and its output is obtained by addition of all intermediate nodes. In the following, the binarization and NAS techniques adapted in this chapter are presented.

Binary convolutional neural networks Given a full precision convolutional layer, its inputs, weights and outputs are denoted as $I \in R^{N\times C_{in}\times H\times W}$, $W \in R^{C_{in}\times C_{out}\times h\times w}$ and $O \in R^{N\times C_{out}\times H\times W}$, respectively, where N, C_{in}, C_{out}, H, W, h and w refer to the batch size, the number of input and output channels, the height and width of the feature maps, and the height and width of the weights, respectively.

Using the binarization method of weights in [37], we approximate the full precision weights W as binary weights b^W with the sign of W and the scaling coefficient s, where the scaling coefficient is computed as the mean of the absolute values of W. Adopting the Straight Through Estimator (STE) [46], the forward and backward propagations of the weights binarization are shown as follows.

$$\begin{aligned} &\text{Forward: } b^W = s \times \text{sign}(W) \\ &\text{Backward: } \frac{\partial L}{\partial W} = \frac{\partial L}{\partial b^W} \times \frac{\partial b^W}{\partial W} \approx s \times \frac{\partial L}{\partial b^W} \end{aligned} \tag{4.1}$$

where L is the total loss.

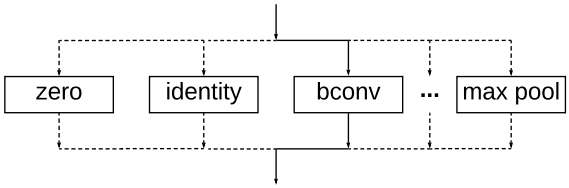


Figure 4.3: A set of operations in every ops.

Using the binarization method of activations in [38], we approximate the full precision activations as binary activations b^I by a piecewise polynomial function. The forward and backward propagations of the activations binarization can be written as follows.

$$\text{Forward: } b^I = \text{sign}(I)$$
$$\text{Backward: } \frac{\partial L}{\partial I} = \frac{\partial L}{\partial b^I} \times \frac{\partial b^I}{\partial I} \text{where} \frac{\partial b^I}{\partial I} = \begin{cases} 2+2I, -1 \le I < 0 \\ 2-2I, 0 \le I < 1 \\ 0, \text{otherwise} \end{cases} \tag{4.2}$$

Gradient based neural architecture search We adapt a gradient-based NAS in [83]. To reduce the memory footprint during training the over-parameterized network, we use the strategy from [80] to binarize and learn the M real-valued architecture parameters α_i.

In the forward propagation, the M real-valued architecture parameters α_i are transformed to the real-valued path weights p_i, and then to the binary gates g_i as follows.

$$p_i = \frac{\exp(\alpha_i)}{\sum_{j=1}^{M} \exp(\alpha_j)} \tag{4.3}$$

$$g_i = \text{binarize}(p_i) = \begin{cases} 1, \text{with probability } p_i \\ 0, \text{with probability } (1-p_i) \end{cases} \tag{4.4}$$

In the backward propagation, the STE [46] is also applied.

$$\frac{\partial L}{\partial p_j} \approx \frac{\partial L}{\partial g_j} \tag{4.5}$$

The gradient w.r.t. architecture parameters can be estimated as follows.

$$\frac{\partial L}{\partial \alpha_i} = \sum_{j=1}^{M} \frac{\partial L}{\partial p_j} \frac{\partial p_j}{\partial \alpha_i} \approx \sum_{j=1}^{M} \frac{\partial L}{\partial g_j} \frac{\partial p_j}{\partial \alpha_i} = \sum_{j=1}^{M} \frac{\partial L}{\partial g_j} p_j(\delta_{ij} - p_i) \tag{4.6}$$

where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$.

4.3.2. NASB STRATEGY

In this section, we present the details of the NASB strategy. To apply NAS for binarizing CNNs, the key innovation is to leverage the NAS technique to find a NASB-convolutional cell as an optimized architecture for binarizing their full precision counterpart, where the NASB-convolutional cell can be a replacement for a binarized convolutional layer, block, group, and network. The NASB strategy consists of the following stages: searching stage, pretraining stage, and finetuning stage. In the following, the search space of a NASB-convolutional cell in the NASB strategy is described, including its connections and operations. The corresponding training algorithm is also presented.

Connections of a NASB-convolutional cell Considering that we are exploring an optimized architecture for a convolutional group as an example, the connections of a NASB-convolutional cell in the NASB strategy are explored at the searching stage as shown in Fig. 4.2.

Fig. 4.2(a) describes all the connections of a NASB-convolutional cell during the training of the searching stage, which consists of a backbone and a NAS-convolutional cell. The left cell is the backbone of the NASB-convolutional cell, which is a standard convolutional group in ResNet [22]. The right cell is considered as a NAS-convolutional cell, which is a directed acyclic graph consisting of five nodes, ten edges, and ten operations for every edge. Here five nodes are used to keep the layer depth of a NASB-convolutional cell in the NASB strategy the same as its full precision counterpart, which will not increase the latency during inference. The connections of the backbone are fixed and there is no need to specify architecture parameters for it. During the training of the searching stage, the model weights of the NASB-convolutional cell and architecture parameters of the NAS-convolutional cell can be updated alternately, and only one operation on every edge in the NAS-convolutional cell is sampled and active at every step. In this way, the inactive paths reduce the memory requirements.

Fig. 4.2(b) is an example of the finalized architecture after completing the training of the searching stage. In the NAS-convolutional cell, we retain only one predecessor for every node and one operation for every edge except for the node with the number 0. Fig. 4.2(c) is a more compact representation of Fig. 4.2(b), showing the output of every node in the NASB-convolutional cell (except for the node with the number 0) defined as the addition of the two inputs from the backbone and the NAS-convolutional cell.

Operations of a NASB-convolutional cell Taking the number of bitwise operations and binary parameters of a 3×3 binary convolution as one unit, the number of bitwise operations and binary parameters of all the operations used in the NASB strategy are unified as shown in Table 4.1. The overhead of Batch Normalization and Relu layer is not included.

The number of bitwise operations and binary parameters of the binary convolution is $NC_{out}HW \times 2C_{in}hw$ and $C_{out}C_{in}hw$, respectively, when no bias is added. Scaling the kernel size of the binary convolution by a scaling coefficient of s_k, both the number of bitwise operations and binary parameters are scaled by s_k^2. Changing the dilation rate will not increase the number of bitwise operations and binary parameters of the binary convolution when the additional cost introduced by padding is omitted. The number of bitwise

Table 4.1: The number of bitwise operations and binary parameters of the operations used in NASB. F and B refer to full precision and binary precision, respectively. Bo and Bp refer to Bitwise operations and Binary parameters, respectively.

Operations	Bo	Bp
op0 = Zero (F)	0	0
op1 = 3×3 average pooling (F)	< 1	0
op2 = 3×3 max pooling (F)	< 1	0
op3 = Identity (F)	0	0
op4 = 1×1 convolution (B)	1/9	1/9
op5 = 3×3 convolution (B)	1	1
op6 = 5×5 convolution (B)	25/9	25/9
op7 = 1×1 dilated convolution (B)	1/9	1/9
op8 = 3×3 dilated convolution (B)	1	1
op9 = 5×5 dilated convolution (B)	25/9	25/9

operations required for computing every individual output of the binary convolution is approximately $2C_{in}hw$, while the number of bitwise operations required for computing every individual output of a 3×3 max and average pooling is $8d$ and $16d$, respectively, where d is the bit-width of pooling operations and $2C_{in}hw \gg 16d$ in general. Pooling will not introduce any parameters.

Three-stage training algorithm As shown in Algorithm 3, the training algorithm of the NASB strategy consists of three stages: the searching stage, pretraining stage, and finetuning stage. The goal of the searching stage is to get an optimized binary CNN architecture, which is done by using NAS to train a binary CNN model M_s from scratch on dataset D. The pretraining stage is used to train a full precision CNN model M_p from scratch on dataset D', whose architecture is finalized from the searching stage. The finetuning stage is used to binarize the pre-trained CNN obtained from the pretraining stage and finetune it on dataset D' to get a binary CNN model M_f.

The binary CNN model finalized from the searching stage is the same as model M_f used in the finetuning stage except for some minor differences because of their different datasets. Performing the searching stage on a small dataset D rather than directly on target dataset D' can be regarded as a proxy task to find the optimized binary architecture model M_f for the finetuning stage, which can enable a large search space and significantly accelerate the computation of the NASB strategy. After binarizing the full precision CNN model M_p from the pretraining stage, we directly get the binary CNN model M_f for the finetuning stage.

4.3.3. VARIANTS OF THE NASB STRATEGY

In this section, a number of variants of NASB are presented to improve the accuracy over state-of-the-art multiple binary CNNs. Taking NASB ResNet18 as an example, there are four NASB-convolutional cells, and each of them is composed of five nodes. We retain only one predecessor for every node and one operation for every edge except for the node

Algorithm 3 Three-stage training algorithm

Input: Dataset $D = \{(X_i, Y_i)\}_{i=1}^{S}$ for the searching stage, dataset $D' = \{(X_i', Y_i')\}_{i=1}^{S}$ for the pretraining and finetuning stages.

Output: Binary CNN model M_s for the searching stage, full precision CNN model M_p for the pretraining stage, and binary CNN model M_f for the finetuning stage.

Stage 1: The searching stage

for $epoch = 1$ to L **do**
  for $batch = 1$ to T **do**
    Randomly sample a mini-batch validation data from D, freeze the model weights of model M_s, and update its architecture parameters.
    Randomly sample a mini-batch training data from D, freeze the architecture parameters of model M_s, and update its model weights.
  end for
end for

Stage 2: The pretraining stage

for $epoch = 1$ to L **do**
  for $batch = 1$ to T **do**
    Randomly sample a mini-batch training data from D' and update the weights of model M_p.
  end for
end for

Stage 3: The finetuning stage

for $epoch = 1$ to L **do**
  for $batch = 1$ to T **do**
    Randomly sample a mini-batch training data from D' and update the weights of model M_f.
  end for
end for

with the number 0. By changing the number of NASB-convolutional cells and operations for every node, different variants of the NASB strategy are explored.

The NASBV1 strategy enlarges the search space of a NASB-convolutional cell. In NASBV1 ResNet18, there are two NASB-convolutional cells, and each of them is composed of nine nodes. In NASBV2 ResNet18, we adapt the method in [83] to retain four operations instead of one operation for the output node of the NASB-convolutional cell. In NASBV3 ResNet18, all the NASB-convolutional cells are copied once to get two binary branches. The two branches can be parallelized thoroughly except that we merge the information of the two branches at the end of every block using an addition operation as in [69]. All the NASB-convolutional cells are different from each other, which can explore the optimized binary architecture for every NASB-convolutional cell. In NASBV4 ResNet18, we retain four operations (except for identity) instead of one operation for every node of the NASB-convolutional cell. In NASBV5 ResNet18, we retain eight operations for the output node and six operations for the other nodes of the NASB-convolutional cell. Fig. 4.2(a) is the connections of a NASB-convolutional cell at the searching stages for the NASB strategy and the NASBV5 strategy. Fig. 4.4 is the derived architecture of the NASBV5 strategy after the searching stage.

Table 4.2: Accuracy of NASB ResNet18 variants

Variants	Top-1	Top-5
NASB ResNet18	60.5%	82.2%
NASBV1 ResNet18	60.3%	82.3%
NASBV2 ResNet18	61.1%	82.7%
NASBV3 ResNet18	62.8%	84.1%
NASBV4 ResNet18	65.3%	85.9%
NASBV5 ResNet18	66.6%	87.0%

Table 4.3: Comparisons of ResNet18 with multiple binary methods.

Model	Top-1	Top-5
Full precision	69.7%	89.4%
ABC-Net ($M = 5$, $N = 5$)	65.0%	85.9%
Group-Net (4 bases)	64.2%	85.6%
Group-Net** (4 bases)	66.3%	86.6%
NASBV4	65.3%	85.9%
NASBV5	66.6%	87.0%

4.4. EXPERIMENTAL RESULTS ON IMAGENET DATASET

We applied our proposed NASB strategy for the binarization of ResNet [22], trained and evaluated on the ILSVRC2012 classification dataset [56]. The ILSVRC2012 classification dataset is one of the most challenging image classification datasets and a 1000-category dataset with over 1.2 million images in the training data and 50 thousand images in the validation data. ResNet is one of the most popular and advanced CNNs.

4.4.1. IMPLEMENTATION DETAILS

During the searching stage, we train model M_s on CIFAR-10. Half of the CIFAR-10 training data is used as a validation set. The Relu layer is not added in the searching stage. We train model M_s for 100 epochs with batch size 64. We use momentum SGD and Adam to optimize the model weights and architecture parameters. The experiments are performed on one GPU. In NASB ResNet18 and NASB ResNet34, all NASB-convolutional cells adopt four nodes and they use three nodes for NASB ResNet50. Due to memory limitations, we remove convolutions and dilated convolutions with kernel size three and five for NASB ResNet50 during this stage.

During the pretraining stage, we train model M_p obtained from the last searching stage on the ILSVRC2012 classification dataset. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted. We do not apply any more sophisticated data augmentation to the training data. We use standard single-crop testing for evaluation. We insert the Relu layer and use the layer order as Conv→Relu→BN, and the $tanh$ function is applied to activation after the Batch Normalization layer.

Table 4.4: Comparisons with single binary CNNs

Model		Full	BNN	XNOR	Bi-Real	NASB
ResNet18	Top-1	69.7%	42.2%	51.2%	56.4%	60.5%
	Top-5	89.4%	67.1%	73.2%	79.5%	82.2%
ResNet34	Top-1	73.2%	–	–	62.2%	64.0%
	Top-5	91.4%	–	–	83.9%	84.7%
ResNet50	Top-1	76.0%	–	–	62.6%	65.7%
	Top-5	92.9%	–	–	83.9%	85.8%

Table 4.5: Comparisons of ResNet18 with fixed-point quantization methods.

Model	W	A	Top-1	Top-5
Full precision	32	32	69.7%	89.4%
Dorefa-Net	2	2	62.6%	84.4%
SYQ	1	8	62.9%	84.6%
Lq-Net	2	2	64.9%	85.9%
NASBV4	1	1	65.3%	85.9%

During the finetuning stage, we binarize and train the pre-trained model M_p from the pre-training stage into model M_f. The weights and activations are binarized using the method described in Section 4.3.1. We keep 1×1 convolution to full-precision in this stage. We adopt Adam as the optimizer and set weight decay to 0 since the binarization can be recognized as a kind of regularization.

4.4.2. EXPERIMENTAL RESULTS OF NASB VARIANTS

The accuracy of different variants is compared in Table 4.2. The accuracy of NASBV1 ResNet18 is almost the same as that of NASB ResNet18. We conjecture that 28/36 of the total edges in NASBV1 ResNet18 are removed rather than 6/10 of the total edges in NASB ResNet18, which will change model M_s too much and remedy the benefits of a larger search space. For other variants of the NASB strategy, we observe the increased operations of NASB-convolutional cell results in a Top-1 accuracy improvement by up to 6.0%. It is expected that with more operations retained, NASB variants can achieve higher accuracy.

We present the finalized architecture of four NASB-convolutional cells in NASBV5 ResNet18, as shown in Fig. 4.4, which is derived from Fig. 4.2(a) after the searching stage. In this figure, we retain eight operations for the output node and six operations for the other nodes of every NASB-convolutional cell. In addition, we have the following observations. 1). Most operations in the discovered architecture are the max pooling layers. The reason that we suspect is the max pooling layer can preserve the features with the highest response. 2). 1×1 convolution is not chosen, which conforms to the experience of manually designing the neural network architecture. 3). The discovered architecture has a larger receptive field since it prefers 3×3 dilated convolution than 3×3 convolutions.

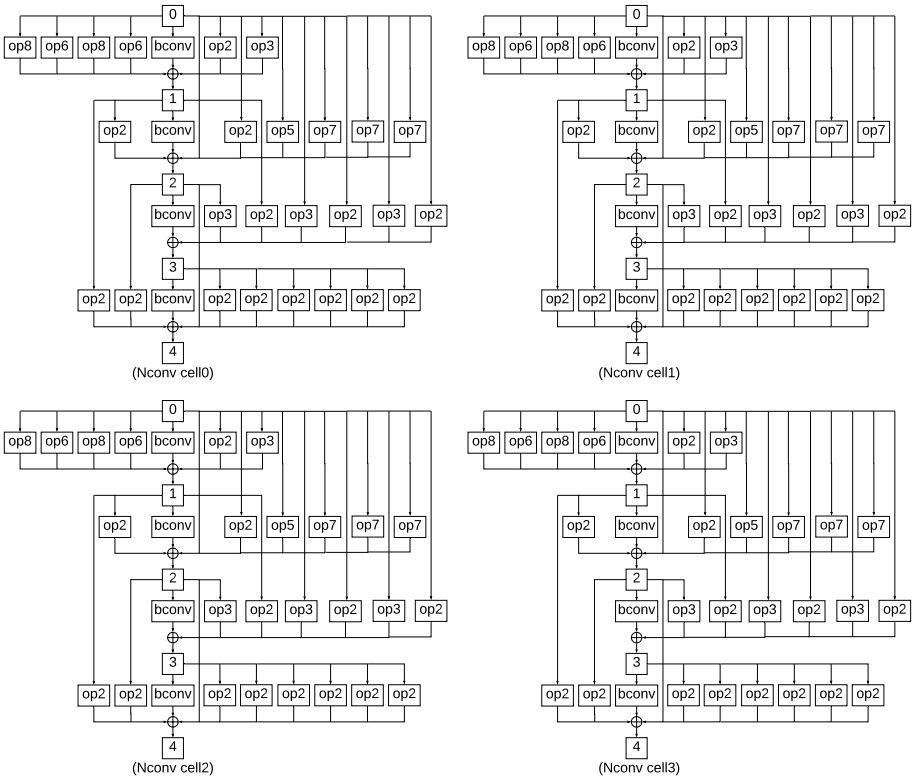


Figure 4.4: Architecture of NASB-convolutional cells in NASBV5 ResNet18. Nconv cell refers to NASB-convolutional cell. conv and bconv refer to full precision and binary convolutional layer, respectively.

4.4.3. COMPARISONS WITH THE STATE-OF-THE-ART QUANTIZED CNNS

As shown in Table 4.4, Table 4.3, and Table 4.5, we compare our NASB strategy with single binary CNNs, multiple parallel binary CNNs, and fixed-point CNNs using different quantization methods, respectively. All the comparison results are directly cited from the corresponding papers.

As shown in Table 4.4, Bi-Real Net [38] is the state-of-the-art single binary CNNs. Compared with Bi-Real ResNet with varying layers from 18 to 50, our proposed NASB ResNet shows consistent accuracy improvement by 4.1%, 1.8%, and 3.1% Top-1 accuracy, respectively.

As shown in Table 4.3, we compare our NASB strategy with ABC-Net and Group-Net, which is a multiple binary CNN and can be implemented in a parallel way. Both NASBV4 and NASBV5 achieve higher accuray than ABC-Net. NASBV4 and NASBV5 show better accuracy performance than Group-Net and Group-Net** by 1.1% and 0.3%, respectively.

Table 4.6: Memory usage and Flops calculation of Bi-Real Net, Group-Net, NASB Net, and full precision models

Model	Memory usage	Memory saving	Flops	Speedup
Bi-Real ResNet18	33.6Mbit	11.14 ×	1.63×10^8	11.06 ×
NASB ResNet18	33.8Mbit	11.07 ×	1.71×10^8	10.60 ×
ResNet18	374.1Mbit	–	1.81×10^9	–
Bi-Real ResNet34	43.7Mbit	15.97 ×	1.93×10^8	18.99 ×
NASB ResNet34	44.0Mbit	15.86 ×	2.01×10^8	18.26 ×
ResNet34	697.3Mbit	–	3.66×10^9	–
Bi-Real ResNet50	176.8Mbit	4.62 ×	5.45×10^8	7.08 ×
NASB ResNet50	178.1Mbit	4.60 ×	6.18×10^8	6.26 ×
ResNet50	817.8Mbit	–	3.86×10^9	–
ABC-Net ($M = 5$, $N = 5$) ResNet18	72.3Mbit	5.17 ×	6.74×10^8	2.70 ×
Group-Net (4 bases) ResNet18	62.1Mbit	6.03 ×	2.62×10^8	6.90 ×
Group-Net** (4 bases) ResNet18	83.9Mbit	4.46 ×	3.38×10^8	5.35 ×
NASBV4 ResNet18	70.7Mbit	5.30 ×	2.81×10^8	6.45 ×
NASBV5 ResNet18	88.3Mbit	4.24 ×	3.52×10^8	5.15 ×
ResNet18	374.1Mbit	–	1.81×10^9	–

Table 4.5 shows Lq-Net is the current best-performing fixed-point method. Multiple binary CNNs with K binary branches are preferable to fixed-point CNNs with $\sqrt{K}$ bit-width considering the computational complexity and memory bandwidth [69]. Thus, NASBV4 with four operations retained per node has less overhead while still achieving better accuracy.

4.4.4. COMPUTATIONAL COMPLEXITY ANALYSIS

To analyze the computational complexity of our proposed NASB strategy, we compare with Bi-Real Net, Group-Net, and full precision models in terms of memory usage and computation speedup as shown in Table 4.6.

The memory usage is computed as the summation of the number of real-valued parameters times thirty two bit and the number of binary parameters times one bit. We use Flops to measure the computation and assume that bitwise XNOR and popcount operations can be calculated as 64-way parallel operations on current CPUs. Thus, the Flops is calculated as the sum of the number of real-valued operations plus $1/64$ of the number of bitwise operations. Following the suggestion from [37, 38, 69], we keep the first convolutional layer, the last fully connected layer, and the downsampling layer as full precision.

Bi-Real Net [38] can be seen as a suboptimal binary CNN architecture of our NASB Net, where one edge connected to its last node is retained for every node and one identity operation remains for every edge. The finalized NAS-convolutional cells in NASB ResNet18 includes twelve max pooling and four identity operations, and they are composed of 20 max pooling and twelve identity operations in NASB ResNet34. In NASB Res50, the NAS-convolutional cells consist of 41 max pooling, six identity, and one 1x1 dilated convolution operations. Compared to Bi-Real Net, the increased computational complexity is mainly due to max pooling. The Flops or the number of bitwise operations of a 3x3 max pooling

is less than that of a 3x3 convolution, and the additional number of trainable parameters introduced by Batch Normalization of max pooling operation is $2C_{out}$.

As shown in Table 4.6, both the additional memory usage and Flops of NASB ResNet of varying depths are negligible compared to Bi-Real Net. ABC-Net requires significantly more Flops than Group-Net and NASB variants. The increased memory usage and Flops of NASBV5 and NASBV4 ResNet18 are insignificant compared to Group-Net** and Group-Net.

4.5. CONCLUSION

In this chapter, we proposed NASB, a strategy to find an accurate architecture for binary CNNs. Specifically, the NASB strategy uses the NAS technique to identify an optimized architecture in a large search space, which is suitable for binarizing CNNs. We use the ImageNet classification dataset to prove the effectiveness of our proposed approach. With insignificant overhead increases, the NASB strategy and its variants achieve up to 4.0% and 1.0% Top-1 accuracy improvement compared with the state-of-the-art single and multiple binary CNNs, respectively, providing a better trade-off between accuracy and efficiency. It is worth to clarify that we can easily extend our proposed NASB strategy to fixed-point quantized convolutional neural networks and other models for computer vision tasks beyond image classification, which can be explored further.

5

UNIFIED EFFECTIVE DEPTH REDUCTION TECHNIQUES

To improve gradient backpropagation, current network engineering for BCNNs relies on enhancing the shortcut Effective Depth Reduction (EDR) technique[1], i.e., increasing the number of shortcuts, to improve the accuracy of BCNNs. Specifically, Bi-Real ResNet and BinaryDenseNet introduce additional shortcuts in addition to the shortcuts already present in their full precision counterparts. However, there is a slight accuracy drop (rather than an increase) when we keep increasing the number of shortcuts further for Bi-Real ResNet and BinaryDenseNet. Rather than relying solely on enhancing the shortcut EDR technique, we take a view of unifying multiple EDR techniques to bring their advantages together into one model to make gradient backpropagate more easily. In particular, we propose to unify both the shortcut and fractal architecture EDR techniques into one model and design two unified architectures for BCNNs: 1. Unified Architectures for binary ResNet (UA-ResNet), and 2. Unified Architectures for binary DenseNet (UA-DenseNet). Gradient path analysis demonstrates that our unified architectures have better gradient paths than Bi-Real ResNet and BinaryDenseNet to make the gradient backpropagate more easily, which cannot be achieved by relying solely on enhancing the shortcut EDR technique. Results show that our proposed unified architectures achieve better accuracy compared with Bi-Real ResNet and BinaryDenseNet. Specifically, the Top-1 accuracy of our proposed UA-ResNet37(41) and UA-DenseNet51(53) on ImageNet outperforms Bi-Real ResNet18(64) and BinaryDenseNet51(32) by 3.29% *and* 1.41%*, respectively, with almost the same computational complexity.*

The content of this chapter is based on [84].

[1] In our chapter, "Effective Depth Reduction (EDR)" refers to a technique of reducing the effective depth of DCNNs to improve gradient backpropagation, including shortcut [22, 23], fractal architecture [85], deep supervision [86], and student-teacher paradigm [87] (In the student-teacher paradigm, there is an interplay between networks of different depths). The EDR techniques share a key characteristic: **large nominal network depth**, but **effectively shorter paths** for gradient propagation during training.

5.1. INTRODUCTION

Convolutional Neural Networks (CNNs) have become the paradigm of choice for visual recognition and have made considerable breakthroughs in a wide range of visual tasks [88], such as image recognition [22, 89], object detection [90], and segmentation [91]. To practically deploy CNNs in the field, their efficiency has become a key differentiator, especially when targeting resource-limited embedded platforms.

Convolutional Neural Networks (CNNs) have become the paradigm of choice for visual recognition [22, 88–91]. A significant amount research has been dedicated to increasing the efficiency of CNNs, including pruning [92, 93], quantization [94], knowledge distillation [87, 95], and efficient network design [75]. In low bit-width quantization, fixed-point integers are used instead of floating-point numbers [6], where binarization is an extreme case of quantization. Binarization [37] is the most efficient among the different bit-widths quantization methods. However, it results in a large accuracy degradation.

The current methods to improve the accuracy of binarization can be divided into two categories [44]: value approximation and structure approximation. In value approximation, we preserve the topology of the full-precision CNNs during the binarization and seek a better local minimum for binarized weights/activations by either minimizing the quantization error [45, 96–99], improving the loss function of the network [29, 67, 100, 101], or improving the quantization functions [35, 38, 102–104]. In structure approximation [38, 44, 60, 105], the architecture of the binary CNNs is redesigned to approximate the original full-precision CNNs. The structure approximation focuses on the architecture design principles for efficient and accurate BCNNs, which is complementary to the value approximation. Regarding structure approximation, Bi-Real ResNet [38] and BinaryDenseNet [105] show significant accuracy improvement without increasing the number of parameters, which indicates that adopting more shortcuts can help the training of BCNNs.

Network engineering for BCNNs [38, 105] can get inspiration from network engineering done for full-precision DCNNs [22, 23, 85–87] since they both try to improve gradient backpropagation to ease training difficulties. To overcome the training difficulty caused by the large depth of full-precision DCNNs, research has shown significant improvements using Effective Depth Reduction (EDR) techniques to improve gradient backpropagation, including shortcut [22, 23], fractal architecture [85], deep supervision [86], and student-teacher paradigm [87]. Besides, BCNNs do not only suffer from the training difficulty caused by their depth, but also the training difficulty caused by their binarization. Thus, we need to further enhance the EDR techniques already used for full-precision DCNNs to improve gradient backpropagation, that focus mainly on shortcut EDR techniques, such as those used for Bi-Real ResNet [38] and BinaryDenseNet [105].

However, relying solely on enhancing the shortcut EDR technique has its limits since there is no accuracy improvement with further increasing the number of shortcuts for Bi-Real ResNet and BinaryDenseNet. Rather than relying solely on enhancing the shortcut EDR technique, we look into the direction to unify multiple EDR techniques into one model to make gradient backpropagate more easily and design unified architectures, that can benefit from both the shortcut and fractal architecture EDR techniques. By unifying the residual connection and fractal architecture EDR techniques into one model, we design Unified Architectures for binary ResNet (UA-ResNet). Also, we design Unified Architectures for binary DenseNet (UA-DenseNet) by unifying the dense connection, residual connection, and

fractal architecture EDR techniques together. Gradient path analysis demonstrates that our unified architectures have better gradient paths than Bi-Real ResNet and BinaryDenseNet to make the gradient backpropagate more easily, which cannot be achieved by relying solely on enhancing the shortcut EDR technique.

To our best knowledge, there has been no research to show and discuss the feasibility of unifying multiple EDR techniques to make gradient backpropagate more easily to solve the training difficulties for BCNNs and full-precision DCNNs. Moreover, there is no previous work on network engineering to unify the shortcut and fractal architecture EDR techniques into one model, since these techniques have only been used independently in ResNet, DenseNet, FractalNet, and their variants. By unifying different EDR techniques, more advanced unified architectures are expected to be explored in the future. The contribution of this chapter is summarized as follows.

- We are the first work to propose the neural network architecture design principle of unifying multiple EDR techniques, rather than relying solely on enhancing the shortcut EDR technique, to make the gradient backpropagate more easily. Based on the experiments of Bi-Real ResNet and BinaryDenseNet, we identify the limitation of relying solely on enhancing the shortcut EDR technique and open the door to unifying multiple EDR techniques to bring their advantages into one model.
- We are the first network engineering effort to unify residual connection, dense connection, and fractal architecture EDR techniques together. We design two unified architectures for BCNNs: 1. UA-ResNet unifying the residual connection and fractal architecture EDR techniques, and 2. UA-DenseNet unifying the dense connection, residual connection, and fractal architecture EDR techniques. Gradient path analysis demonstrates that our unified architectures have better gradient paths than Bi-Real ResNet and BinaryDenseNet to make the gradient backpropagate more easily, which cannot be achieved by relying solely on enhancing the shortcut EDR technique.
- Under a given computational complexity budget, our unified architectures on classification tasks improve the accuracy of state-of-the-art Bi-Real ResNet and BinaryDenseNet.

5.2. RELATED WORK

In this section, we review and compare the recent work of compact architecture design and quantized CNNs.

5.2.1. COMPACT ARCHITECTURE DESIGN

Efficient architecture design has attracted lots of attention from researchers. 3×3 convolution has been replaced with 1×1 convolution in GoogLeNet [71] and SqueezeNet [61] to reduce the computational complexity. Group convolution [106], depthwise separable convolution [75], shuffle operation [76], and shift operation [107] have been shown to reduce the computational complexity of traditional convolution. Instead of relying on human experts, neural architecture search techniques [81, 82] can automatically provide optimized platform-specific architectures, achieving state-of-the-art efficiency.

Table 5.1: Binary ResNet and DenseNet variants on CIFAR-100. There are two blocks in this Table. **First block:** ResNet variants adopting the residual connection EDR technique. **Second block:** DenseNet variants adopting the dense connection EDR technique.

Model	Bit-width	Top-1	Top-5	Training difficulty	Shortcuts
ResNet18(64)	$b=32$	23.54%	6.55%	$D_{depth=18}$	5
Bi-Real ResNet18(64)	$b=32$	23.01%	6.24%	$D_{depth=18}$	13
Enhanced Bi-Real ResNet18(64)	$b=32$	23.07%	6.20%	$D_{depth=18}$	18
ResNet18(64)	$b=2$	29.59%	9.17%	$D_{depth=18}$, $D_{width=2}$	5
Bi-Real ResNet18(64)	$b=2$	26.71%	7.46%	$D_{depth=18}$, $D_{width=2}$	13
Enhanced Bi-Real ResNet18(64)	$b=2$	27.26%	7.58%	$D_{depth=18}$, $D_{width=2}$	18
ResNet18(64)	$b=1$	32.57%	10.18%	$D_{depth=18}$, $D_{width=1}$	5
Bi-Real ResNet18(64)	$b=1$	28.48%	8.65%	$D_{depth=18}$, $D_{width=1}$	13
Enhanced Bi-Real ResNet18(64)	$b=1$	29.94%	8.88%	$D_{depth=18}$, $D_{width=1}$	18
BinaryDenseNet51(32)	$b=32$	25.41%	7.30%	$D_{depth=51}$	46
Enhanced BinaryDenseNet51(32)	$b=32$	25.44%	7.27%	$D_{depth=97}$	92
BinaryDenseNet51(32)	$b=2$	26.61%	7.57%	$D_{depth=51}$, $D_{width=2}$	46
Enhanced BinaryDenseNet51(32)	$b=2$	26.71%	7.60%	$D_{depth=97}$, $D_{width=2}$	92
BinaryDenseNet51(32)	$b=1$	27.16%	7.77%	$D_{depth=51}$, $D_{width=1}$	46
Enhanced BinaryDenseNet51(32)	$b=1$	27.35%	7.88%	$D_{depth=97}$, $D_{width=1}$	92

5.2.2. QUANTIZED CONVOLUTIONAL NEURAL NETWORKS

Low bit-width quantization has been extensively explored in recent work, including reducing the gradient error [108], improving the loss function of the network [51, 109], and minimizing the quantization error [50]. Moreover, mixed-precision quantized neural networks are developed to improve the performance further for low bit-width quantized neural networks. Using neural architecture search, mixed-precision neural networks [110–112] are developed to find the optimal bit-width (i.e., precision) for weights and activations of each layer efficiently.

Improving network loss function [100], minimizing the quantization error [97], and reducing the gradient error [104] have been studied to provide a better value approximation for BCNNs. Channel-wise Interaction based Binary Convolutional Neural Network (CI-BCNN) [100] uses a reinforcement learning model to mine the channel-wise interactions and impose channel-wise priors to alleviate the inconsistency of signs in binary feature maps. [97] obtains significant accuracy gains by minimizing the discrepancy between the output of the binary and the corresponding real-valued convolution. Information Retention Network (IR-Net) [104] has been proposed to retain the information that consists of the forward activations and backward gradients. Regarding structure approximation, [38, 105] enhances the shortcut EDR technique, i.e., adopting more shortcuts to help the training of BCNNs.

5.3. UNIFIED ARCHITECTURES

In this section, we identify the limitation of relying solely on enhancing the shortcut EDR technique. Rather than relying solely on enhancing the shortcut EDR technique, we propose to unify the shortcut and fractal architecture EDR techniques to make the gradient backpropagate more easily and introduce our unified architectures for BCNNs. Besides, gradient path analysis demonstrates that our unified architectures have better gradient paths than Bi-Real ResNet and BinaryDenseNet to make the gradient backpropagate more easily, which cannot be achieved by relying solely on enhancing the shortcut EDR technique. In addition, to ensure a fair comparison we scale the number of base channels or the growth rate of our unified architectures to have almost the same computational complexity as Bi-Real ResNet and BinaryDenseNet.

5.3.1. LIMITATION OF SHORTCUT EDR TECHNIQUE

For full-precision DCNNs, there is a training difficulty caused by their large depth (D_{depth}). For BCNNs, there is a training difficulty caused by the large depth (D_{depth}) and a training difficulty caused by the binarization (D_{width}). Comparing Bi-Real ResNet to ResNet in Table 5.1, we have two observations. The first observation is that the Top-1 accuracy of ResNet18 with different bit-widths will improve with increasing the number of shortcuts, by 0.53% for 32 bit-width, 2.88% for 2 bit-width, and 4.09% for 1 bit-width. The second observation is that the accuracy benefit of increasing the number of shortcuts, will be more obvious when the training difficulty increases, i.e., 4.09% > 2.88% > 0.53% with $D_{width=1} > D_{width=2} > D_{width=32}$.

More importantly, we identify one limitation when comparing Enhanced Bi-Real ResNet and Enhanced BinaryDenseNet to Bi-Real ResNet and BinaryDenseNet. Enhanced Bi-Real ResNet is obtained by adding more shortcuts to Bi-Real ResNet using the method in [113], and Enhanced BinaryDenseNet is obtained by adding more shortcuts to BinaryDenseNet following the method in [105]. Therefore, Enhanced BinaryDenseNet51(32) is the same as BinaryDenseNet97(16). The Top-1 accuracy of Bi-Real ResNet will degrade with increasing the number of shortcuts, by 0.06% for 32 bit-width, 0.55% for 2 bit-width, and 1.46% for 1 bit-width. Similarly, the Top-1 accuracy of Enhanced BinaryDenseNet51 is worse than that of BinaryDenseNet51 by 0.03% for 32 bit-width, 0.10% for 2 bit-width, and 0.19% for 1 bit-width. There is a limitation of relying solely on enhancing the shortcut EDR technique since there is a slight accuracy drop when we keep increasing the number of shortcuts for Bi-Real ResNet and BinaryDenseNet. Rather than relying solely on enhancing the shortcut EDR technique, i.e., increasing the number of shortcuts, the key to our proposal is to unify multiple EDR techniques into one model to make gradient backpropagate more easily and design unified architectures.

5.3.2. UNIFIED ARCHITECTURES FOR BINARY RESNET

The block of UA-ResNet unifies the residual connection and fractal architecture EDR techniques in one model. In Bi-Real ResNet, enhancing the residual connection EDR technique is achieved by introducing more residual connections, which uses the summation as the operation of feature aggregation. Rather than relying solely on enhancing the residual connection EDR technique, the block of UA-ResNet brings the advantages of residual connection

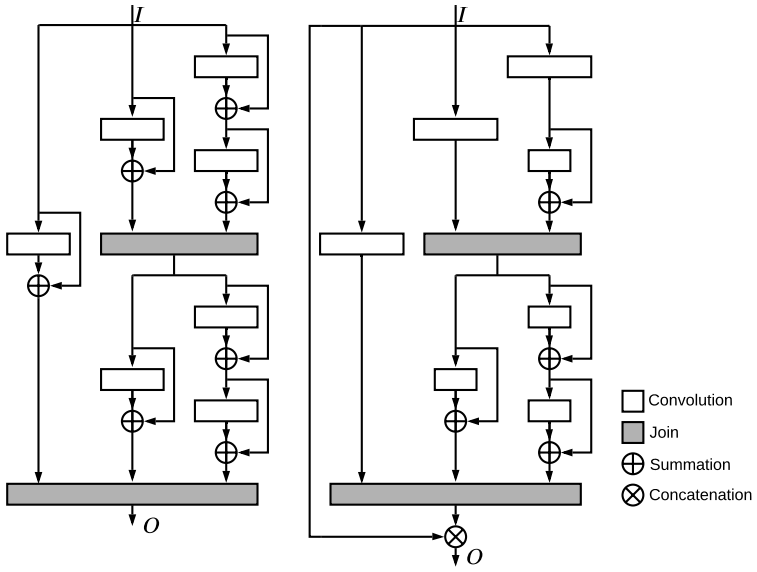


Figure 5.1: The diagrams of the blocks of unified architectures. **Left:** The block description of binary UA-ResNet. **Right:** The block description of binary UA-DenseNet.

and fractal architecture EDR techniques into one unified model to resolve the training difficulty of BCNNs. In UA-ResNet, a block (c3d4) is shown to the left of Figure 5.1, where the number of columns and the longest depth between the input and output of the block is $c = 3$ and $d = 4$, respectively. Comparing this block to a block of FractalNet, we add a residual connection to every convolutional layer. We define the base case and the iteration rule for the block of UA-ResNet. Specifically, the base case is as follows where $\oplus$ refers to feature aggregation of summation.

$$F_1^R(I) = \text{Conv}(I) \oplus I \tag{5.1}$$

Besides, we have successive fractals recursively as follows where $\circledast$ denotes composition operation and $\odot$ represents the join layer.

$$F_{C+1}^R(I) = (F_C^R \circledast F_C^R(I)) \odot (\text{Conv}(I) \oplus I) \tag{5.2}$$

5.3.3. UNIFIED ARCHITECTURES FOR BINARY DENSENET

The block of UA-DenseNet unifies the residual connection, dense connection, as well as fractal architecture EDR techniques in one model. Since the feature maps of all preceding convolutional blocks in DenseNet will be concatenated and reused, the fractal architecture EDR technique is applied to produce new feature maps. Moreover, all the convolutional layers, where the number of input channels is the same as the number of output channels, adopt the residual connection EDR technique, so the shortcut EDR technique is used as often as possible.

Table 5.2: Details of gradient paths in binary model blocks. $(\cdot|\cdot)$ refers to the number of binary parameters on a gradient path and the number of layers for the gradient path length. For example, $2 \times P^{BR}|2$ indicates that the number of binary parameters on the gradient path GP_1 in the Bi-Real ResNet block is $2 \times P^{BR}$ and the length of the gradient path is two layers. The details of gradient path GP_5 in binary UA-ResNet block is 0|0, which is not listed in the table.

Block	GP_1	GP_2	GP_3	GP_4
Bi-Real ResNet	$2 \times P^{BR}\|2$	$P^{BR}\|1$	$P^{BR}\|1$	0|0
UA-ResNet	$2 \times P^{UR}\|2$	$P^{UR}\|1$	$P^{UR}\|1$	$P^{UR}\|1$
BinaryDenseNet	$P_1^{BD} + P_2^{BD}\|2$	$P_1^{BD}\|1$	$P_2^{BD}\|1$	0|0
UA-DenseNet	$P_1^{UD} + P_2^{UD}\|2$	$P_1^{UD}\|1$	$P_1^{UD}\|1$	0|0

In UA-DenseNet, a block (c3d4) is shown to the right of Figure 5.1, where the backbone, i.e., the convolutional and join layers, is the same as that in a block of FractalNet. Two characteristics need to be clarified for the block of UA-DenseNet. The fractal architecture EDR technique is used to produce new feature maps, that will concatenate with the feature maps of all preceding convolutional layers. All the convolution layers, where the number of input channels is the same as the number of output channels, are associated with the residual connections, so the shortcut EDR technique is used as often as possible. ${F_1}^D$ is the base case for the block of UA-DenseNet, where only one convolutional layer is used. ${F_1}^D$ is calculated as follows.

$$ {F_1}^D(I) = \text{Conv}(I) \tag{5.3} $$

We define the truncated fractal ${F_2}^D$ as follows where three convolutional layers are used and F_1^R is the truncated fractal in the block of UA-ResNet.

$$ {F_2}^D = {F_1}^D \circledast F_1^R(I) \odot \text{Conv}(I) \tag{5.4} $$

We define the truncated fractal ${F_3}^D$ as follows where seven convolutional layers are used.

$$ {F_3}^D = {F_2}^D \circledast F_2^R(I) \odot \text{Conv}(I) \tag{5.5} $$

At the end of the block of UA-DenseNet, we use feature aggregation of concatenation as follows where $\otimes$ refers to feature aggregation of concatenation.

$$ O = {F_C}^D \otimes I \tag{5.6} $$

5.3.4. GRADIENT PATH ANALYSIS

This section demonstrates that our unified architectures have better gradient paths than Bi-Real ResNet and BinaryDenseNet to make the gradient backpropagate more easily, which cannot be achieved by relying solely on enhancing the shortcut EDR technique.

We use three metrics to evaluate the quality of gradient paths in a binary model: the total number of gradient paths, the average length of the gradient paths, and the average number

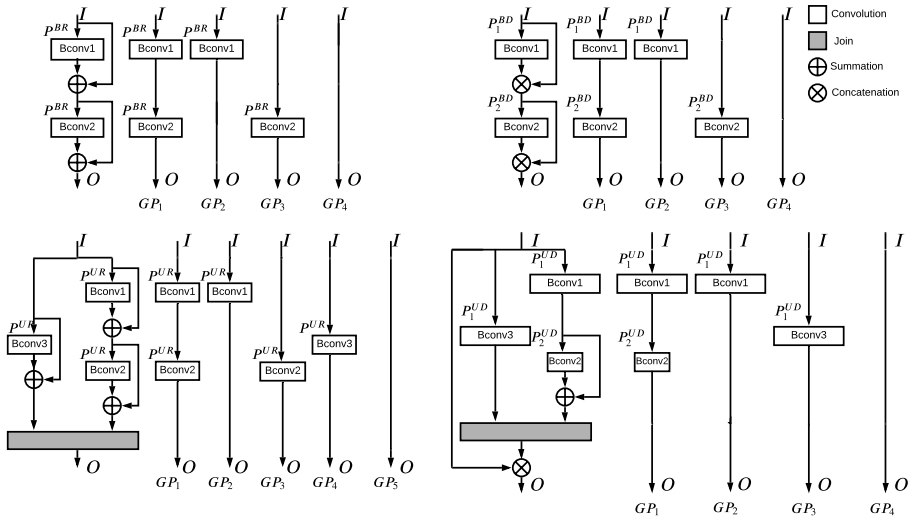


Figure 5.2: Analysis of gradient paths in binary ResNet and DenseNet variants. **Top left:** Analysis of gradient paths in a Bi-Real ResNet block. **Bottom left:** Analysis of gradient paths in a binary UA-ResNet block. **Top right:** Analysis of gradient paths in a BinaryDenseNet block. **Bottom right:** Analysis of gradient paths in a binary UA-DenseNet block. *GP* and Bconv refer to gradient path and binary convolutional layer, respectively. P^{BR} and P^{UR} represent the number of binary parameters of a convolutional layer in Bi-Real ResNet and binary UA-ResNet, respectively. P^{BD} and P^{UD} represent the number of binary parameters of a convolutional layer in BinaryDenseNet and binary UA-DenseNet, respectively.

of binary parameters on the gradient paths of the same length. Gradient backpropagation of full-precision layers is much easier than that of binary convolutional layers. Thus, we consider binary convolutional layers and ignore full-precision layers to compute the three metrics. We present the gradient paths in the blocks of the binary ResNet and DenseNet variants in Figure 5.2. We take binary model blocks with a depth of two layers as an example to illustrate the analysis of gradient paths, which will work for other binary neural network architecture configurations. We show the details of the gradient paths in Table 5.2.

Why do gradient paths in binary UA-ResNet blocks improve? The binary UA-ResNet block has better gradient paths than the Bi-Real ResNet block for two reasons. 1. Binary UA-ResNet block has more gradient paths. 2. A smaller average number of binary parameters on the gradient paths of the same length in the binary UA-ResNet block needs to be updated.

The gradient path analysis in Bi-Real ResNet and binary UA-ResNet blocks is summarized as follows. 1. The total number of gradient paths in Bi-Real ResNet and Binary UA-ResNet blocks is 4 and 5, respectively. 2. The average length of the gradient paths in Bi-Real ResNet and binary UA-ResNet blocks is one layer. 3. The average number of

binary parameters on the gradient paths of one layer in Bi-Real ResNet and binary UA-ResNet blocks is P^{BR} and P^{UR}, respectively. Considering the gradient paths of two layers, the average number of binary parameters in Bi-Real ResNet and binary UA-ResNet blocks is $2 \times P^{BR}$ and $2 \times P^{UR}$, respectively. To ensure a fair comparison, we set the numbers of binary parameters of model blocks to be roughly the same, i.e., $3 \times P^{UR} \approx 2 \times P^{BR}$. Thus, $P^{BR} > P^{UR}$.

Why do gradient paths in binary UA-DenseNet blocks improve? Compared with the BinaryDenseNet block, the UA-DenseNet block has a smaller average number of binary parameters on the gradient paths of the same length to be updated, which results in improved gradient paths.

The gradient path analysis in BinaryDenseNet and binary UA-DenseNet blocks is summarized as follows. 1. The total number of gradient paths in BinaryDenseNet and binary UA-DenseNet blocks is 4. 2. The average length of gradient paths in BinaryDenseNet and binary UA-DenseNet blocks is one layer. 3. Regarding the gradient paths of one layer, the average number of binary parameters in BinaryDenseNet and binary UA-DenseNet blocks is $\frac{P_1^{BD}+P_2^{BD}}{2}$ and P_1^{UD}, respectively. In terms of the gradient paths of two layers, the average number of binary parameters in BinaryDenseNet and binary UA-DenseNet blocks is $P_1^{BD}+P_2^{BD}$ and $P_1^{UD}+P_2^{UD}$, respectively. To ensure a fair comparison, we set the numbers of binary parameters of model blocks to be roughly the same, i.e., $P_1^{BD}+P_2^{BD} \approx 2\times P_1^{UD}+P_2^{UD}$. Therefore, $\frac{P_1^{BD}+P_2^{BD}}{2} > P_1^{UD}$ and $P_1^{BD}+P_2^{BD} > P_1^{UD}+P_2^{UD}$.

Why does it not work to enhance the shortcut EDR technique further for Bi-Real ResNet and BinaryDenseNet? The total number of gradient paths, the average length of the gradient paths, and the average number of binary parameters on the gradient paths of the same length in the Enhanced Bi-Real ResNet block are the same as those in the Bi-Real ResNet block. Thus, increasing the number of residual connections for Bi-Real ResNet cannot improve the gradient paths. The Enhanced BinaryDenseNet block has more gradient paths, but a larger average length of the gradient paths than the BinaryDenseNet block. Besides, the average number of binary parameters on the gradient paths of three layers and four layers in the Enhanced BinaryDenseNet block is much larger than that in the BinaryDenseNet block although there is a smaller average number of binary parameters on the gradient paths of one layer and two layers in the Enhanced BinaryDenseNet block. Thus, increasing the dense connection further for BinaryDenseNet does not lead to better gradient paths. More information and analysis of gradient paths in Bi-Real ResNet, BinaryDenseNet, and their enhanced versions can be found in the appendix.

We present the gradient paths in the blocks of the binary ResNet and DenseNet variants in Figure 5.3. We take binary model blocks with a depth of two layers as an example to illustrate the analysis of gradient paths, which will work for other binary neural network architecture configurations. We show the details of the gradient paths in Table 5.3.

Comparison between Bi-Real ResNet and Enhanced Bi-Real ResNet. The gradient path analysis in Bi-Real ResNet and Enhanced Bi-Real ResNet blocks is summarized as follows. 1. The total number of gradient paths in Bi-Real ResNet and Enhanced Bi-Real

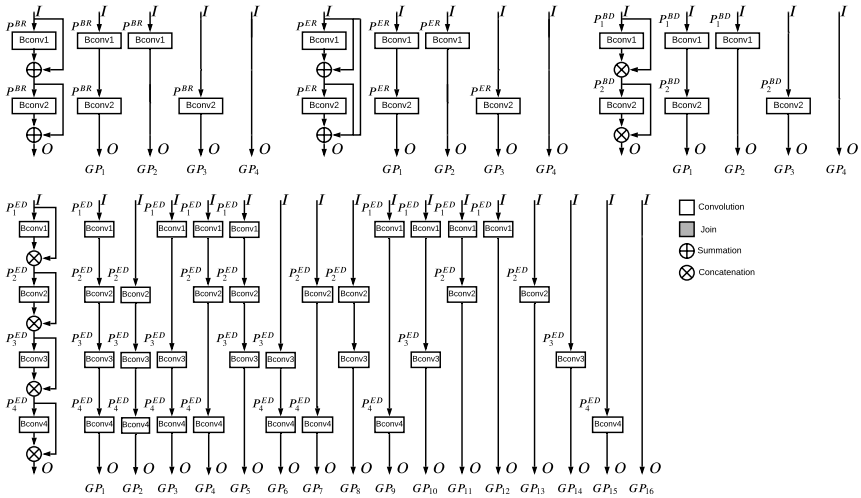


Figure 5.3: Analysis of gradient paths in binary ResNet and DenseNet variants. **Top left:** Analysis of gradient paths in Bi-Real ResNet block. **Top middle:** Analysis of gradient paths in Enhanced Bi-Real ResNet block. **Top right:** Analysis of gradient paths in BinaryDenseNet block. **Bottom:** Analysis of gradient paths in Enhanced BinaryDenseNet block. GP and Bconv refer to gradient path and binary convolutional layer, respectively. P^{BR} and P^{ER} represent the number of binary parameters of a convolutional layer in Bi-Real ResNet and Enhanced Bi-Real ResNet, respectively. P^{BD} and P^{ED} represent the number of binary parameters of a convolutional layer in BinaryDenseNet and Enhanced BinaryDenseNet, respectively.

ResNet blocks is equal to 4. 2. The average length of the gradient paths in Bi-Real ResNet and Enhanced Bi-Real ResNet blocks is one layer. 3. The average number of binary parameters on the gradient paths of one layer in Bi-Real ResNet and Enhanced Bi-Real ResNet blocks is P^{BR} and P^{ER}, respectively. Considering the gradient paths of two layers, the average number of binary parameters in Bi-Real ResNet and Enhanced Bi-Real ResNet blocks is $2 \times P^{BR}$ and $2 \times P^{ER}$, respectively. To ensure a fair comparison, we set the numbers of binary parameters of model blocks to be roughly the same, i.e., $2 \times P^{UR} \approx 2 \times P^{BR}$. Thus, $P^{BR} = P^{ER}$.

Comparison between BinaryDenseNet and Enhanced BinaryDenseNet. The gradient path analysis in BinaryDenseNet and Enhanced BinaryDenseNet blocks is summarized as follows. 1. The total number of gradient paths in BinaryDenseNet and Enhanced BinaryDenseNet blocks is 4 and 16, respectively. 2. The average length of gradient paths in BinaryDenseNet and Enhanced BinaryDenseNet blocks is one layer and two layers, respectively. 3. Regarding the gradient paths of one layer, the average number of binary parameters in BinaryDenseNet and Enhanced BinaryDenseNet blocks is $\frac{P_1^{BD}+P_2^{BD}}{2}$ and

Table 5.3: Details of gradient paths in binary model blocks. $(\cdot|\cdot)$ refers to the number of binary parameters on a gradient path and the number of layers for the gradient path length. For example, $2 \times P^{BR}|2$ indicates that the number of binary parameters on the gradient path GP_1 in the Bi-Real ResNet block is $2 \times P^{BR}$ and the length of the gradient path is two layers. BR and BD represent Bi-Real ResNet and BinaryDenseNet, respectively.

Block	GP_1	GP_2	GP_3	GP_4
BR	$2 \times P^{BR}\|2$	$P^{BR}\|1$	$P^{BR}\|1$	$0\|0$
Enhanced BR	$2 \times P^{ER}\|2$	$P^{ER}\|1$	$P^{ER}\|1$	$0\|0$
BD	$P_1^{BD} + P_2^{BD}\|2$	$P_1^{BD}\|1$	$P_2^{BD}\|1$	$0\|0$
Enhanced BD	$P_1^{ED} + P_2^{ED} + P_3^{ED} + P_4^{ED}\|4$	$P_2^{ED} + P_3^{ED} + P_4^{ED}\|3$	$P_1^{ED} + P_3^{ED} + P_4^{ED}\|3$	$P_1^{ED} + P_2^{ED} + P_4^{ED}\|3$
Block	GP_5	GP_6	GP_7	GP_8
Enhanced BD	$P_1^{ED} + P_2^{ED} + P_3^{ED}\|3$	$P_3^{ED} + P_4^{ED}\|2$	$P_2^{ED} + P_4^{ED}\|2$	$P_2^{ED} + P_3^{ED}\|2$
Block	GP_9	GP_{10}	GP_{11}	GP_{12}
Enhanced BD	$P_1^{ED} + P_4^{ED}\|2$	$P_1^{ED} + P_3^{ED}\|2$	$P_1^{ED} + P_2^{ED}\|2$	$P_1^{ED}\|1$
Block	GP_{13}	GP_{14}	GP_{15}	GP_{16}
Enhanced BD	$P_2^{ED}\|1$	$P_3^{ED}\|1$	$P_4^{ED}\|1$	$0\|0$

$\frac{P_1^{ED}+P_2^{ED}+P_3^{ED}+P_4^{ED}}{4}$, respectively. In terms of the gradient paths of two layers, the average number of binary parameters in BinaryDenseNet and Enhanced BinaryDenseNet blocks is $P_1^{BD} + P_2^{BD}$ and $\frac{P_1^{ED}+P_2^{ED}+P_3^{ED}+P_4^{ED}}{2}$, respectively. In terms of the gradient paths of three layers, the average number of binary parameters in BinaryDenseNet and Enhanced BinaryDenseNet blocks is 0 and $\frac{3\times(P_1^{ED}+P_2^{ED}+P_3^{ED}+P_4^{ED})}{4}$, respectively. In terms of the gradient paths of four layers, the average number of binary parameters in BinaryDenseNet and Enhanced BinaryDenseNet blocks is 0 and $P_1^{ED} + P_2^{ED} + P_3^{ED} + P_4^{ED}$, respectively. To ensure a fair comparison, we set the numbers of binary parameters of model blocks to be roughly the same, i.e., $P_1^{BD} + P_2^{BD} \approx P_1^{ED} + P_2^{ED} + P_3^{ED} + P_4^{ED}$. Therefore, $\frac{P_1^{BD}+P_2^{BD}}{2} > \frac{P_1^{ED}+P_2^{ED}+P_3^{ED}+P_4^{ED}}{4}$. The average number of binary parameters on the gradient paths of one layer and two layers in the Enhanced BinaryDenseNet block is half of that in the BinaryDenseNet block. However, there is a much larger average number of binary parameters on the gradient paths of three layers and four layers in the Enhanced BinaryDenseNet block than that in the BinaryDenseNet block.

Why is it reasonable to expect that a smaller average number of binary parameters on the gradient paths of the same length results in an improvement of gradient paths? Considering the gradient paths of the same length, we can interpret the average number of binary parameters on these gradient paths as the average width of these gradient paths. It is reasonable to expect that a smaller average number of binary parameters on the gradient paths of the same length (or a shorter average width of these gradient paths, which are of the same length) results in an improvement of these gradient paths. Reducing gradient

path length has been adopted for improving gradient backpropagation during training since the gradient information received by earlier layers from a loss at the end of the model is noisier than that received by deeper layers [85, 114, 115]. The gradient information noise in a BCNN is more than that in a full-precision DCNN because of the gradient mismatch problem introduced by the binarization. We do not have to consider the gradient vanishing or exploding problem since we use ReLu as the no-linear function. The gradient information noise accumulates with computing gradient backpropagation on the gradient paths. In particular, a smaller amount of computation (i.e., the smaller number of multiplication and addition) required for gradient backpropagation on the gradient paths indicates less accumulation of gradient information noise. A smaller average number of binary parameters on the gradient paths of the same length and a shorter average length of the gradient paths implies a smaller amount of computation needed for gradient backpropagation on the gradient paths. Thus, reducing the average length of the gradient paths and decreasing the average number of binary parameters on the gradient paths of the same length can lead to improving the gradient paths.

Why can the three metrics evaluate the quality of gradient paths? We summarize how the three metrics are used to evaluate the quality of gradient paths as follows. Increasing the total number of gradient paths, reducing the average length of the gradient paths, and decreasing the average number of binary parameters on the gradient paths of the same length can improve the gradient paths. The summation of a larger total number of gradient paths can reduce gradient information random noise, improving these gradient paths. A shorter average length of the gradient paths and a smaller average number of binary parameters on the gradient paths of the same length indicates a smaller amount of computation used for gradient backpropagation, which reduces the accumulation of gradient information noise to improve the gradient paths.

Why is there no contradiction between increasing representation capability and improving gradient paths in a binary model? It is worth clarifying that there is no contradiction between increasing representation capability and improving gradient paths in a binary model. The representation capability is closely related to the number of binary parameters of a binary model. Specifically, increasing the number of binary parameters of a binary model can enlarge the representation capability. We evaluate the gradient path quality with the average length of these gradient paths and the average number of binary parameters on these gradient paths of the same length (or the average width of these gradient paths, which are of the same length). In particular, reducing the average length of the gradient paths and decreasing the average number of binary parameters on the gradient paths of the same length can improve the gradient paths. It is worth noting that the summation of the binary parameters on all the gradient paths in a binary model is not equal to the number of binary parameters of the binary model. Thus, we can enlarge the representation capability by increasing the number of binary parameters of a binary model and simultaneously improve the gradient paths by reducing the average length of these gradient paths and decreasing the average number of binary parameters on these gradient paths of the same length.

Table 5.4: Binary ResNet and DenseNet variants on ImageNet. There are four blocks in this Table. **First block:** ResNet18(64) and UA-ResNet variants to compare with ResNet18(64). **Second block:** ResNet34(64) and UA-ResNet variants to compare with ResNet34(64). **Third block:** BinaryDenseNet51(32) and UA-DenseNet to compare with BinaryDenseNet51(32). **Fourth block:** BinaryDenseNet69(32) and UA-DenseNet variants to compare with BinaryDenseNet69(32).

Model	Bit-width	Top-1	Top-5	Storage	Computation	Run-time memory
Bi-Real ResNet18(64)	$b=1$	40.42%	18.29%	33.18Mbit	1.64×10^8Flops	154.14MB
UA-ResNet21(53)	$b=1$	37.58%	16.06%	32.63Mbit	1.46×10^8Flops	170.20MB
UA-ResNet37(41)	$b=1$	**37.13%**	**15.63%**	32.24Mbit	1.28×10^8Flops	164.58MB
UA-ResNet69(31)	$b=1$	37.66%	15.77%	32.16Mbit	1.14×10^8Flops	149.32MB
Bi-Real ResNet34(64)	$b=1$	36.74%	15.36%	43.28Mbit	1.93×10^8Flops	154.14MB
UA-ResNet41(48)	$b=1$	**35.62%**	**14.53%**	42.61Mbit	1.64×10^8Flops	154.14MB
UA-ResNet77(35)	$b=1$	36.66%	15.07%	41.53Mbit	1.44×10^8Flops	140.49MB
BinaryDenseNet51(32)	$b=1$	38.14%	16.80%	34.80Mbit	2.70×10^8Flops	359.66MB
UA-DenseNet51(53)	$b=1$	**36.73%**	**15.54%**	34.53Mbit	2.97×10^8Flops	306.68MB
BinaryDenseNet69(32)	$b=1$	36.26%	15.24%	41.95Mbit	2.82×10^8Flops	359.66MB
UA-DenseNet69(48)	$b=1$	**35.20%**	**14.59%**	41.52Mbit	3.06×10^8Flops	282.59MB

5.3.5. COMPUTATIONAL COMPLEXITY ANALYSIS

To guarantee the fairness of the comparison, we scale the number of base channels or the growth rate of unified architectures to match the computational complexity of binary ResNet and DenseNet baselines. The computational complexity is analyzed in terms of storage in Mbit, computation in Flops, and run-time memory in MB. Given a computation complexity budget, we build more unified architectures with different base channels or the growth rate (or depths, or numbers of columns in a block, or numbers of convolutional layers on the longest path in a block) to show that our unified EDR techniques and unified architectures are robust to architectural hyperparameters of neural networks. Taking UA-ResNet41(48) and UA-DenseNet51(53) as examples, 41 and 51 are the depths of unified architectures, while 48 and 53 refer to the number of base channels and the growth rate after scaling, respectively.

Storage and computation. We adopt the number of parameters as the metric for storage usage, and the number of Flops as the metric for computational efficiency. The number of parameters is measured as the summation of 32bits times the number of floating-point parameters and 1bit times the number of binary parameters in the model. The XNOR and Popcount bitwise operations can be executed by the current CPUs with a parallelism of 64. Therefore, the Flops is calculated by the number of floating-point multiplications plus $1/64$ of the number of binary multiplication.

Table 5.5: Binary ResNet and DenseNet variants on CIFAR-100. There are four blocks in this Table. **First block:** ResNet18(64) and UA-ResNet variants to compare with ResNet18(64). **Second block:** ResNet34(64) and UA-ResNet variants to compare with ResNet34(64). **Third block:** BinaryDenseNet51(32) and UA-DenseNet variants to compare with BinaryDenseNet51(32). **Fourth block:** BinaryDenseNet69(32) and UA-DenseNet variants to compare with BinaryDenseNet69(32).

Model	Bit-width	Top-1	Top-5	Storage	Computation	Run-time memory
Bi-Real ResNet18(64)	$b = 1$	28.48%	8.65%	18.18Mbit	1.67×10^7Flops	50.33MB
UA-ResNet21(50)	$b = 1$	**26.34%**	**7.89%**	18.07Mbit	1.53×10^7Flops	52.43MB
UA-ResNet37(36)	$b = 1$	26.67%	7.51%	17.57Mbit	1.42×10^7Flops	47.19MB
UA-ResNet69(26)	$b = 1$	26.85%	7.57%	17.63Mbit	1.38×10^7Flops	40.89MB
Bi-Real ResNet34(64)	$b = 1$	27.93%	8.37%	28.28Mbit	2.61×10^7Flops	50.33MB
UA-ResNet41(45)	$b = 1$	**25.36%**	**7.26%**	27.64Mbit	2.28×10^7Flops	47.19MB
UA-ResNet77(32)	$b = 1$	25.57%	6.86%	27.94Mbit	2.24×10^7Flops	41.94MB
UA-ResNet149(22)	$b = 1$	26.38%	7.83%	26.35Mbit	2.08×10^7Flops	34.60MB
BinaryDenseNet51(32)	$b = 1$	27.16%	7.77%	17.65Mbit	5.13×10^7Flops	117.44MB
UA-DenseNet51(48)	$b = 1$	**26.72%**	**7.51%**	17.51Mbit	5.32×10^7Flops	92.27MB
BinaryDenseNet69(32)	$b = 1$	26.88%	7.52%	23.70Mbit	5.50×10^7Flops	117.44MB
UA-DenseNet69(44)	$b = 1$	**26.38%**	**7.32%**	23.33Mbit	5.67×10^7Flops	85.98MB

Run-time memory consumption. To estimate the run-time memory requirement, we calculate the size of the weights and activations for a layer or an operation, plus all the activations of shortcut connections that cross past that layer or operation. The type of that layer can be either a convolutional layer or a join layer. The type of that operation can be either a summation operation or a concatenation operation. We report the largest among such values for each model, which would serve as a lower bound for the run-time memory disregarding the layer or operation schedule even though the actual usage would largely depend on individual hardware and framework implementation. The reported run-time memory is estimated with a batch size of 64.

Computational complexity of shortcuts and join layers in unified architectures. The shortcuts and join layers in unified architectures need more run-time memory and more computation than those in Bi-Real ResNet and BinaryDenseNet since there are fewer shortcuts and no join layers in Bi-Real ResNet and BinaryDenseNet. However, the binary convolutional layers in unified architectures consume less run-time memory and less computation than those in Bi-Real ResNet and BinaryDenseNet. To ensure a fair comparison, we scale the number of base channels or the growth rate of unified architectures to match the computational complexity of Bi-Real ResNet and BinaryDenseNet baselines. For example, we build UA-ResNet21 with a scaled number of base channels equal to 53 to match the computational complexity of Bi-Real ResNet18 with the number of base channels equal to 64. Thus, our unified architectures require almost the same run-time memory and computation

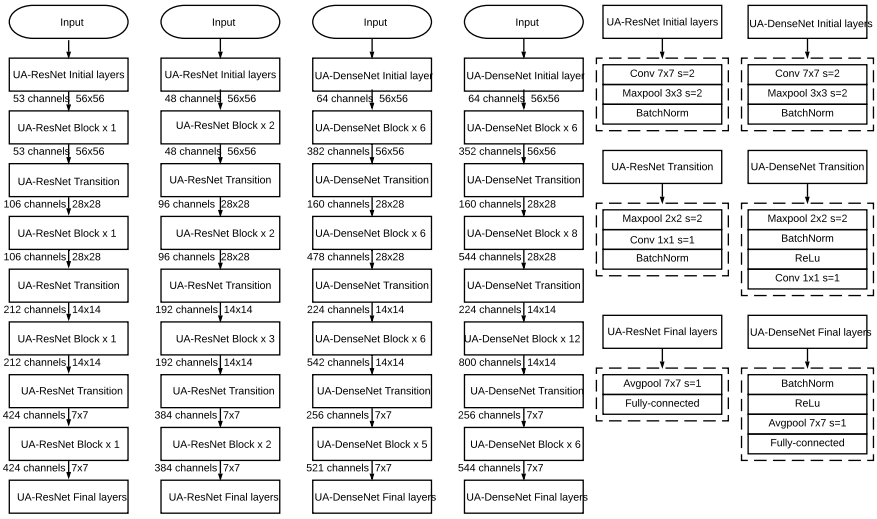


Figure 5.4: The building blocks and an exemplary network structure of our unified architectures. **Left two columns:** In UA-ResNet, a block (c3d4) has 3 columns and 4 convolutional layers on the longest path. **Middel two columns:** In UA-DenseNet, a block (c2d2) has 2 columns and 2 convolutional layers on the longest path.

to execute as Bi-Real ResNet and BinaryDenseNet.

5.3.6. OVERVIEW OF UNIFIED ARCHITECTURES

As shown in Figure 5.4, we describe the overall view of our unified architectures with the input images of size 224×224. To ensure a fair comparison when we build our unified architectures, we scale the number of base channels or the growth rate of our unified architectures to have almost the same computational complexity as Bi-Real ResNet and BinaryDenseNet.

The left two columns are the UA-ResNet, i.e., UA-ResNet21(53) and UA-ResNet41(48), respectively. 21 and 41 represent the depths of unified architectures, while 53 and 48 refer to their base number of channels, which are scaled to match the computational complexity of ResNet18 and ResNet34 after binarization, respectively. Similarly, we build UA-DenseNet51(53) and UA-DenseNet69(48) to match the computational complexity of DenseNet51(32) and DenseNet69(32) after binarization [105], respectively. 51 and 69 refer to the depths of unified architectures, while 53 and 48 refer to the growth rate after scaling. We calculate the model depth with the criteria that every convolutional layer is recognized as one layer, which is different from that in [105] (i.e., every block is recognized as a layer). To ensure consistency, BinaryDenseNet28(64) and BinaryDenseNet37(64) in [105] are renamed as BinaryDenseNet51(32) and BinaryDenseNet69(32) in our chapter. The right two columns present the composition of the initial layers, transition block, and final layers in

unified architectures.

5.4. EXPERIMENTAL RESULTS

Compared with both Bi-Real ResNet and BinaryDenseNet on ImageNet and CIFAR-100, our unified architectures with different base channels or growth rate consistently show significant performance improvement, which necessitates the fractal EDR technique for our unified architectures. Besides, we demonstrate the essential role of the shortcut EDR technique for our unified architectures, which supports that the key of our proposal is to unify multiple EDR techniques into one model.

5.4.1. EXPERIMENTAL RESULTS ON IMAGENET

In this section, we present the experimental results of our unified architectures on ImageNet. Compared with both Bi-Real ResNet and BinaryDenseNet, our unified architectures with different base channels or growth rate consistently show significant performance improvement, which necessitates the fractal EDR technique for our unified architectures. Besides, we demonstrate the essential role of the shortcut EDR technique for our unified architectures, which supports that the key of our proposal is to unify multiple EDR techniques into one model.

ResNet variants on ImageNet. As shown in Table 5.4, we present the experimental results of UA-ResNet on ImageNet. Our UA-ResNet variants with different base channels, including UA-ResNet21(53), UA-ResNet37(41), and UA-ResNet69(31), consistently achieve significant performance improvement compared with Bi-Real ResNet18. In particular, UA-ResNet37(41) and UA-ResNet41(48) improve the Top-1 accuracy by 3.29% and 1.12% compared with Bi-Real ResNet18(64) and Bi-Real ResNet34(64), respectively. Regarding the computational complexity, UA-ResNet37(41) increases the run-time memory size by 10.44MB, but saves the number of parameters by 0.94Mbit and the number of Flops by 0.36×10^8 (21.95%) compared with Bi-Real ResNet18(64). Similarly, the number of parameters and the number of Flops required for our proposed UA-ResNet41(48) are 0.67Mbit and 0.29×10^8 less than those needed for Bi-Real ResNet34(64).

DenseNet variants on ImageNet. As shown in Table 5.4, we present the experimental results of our UA-DenseNet on ImageNet. The Top-1 accuracy of UA-DenseNet51(53) and UA-DenseNet69(48) are 1.41% and 1.06% better than those of BinaryDenseNet51(32) and BinaryDenseNet69(32), respectively. In terms of the computational complexity, UA-DenseNet51(53) and UA-DenseNet69(48) require 0.27×10^8 Flops and 0.24×10^8 Flops less compared with BinaryDenseNet51(32) and BinaryDenseNet69(32), respectively, while they save the number of parameters by 0.37Mbit and 0.37Mbit, respectively, and decrease the run-time memory size by 52.98MB and 77.07MB, respectively.

5.4.2. EXPERIMENTAL RESULTS ON CIFAR-100

In this section, we present the experimental results of binary ResNet and DenseNet variants on CIFAR-100, which shows that our proposed unified architectures with different base

Table 5.6: Ablation study results of CIFAR-100. **First block:** DenseNet variants. **Second block:** ResNet variants.

Model	Bit-width	Top-1	Top-5
UA-DenseNet51(48)	$b = 1$	26.72%	7.51%
A-DenseNet51(48)	$b = 1$	27.19%	7.26%
UA-DenseNet69(44)	$b = 1$	26.38%	7.32%
A-DenseNet69(44)	$b = 1$	26.63%	7.34%
UA-ResNet21(50)	$b = 1$	26.34%	7.89%
A-ResNet21(50)	$b = 1$	31.82%	9.70%
UA-ResNet41(45)	$b = 1$	25.36%	7.26%
A-ResNet41(45)	$b = 1$	40.14%	15.19%

channels or growth rate can consistently improve the accuracy of binary ResNet and binary DenseNet significantly.

ResNet variants on CIFAR-100. As shown in Table 5.5, we present the accuracy of unified architectures for binarizing ResNet18 and ResNet34. UA-ResNet variants with different base channels consistently outperform Bi-Real ResNet baselines. Compared with Bi-Real ResNet18(64) and Bi-Real ResNet34(64), the Top-1 accuracy of our UA-ResNet21(50) and UA-ResNet41(45) are improved by 2.14% and 2.57%, respectively. Considering the computational complexity, our UA-ResNet21(50) use 0.11Mbit less for storage, 0.14×10^7 Flops less for computation, and 2.10MB more for run-time memory than Bi-Real ResNet18(64). Our UA-ResNet41(45) saves the storage by 0.64Mbit, the computation by 0.33×10^7 Flops, and the run-time memory by 3.14MB compared with Bi-Real ResNet34(64), respectively.

DenseNet variants on CIFAR-100. As shown in Table 5.5, we present the accuracy of our unified architectures for binary DenseNet51(32) and DenseNet69(32). The Top-1 accuracy of our proposed UA-DenseNet51(48) and UA-DenseNet69(44) are 0.44% and 0.50% better than those of BinaryDenseNet51(32) and BinaryDenseNet69(32), respectively. The increased number of Flops required for our proposed UA-DenseNet51(48) and UA-DenseNet69(44) is 0.19×10^7 and 0.17×10^7, respectively, while the decreased number of parameters required for them is 0.06Mbit and 0.37Mbit, respectively, and the decreased run-time memory size needed for them is 25.17MB and 31.46MB, respectively, compared with BinaryDenseNet51(32) and BinaryDenseNet69(32).

5.4.3. ABLATION STUDY

In the above section, we have shown the advantage of unified architectures over Bi-Real ResNet and BinaryDenseNet, which indicate the benefits of unifying the fractal architecture EDR technique. In this section, we explore the role of shortcut EDR technique for unified architectures.

Table 5.7: Comparison with state-of-the-art methods on ImageNet. * refers to the baseline from the published papers. # indicates the downsampling layers are binarized. Complexity (a/b) means a Mbit and b GFlops.

Model	Top-1/Top-5	Complexity
BNN ResNet18 #* [33]	57.80%/30.80%	27.9/0.14
XNOR-Net ResNet18 #* [37]	48.80%/26.80%	28.0/0.14
TBN-ResNet18 #* [49]	44.40%/25.80%	27.9/0.17
Trained Bin ResNet18 #* [116]	45.80%/22.10%	27.9/0.14
CI-Net ResNet18 #* [100]	43.30%/19.90%	27.9/0.14
XNOR-Net++ ResNet18 #* [98]	42.90%/20.10%	28.0/0.14
Bi-Real ResNet18 * [38]	43.60%/20.50%	33.2/0.16
CI-Net ResNet18 * [100]	40.10%/15.80%	33.2/0.16
Real-to-Bin ResNet18 * [97]	**34.60%/13.80%**	33.2/0.16
Bi-Real ResNet18(64) [38]	40.42%/18.29%	33.2/0.16
UA-ResNet37(41)	37.13%/15.63%	32.2/0.13
TBN-ResNet34 #* [49]	41.80%/19.00%	38.0/0.23
Bi-Real ResNet34 * [38]	37.80%/16.10%	43.3/0.19
Bi-Real ResNet34(64) [38]	36.74%/15.36%	43.3/0.19
UA-ResNet41(48)	**35.62%/14.53%**	42.6/0.16
BinaryDenseNet51(32) * [105]	39.30%/17.60%	34.8/0.27
BinaryDenseNet51(32) [105]	38.14%/16.80%	34.8/0.27
UA-DenseNet51(53)	**36.73%/15.54%**	34.5/0.30
BinaryDenseNet69(32) * [105]	37.50%/16.10%	42.0/0.28
BinaryDenseNet69(32) [105]	36.26%/15.24%	42.0/0.28
UA-DenseNet69(48)	**35.20%/14.59%**	41.5/0.31
Full-precision ResNet18 *	30.70%/10.80%	374.1/1.81
Full-precision ResNet34 *	26.80%/8.60%	697.3/3.66

The architectures of A-DenseNet51(48) and A-ResNet21(50) are obtained by removing all the residual connections from UA-DenseNet51(48) and UA-ResNet21(50), respectively. As shown in Table 5.6, the residual connection EDR technique can improve the Top-1 accuracy of A-DenseNet51(48) and A-DenseNet69(44) by 0.47% and 0.25%, respectively. Similarly, the Top-1 accuracy degradation of A-ResNet21(50) and A-ResNet41(45) is 5.48% and 14.78% without residual connections.

5.4.4. COMPARISON TO STATE-OF-THE-ART

As shown in Table 5.7, we compare with state-of-the-art BCNNs on ImageNet. Except for the Real-to-Bin ResNet18 [97], the Top-1 accuracy of UA-ResNet37(41), UA-ResNet41(48), UA-DenseNet51(53), and UA-DenseNet69(48) achieve 37.13%, 35.62%, 36.73%, and 35.20%, respectively, and outperform other binary ResNet and DenseNet variants by a large margin. More importantly, Real-to-Bin focuses on the minimization of quantization error between the BCNNs and their full precision counterparts, while unified architectures work towards the architecture design for BCNNs. Thus, it is also reasonable

to expect that the performance of Real-to-Bin can be improved further when applying our proposed unified architectures.

5.5. CONCLUSION

In this chapter, we identify the limitation of relying solely on enhancing the shortcut EDR technique. Rather than relying solely on enhancing the shortcut EDR technique, we propose to unify multiple EDR techniques to make the gradient backpropagate more easily and design unified architectures. In particular, UA-ResNet unifies the residual connection and fractal architecture EDR techniques into one model. UA-DenseNet unifies the residual connection, dense connection, and fractal architecture EDR techniques together. Gradient path analysis demonstrates that our unified architectures have better gradient paths than Bi-Real ResNet and BinaryDenseNet to make the gradient backpropagate more easily, which cannot be achieved by relying solely on enhancing the shortcut EDR technique. Our unified architectures with different base channels or growth rate can consistently improve the performance of binary ResNet and binary DenseNet by a large margin, indicating the necessity of unifying the fractal architecture EDR technique for our unified architectures. The ablation study shows that it is essential to unifying the shortcut EDR technique for our unified architectures. Thus, we conclude that the key of our proposal is to unify multiple EDR techniques to make the gradient backpropagate more easily and each EDR technique is indispensable to our unified architectures. Under a given computational complexity budget, the Top-1 accuracy of our proposed unified architectures surpasses the state-of-the-art Bi-Real ResNet18(64) by 3.29%, Bi-Real ResNet34(64) by 1.12%, BinaryDenseNet51(32) by 1.41%, and BinaryDenseNet69(32) by 1.06% on ImageNet classification.

6

REDUCING FEATURE REUSE WITHIN CONVOLUTION

High-level feature maps of Convolutional Neural Networks are computed by reusing their corresponding low-level feature maps, which brings into full play feature reuse to improve the computational efficiency. This form of feature reuse is referred to as feature reuse between *convolutional layers. The second type of feature reuse is referred to as feature reuse* within *the convolution, where the channels of the output feature maps of the convolution are computed by reusing the same channels of the input feature maps, which results in an approximation of the channels of the output feature maps. To compute them accurately, we need specialized input feature maps for every channel of the output feature maps. In this chapter, we first discuss the approximation problem introduced by full feature reuse* within *the convolution and then propose a new feature reuse scheme called Reducing Approximation of channels by Reducing Feature reuse (REAF). The chapter also shows that group convolution is a special case of our REAF scheme and we analyze the advantage of REAF compared to such group convolution. Moreover, we develop the REAF+ scheme and integrate it with group convolution-based models. Compared with baselines, experiments on image classification demonstrate the effectiveness of our REAF and REAF+ schemes. Under the given computational complexity budget, the Top-1 accuracy of REAF-ResNet50 and REAF+-MobileNetV2 on ImageNet will increase by* 0.37% *and* 0.69% *respectively.*

The content of this chapter is based on [117].

6.1. INTRODUCTION

Convolutional Neural Networks (CNNs) have achieved a series of breakthroughs on non-trivial visual tasks [90, 118–121]. The features in a dataset can be learned in an end-to-end manner by CNNs with minimal human effort and can be transferred to diverse visual tasks [122]. Accordingly, researchers are dedicated to designing better networks for learning representations [123–126] instead of handcrafted features.

To enrich the representational power of CNNs, recent work investigates various aspects of CNN network architecture [71, 127–129]. Constructing deep CNNs by stacking building blocks of the same shape is an effective strategy [71] since higher layers learn more abstract and invariant representations [87, 130]. Inherited from this, networks with skip connections [22, 127] enable the training CNNs with extreme depth. Inspired from the Hebbian principle and multi-scale processing, multi-branch CNNs [72] achieve compelling accuracy if the topology of each branch is carefully designed, and multiple branches are expected to approximate large and dense layers of powerful representational capability. Except for depth, the width of a network is an essential dimension to increase the model capability [131]. Exposing the new dimension of cardinality [132] or deploying the new approaches [124, 133, 134] can enlarge the representational ability of the model. To address the limitation of spatial locality in convolution, the attention mechanism [133, 135] is used to capture a larger feature interaction. Using automated strategy, Neural Architecture Search achieves state-of-the-art accuracy [134] and platform-aware efficiency [81].

When it comes to the efficiency of designing CNNs, feature reuse is key to making it feasible [74, 130, 136, 137]. More precisely, the features computed by earlier layers will be reused by the latter layers, which is the feature reuse *between* convolutional layers and is popular in both a plain network [138] and a multi-branch network [72]. The pursuit of maximizing feature reuse between convolutional layers is an important concept in designing these networks. Deeper CNNs encourage more feature reuse, which resulted in designing VGG-net [138] and ResNet [22]. Deep CNNs with identity mapping have a problem of diminishing feature reuse in [139], which motivates the development of wide ResNet [131], ResNet with stochastic depth [140], and DenseNet [23]. Also, feature reuse plays a critical role *within* the convolution in addition to *between* the convolutional layers. Specifically, all the channels of the output activations of the convolution are computed by reusing all and the same channels of the input activations.

Feature reuse is at the heart of the theoretical advantages behind deep learning and explains the power of distributed representations [130]. However, it is a limitation of the representational capability of networks since the reused features are an approximation of accurate features. Reducing or eliminating feature reuse from a given model will result in comparable or higher model accuracy, which supports the approximation drawback of feature reuse. There is some research investigating eliminating feature reuse *between* convolutional layers. For example, CondenseNet [74] removes such connections between layers to avoid superfluous feature reuse in the network architecture. Besides, the lottery ticket [141], selective allocation of channels [142], pruning [143, 144], and group convolution [106, 128] can all be regarded as methods for reducing feature reuse *within* the convolution as analyzed in our work, which has not been pointed out in their original illustration. We initially pointed out that feature reuse *within* the convolution leads to the problem of approximation of the channels. Thus, by reducing feature reuse we can reduce the approximation *within*

the convolution, which makes the calculation of the channels more accurate.

Our main contributions are summarized as follows.

- To our best knowledge, we are the first to point out and analyze the approximation problem introduced by the feature reuse *within* the convolution. To solve the problem, we propose the Reducing Approximation of channels by Reducing Feature reuse (REAF) scheme, which is a moderate version of feature reuse *within* the convolution.
- We compare our REAF scheme with group convolution and show that there are more merged channels in our REAF scheme than those in group convolution even though group convolution is a special case of our REAF scheme.
- We develop our REAF+ scheme with Bn and Relu layers as the parameterized operations and integrate it with group convolution-based models.
- We use extensive experiments to demonstrate the effectiveness of our REAF and REAF+ schemes.

6.2. RELATED WORK

In this section, we describe the network engineering, including the multi-branch and convolution variants.

6.2.1. MULTI-BRANCH CONVOLUTIONAL NETWORKS

To ease the difficulty of training deep neural networks, an adaptive gating unit is used in Highway networks [139], which evolves into identity mapping in ResNet [22]. Replacing the identity mapping with more residual blocks, shake-shake networks [145], and multi-residual networks [146] are extended to improve the accuracy and speed. FractalNets [85] and Multilevel ResNets [113] expand the multiple paths in a fractal and recursive way, respectively. The Inception series [72] aggregate the multifarious features of multi-scale with a careful configuration for each branch.

6.2.2. CONVOLUTION VARIANTS

To enrich the representational capability, deformable convolution [123, 147] and active convolution [148] augment the spatial sampling locations with additional offsets. Considering the computational complexity, group convolution [149], flattened convolution [150], tiled convolution [151], octave convolution [152], and dilated convolution [118] are developed as variants. Depthwise convolution [153] is an example of group convolution with the number of groups being the same as the number of channels. Based on these convolution variants, compact models are built, including IGCV series [154], MobileNet series [75], ShuffleNet series [76], and Espnet series [77].

6.3. METHOD

In this section, we analyze the limitation of the convolution, i.e., the approximation of the channels caused by the full feature reuse *within* the convolution. To address this problem,

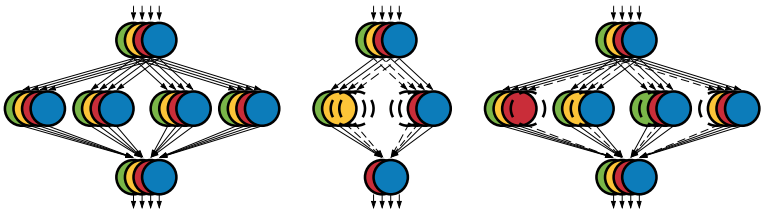


Figure 6.1: Illustration of convolution using different feature reuse schemes. Every circle stands for every channel of the feature maps. Circles on the top, in the middle, and on the bottom refer to the input feature maps, the input feature maps to compute every channel of the output feature maps, and the output feature maps, respectively. Circles in dashed lines indicate the absence of the channels.

we propose our REAF scheme for convolution, which introduces specialization to compute the channels of the output feature maps. We optimize the REAF scheme and compare it with group convolution since group convolution is a special case of our scheme. Finally, we develop the REAF+ scheme to improve the performance of the group convolution-based models.

6.3.1. PROBLEM DEFINITION

The output activations $O \in R^{C_{out} \times H \times W}$ of the convolution are convolved between the input activations $I \in R^{C_{in} \times H \times W}$ and the weights $W \in R^{C_{out} \times C_{in} \times h \times w}$, where the batch size N is omitted. The channel-wise representation of the output activations O is shown as follows, where O_j refers to the j^{th} channel of the output activations O and $j = 0, ..., C_{out} - 1$.

$$O = \{O_0, O_1, ..., O_{C_{out}-1}\} \tag{6.1}$$

To study feature reuse for every individual channel, we construct a variable $A \in R^{C_{out} \times C_{in} \times H \times W}$ to compute all the channels of the output activations, where A_j is the input activation to compute the j^{th} channel of the output activation O_j. The feature reuse *within* the convolution is shown in the left of Fig. 6.1, where we reuse all the channels of the input activations I to calculate every individual channel of the output activations O. Therefore, O_j is convolved by the input activations A_j and the weights W_j as follows and $i = 0, ..., C_{in} - 1$.

$$O_j = \sum_{i=0}^{C_{in}-1} A_{ji} \times W_{ji} = \sum_{i=0}^{C_{in}-1} I_i \times W_{ji} \tag{6.2}$$

The information of the CNN transitions gradually from spatial coding to channel coding by a hierarchy of representations. Regarding the feature reuse *between* the convolutional layers, the learned hierarchical representation makes it reasonable to save computational complexity for CNNs since the high-level features are composed of the low-level features. However, when it comes to feature reuse *within* the convolution, i.e., every channel of the output activations is computed by reusing the same input activations. The reused input

Table 6.1: ResNet50 and REAF-ResNet50 with a 4-3-4 template using the reformulation of the REAF scheme. Inside the brackets is the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=72" refers to the base width of the mode. “4-3-4” suggests that the REAF scheme with the configurations of $G_I = 4$, $G_M = 3$, and $G_O = 4$ is applied to the convolution. The numbers of parameters and FLOPs are comparable between these two models.

Stage	Output	ResNet50	REAF-ResNet50 (72-72, 4-3-4)
conv1	112 × 112	7 × 7, 64, stride 2	7 × 7, 64, stride 2
	56 × 56	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2
conv2	56 × 56	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 72 \\ 3\times3, 72, \text{4-3-4} \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28 × 28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 144 \\ 3\times3, 144, \text{4-3-4} \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14 × 14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 288 \\ 3\times3, 288, \text{4-3-4} \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7 × 7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 576 \\ 3\times3, 576, \text{4-3-4} \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1 × 1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
Parameters		25.56×10^6	26.11×10^6
FLOPs		3.86×10^9	3.97×10^9

activations are approximated tensors for all the channels of the output activations. As every channel of feature maps is considered as a feature detector [155], the input feature maps to compute every channel of the output feature maps of the convolution are expected to be customized and specialized to make the computation more accurate.

6.3.2. REDUCING APPROXIMATION OF CHANNELS BY REDUCING FEATURE REUSE

The approximation problem of the channels introduced by the feature reuse *within* the convolution can be expressed as $A_j = A_{j'}$ and $j, j' = 0, ..., C_{out} - 1$. To introduce specialized input feature maps for every channel of the output feature maps, a straightforward scheme for the convolution is shown in the middle of Fig. 6.1, where there is no feature reuse *within* the convolution at all. The input activations are divided into G_I groups, where $C_{in} = G_I \times A_{in}$ and A_{in} is the number of channels of every group to compute every channel of the output

Table 6.2: ResNeXt50 and REAF-ResNet50 with a 4-3-4 template using the reformulation of the REAF scheme. Inside the brackets is the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=64" refers to the base width of the mode. “4-3-4” suggests that the REAF scheme with the configurations of $G_I = 4$, $G_M = 3$, and $G_O = 4$ is applied to the convolution. "4" refers to the number of groups $G = 4$ in ResNeXt50. The numbers of parameters and FLOPs are comparable between these two models.

Stage	Output	ResNeXt50	REAF-ResNet50 (64-64, 4-3-4)
conv1	112 × 112	7 × 7, 64, stride 2	7 × 7, 64, stride 2
	56 × 56	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2
conv2	56 × 56	$\begin{bmatrix} 1\times1, 64, 4 \\ 3\times3, 64, 4 \\ 1\times1, 256, 4 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 72, 4\text{-}3\text{-}4 \\ 3\times3, 72, 4\text{-}3\text{-}4 \\ 1\times1, 256, 4\text{-}3\text{-}4 \end{bmatrix} \times 3$
conv3	28 × 28	$\begin{bmatrix} 1\times1, 128, 4 \\ 3\times3, 128, 4 \\ 1\times1, 512, 4 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 144, 4\text{-}3\text{-}4 \\ 3\times3, 144, 4\text{-}3\text{-}4 \\ 1\times1, 512, 4\text{-}3\text{-}4 \end{bmatrix} \times 4$
conv4	14 × 14	$\begin{bmatrix} 1\times1, 256, 4 \\ 3\times3, 256, 4 \\ 1\times1, 1024, 4 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 288, 4\text{-}3\text{-}4 \\ 3\times3, 288, 4\text{-}3\text{-}4 \\ 1\times1, 1024, 4\text{-}3\text{-}4 \end{bmatrix} \times 6$
conv5	7 × 7	$\begin{bmatrix} 1\times1, 512, 4 \\ 3\times3, 512, 4 \\ 1\times1, 2048, 4 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 576, 4\text{-}3\text{-}4 \\ 3\times3, 576, 4\text{-}3\text{-}4 \\ 1\times1, 2048, 4\text{-}3\text{-}4 \end{bmatrix} \times 3$
	1 × 1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
Parameters		20.89×10^6	20.39×10^6
FLOPs		2.98×10^9	2.72×10^9

activations. We have $G_I = C_{out}$. The input activations to compute the j^{th} channel of the output activations are A_j, where A_j is part of the input activations I as follows. Index refers to the function that indexes the j^{th} group from I.

$$A_j = \text{Index}(I, j, A_{in}) \tag{6.3}$$

In this way, we do not reuse any channels of the input activations I to compute every channel of the output activations O. O_j is convolved by the input activations A_j and the weights W_j as follows.

$$O_j = \sum_{i=0}^{A_{in}-1} A_{ji} \times W_{ji} = \sum_{i=0}^{A_{in}-1} \text{Index}(I, j, A_{in}) \times W_{ji} \tag{6.4}$$

Considering computational complexity and distributed representations, it is inadvisable to remove feature reuse totally *within* the convolution. To keep feature reuse and introduce customized input activations to compute the channels, we introduce our scheme called REAF, which is a moderate version of feature reuse for convolution as shown in the right part of Fig. 6.1.

All the channels of the input activations I and the output activations O are divided into G_I and G_O groups respectively, and there is only one channel in every group as shown in the right part of Fig. 6.1. $C_{out} = G_O \times B_{out}$ and B_{out} is the number of channels of every group of the output activations. To keep feature reuse and introduce specialization, we draw G_M groups from the G_I groups of the input activations to compute every group of the output activations. If all the channels of the input activations I are considered as a set S of G_I elements, every element is a group. The number of the (G_M)-combinations is denoted using elementary combinatorics text as $C(G_I, G_M)$. All the (G_M)-combinations are enumerated as E_0, ..., E_l, ..., E_{G_O-1}. Here $l = \lfloor j/B_{out} \rfloor$ and $l = 0, ..., G_O - 1$. Since it is hard to get prior knowledge on how many times the groups of the input activations should be reused, we try to keep the homogeneity of computing channels. Therefore, we have $C(G_I, G_M) = G_O$. Index refers to the function that indexes G_M groups from I based on E_l and concatenate them together.

$$A_j = \text{Index}(I, E_l, A_{in}) \tag{6.5}$$

To compute different groups of the output activations, the reused G_M groups of the input activations are different from each other. O_j is convolved by the input activations A_j and the weights W_j as follows.

$$O_j = \sum_{i=0}^{A_{in} \times G_M - 1} A_{ji} \times W_{ji} = \sum_{i=0}^{A_{in} \times G_M - 1} \text{Index}(I, E_l, A_{in}) \times W_{ji} \tag{6.6}$$

6.3.3. OPTIMIZING THE CONFIGURATIONS OF OUR SCHEME

Given a convolution with a computational complexity budget, the configurations of $G_I - G_M - G_O$ can be optimized since they have an influence on the feature reuse *within* the convolution and the number of the merged channels. When $G_I - G_M$ remains unchanged and G_O increases, the feature reuse within the group reduces since every group of the output activations is computed by reusing different input activations. On the contrary, the feature reuse between the groups increases since the number of different channels between the reused input activations of computing different groups of the output activations decreases. When G_O remains unchanged and $G_I - G_M$ increases, the feature reuse *within* the convolution reduces while the number of the merged channels decreases. In this chapter, we focus on studying the extreme case of $G_M = G_I - 1$ since $G_M = 1$ (i.e., group convolution) has been explored in [132]. Given a convolutional neural network, we apply our scheme to its convolutions and optimize the configurations. Taking ResNet50 as an example, and the overall REAF-ResNet50 architecture, i.e., apply our scheme on ResNet50, is listed in as shown in Table 6.1. We keep the topology and computational complexity of the model unchanged for a fair comparison. We adopt the REAF scheme for the middle convolution of the bottleneck and adjust its width.

As shown in Table 6.3, the experimental results of REAF-ResNet50 on CIFAR-100 classification are presented. C and C' refer to the base number of the input and output channels of the middle convolution of the bottleneck and $C = C'$ in default. When applying the REAF scheme to the middle convolution, the accuracy will improve with a wide range of configurations. When $G_I - G_M = 1$, REAF-ResNet50 with $G_I = 4$ achieves the best Top-1 accuracy among all the variants and 1.48% better than the baseline. With the increase and decrease of the number of the groups from $G_I = 4$, the accuracy will degrade, which suggests that $G_I = 4$ refers to an optimized configuration of the REAF scheme for the ResNet50.

6.3.4. ADVANTAGE OVER GROUP CONVOLUTION

Group convolution [132] has been widely adopted in the design of CNNs since it exposes a new dimension, i.e., the size of the set of transformations. Meanwhile, group convolution benefits from reducing feature reuse *within* the convolution as explained in our work, which has not been pointed out by the original reference [132]. Group convolution is an example of our REAF scheme when $G_M = 1$. The number of merged channels in our proposed scheme is $G_M - 1$ times larger than that in group convolution. Also note that the number of merged channels in the REAF scheme with the optimized configuration is much more than that in group convolution with the optimized number of groups G. Based on the experiments in [132], a larger number of cardinality indicates a more effective choice for a given model and the number of merged channels is 4 in group convolution with the optimized groups $G = 32$. For example, in our proposed scheme with the optimized configuration 4-3-4 and the number of merged channels is 54.

6.3.5. REAF+ SCHEME

We propose the REAF+ scheme, which adopts parameterized or parameter-free operations to enable $A_j \neq A_{j'}$. Applying the REAF+ scheme, the performance of the models, including the group convolution-based models or REAF scheme-based models, will improve further. When the Bn and Relu layers are included, O_j is convolved by the output activations O' of the last convolution and the weights W_j as follows.

$$O_j = \text{Conv}(A_j, W_j) = \text{Conv}(I, W_j) = \text{Conv}(\text{Relu}(\text{Bn}(O^{'})), W_j) \tag{6.7}$$

Taking the parameterized operations of Bn and Relu layers as an example, the REAF+ scheme can be expressed as follows. There are g_O pairings of Bn and Relu layers introduced when the output activations are divided into g_O groups.

$$O_j = \text{Conv}(A_j, W_j) = \text{Conv}(\text{Relu}_l(\text{Bn}_l(O^{'})), W_j) \tag{6.8}$$

6.4. EXPERIMENTAL RESULTS

We trained and evaluated our REAF-Net and REAF+-Net modes, i.e., applying our REAF and REAF+ schemes on baseline models, on CIFAR-100 and ImageNet ILSVRC2012 classification dataset [56].

Table 6.3: Experimental results of REAF-ResNet50 with different configurations.

Model	Top-1 err.	Top-5 err.	GFLOPs	#Params (M)
ResNet50 Baseline	22.06%	5.54%	1.22	23.71
REAF-ResNet50 (C=C'=64, 16-15-16)	21.70%	5.52%	1.18	23.01
REAF-ResNet50 (C=C'=64, 8-7-8)	21.57%	5.54%	1.14	22.30
REAF-ResNet50 (C=C'=70, 5-4-5)	20.89%	5.14%	1.22	24.11
REAF-ResNet50 (C=C'=72, 4-3-4)	20.58%	5.11%	1.24	24.32
REAF-ResNet50 (C=C'=72, 3-2-3)	21.80%	5.42%	1.20	23.20
REAF-ResNet50 (C=C'=80, 2-1-2)	22.52%	5.72%	1.22	23.75
REAF-ResNet50 (C=88, C'=56, 8-6-28)	22.78%	5.73%	1.23	23.60
REAF-ResNet50 (C=77, C'=63, 7-5-21)	21.94%	5.65%	1.18	22.76
REAF-ResNet50 (C=72, C'=75, 6-4-15)	22.26%	5.44%	1.23	23.74
REAF-ResNet50 (C=80, C'=70, 5-3-10)	22.14%	5.84%	1.21	23.23
REAF-ResNet50 (C=88, C'=72, 4-2-6)	21.51%	5.69%	1.23	23.40
REAF-ResNet50 (C=C'=90, 3-1-3)	21.50%	5.61%	1.25	23.66
REAF-ResNet50 (C=96, C'=56, 8-5-56)	20.82%	5.29%	1.23	23.21
REAF-ResNet50 (C=84, C'=70, 7-4-35)	20.91%	5.24%	1.23	23.50
REAF-ResNet50 (C=78, C'=80, 6-3-20)	20.77%	5.36%	1.21	23.22
REAF-ResNet50 (C=90, C'=80, 5-2-10)	20.80%	5.27%	1.23	23.36
REAF-ResNet50 (C=C'=96, 4-1-4)	21.30%	5.39%	1.25	23.44
REAF-ResNet50 (C=88, C'=70, 8-4-70)	20.56%	4.96%	1.21	23.00
REAF-ResNet50 (C=98, C'=70, 7-3-35)	21.35%	5.28%	1.24	23.29
REAF-ResNet50 (C=C'=90, 6-2-15)	20.58%	5.11%	1.25	23.56
REAF-ResNet50 (C=C'=100, 5-1-5)	21.58%	5.96%	1.24	23.19
REAF-ResNet50 (C=64, C'=112, 8-3-56)	20.71%	5.33%	1.21	23.59
REAF-ResNet50 (C=98, C'=84, 7-2-21)	21.38%	5.41%	1.21	22.76
REAF-ResNet50 (C=C'=102, 6-1-6)	**20.28**%	**5.06**%	1.21	22.74
REAF-ResNet50 (C=90, C'=84, 9-3-84)	21.28%	5.23%	1.20	22.68
REAF-ResNet50 (C=104, C'=84, 8-2-28)	21.11%	5.06%	1.21	22.71
REAF-ResNet50 (C=C'=105, 7-1-7)	21.08%	5.27%	1.22	22.75
REAF-ResNet50 (C=90, C'=108, 9-2-36)	20.59%	5.23%	1.24	23.58
REAF-ResNet50 (C=C'=112, 8-1-8)	20.46%	5.18%	1.28	23.75

6.4.1. EXPERIMENTS ON CIFAR-100 CLASSIFICATION

We apply our scheme on ResNet models and conduct experiments on low-resolution imagery CIFAR-100 datasets. We adopt crop translation and flipping data augmentation. We train 200 epochs in total and the learning rate decays at the steps of 60, 120, and 160 with a factor of 0.2. As the experimental results presented in Table 6.4, the Top-1 accuracy of our REAF-ResNet outperforms that of ResNet baseline with various depths using comparable computational complexity. Especially, our REAF-ResNet101 achieves a 1.76% top-1

Table 6.4: Comparisons of REAF-ResNet and ResNet on CIFAR-100 classification.

Model	Top-1 err.	Top-5 err.	GFLOPs	#Params (M)
ResNet18 Baseline	22.21%	6.12%	0.56	11.22
REAF-ResNet18 (C=72, 4-3-4)	**21.86%**	**5.82%**	0.56	11.17
ResNet34 Baseline	21.30%	5.32%	1.16	21.33
REAF-ResNet34 (C=72, 4-3-4)	**20.86%**	**5.23%**	1.13	18.10
ResNet101 Baseline	21.20%	5.40%	2.44	42.70
REAF-ResNet101 (C=72, 4-3-4)	**20.44%**	**4.89%**	2.48	43.81

Table 6.5: Comparisons of REAF-ResNet and ResNet on ImageNet classification.

Model	Top-1 err.	Top-5 err.	GFLOPs	#Params (M)
ResNet18 Baseline	29.56%	10.36%	1.81	11.69
REAF-ResNet18 (C=72, 4-3-4)	**29.29%**	**10.28%**	1.95	11.74
ResNet50 Baseline	23.65%	6.89%	3.86	25.56
REAF-ResNet50 (C=88, C'=70, 8-4-70)	**23.18%**	**6.90%**	3.74	24.85
REAF-ResNet50 (C=72, 4-3-4)	23.28%	6.74%	3.97	26.11
WideResNet50 Baseline (widen=2.0)	22.09%	6.08%	11.44	68.88
REAF-WideResNet50 (C=136, 4-3-4)	**21.89%**	**6.10%**	10.50	63.11
ResNet101 Baseline	22.09%	6.18%	7.57	44.55
REAF-ResNet101 (C=72, 4-3-4)	**21.84%**	**5.98%**	7.79	45.83

accuracy improvement compared with the ResNet101 baseline.

6.4.2. EXPERIMENTS ON IMAGENET CLASSIFICATION

To show the performance of our proposed scheme on high-resolution images and large datasets, we experiment with ResNet and REAF-ResNet on ImageNet classification. We train 100 epochs in total with a batch size of 256. The learning rate starts at 0.1 and decays every 30 epochs with a factor of 0.1. The weight decay is $1e-4$ and the momentum is 0.9. The image is resized for scale augmentation. A 224×224 crop is randomly sampled from an image or its horizontal flip, with per-pixel normalization. As summarized in Table 6.5, the Top-1 accuracy of our REAF-ResNet increases by 0.27%, 0.37%, and 0.25% for the depth of 18, 50 and 101 layers respectively compared to the ResNet baseline.

6.4.3. COMPARISONS WITH GROUP CONVOLUTION

This subsection reports on experiments on CIFAR-100 and ImageNet classification datasets to show the advantage of our proposed scheme over group convolution. We build ResNeXt

Table 6.6: Comparisons between our REAF scheme and group convolution on classification.

Model	Top-1 err.	Top-5 err.	GFLOPs	#Params (M)
ResNeXt50 on CIFAR-100	22.89%	6.19%	1.02	19.04
REAF-ResNet50 on CIFAR-100	**22.15%**	**5.54%**	1.00	18.73
ResNeXt101 on CIFAR-100	21.70%	5.62%	1.96	33.60
REAF-ResNet101 on CIFAR-100	**20.59%**	**5.21%**	1.92	33.01
ResNeXt50 on ImageNet	25.04%	7.80%	2.98	20.89
REAF-ResNet50 on ImageNet	**24.30%**	**7.32%**	2.72	20.39

Table 6.7: Results of group convolution-based and our REAF+ scheme-based models on classification

Model	Top-1 err.	Top-5 err.	GFLOPs	#Params (M)
MobileNetV2 on CIFAR-100	31.53%	9.34%	67.59×10^{-3}	2.37
REAF+-MobileNetV2 on CIFAR-100	**30.98%**	**8.46%**	67.90×10^{-3}	2.38
ResNeXt50 on CIFAR-100	20.13%	4.99%	1.36	23.18
REAF+-ResNeXt50 on CIFAR-100	**19.70%**	**4.76%**	1.36	23.19
MobileNetV2 on ImageNet	35.02%	13.62%	3.14	3.50
REAF+-MobileNetV2 on ImageNet	**34.33%**	**13.46%**	3.17	3.52

and REAF-ResNet according to a given computational complexity budget, and their architectures can be found as shown in Table 6.2. The network architecture of ResNeXt50 and REAF-ResNet50 in the section of comparisons with group convolution are listed. The base width of ResNeXt and REAF-ResNet is $C = 64$. In ResNeXt, all the convolutions in the building bottlenecks of ResNet are replaced with the group convolutions $G = 4$. Specifically, the three convolutions in a bottleneck are replaced with group convolutions. Similarly, all the convolutions in the building bottlenecks of REAF-ResNet adopt our proposed scheme with a configuration of $4-3-4$. In this way, the difference between group convolution and our proposed scheme, introduced by the number of merged channels, can be observed clearly. On the CIFAR-100 dataset, the Top-1 accuracy of the REAF-ResNet50 model is 0.74% better than that of ResNeXt50, and the gap is 1.11% for 101 layers as in Table 6.6. Compared to ResNeXt50, the Top-1 accuracy of REAF-ResNet50 increases by 0.74% on the ImageNet dataset.

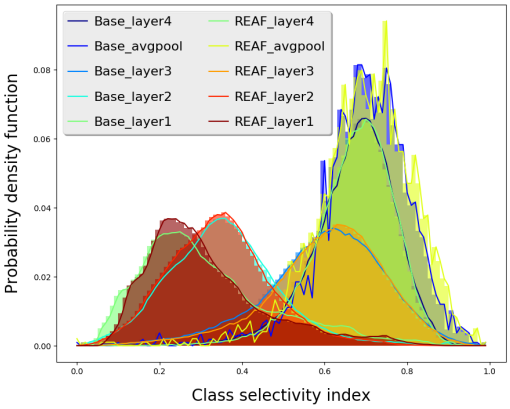


Figure 6.2: ResNet50+REAF-ResNet50.

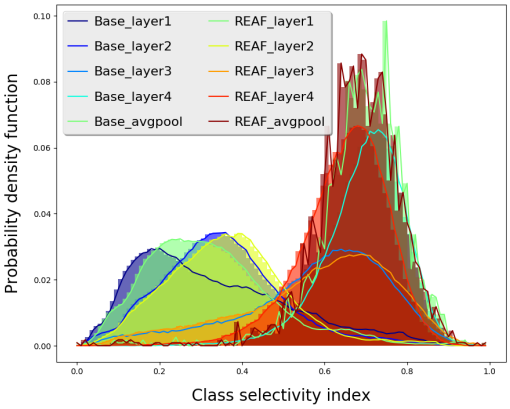


Figure 6.3: ResNet101+REAF-ResNet101.

6.4.4. EXPERIMENTS OF REAF+ SCHEME

As shown in Table 6.7, we apply the REAF+ scheme to a group convolution-based model and conduct experiments on the classification to show accuracy improvement. Applying the REAF+ scheme with $g_O = 2$ to the third convolution (i.e., the second pointwise convolution) of the bottleneck, the Top-1 accuracy of MobileNetV2 and ResNeXt50 on CIFAR-100 will improve by 0.55% and 0.43% respectively. The Top-1 accuracy of MobileNetV2 on ImageNet will increase by 0.69%.

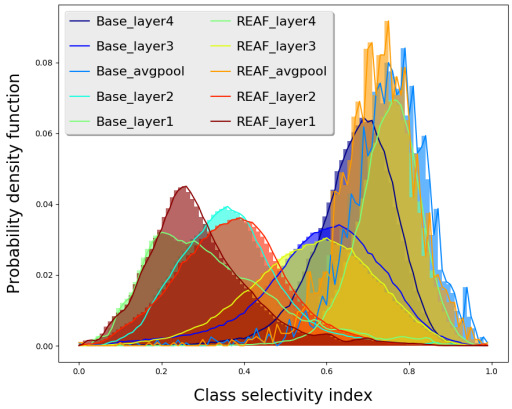


Figure 6.4: ResNet50+ResNeXt50.

6.4.5. EFFECTS ON LEARNED REPRESENTATION

In this section, we compare the class selectivity index for the features in the ResNet baseline and REAF-ResNet to interpret the effect of our proposed scheme on learned representation. The class selectivity index is computed for every channel of the feature maps, as *selectivity* $= (u_{max} - u_{-max})/(u_{max} + u_{-max})$. u_{max} represents the highest class-conditional mean activity and u_{-max} represents the mean of class-conditional mean activity across all other classes over given data distribution. Selectivity provides the degree to which features are being shared across classes, which is a central property of distributed representations and measure the extent of feature reuse. Using the validation dataset of the ImageNet, we calculate the selectivity for the features of "layer1", "layer2", "layer3", "layer4", and "avgpool", which are the output of the stage2, stage3, stage4, stage5, and AvgPool2d, respectively.

One trend is that the selectivity for the features of deep layers is more than that of earlier layers, which conforms to the observations in [135, 156] and can be attributed to the results of feature reuse *between* convolutional layers. Another trend identified in our work is that our REAF scheme increases the selectivity for the features, which verifies its reduction of feature reuse *within* the convolution. The distribution of the selectivity for ResNet and REAF-ResNet appears to be closely matched, so we only analyze the layer, at which the distribution separates most. Figures 6.2 and 6.3 present the selectivity comparison for the features in ResNet and REAF-ResNet for 50 and 101 layers. The most distinct distribution of the class selectivity appears at "layer1" in Figures 6.2 and 6.3 and the selectivity for the features of the "layer1" in REAF-ResNet are more than that in ResNet. We compare the selectivity for the features in ResNet50 and ResNeXt50 (i.e., REAF-ResNet50 with the configuration of $G_M = 1$) in Figure 6.4, respectively. Since ResNeXt is an example of our REAF scheme, its learned representation is expected to increase the selectivity. The largest mismatch of the selectivity distribution for the features between ResNet50 and ResNeXt50 falls to "layer4", where ResNeXt50 exhibits more selectivity that ResNet50.

6.5. CONCLUSIONS

In this chapter, we analyze the approximation problem introduced by the full use of feature reuse *within* the convolution, where the channels of the output activations are computed by reusing the same input activations. To make computing channels more accurate, we propose the REAF scheme, which is a moderate feature reuse version for convolution. Moreover, the configuration of the REAF scheme is optimized for a given CNN. Besides, we clarify the advantage of keeping more merged channels over group convolution. Also, we develop the REAF+ scheme and integrate it with the group convolution-based models. Last but not least, we present sufficient experiments on the classification to support our analysis concerning the REAF and REAF+ schemes.

7

ATTENTION MODULE

Convolution can extract features by fusing spatial and channel-wise information within a local receptive field, which matches well with the statistics of the natural images and makes convolutional neural networks the most successful paradigm in many computer vision tasks. On the contrary, operating in a local neighborhood prevents the convolution from capturing long-range information. Attention mechanisms have been regarded as an advanced technique to capture long-range feature interactions and to boost the representation capability for convolutional neural networks. However, we found two ignored problems in current attentional activations-based models: the approximation problem and the insufficient capacity problem of the attention maps. To solve the two problems together, we propose an attention module for convolutional neural networks by developing an AW-convolution, where the shape of attention maps matches that of the weights rather than the activations. Our proposed attention module is a complementary method to previous attention-based schemes, such as those that apply the attention mechanism to explore the relationship between channel-wise and spatial features. Experiments on several datasets for image classification and object detection tasks show the effectiveness of our proposed attention module. In particular, our proposed attention module achieves 1.00% *Top-1 accuracy improvement on ImageNet classification over a ResNet101 baseline and* 0.63 *COCO-style Average Precision improvement on the COCO object detection on top of a Faster R-CNN baseline with the backbone of ResNet101-FPN. When integrating with the previous attentional activations-based models, our proposed attention module can further increase their Top-1 accuracy on ImageNet classification by up to* 0.57% *and COCO-style Average Precision on the COCO object detection by up to* 0.45.

The content of this chapter is based on [157].

7.1. INTRODUCTION

Convolutional neural networks have demonstrated to be the gold-standard to solving various problems in the field of computer vision, including image classification [22], object detection [158, 159], and segmentation [118]. To improve their performance, many researchers explored various aspects of CNN design and implementation [160].

To enrich the representation power of CNNs, we can build substantially deeper convolutional neural networks. For example, VGGNet [161] stacks very small 3×3 convolutional layers, while ResNet [22] stacks residual blocks with skip connections. GoogLeNet [71] uses multi-scales of processing to capture spatial correlation. Wide ResNet [162] shows that the width increase of residual networks can enlarge the representation capability and reuse the features better. Xception [73] and ResNeXt [132] expose new dimensions to increase cardinalities. Besides, recent literature [163–165] have investigated the attention mechanism since it can improve not only the representation power but also the representation of interests. Convolutional neural networks can extract informative features by blending cross-channel and spatial information [135]. Attention modules [166, 167] can learn "where" and "what" to attend in channel and space axes, respectively, by focusing on important features and suppressing unnecessary ones of the activations. Dynamic Filter Networks [124, 168] generate the filters conditioned on the input and show the flexibility power of such filters because of their adaptive nature, which has become popular in prediction [169] and Natural Language Processing [170]. Both Dynamic Filter Networks and attention-based models are adaptive based on the inputs, but there are significant differences between them. Attention-based models [135, 166] produce attention maps using the attention mechanism to operate on the activations of convolution. On the contrary, Dynamic Filer Networks [171, 172] generate input information-specific kernels, such as position-specific kernels [171], semantic label map-specific kernels [172], and few-shot learning setting-specific kernels [173], which work as the weights of convolution. Our proposed attention module leverages the attention mechanism to compute the attention maps for attending the activations of convolution, so it is clear to categorized the models applied with our proposed attention module as attention-based models instead of Dynamic Filter Networks.

Based on our analysis, the approximation problem and the insufficient capacity problem of the attention maps are ignored in current attentional activations-based models. Motivated by solving the two problems together, we develop an attention module and inspect the complementary relationship between our proposed attention module and previously published attention-based models, such as the attention augmented models and the attentional activations-based models. Our contributions are summarized as follows.

- We point out and analyze two ignored problems of the current attentional activations-based models: the approximation problem and the insufficient capacity problem of the attention maps. To address the two problems together, we originally propose an attention module by developing an AW-convolution, where the shape of the attention maps matches that of the weights instead of the activations.

- Our proposed attention module is a complementary method to previous attention mechanism-based modules, such as Attention Augmented (AA) convolution [174], the SE [175] and CBAM [166] modules in the attentional activations-based models.

Integrating with our proposed attention module, the accuracy of AA-Net, SE-Net, and CBAM-Net will be improved further.

- We use image classification and object detection tasks to demonstrate the effectiveness of our proposed attention module. With negligible computational complexity increase, our proposed attention module can boost the image classification and object detection task performance, and it can achieve better accuracy when integrating with other attention-based models.

7.2. RELATED WORK

In this section, we discuss the recent developments of network engineering and attention mechanism.

7.2.1. NETWORK ENGINEERING

"Network engineering" has been one of the most active research areas since it targets building powerful convolutional neural networks on image classification, which are the backbones of various computer vision tasks and ensure their remarkable performance [122]. Increasing the depth of convolutional neural networks has been regarded as an intuitive way to boost performance, which is the philosophy of VGGNet [161] and ResNet [22]. In addition, since the skip connection from ResNet shows a strong ability to assist the gradient flow, WideResNet [162], PyramidNet [176], Inception-ResNet [72], and ResNeXt [127, 132] are ResNet-based versions proposed to explore further the influence of the width, the increase of the width, the multi-scale and the cardinality of convolution, respectively. In terms of efficiency, DenseNet [23] reuses the feature maps by concatenating the feature maps from different layers. In particular, MobileNet [153, 177] and ShuffleNet [76] series present the advantage of depthwise convolution and the shuffle operation between various group convolutions, respectively. Another design approach uses automated neural architecture search, which achieves state-of-the-art performance regarding both accuracy and efficiency across a range of computer vision tasks [81].

7.2.2. ATTENTION MECHANISM

The attention mechanism plays an important role in the human vision perceptron since it can allocate the available resources to selectively focus on processing the salient part instead of the whole scene [178, 179]. Multiple attention mechanisms are used to address a known weakness in convolution [71, 152, 167, 175, 180, 181], by capturing long-range information interactions [182, 183]. The Inception family of architectures [71, 72], Multi-grid Neural Architectures [181], and Octave Convolution [152] aggregate the scale-space information, while Squeeze-and-Excitation Networks [175] and Gather-Excite [135] adaptively recalibrate channel-wise response by modeling interdependency between channels. GALA [167], CBAM [166], and BAM [184] refine the feature maps separately in the channel and spatial dimensions. Attention Modules [185] and self-attention [174, 186] can be used to exploit global context information. Precisely, non-local networks [187] deploy self-attention as a generalized global operator to capture the relationship between all pairwise convolutional feature maps interactions. Except for applying the attention mechanism to

computer vision tasks [188], it has been a widespread adoption to modeling sequences in Natural Language Processing [189].

7.3. PROPOSED ATTENTION MODULE

In this section, we analyze the two ignored problems in current attentional activations-based models: the approximation problem and the insufficient capacity problem of the attention maps. To address the two problems together, we develop an attention module that mainly refers to the AW-convolution, where the shape of attention maps matches that of the weights rather than the activations. Besides, we refine the branch of calculating the attention maps to achieve a better trade-off between efficiency and accuracy. Last but not least, we integrate our proposed attention module with other attention-based models to enlarge their representational capability.

7.3.1. MOTIVATION

First, we define basic notations in a traditional convolutional layer. In a traditional convolutional layer, the input activations, weights, and output activations are denoted as I, K, and O, respectively. For the input activations $I \in R^{N\times C_1\times H\times W}$, N, C_1, H, and W refer to the batch size, the number of input channels, the height, and width of the input feature maps, respectively. For the weights $K \in R^{C_2\times C_1\times h\times w}$, C_2, h and w refer to the number of output channels, the height and width of the weights, respectively. For the output activations $O \in R^{N\times C_2\times H\times W}$, it is computed as the convolution between the input activations I and the weights K. In particular, every individual value of the output activations $O_{[l,p,m,n]}$ is calculated as follows.

$$O_{[l,p,m,n]} = \text{Convolution}(I,K) = \sum_{o=1}^{C_1}\sum_{j=1}^{h-1}\sum_{k=1}^{w-1} I_{[l,o,m'+j,n'+k]} \times K_{[p,o,j,k]} \tag{7.1}$$

where $l = 0,...,N-1$, $m = 0,...,H-1$, $n = 0,...,W-1$, $o = 0,...,C_1-1$, $p = 0,...,C_2-1$, $m' = m - \frac{h-1}{2}$, $n' = n - \frac{w-1}{2}$.

To apply the attention mechanism on the input activations I, previous attentional activations-based models produce the channel attention maps $A_c \in R^{N\times C_1\times 1\times 1}$ and spatial attention maps $A_s \in R^{N\times 1\times H\times W}$ separately. For example, applying the channel attention maps A_c on the input activations I is presented as $O = \text{Convolution}((I \odot A_c), K)$, where $\odot$ refers to the Hadamard product and broadcasting during element-wise multiplication is omitted.

Approximation problem of the attention maps: Instead of directly computing the three-dimensional attention map (N is omitted, otherwise the attention maps are of four dimensions.), all the current attentional activations-based models produce the attention maps separately into the channel attention maps A_c and spatial attention maps A_s, which leads to the approximation problem of attention maps. However, to thoroughly attend the input activations I, we need to compute the attention maps $A_f \in R^{N\times C_1\times H\times W}$ and apply it as $O = \text{Convolution}((I \odot A_f), K)$, which requires too much computational and parameter overhead.

Inspired by convolution, we adopt local connection and attention maps sharing to reduce the size of the attention maps. We compute the attention maps $A_a \in R^{N\times C_1\times h\times w}$ as follows, where $\otimes$ is a special element-wise multiplication since it only works associated with convolution.

$$\begin{aligned} O_{[l,p,m,n]} &= \text{Convolution}(I \otimes A_a, K) \\ &= \sum_{o=1}^{C_1}\sum_{j=1}^{h-1}\sum_{k=1}^{w-1}(I_{[l,o,m'+j,n'+k]} \times A_{a[l,o,j,k]}) \times K_{[p,o,j,k]} \end{aligned} \tag{7.2}$$

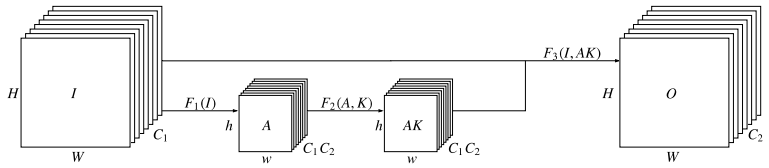
Insufficient capacity problem of the attention maps To compute different channels of the output activations of the convolution, the input activations are constrained to be recalibrated by the same attention maps, which indicates the insufficient capacity of the attention maps. As each channel of the feature maps is considered as a feature detector, different channels of the output activations of the convolution expect the input activations to be adapted by different attention maps.

Take two channels of output activations of a convolutional layer as an example, the two channels are responsible for recognizing rectangle shape and triangle shape, respectively. Thus, it is reasonable for the two channels to expect that there are different attention maps for attending the input activations of the convolution (i.e., the attention maps to compute the channel of recognizing the rectangle shape should be different from the attention maps to compute the channel of recognizing the triangle shape). To meet this expectation, we need to compute the attention maps $A_{ic} \in R^{N\times C_2\times C_1\times 1\times 1}$ and apply it on the input activations as follows.

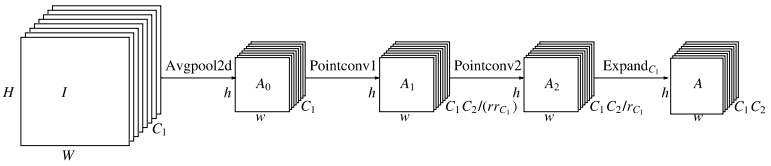
$$\begin{aligned} O_{[l,p,m,n]} &= \text{Convolution}(I \odot A_{ic[l,p,:,:,:]}, K) \\ &= \sum_{o=1}^{C_1}\sum_{j=1}^{h-1}\sum_{k=1}^{w-1}(I_{[l,o,m'+j,n'+k]} \times A_{ic[l,p,o,0,0]}) \times K_{[p,o,j,k]} \end{aligned} \tag{7.3}$$

To solve the approximation problem and the insufficient capacity problem of the attention maps together (i.e., combining the solution of Equation.7.2 and the solution of Equation. 7.3), we introduce our proposed attention module by developing the AW-convolution. Specifically, we propose to compute the attention maps $A \in R^{N\times C_2\times C_1\times h\times w}$ and apply it as follows where the attention maps $A_{[l,:,:,:,:]}$ has the same shape as that of the weights instead of the input activations. In this chapter, "Attentional weights" refers to the element-wise multiplication result between the attention maps and the weights. Similarly, "Attentional activations" refers to the element-wise multiplication result between the attention maps and the activations in previous attentional activations-based models. Thus, $I \otimes A$ and $A_{[l,:,:,:,:]} \odot K$ represent the attentional activations and attentional weights, respectively. To reduces half the number of element-wise multiplications, we calculate attentional weights instead of attentional activations as follows.

$$\begin{aligned} O_{[l,p,m,n]} &= \text{Convolution}(I \otimes A, K) \\ &= \sum_{o=1}^{C_1}\sum_{j=1}^{h-1}\sum_{k=1}^{w-1} I_{[l,o,m'+j,n'+k]} \times (A_{[l,p,o,j,k]} \times K_{[p,o,j,k]}) \\ &= \text{Convolution}(I, A_{[l,:,:,:,:]} \odot K) = \text{AW-Convolution}(I, A \odot K) \end{aligned} \tag{7.4}$$



(a) The AW-convolution architecture.



(b) The architecture of calculating attention maps A.

Figure 7.1: The architecture of our proposed attention module.

7.3.2. AW-CONVOLUTION IN PROPOSED ATTENTION MODULE

The AW-convolution in our proposed attention module is presented in Figure 7.1a. In this figure, the attention maps A has five dimensions, which is computed from the input activations I as $A = F_1(I)$. F_1 is a function to calculate the attention maps A given the input activations I. Then, the attentional weights $AK \in R^{N\times C_2\times C_1\times h\times w}$ is calculated as $AK = F_2(A, K) = K + A \odot K$. F_2 is a function to calculate the attentional weights AK given the weights K and the attention maps A. Finally, the output activations O is calculated from the input activations I and the attentional weights AK as follows.

$$\begin{aligned} O_{[l,p,m,n]} &= F_3(I, AK) = \text{AW-Convolution}(I, AK) \\ &= \sum_{i=1}^{C_1}\sum_{j=1}^{h-1}\sum_{k=1}^{w-1} I_{[l,o,m'+j,n'+k]} \times AK_{[l,p,o,j,k]} = \text{Convolution}(I, AK_{[l,:,:,:,:]}) \end{aligned} \tag{7.5}$$

where F_3 is a function to calculate the output activations O given the input activations I and the attentional weights AK. Compared with the traditional convolution, the attentional weights AK of the AW-convolution in our proposed attention module has five dimensions rather than four dimensions, which are different from each other for every individual sample of the input activations batch to convolute.

It is also worth explaining the definition of the function F_2. $AK = K + A \odot K$ instead of $AK = A \odot K$ is used to describe the function F_2 since it can be regarded as a residual design as follows.

$$\begin{aligned} O &= F_3(I, AK) = \text{AW-Convolution}(I, F_2(A, K)) \\ &= \text{Convolution}(I, K) + \text{AW-Convolution}(I, A \odot K) \end{aligned} \tag{7.6}$$

7.3.3. CALCULATING THE ATTENTION MAPS

As shown in Figure 7.1b, the architecture to compute the attention maps A (i.e., the definition of the function F_1) is presented, which can be expressed as follows. Avgpool2d

aggregates feature responses from the whole spatial extent and embeds them into A_0, and Pointconv1 and Pointconv2 followed by Relu redistribute the pooled information to capture the dynamic and no-linear dependencies between channels and spatial spaces.

$$\begin{aligned} A = F_1(I) &= \text{Expand}_{C_1}(A_2) = \text{Expand}_{C_1}(\text{Pointconv2}(A_1)) \\ &= \text{Expand}_{C_1}(\text{Pointconv2}(\text{Pointconv1}(A_0))) \\ &= \text{Expand}_{C_1}(\text{Pointconv2}(\text{Pointconv1}(\text{Avgpool2d}(I)))) \end{aligned} \tag{7.7}$$

where Pointconv1 and Pointconv2 are pointwise convolutions. We add Batch Normalization and Relu layers after Pointconv1, while adding Batch Normalization and Sigmoid layers after Pointconv2, and they are omitted here to provide a clear expression.

In Figure 7.1b, Expand function along C_1 dimension, denoted as Expand_{C_1}, is used as an example, and Expand function can be also executed along N, C_2, h, and w dimensions in a similar way. Expand_{C_1} function is used to expand the tensor $A_2 \in R^{N\times(C_2C_1/r_{C_1})\times h\times w}$ into the attention maps $A \in R^{N\times C_2\times C_1\times h\times w}$ with the reduction ratio r_{C_1}, including necessary squeeze, reshape, and expand operations. Expand_{C_1} can be expressed as follows.

$$\begin{aligned} A = \text{Expand}_{C_1}(A_2) = A_2.\text{reshape}(N, C_2, C_1/r_{C_1}, h, w).\text{unsqueeze(dim=3)} \\ .\text{expand}(N, C_2, C_1/r_{C_1}, r_{C_1}, h, w).\text{reshape}(N, C_2, C_1, h, w) \end{aligned} \tag{7.8}$$

Calculating the five-dimension attention maps A is not an easy computational task without careful design. Thus, we analyze the additional computational complexity of an AW-convolution compared with a traditional convolution as a reference to refine this design. Considering the trade-off between computational complexity and accuracy, all the experiments in the remainder of this chapter use the same settings for the architecture of calculating the attention maps A in our proposed attention module, including $r_{C1} = C_1$, $r_{C_2} = r_{hw} = 1$, $r = 16$, used in all the stages, and $AK = K + A \odot K$ as the definition for the function F_2.

7.3.4. REFINING THE ARCHITECTURE OF CALCULATING THE ATTENTION MAPS

Compared with a traditional convolution, the additional parameters P_{AW} introduced in the AW-convolution are included in the Pointconv1, Pointconv2, and Batch Normalization layers, as follows.

$$P_{AW} = C_1^2C_2^2/(rr_{C_1}^2) + C_1^2C_2/(rr_{C_1}) + 2C_1C_2(1/r+1)/r_{C_1} \tag{7.9}$$

where r is a reduction ratio between the two pointwise convolutions, and r_{C_1} is the one in the Expand_{C_1} function, reducing the computational complexity.

The additional number of FLOPs F_{AW} needed to compute Avgpool2d, Pointconv1, Pointconv2, Batch Normalization, non-linear functions, and elementwise-multiplication, is calculated as follows.

$$F_{AW} = HWC_1 + 9C_1^2C_2^2/(r_{C_1}^2 r) + 9C_1^2C_2/(r_{C_1} r) + 45C_1C_2(1/r+1)/r_{C_1} + 9C_1C_2 \tag{7.10}$$

To refine the architecture of calculating the attention maps A, we attempted several designs to reduce the computational complexity and minimize the accuracy drop. In particular, we mainly explore the design space by expanding along different dimensions. Besides,

Table 7.1: Comparisons of AW-ResNet50 on CIFAR-100 image classification with different settings for the architecture of calculating the attention maps A. The default settings in this table are: $r_{C_1} = r_{C_2} = r_{hw} = r = 1$, used in all the stages, and $AK = K + A \odot K$ as the definition of the function F_2.

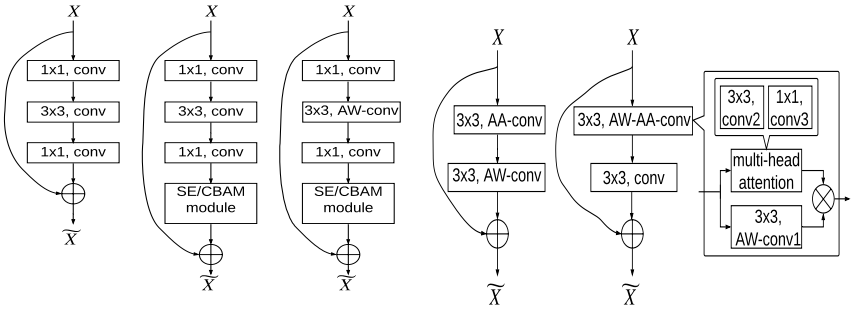
Model	Top-1 Error	Top-5 Error	GFLOPs	Parameters (M)
ResNet50 Baseline [22]	22.33%	5.83%	1.22	23.71
AW-ResNet50 ($r_{C_1} = C_1/2$)	19.51%	4.65%	1.30	31.28
AW-ResNet50 ($r_{C_2} = C_2/2$)	20.40%	4.97%	1.30	31.28
AW-ResNet50 ($r_{C_1} = C_1/2$, $r_{hw} = hw$)	20.37%	5.13%	1.24	31.28
AW-ResNet50 (s2, $r_{C_1} = C_1/2$)	20.57%	5.10%	1.22	23.78
AW-ResNet50 (s3, $r_{C_1} = C_1/2$)	20.57%	5.01%	1.22	24.10
AW-ResNet50 (s4, $r_{C_1} = C_1/2$)	20.99%	5.19%	1.25	26.08
AW-ResNet50 (s5, $r_{C_1} = C_1/2$)	21.18%	5.40%	1.27	28.44
AW-ResNet50 ($r_{C_1} = C_1/2$, $AK = A \odot K$)	20.14%	4.82%	1.30	31.28
AW-ResNet50 ($r_{C_1} = C_1$, $r = 16$)	19.87%	4.76%	1.23	23.87

we explore the influence of our proposed attention module for every single stage, which refers to a collection of bottlenecks operating on the feature maps with a universal spatial dimension. Also, we investigate the definition of the function F_2. Finally, we adjust the hyperparameter reduction ratios to achieve a better trade-off between computational complexity and accuracy. The refined architecture is suggested based on the experimental results of ResNet50 [22] on CIFAR-100 image classification [190], where we adapt our proposed attention module by replacing a 3×3 traditional convolution with our proposed 3×3 AW-convolution in the bottlenecks.

According to the results shown in Table 7.1, we get a couple of indications. Expand_{C_1} is a better choice (i.e., 0.89%) than Expand_{C_2}, while Expand_{hw} will lead to an accuracy drop (i.e., 0.86%) because of the loss of capturing spatial information for attentional weights. Applying our proposed attention module in the early stage will give a better performance, and we achieve the best result when all the stages adopt our proposed attention module. Using $AK = K + A \odot K$ as the definition of the function F_2 provides a better result (i.e., 0.63%) than $AK = A \odot K$ since the residual design can help the training of deep models. Considering the trade-off between computational complexity and accuracy, all the experiments in the remainder of this chapter use the same settings for the architecture of calculating the attention maps A in our proposed attention module, including $r_{C1} = C_1$, $r_{C_2} = r_{hw} = 1$, $r = 16$, used in all the stages, and $AK = K + A \odot K$ as the definition for the function F_2.

After refining the architecture, the additional parameters P_{AW} and FLOPs F_{AW} introduced in an AW-convolution are as follows.

$$\begin{aligned} P_{AW} &= C_2^2/r + C_1C_2/r + 2C_2(1/r + 1) \\ F_{AW} &= HWC_1 + 9C_2^2/r + 9C_1C_2/r + 45C_2(1/r + 1) + 9C_1C_2 \end{aligned} \tag{7.11}$$



(a) Integrating with SE-ResNet/CBAM-ResNet.

(b) Integrating with AA-Wide-ResNet.

Figure 7.2: The schema of bottlenecks and blocks when integrating with our proposed attention module.

7.3.5. INTEGRATING WITH OTHER ATTENTION-BASED MODULES

In this section, we show how to integrate our proposed attention module with the previous attention-based convolutional neural networks to demonstrate the complementary relationship between our proposed attention module and other attention-based modules. Since applying our proposed attention module is using the AW-convolution to replace the traditional convolution, we can easily integrate our proposed attention module with any convolutional neural networks consisting of traditional convolution, including all the recently developed attention-based models [166, 167, 174, 175, 184].

We choose the recent attentional activations-based models, i.e., SE-Net and CBAM-Net, as examples to show how to integrate our proposed attention module with other attention-based models. Here we use the popular ResNet [22] as the backbone to apply the attention mechanism. As shown in Figure 7.2a, the left side is the structure of a primary bottleneck in ResNet. The middle one is the structure of a bottleneck with SE/CBAM modules in SE-ResNet/CBAM-ResNet. Integrating the central bottleneck with our proposed attention module is completed by replacing its 3×3 convolution with a 3×3 AW-convolution, and its final structure in AW-SE-ResNet/AW-CBAM-ResNet is shown on the right side. In summary, our proposed attention module is a general module to be integrated seamlessly with any CNNs architectures, including previous attention-based CNNs.

In Figure 7.2b, we integrate our proposed attention module with Attention Augmented (AA) convolutional networks and describe the architecture of their possible AW-AA-blocks. Experiments of AW-AA-Wide-ResNet [162, 174] on CIFAR-100 image classification [190], as shown in Table 7.3, suggest that integrating our proposed attention module with attention-based models should be explored carefully since different integration architectures achieve different accuracy. In some cases, an improper integration architecture leads to a small accuracy drop. With a careful design for integration, our proposed attention module is complementary to AA convolution and improves the accuracy of AA-Net further.

In particular, we augment Wide-ResNet-28-10 by replacing the first convolution of all the residual blocks with Attention Augmented convolution, which is the AA-Wide-ResNet baseline. Here we set $N_h = 8$ heads, $k = 2v = 0.2$, and a minimum of 20 dimensions per

Table 7.2: Comparisons of attention-based models on ImageNet classification. * refers to the baseline results from [166]. All the rest results are produced using the source code from [166].

Model	Top-1 Error	Top-5 Error	GFLOPs	Parameters (M)
ResNet50 [22] *	24.56%(+0.00%)	7.50%	3.86	25.56
AW-ResNet50	23.38%(+1.18%)	6.79%	3.87	25.72
SE-ResNet50 [175] *	23.14%(+1.42%)	6.70%	3.87	28.09
AW-SE-ResNet50	22.72%(+1.84%)	6.47%	3.88	28.25
AW-CBAM-ResNet50 (MaxPool)	22.82%(+1.74%)	6.41%	3.89	28.25
AW-CBAM-ResNet50 (Spatial)	23.20%(+1.36%)	6.58%	3.90	28.25
ResNet101 Baseline [22] *	23.38%(+0.00%)	6.88%	7.57	44.55
AW-ResNet101	22.38%(+1.00%)	6.21%	7.58	44.95
SE-ResNet101 [175] *	22.35%(+1.03%)	6.19%	7.58	49.33
AW-SE-ResNet101	21.78%(+1.60%)	5.74%	7.59	49.73
AW-CBAM-ResNet101 (MaxPool)	21.64%(+1.74%)	5.76%	7.60	49.73
AW-CBAM-ResNet101 (Spatial)	22.32%(+1.06%)	6.18%	7.61	49.73
MobileNet Baseline [153] *	31.39%(+0.00%)	11.51%	0.569	4.23
SE-MobileNet [175] *	29.97%(+1.42%)	10.63%	0.581	5.07
AW-SE-MobileNet	29.41%(+1.98%)	10.59%	0.623	5.52
CBAM-MobileNet [166]	29.01%(+2.38%)	9.99%	0.611	5.07
AW-CBAM-MobileNet (Spatial)	28.82%(+2.57%)	9.98%	0.652	5.52

head for the keys as in [174]. In this figure, we can develop four possible architectures to integrate our proposed attention module with AA-Wide-ResNet. On the left side of this figure, we can build AA-Wide-ResNet-0 by replacing the second 3×3 convolution with our AW-convolution in an AA-block. On the right side, there are three traditional convolutions in an AA convolution, including conv1, conv2, conv3. The conv1 is parallel to the multi-head attention, while the conv2 and conv3 are used to calculate QKV (i.e., queries, keys, and values) and to output attention maps, respectively. On the medial side, we can replace one traditional convolution from conv1, conv2, or conv3 with our AW-conv1, AW-conv2, or AW-conv3 to construct AW-AA-Wide-ResNet-1, AW-AA-Wide-ResNet-2, or AW-AA-Wide-ResNet-3, respectively. The Top-1 accuracy of AW-AA-Wide-ResNet-0 is 1.87% higher than the AA-Wide-ResNet baseline, while AW-AA-Wide-ResNet-3 shows worse performance by 0.10% drop.

7.4. EXPERIMENTAL RESULTS

In this section, we use extensive experiments to demonstrate the effectiveness of our proposed attention module. We use ResNet [22], MobileNet [153], SSD300 [158], and Faster R-CNN [159] as the baseline models, and various variants of these models are developed, including using our proposed attention module for these baseline models and integrating our proposed attention module with their attentional activations-based models. The datasets to train these models include CIFAR-100 classification [190], ImageNet classification [191],

VOC object detection datasets [192], and COCO object detection datasets [193].

7.4.1. DATA AUGMENTATION AND TRAINING SETTINGS

All the experiments in our work are implemented with the same settings for our proposed attention module, including $r_{C1} = C_1$, $r_{C_2} = r_{hw} = 1$, $r = 16$, used in all the stages, and $AK = K + A \odot K$.

Data augmentation and training setting for CIFAR-100 classification. CIFAR-100 classification dataset [190] contains 50K training images and 10K test images and consists of 32×32 color images drawn from 100 classes. Images are first zero-padded on each side with four pixels. A random 32×32 patch is cropped from its padded image or its horizontal flip before applying mean/std normalization. During testing, we use only mean/std normalization. We set the batch size to 128 and use an SGD optimizer with a momentum value of 0.9 and weight decay $5 \cdot 10^{-4}$. We train 200 epochs in total, and the learning rate is started at 0.1 and decayed at steps of 60, 120, and 160 by a decay factor of 0.2.

Data augmentation and training setting for ImageNet classification. ImageNet classification dataset [191] consists of 1.2 million images in the training dataset and 50K images in the validation dataset. During the training, a 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted. When validating, we perform data augmentation by resizing a shorter edge to 256 and center-cropping to 224×224 pixels, and similarly, normalizing the input images with mean channel subtraction. We train 90 epochs with a batch size of 256 and use an SGD optimizer with a momentum of 0.9 and a weight decay of $1 \cdot 10^{-4}$. The learning rate starts from 0.1 and will be divided by 10 every 30 epochs. We report singe-crop results on the validation dataset.

Data augmentation and training setting for PASCAL VOC2007. PASCAL VOC2007 [192] consists of 20 classes and is a popularly used benchmark for object detection. VGG16 is pre-trained on the ImageNet classification dataset [191], and we finetune the resulting SSD300 model using an SGD optimizer with a momentum value of 0.9 and a weight decay of $5e-4$, and a batch size of 32. We train 120K steps in total and schedule the learning rate to start at $1e-3$ and decay at steps of 80K and 100K by a decay factor of 0.1. We train SSDLite300 with MobileNet as the backbone for 200 epochs, and the learning rate is started at 0.01 and scheduled by a cosine scheduler.

Data augmentation and training setting for COCO. COCO [193] is a large-scale object detection dataset, including 80 object categories. ResNet101 is pre-trained on the ImageNet classification, which is loaded in Faster R-CNN to finetune. All the batch normalization in the AW-convolution of FPN is replaced with group normalization with the number of channels as 16. We train $90K$ iterations in total with a batch size of 8, an SGD optimizer with the momentum value of 0.9, and a weight decay of $1e-4$. The base learning rate is 0.01, and a linear warming up is used in the first $1K$ iterations with a factor of 0.001. The base learning rate is decayed at the step of $70K$ and $83.333K$ with a factor of 0.1.

7.4.2. COMPUTATION COMPLEXITY

In this section, we explain how we calculate the additional computation complexity for the SE module [175], CBAM module [166], and AW-convolution in our work. A residual bottleneck is the basic unit of ResNet with more than 50 layers [22]. Taking a bottleneck in ResNet [175] as an example, we use FLOPs and the number of parameters as the metrics to report the computation complexity of convolution. We take all the additional operations into considerations, including pooling, no-linear function, etc.

Computation complexity for the SE module. In a bottleneck of the SE-Net, we use the additional FLOPs to compute the Average pooling, Full-connected, Relu, Sigmoid, Scale, and Elementwise multiplication. We calculate the additional computation complexity to introduce the SE module for convolution as follows.

$$F_{SE} = 2HWC_s + 2C_s^2/r + C_s/r + C_s \tag{7.12}$$

where r, C_s, H, and W refers to the reduction ratio, the dimension of the output channels, and the height and width of the output activations, respectively. It is worth to note here that $C_s = e \times C$, and $e = 4$ is the expansion ratio of the bottleneck..

The additional parameters P_{SE} introduced for a bottleneck come from the two fully-connected layers of the gating mechanism as follows.

$$P_{SE} = 2(eC)^2/r + eC(1/r + 1) \tag{7.13}$$

Computation complexity for the CBAM module. In a bottleneck of the CBAM-Net, we need to compute the channel and spatial attention maps separately. To calculate the channel attention maps, we need to introduce the following operations, including the Average pooling, Max pooling, Full-connected, Relu, Sigmoid, Scale, and Elementwise multiplication. To calculate the spatial attention maps, we need to introduce the following operations, including the Average pooling, Max pooling, Convolution, Batch Normalization, Sigmoid, Scale, and Elementwise multiplication. We calculate the increased Flops as follows.

$$F_{CBAM} = 6HWC_s + 4C_s^2/r + 2C_s/r + 2C_s + 105HW \tag{7.14}$$

The additional parameters P_{CBAM} introduced for a bottleneck come from the two fully-connected layers of the gating mechanism as follows.

$$P_{CBAM} = 2(eC)^2/r + eC(1/r + 1) \tag{7.15}$$

Computation complexity for the AW-convolution. In a bottleneck of our AW-Net, we require to add the following operations, including the Average pooling, Pointconv1 convolution, Pointconv2 convolution, Batch Normalization, non-linear functions, scale, and elementwise-multiplication. To apply our proposed attention module on a bottleneck, we calculate the number of FLOPs as follows.

$$F_{AW} = HWC + 18C^2/r + 45C(1/r + 1) + 9C^2 \tag{7.16}$$

Table 7.3: Comparisons of attention-based models on CIFAR-100 classfication.

Model	Top-1 Error	Top-5 Error	GFLOPs	Parameters (M)
AA-Wide-ResNet Baseline [174]	28.01%(+0.00%)	7.92%	3.89	8.43
AW-AA-Wide-ResNet-0	26.14%(+1.87%)	7.43%	3.90	8.50
AW-AA-Wide-ResNet-1	27.17%(+0.84%)	7.49%	3.89	8.48
AW-AA-Wide-ResNet-2	27.09%(+0.92%)	8.00%	3.89	8.45
AW-AA-Wide-ResNet-3	28.11%(−0.10%)	8.24%	3.89	8.44
ResNet50 Baseline [22]	22.33%(+0.00%)	5.83%	1.22	23.71
AW-ResNet50	19.87%(+2.46%)	4.76%	1.23	23.87
SE-ResNet50 [175]	20.43%(+1.90%)	5.01%	1.23	26.24
AW-SE-ResNet50	19.00%(+3.33%)	4.51%	1.24	26.40
CBAM-ResNet50 [166]	19.46%(+2.87%)	4.56%	1.24	26.24
AW-CBAM-ResNet50	18.94%(+3.39%)	4.76%	1.25	26.40

The AW-convolution in a bottleneck increases the parameters compared with a traditional convolution, as follows.

$$P_{AW} = 2C^2/r + 2C(1/r + 1) \tag{7.17}$$

Considering the analysis of the bottleneck above, the number of parameters when adopting the SE or CBAM module is approximately $e^2 = 16$ times that when applying our proposed attention module.

7.4.3. IMAGENET IMAGE CLASSIFICATION

To investigate the performance of our proposed attention module on high-resolution images, we train ResNet50 and ResNet101 [22] and their attention-based variants on the ImageNet classification dataset [191]. According to the results shown in Table 7.2, our proposed attention module is complementary to other attentional activations-based models. AW-ResNet50 achieves a 1.18% Top-1 error reduction compared with the ResNet50 baseline. Integrating with our proposed attention module, SE-ResNet50 [175] can improve further by 0.42% Top-1 accuracy. The Top-1 accuracy of our AW-SE-ResNet101 is 1.60% and 0.57% higher than that of ResNet101 and SE-ResNet101, respectively. To integrate with CBAM-ResNet [166] more carefully, we define CBAM-ResNet (MaxPool) and CBAM-ResNet (Spatial) separately. In CBAM-ResNet (MaxPool), we do not deploy the spatial attention maps, while we do not use max-pooled features in CBAM-ResNet (Spatial). The Top-1 accuracy of AW-CBAM-ResNet50 (MaxPool) and AW-CBAM-ResNet50 (Spatial) are better than AW-ResNet50 by 0.56% and 0.18%, respectively, but worse than AW-SE-ResNet50. The number of additional parameters for our proposed attention module is 0.16 M, which is much smaller than 2.83 M (i.e., one-sixteenth) of SE and CBAM modules. Moreover, it takes only 0.01 GFLOPs to apply our proposed attention module on the ResNet50 model on ImageNet classification, which is comparable with 0.01 GFLOPs and 0.04 to adopt the SE and CBAM modules and is negligible in terms of FLOPs to implement the baseline model.

Table 7.4: Comparisons of attention-based ResNet various depths on CIFAR-100 classfication.

Model	Top-1 Error	Top-5 Error	GFLOPs	Parameters (M)
ResNet18 Baseline [22]	22.33%	5.93%	0.557	11.22
AW-ResNet18	21.29%	6.08%	0.564	11.31
SE-ResNet18 [175]	22.10%	6.25%	0.558	11.31
AW-SE-ResNet18	21.84%	5.87%	0.565	11.40
CBAM-ResNet18 [166]	22.12%	6.22%	0.559	11.31
AW-CBAM-ResNet18	22.18%	6.37%	0.566	11.40
ResNet101 Baseline [22]	20.02%	4.88%	2.44	42.70
AW-ResNet101	19.59%	4.96%	2.47	43.01
SE-ResNet101 [175]	19.88%	4.94%	2.45	47.48
AW-SE-ResNet101	19.63%	4.72%	2.48	47.79
CBAM-ResNet101 [166]	19.70%	4.75%	2.47	47.48
AW-CBAM-ResNet101	19.07%	4.61%	2.49	47.79

Table 7.5: Comparisons of attention-based Faster R-CNN on COCO.

Backbone	Detector	mAP@[0.5, 0.95]	mAP@0.5	mAP@0.75
ResNet101-FPN [194]	Faster R-CNN	37.13(+0.00%)	58.28	40.29
ResNet101-AW-FPN	Faster R-CNN	37.76(+0.63%)	59.17	40.91
ResNet101-SE-FPN [175]	Faster R-CNN	38.11(+0.98%)	59.41	41.33
ResNet101-AW-SE-FPN	Faster R-CNN	38.45(+1.32%)	59.70	41.86
ResNet101-CBAM-FPN [166]	Faster R-CNN	37.74(+0.61%)	58.84	40.77
ResNet101-AW-CBAM-FPN	Faster R-CNN	38.19(+1.06%)	59.52	41.43

RESOURCE-CONSTRAINED ARCHITECTURE

Driven by demand for mobile applications, many depthwise convolution-based models are developed to take care of the trade-off between accuracy and efficiency. To inspect the generalization of our proposed attention module in this resource-constrained scenario, we conduct the ImageNet classification [191] with the MobileNet architecture [153]. Since our proposed attention module is efficient in terms of both the storage and computational complexity, integrating it into the light-weight architecture is worth exploring. We apply our proposed attention module to pointwise convolution instead of depthwise convolution in every two depthwise separable convolutions. When integrating with the CBAM models [166], we remove the max-pooled features and keep spatial attention maps. As shown in Table 7.2, AW-SE-MobileNet and AW-CBAM-MobileNet achieve 0.56% and 0.19% Top-1 accuracy improvements compared with SE-MobileNet [175] and CBAM-MobileNet, respectively. It is an impressive result that the Top-1 accuracy of AW-CBAM-MobileNet is 2.57% better than that of the MobileNet baseline. For the MobileNet model, our proposed attention module increases the computation by 0.041 GFLOPs, while SE and CBAM modules need 0.012 and 0.041 GFLOPs, respectively. Also, the required parameters for our

Table 7.6: Comparisons of attention-based SSD300 and SSDLite300 on VOC2007 test.

Backbone	Detector	mAP@0.5	GFLOPs	Parameters
VGG16	SSD300 [158]	77.40	35.20	26.29
VGG16	AW-SSD300	77.77	35.21	26.75
VGG16	SE-SSD300 [175]	77.70	35.20	26.51
VGG16	AW-SE-SSD300	78.25	35.21	26.97
VGG16	CBAM-SSD300 [166]	78.18	35.21	26.51
VGG16	AW-CBAM-SSD300	78.10	35.22	26.97
MobileNet	SSDLite300 [25]	68.71	0.800	3.39
MobileNet	AW-SSDLite300	69.09	0.812	4.00
MobileNet	SE-SSDLite300 [175]	68.85	0.803	3.69
MobileNet	CBAM-SSDLite300 [166]	68.40	0.809	3.69
MobileNet	AW-SE-SSDLite300	69.49	0.815	4.30

proposed attention module are 0.45 M, which is much less than 0.84 M for SE and CBAM modules.

7.4.4. CIFAR-100 IMAGE CLASSIFICATION

We perform experiments of the CIFAR-100, which is a standard benchmark for low-resolution images classification. According to the results shown in Table 7.3, we conclude that our proposed attention module can boost the CIFAR-100 [190] accuracy of both the ResNet50 baseline model [22] and their attentional activations-based models with negligible additional computational complexity. AW-ResNet50 achieves a 2.46% Top-1 error reduction compared with the ResNet50 baseline. Integrating with our proposed attention module, SE-ResNet50 [175] and CBAM-ResNet50 [166] can increase Top-1 accuracy by 1.43% and 1.52%, respectively. In terms of the computational complexity, our proposed attention module requires 0.01 GFLOPs, which is acceptable compared with 0.01 GFLOPs for the SE module, 0.02 GFLOPs for the CBAM module, and 1.22 GFLOPs for the baseline model. Besides, we only introduce 0.16 M parameters for our proposed attention module, which is less than 2.53 M parameters for the SE and CBAM modules.

Generalized to Networks with various depths. We train ResNet with various depths on CIRAR-100 image classification to show that our proposed attention module works for CNNs with different depths. As shown in Table 7.4, AW-ResNet18 achieves the highest Top-1 accuracy, scoring 1.04% better than the ResNet18 baseline. Integrating with our proposed attention module, the Top-1 accuracy of SE-ResNet18 improves further by 0.26%. Regarding deeper networks, the Top-1 accuracy of ResNet101, SE-ResNet101, and CBAM-ResNet101 increase by 0.43%, 0.15%, and 0.63%, respectively when applying our proposed attention module on top of them. Besides, our proposed attention module needs negligible parameters when there are bottlenecks in deeper networks. The number of added parameters for AW-ResNet101 is 0.31 M while it becomes 4.78 M for SE-ResNet101.

7.4.5. OBJECT DETECTION ON COCO

To show the generalization of our proposed attention module, we apply it to object detection tasks. We evaluate our proposed attention module further on the COCO dataset [193], which contains $118K$ images (i.e., train2017) for training and $5K$ images (i.e., val2017) for validating. We use Faster R-CNN [159] as the detection method with the ResNet101-FPN backbone [194]. Here we intend to evaluate the benefits of applying our proposed attention module on the ResNet101-FPN backbone [194], where all the lateral and output convolutions of the FPN adopt our AW-convolution. The SE and CBAM modules are placed right before the lateral and output convolutions. As shown in Table 7.5, applying our proposed attention module on ResNet101-FPN boosts mAP@[0.5, 0.95] by 0.63 for the Faster R-CNN baseline. Integrating with attentional activations-based models, Faster R-CNNs with the backbones of ResNet101-AW-SE-FPN and ResNet101-AW-CBAM-FPN outperform Faster R-CNNs with the backbones of ResNet101-SE-FPN and ResNet101-CBAM-FPN by 0.34 and 0.45 on COCO's standard metric AP.

7.4.6. GENERALIZED TO OBJECT DETECTION ON VOC2007 TEST

To show the generalization of our proposed attention module, we apply it to object detection tasks. We train SSD300 [158] model with the backbone of VGG16 [161] and SSDLite300 [25] model with the backbone of MobileNet on the training union dataset of VOC2007 trainval and VOC2012 trainval (i.e., 07+12), and evaluate on the testing dataset of VOC2007 test [192].

Since improving the performance of the backbone can typically lead to improvements in more sophisticated computer vision tasks, we apply our proposed attention module to the detector instead of the base networks here. In SE-SSD300/CBAM-SSD300 models, we follow the same design in [166] and plug SE/CBAM modules exactly before every classifier. While in AW-SSD300 model, we replace the traditional convolution of every classifier as our proposed AW-convolution, where $r_{C_1} = 1$ and $r_{C_2} = C_2$.

According to the results shown in Table 7.6, our proposed attention module shows a good generalization when it comes to object detection tasks. Using VGG16 as the base network, AW-SSD300 achieves 0.37 mAP accuracy improvement compared with the SSD300 baseline and increases acceptable computational complexity by 0.01 GFLOPs and 0.46 M parameters. Integrating our proposed attention module, the performance of SE-SSD300 improves by 0.55 mAP further, which is 0.07 mAP better than that of CBAM-SSD300. Similarly, AW-SSDLite300, with the backbone of MobileNet, increases the performance of the SSDLite300 baseline by 0.38 mAP. The accuracy of AW-SE-SSDLite300 is 0.64 mAP better than that of SE-SSDLite300.

7.5. CONCLUSION

In this chapter, We analyze the two ignored problems in attentional activations-based models: the approximation problem and the insufficient capacity problem of the attention maps. To address the two problems together, we propose an attention module by developing the AW-convolution, where the shape of the attention maps matches that of the weights rather than the activations, and integrate it with attention-based models as a complementary

method to enlarge their attentional capability. We have implemented extensive experiments to demonstrate the effectiveness of our proposed attention module, both on image classification and object detection tasks. Our proposed attention module alone shows noticeable accuracy improvement compared with baseline models. More importantly, integrating our proposed module with previous attention-based models, such as AA-Net [174], SE-Net [175], and CBAM-Net [166], will further boost their performance.

8

RECONSTRUCTION FOR DATA-FREE COMPRESSION

The previous chapters are the algorithms to improve the inference efficiency of CNNs. In this chapter, our work is towards a new problem of missing the original training dataset when improving the inference efficiency of CNNs.

Data-free compression raises a new challenge because the original training dataset for a pre-trained model to be compressed is not available due to privacy or transmission issues. Thus, a common approach is to compute a reconstructed training dataset before compression. The current reconstruction methods compute the reconstructed training dataset with a generator by exploiting information from the pre-trained model. However, current reconstruction methods focus on extracting more information from the pre-trained model but do not leverage network engineering. This work is the first to consider network engineering as an approach to design the reconstruction method. Specifically, we propose the AutoReCon method, which is a neural architecture search-based reconstruction method. In the proposed AutoReCon method, the generator architecture is designed automatically given the pre-trained model for reconstruction. Experimental results show that using generators discovered by AutoRecon method always improve the performance of data-free compression.

The content of this chapter is based on [195].

8.1. INTRODUCTION

To be deployed on resources-constrained hardware for real-time applications, the efficiency of deep convolutional neural networks has been improved significantly by various model compression techniques [21, 75, 196, 197]. Without altering the model architecture, quantized neural networks [21] use a low bit width representation instead of full-precision floating-point, saving expensive multiplications. Pruning [196] is an approach to remove the weights or neurons based on certain criteria. In terms of efficient neural network architectures, the MobileNet [75], ShuffleNet, and ESPNet [77] series make use of depthwise-separable convolution, grouped convolution with shuffle operation, and efficient spatial pyramid. The knowledge distillation paradigm [197] transfers the information from a pre-trained teacher network to a portable student network.

Data-free compression [94, 198] has been an active research area when the original training dataset for the given pre-trained model is unavailable because of privacy or storage concerns. Given the pre-trained model to be compressed, it is an essential step to reconstruct the original training dataset by inverting representation. For example, accuracy degradation of ultra-low precision quantized models [199–201] is unacceptable without fine-tuning on the reconstructed training dataset. The reconstruction method computes a reconstructed training dataset by leveraging some extra metadata [202] or by extracting some prior information [203] from the pre-trained model. Instead of computing the reconstructed training dataset directly [94, 202, 204], recent reconstruction methods [198, 200, 203, 205–207] employ a generator to generate a reconstructed training dataset in an end-to-end manner and show better performance for data-free compression.

The quality of the reconstruction closely relates to the extracted information from the pre-trained model. When more information is exploited from the pre-trained model, data-free compression achieves better performance. Thus, the current reconstruction methods [198, 200, 203, 204, 206, 207] focus on exploiting as much prior information as possible from the pre-trained model. However, how the network engineering will contribute to the reconstruction method remains unknown. Thus, we consider network engineering of the reconstruction method for the first time in the literature. This work aims to seek an optimized generator architecture, with which data-free compression shows performance improvement. It is worth mentioning that network engineering of the reconstruction and exploiting more prior information from the pre-trained model are complementary rather than contradictory. Both are important and should be explored for improving data-free compression. The contribution of this chapter is summarized as follows.

- To our best knowledge, we are the first work to consider network engineering of the reconstruction method.
- We propose the AutoReCon method, which is a neural architecture search-based reconstruction method to optimize generator architecture for reconstruction.
- Using the discovered generator, diverse experiments are conducted to demonstrate the effectiveness of the AutoReCon method for data-free compression.

8.2. RELATED WORK

In this section, we review the current work on neural architecture search and data-free compression.

8.2.1. NEURAL ARCHITECTURE SEARCH

Neural architecture search has attracted a lot of attention since it can automatically search for an optimized architecture for a certain task and achieve remarkable performance [60, 78, 83, 208]. The optimization algorithms of neural architecture search include reinforcement learning [78], evolutionary algorithm, random search [209], and gradient-based algorithm [83]. There is a lot of work towards reducing the computational resources required by searching, including weight sharing [78], progressive search, one-short mechanism [83], and using a proxy task. The performance of the discovered architecture by neural architecture search has surpassed human-designed architecture in many computer vision tasks, including classification [83], detection [210], and image generation [208].

8.2.2. DATA-FREE MODEL COMPRESSION

Data-free compression covers data-free quantization and data-free knowledge distillation. Without a generator, the reconstructed training dataset is computed directly in [94, 201, 202, 204]. [202] present a method for data-free knowledge distillation, where the reconstructed training dataset is computed based on some extra recorded activations statistics from the pre-trained model. In data-free knowledge distillation [204], the class similarities are computed from the pre-trained model and the output space is modeled via Dirichlet Sampling. [94] calculates the reconstructed training dataset to match the statistics of the batch normalization layers of the pre-trained model and introduces the Pareto frontier to enable mixed-precision quantization. [201] improves data-free quantization by equalizing the weight ranges and correcting the biased quantization error.

The performance of data-free compression can be improved by employing a generator for the reconstruction [198, 200, 203, 205–207]. [198] proposes a framework for data-free knowledge distillation by exploiting generative adversarial networks, where the reconstructed training dataset derivated from the generator is expected to lead to maximum response on the discriminator of the pre-trained model. The KEGNET [206] framework use the generator and decoder networks to estimate the conditional distribution of the original training dataset for data-free knowledge distillation. In data-free knowledge distillation [207], an adversarial generator is used to produce and search for the reconstructed training dataset on which the student poorly match the teacher. In this chapter, we improve on the work of [200], which proposes a knowledge matching generator to produce a reconstructed training dataset by exploiting classification boundary knowledge and distribution information from the pre-trained model.

8.3. AUTORECON METHOD FOR DATA-FREE COMPRESSION

In this section, we define the reconstruction method for data-free compression. Then, we introduce our proposed AutoReCon method, a neural architecture search-based reconstruc-

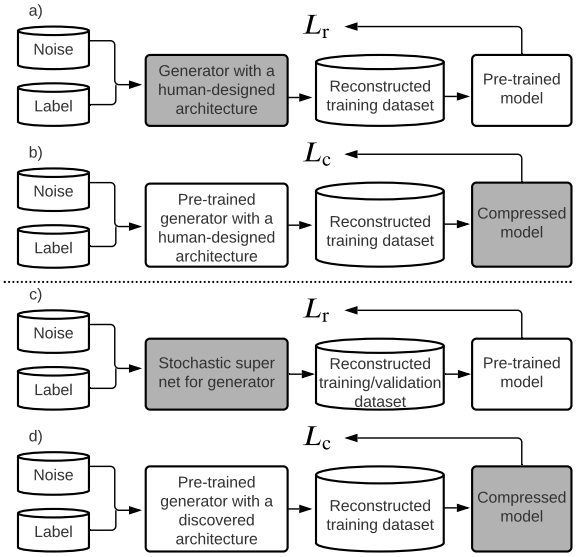


Figure 8.1: The comparison between the current reconstruction method and the AutoReCon method for data-free compression. The goal of every subfigure is to update the models in gray color, given the pre-trained and fixed models in white color. **a)** an overview of the current reconstruction method to update the generator by minimizing the reconstruction loss L_r, where the generator has a human-designed architecture. **b)** an overview of the current reconstruction for data-free compression to update the compressed model by minimizing the compression loss L_c, after the generator with the human-designed architecture has been trained in subfigure **a)**. **c)** an overview of the AutoReCon method to update the generator by minimizing L_r, where there is a super net for the generator. **d)** an overview of the AutoReCon method for data-free compression to update the compressed model by minimizing L_c, after the generator with a discovered architecture has been trained in subfigure **c)**.

tion method, and present its search space and search algorithm. Also, the training process of the AutoReCon method for data-free compression is described.

8.3.1. DEFINITION OF RECONSTRUCTION METHOD

The pre-trained model M_p is obtained by training on the original training dataset $T_o = \{x_o, y_o\}$. Given the pre-trained model M_p, we compute the reconstructed training dataset $T_r = \{x_r, y_r\}$ with the reconstruction method Φ, i.e., $T_r = \Phi(M_p)$.

Considering the reconstruction method with a generator as shown in Figure 8.1a), the pre-trained model M_p is fixed while the weights of the generator are updated by minimizing the reconstruction loss L_r. The prior information extracted from the pre-trained model M_p by the current methods is mainly the class boundary information and distribution information.

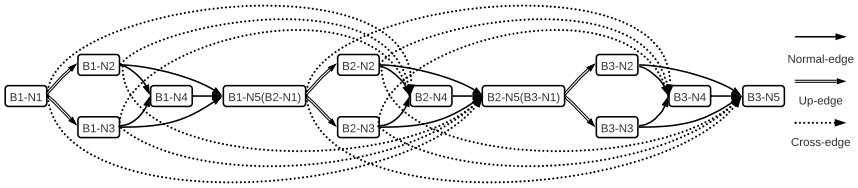


Figure 8.2: The macro-architecture for the generator. The macro-architecture is a directed acyclic graph consisting of an ordered sequence of nodes. For example, the rectangle with the tag "B1-N1" represents the 1st node of the 1st convolutional block. "B1-N5(B2-N1)" indicates the 5^{th} node of the 1st convolutional block is the same as the 1st node of the 2nd convolutional block.

If more prior information can be extracted from the pre-trained model, the reconstruction method can be easily adjusted by incorporating more loss terms to the reconstruction loss. Current reconstruction method Φ can be expressed as follows.

$$\begin{aligned} &\min_{W_g} L_r(W_g) = \min_{W_g} \mathbb{E}_{y_o \sim P_{y_o}(y_o), z \sim P_z(z)} \\ &[L_{class}(M_p(M_g(z|y_o); W_g), y_o) + L_{bns}(BN_r, BN_o)] \end{aligned} \tag{8.1}$$

where z and W_g are the random noise input vector and weights of the generator, and $L_{class}(\cdot,\cdot)$ is the cross-entropy loss function. $L_{bns}(\cdot,\cdot)$ measures the distribution distance between the batch normalization statistics of the original training dataset BN_o and the batch normalization statistics of the reconstructed training dataset BN_r.

8.3.2. AUTORECON METHOD

As shown in Figure 8.1a) and c), we present an overview of current reconstruction and the AutoReCon method. The current reconstruction method includes a pre-trained model M_p and a generator M_g with a human-designed architecture. In the AutoReCon method, we aim to search for a superior generator architecture automatically for reconstruction.

Regarding the reconstruction task, our training objection function is written as follows, where both weights W_g and architecture A_g of the generator can be updated by minimizing the reconstruction loss.

$$\begin{aligned} &\min_{A_g} L_r^{\text{val}}(A_g, W_g^*(A_g)) \\ &s.t.\ W_g^*(A_g) = \operatorname*{argmin}_{W_g} L_r^{\text{train}}(A_g, W_g) \end{aligned} \tag{8.2}$$

where L_r^{train} and L_r^{val} refer to the reconstruction loss function on the reconstructed training dataset and the reconstructed validation dataset, respectively. $W_g^*(A_g)$ are the optimal weights of the generator given the generator architecture A_g. $A_g \in S$ and S is the whole search space of the generator.

Algorithm 4 The AutoReConmethod for data-free compression

Input: Pre-trained model M_p.
Output: Discovered generator M_g, compressed model M_c.
Stage 1: Searching for generator architecture.
for $epoch = 1$ to L_1 **do**
  for $batch = 1$ to T_1 **do**
    Obtain random noise $z \sim N(0,1)$ and label y_o.
    Generate reconstructed training dataset T_r with stochastic super net M_s.
    Update weights of stochastic super net by minimizing reconstruction loss L_r.
  end for
  for $batch = 1$ to V_1 **do**
    Obtain random noise $z \sim N(0,1)$ and label y_o.
    Generate reconstructed validation dataset V_r with stochastic super net M_s.
    Update architecture parameters of stochastic super net by minimizing reconstruction loss L_r.
  end for
end for
Stage 2: Compression with discovered generator.
for $epoch = 1$ to L_2 **do**
  for $batch = 1$ to T_2 **do**
    Obtain random noise $z \sim N(0,1)$ and label y_o.
    Generate reconstructed training dataset T_r with the discovered generator M_g.
    Update weights of compressed model M_c by minimizing compression loss L_c.
  end for
end for

The search space: We construct a layer-wise search space with a fixed macro-architecture for the generator. The macro-architecture defines the type of the edge, the number of edges, the node connection, and the input/output dimension of each node. The macro-architecture is shown in Figure 8.2, where there are three convolutional blocks and five nodes in every convolutional block. We denote the generator as $M_g(e_1, ..., e_i, ..., e_E)$, where e_i represents the i^{th} edge and E is the number of edges. The nodes refer to the feature maps and we calculate them as the summation of the outputs of their previous connected edges. There are three types of edges: normal-edge, up-edge, and cross-edge. Normal-edge connects two nodes with the same dimension. Up-edge is used to increase the spatial resolution. Normal-edge and Up-edge are within a convolutional block. Cross-edge connects two adjacent convolutional blocks.

To construct a layer-wise search space for the generator, we set each edge as a mixture of candidate operations, which has several parallel operations instead of one specific operation. Thus, the over-parameterized generator is expressed as $M_g(e_1 = C_1, ..., e_i = C_i, ..., e_E = C_E)$ and C_i is the mixture of candidate operations for the edge e_i. As shown in Table 8.1, different types of edges use different mixtures of candidate operations. Taking the edge C_i as an example, we compute its output by summing the outputs of the mixture of candidate operations as follows.

$$X^i_{out} = C_i(X^i_{in}) = \sum_{j=1}^{F} O^i_j(X^i_{in}) \tag{8.3}$$

Table 8.1: For different types of edges, there are different mixtures of candidate operations.

Edge type	Mixture of candidate operations
Normal-edge	Convolution 1 × 1, dilation=1
	Convolution 3 × 3, dilation=1
	Convolution 5 × 5, dilation=1
	Convolution 3 × 3, dilation=2
	Convolution 5 × 5, dilation=2
	Identity
	None
Up-edge	Nearest Neighbor Interpolation
	Bilinear Interpolation
Cross-edge	Nearest Neighbor Interpolation
	Bilinear Interpolation
	None

where X_{in}^{i} and X_{out}^{i} are the input and output of the i^{th} edge. O_j^i denotes the j^{th} candidate operation of the i^{th} edge and $j = 1, ..., F$. F is the number of candidate operations for an edge.

The search algorithm: The search algorithm represents the search space as a stochastic super net M_s. In the stochastic super net M_s, O_j^i is associated with an architecture parameter α_j^i. To derive a generator A_g from the stochastic super net M_s, the candidate operation O_j^i is sampled with the probability p_j^i, which is computed as follows.

$$p_j^i(O_j^i; \alpha^i) = \text{softmax}(\alpha^i) = \frac{exp(\alpha_j^i)}{\sum_{j=1}^{F} exp(\alpha_j^i)} \tag{8.4}$$

Since sampling from the mixture of candidate operations for each edge is independent, the probability of sampling a generator architecture A_g can be described as follows.

$$P(A_g; \alpha_g) = \prod_{i=1}^{E} p_j^i(O_j^i; \alpha^i) \tag{8.5}$$

In this case, we can approximate the problem of finding an optimized discrete generator architecture by finding optimized sampling probabilities. The training objective function of the AutoReCon method is re-written from Equation 8.2 as follows.

$$\begin{aligned} &\min_{a_g} \mathbb{E}_{a_g \sim P_{a_g}(a_g)}[L_r^{\text{val}}(a_g, W_g^*(a_g))] \\ &s.t.\ W_g^*(a_g) = \underset{W_g}{\text{argmin}}\ L_r^{\text{train}}(a_g, W_g) \end{aligned} \tag{8.6}$$

To make the reconstruction loss differentiable to the sampling probabilities, we compute continuous variables m_j^i by the Gumbel Softmax function as an alternative as follows.

$$m_j^i = \text{GumbelSoftmax}(p^i) = \frac{exp\left[(p_j^i + g_j^i)/\tau\right]}{\sum_{j=1}^{F} exp\left[(p_j^i + g_j^i)/\tau\right]} \tag{8.7}$$

where g_j^i is the noise sampled from the Gumbel distribution $(0, 1)$ and τ is a temperature parameter to control the sampling operation. Then, the continuous variables m_j^i are directly differentiable with respect to the sampling probabilities. Thus, the computation of the edge C_i in Equation 8.3 can be expressed as follows.

$$X_{out}^i = C_i(X_{in}^i) = \sum_{j=1}^{F} m_j^i O_j^i(X_{in}^i) \tag{8.8}$$

8.3.3. TRAINING PROCESS

Using the AutoReCon method, the training process for data-free compression is illustrated as shown in Algorithm 4. The first stage of the training process is to search for generator architecture with our AutoReCon method, as shown in Figure 8.1c). The goal of the first stage is to seek an optimized generator architecture from the stochastic super net. The second stage of the training process is to compress the pre-trained model M_p with the discovered generator M_g. The compression loss L_c can be introduced from quantization and/or knowledge distillation. Compared with the current reconstruction methods, our AutoReCon method considers network engineering and search for an optimized generator architecture for reconstruction.

8.4. EXPERIMENTS

In this section, we run data-free compression experiments on image classification tasks.

8.4.1. IMPLEMENTATION DETAILS

Our interest is to show the performance improvement of data-free compression, which is brought by the AutoReCon method. We adopt the GDFQ data-free compression method [200] as a baseline for the following three reasons. First, it exploits both class boundary information and distribution information from the pre-trained model M_p, compared to other methods that use only one type of information [94, 198, 204, 206]. Second, it includes both data-free quantization and data-free knowledge distillation, where knowledge distillation is applied for the output layer (i.e., knowledge distillation is not applied for the intermediate layers). Third, it achieves state-of-the-art performance. We use the same experimental settings as the GDFQ method to observe the influence of the generator architecture. In the GDFQ method, the human-designed generator architecture for both CIFAR-100 and ImageNet classification follows ACGAN. Besides, the human-designed generator for ImageNet classification adopts the categorical conditional batch normalization layer to fuse label information following SN-GAN.

Table 8.2: Experimental results of data-free compression on CIFAR-100 and ImageNet classification. $w4a4$ means that the weights and activations are quantized to 4-bit precision. Both our data-free compression method and the GDFQ adopt knowledge distillation for the output layer. In each block, the first row presents the accuracy of the full-precision pre-trained model on CIFAR-100. The second row shows the accuracy of the full-precision pre-trained model on ImageNet. In Top-1 colomn, (a/b) represents (CIFAR-100/ImageNet).

Method	Pre-trained model	Generator	Quantization	Top-1
-	ResNet18	-	-	78.83%/-
-	ResNet18	-	-	-/71.47%
GDFQ	ResNet18	Human-designed	w6a6	78.00%/70.10%
GDFQ	ResNet18	Human-designed	w5a5	75.93%/68.38%
GDFQ	ResNet18	Human-designed	w4a4	60.23%/60.70%
GDFQ	ResNet18	Human-designed	w3a3	28.71%/20.69%
Ours	ResNet18	In AutoReCon	w6a6	78.52%/70.61%
Ours	ResNet18	In AutoReCon	w5a5	77.22%/68.88%
Ours	ResNet18	In AutoReCon	w4a4	71.02%/61.32%
Ours	ResNet18	In AutoReCon	w3a3	46.44%/23.37%
-	MobileNetV2	-	-	70.72%/-
-	MobileNetV2	-	-	-/73.03%
GDFQ	MobileNetV2	Human-designed	w6a6	69.59%/71.18%
GDFQ	MobileNetV2	Human-designed	w5a5	65.27%/67.81%
GDFQ	MobileNetV2	Human-designed	w4a4	53.91%/59.80%
GDFQ	MobileNetV2	Human-designed	w3a3	8.50%/2.31%
Ours	MobileNetV2	In AutoReCon	w6a6	70.57%/71.53%
Ours	MobileNetV2	In AutoReCon	w5a5	67.95%/68.40%
Ours	MobileNetV2	In AutoReCon	w4a4	58.42%/60.13%
Ours	MobileNetV2	In AutoReCon	w3a3	10.21%/14.30%
-	ResNet50	-	-	79.36%/-
-	ResNet50	-	-	-/77.72%
GDFQ	ResNet50	Human-designed	w6a6	78.79%/76.40%
GDFQ	ResNet50	Human-designed	w5a5	76.17%/70.79%
GDFQ	ResNet50	Human-designed	w4a4	61.44%/55.94%
GDFQ	ResNet50	Human-designed	w3a3	26.51%/1.20%
Ours	ResNet50	In AutoReCon	w6a6	79.12%/76.76%
Ours	ResNet50	In AutoReCon	w5a5	77.06%/74.13%
Ours	ResNet50	In AutoReCon	w4a4	68.20%/64.37%
Ours	ResNet50	In AutoReCon	w3a3	36.17%/1.63%

8.4.2. RESULTS ON IMAGE CLASSIFICATION

Results on CIFAR-100 classification: As shown in Table 8.2, we report the experimental results of data-free compression on the CIFAR-100 classification dataset. Using various pre-trained models and low-bit width quantization, our data-free compression with an optimized generator architecture achieves better accuracy than the GDFQ method with a

Table 8.3: Experimental results of data-free compression on CIFAR-100 classification. The GDFQ method uses a human-designed generator. Our data-free compression uses the generator discovered by the AutoRe method.

Method	Scale	Top-1	Top-5
GDFQ	$s = 4$	64.87%	86.76%
GDFQ	$s = 3$	65.04%	86.93%
GDFQ	$s = 2$	65.22%	87.19%
GDFQ	$s = 1$	65.27%	87.30%
GDFQ	$s = 0.5$	63.72%	86.21%
Ours	$s = 4$	68.78%(+3.91%)	88.62%
Ours	$s = 3$	68.09%(+3.05%)	89.01%
Ours	$s = 2$	67.95%(+2.73%)	88.76%
Ours	$s = 1$	67.58%(+2.31%)	88.42%
Ours	$s = 0.5$	66.30%(+2.58%)	88.09%

Table 8.4: Experimental results of data-free compression on CIFAR-100 classification. The first row is the accuracy of the pre-trained teacher model.

Method	Generator	Top-1
-	-	77.50%
DAFL	Human-designed	61.40%
DFAD	Human-designed	67.70%
Ours	Discovered by AutoReCon	69.98%(+2.28%)

human-designed generator. Using ResNet18 as the pre-trained model and 3-bit width quantization, the Top-1 accuracy of the GDFQ method will improve by 17.73% if the human-designed generator is replaced with the generator discovered by the AutoReCon method. Using MobileNetV2 and 5-bit width quantization, the Top-1 accuracy of our data-free compression shows an improvement of 4.51% compared with the GDFQ method. The Top-1 accuracy improvement becomes 9.66% using ResNet50 as the pre-trained model and 4-bit width quantization.

Results on ImageNet classification: As shown in Table 8.2, we report the experimental results of data-free compression on the ImageNet classification dataset. Replacing the human-designed generator with the generator discovered by our AutoReCon method, the accuracy of the GDFQ method increases consistently using different pre-trained models and low-bit width quantization. Using ResNet18 as the pre-trained model and 3-bit width quantization, the Top-1 accuracy of the GDFQ method can increase by 2.68% when using the generator discovered by the AutoReCon method. The Top-1 accuracy of the GDFQ method increases by 11.99% using MobileNetV2 as the pre-trained model, 3-bit width quantization, and the generator discovered by the AutoReCon method. Using ResNet50 as the pre-trained

Table 8.5: Comparison of different data-free compression methods on ImageNet classification. $w4a4$ means that the weights and activations are quantized to 4-bit precision. The first row of each block is the accuracy of the full-precision pre-trained model.

Method	Pre-trained model	Quantization	Top-1
-	ResNet18	-	71.47%
DFQ	ResNet18	w4a4	0.10%
ZeroQ	ResNet18	w4a4	26.04%
DFC	ResNet18	w4a4	55.49%
GDFQ	ResNet18	w4a4	60.70%
Ours	ResNet18	w4a4	61.60%
-	MobileNetV2	-	73.03%
DFQ	MobileNetV2	w4a4	0.11%
ZeroQ	MobileNetV2	w4a4	3.31%
GDFQ	MobileNetV2	w4a4	59.80%
Ours	MobileNetV2	w4a4	60.02%
-	ResNet50	-	77.72%
ZeroQ	ResNet50	w4a4	0.12%
GDFQ	ResNet50	w4a4	55.94%
Ours	ResNet50	w4a4	57.49%

model and 5-bit width quantization, the Top-1 accuracy of our data-free compression with an optimized generator surpasses the GDFQ method by 8.43%.

8.4.3. ABLATION STUDY

Scalability of discovered generator architectures: We explore the scalability of the discovered generator architecture for data-free compression on the CIFAR-100 classification dataset. We scale the base channels by a factor from $s = 0.5$ to $s = 4$ for the discovered generator and the human-designed generator. The data-free compression results using MobileNetV2 as the pre-trained model, 5-bit width quantization, and knowledge distillation applied for the output layer are shown in Table 8.3. Without modifying the optimized generator architecture, the performance of our data-free compression keeps increasing and is always better than the GDFQ method when scaling the base channels by the factor from $s = 0.5$ to 4.0. The accuracy of the GDFQ method decreases when we scale the base channels for the human-designed generator. Thus, we conclude that our searched generator architecture has superior scalability compared to the human-designed generator for data-free compression.

Generalization of AutoReCon method: Except for the GDFQ method, we use the GFAD[205] method as a baseline to show the generalization of our AutoReCon method. The generation loss in the GFAD method is replaced with the reconstruction loss of Equation 6, which enables the exploration of generator architecture. We use ResNet34 as the pre-

Table 8.6: Experimental results of data-free compression on CIFAR-100 classification. $w4a4$ means that the weights and activations are quantized to 4-bit precision. The first row of each block is the accuracy of the full-precision pre-trained model.

Method	Generator	Quantization	Top-1	Top-5
-	-	-	70.33%	91.46%
GDFQ	Human-designed	w8a8	70.19%	91.44%
GDFQ	Human-designed	w7a7	70.01%	91.20%
GDFQ	Human-designed	w6a6	68.98%	90.98%
GDFQ	Human-designed	w5a5	67.40%	89.57%
GDFQ	Human-designed	w4a4	63.01%	86.70%
GDFQ	Human-designed	w3a3	42.62%	71.70%
GDFQ	Human-designed	w2a2	1.23%	5.66%
Ours	Discovered by AutoReCon	w8a8	70.20%(+0.01%)	91.39%
Ours	Discovered by AutoReCon	w7a7	70.02%(+0.01%)	91.18%
Ours	Discovered by AutoReCon	w6a6	69.71%(+0.73%)	91.01%
Ours	Discovered by AutoReCon	w5a5	68.75%(+1.35%)	90.19%
Ours	Discovered by AutoReCon	w4a4	63.60%(+0.59%)	87.25%
Ours	Discovered by AutoReCon	w3a3	48.66%(+6.04%)	76.74%
Ours	Discovered by AutoReCon	w2a2	1.43%(+0.20%)	6.06%

trained teacher model and ResNet18 as the student model. The experimental results of data-free knowledge distillation on CIFAR-100 is shown in Table 8.4. With a human-designed generator, the GFAD method achieves better accuracy than the DAFL[198] method. The Top-1 accuracy of our data-free knowledge distillation with a discovered generator is 2.28% better than the baseline of the GFAD method with a human-designed generator.

Impact of different bit width quantiztion: As shown in Table 8.6, we report the accuracy improvement of our data-free compression method compared to the GDFQ method [200] using ResNet20 as the pre-trained model, knowledge distillation applied for the output layer and different low-bit width quantization. We are interested in the impact of different low-bit width quantization on the accuracy improvement of our method. From 2-bit width to 8-bit width quantization, our data-free compression method is consistently better than the GDFQ method in terms of accuracy since the AutoReCon method searched for an optimized generator for the reconstruction. Using 8-bit width and 7-bit width quantization, the accuracy improvement of our method is negligible since there is a small accuracy drop introduced by quantization. Besides, the accuracy improvement of our method is small using 2-bit width quantization because of the large quantization error. Thus, our method shows the largest Top-1 accuracy improvement of 6.04% using 3-bit width quantization compared with the GDFQ method.

8.4.4. COMPUTATION COMPLEXITY OF GENERATOR

The goal of the AutoReCon method is not to reduce the computational complexity of the generator. However, we report the comparison of the human-designed generator and the

Table 8.7: The computational complexity of different generators.

Generator	Pre-trained model	Parameters	Flops
Human-designed generator	-	40.36Mbit	3.83×10^9
Discovered by AutoReCon	ResNet18	40.17Mbit	3.63×10^9
Discovered by AutoReCon	MobileNetV2	40.25Mbit	1.43×10^9
Discovered by AutoReCon	ResNet50	40.28Mbit	2.76×10^9

discovered generators as shown in Table 8.7. The generators are used for data-free compression on the ImageNet classification dataset. With different pre-trained models, the human-designed generator has the same architecture. There is a specific discovered generator architecture for a pre-trained model. In terms of model size, our discovered generator requires similar parameters compared with the human-designed generator. The number of Flops to compute our discovered generator is smaller compared with the human-designed generator. For example, the discovered generator for the pre-trained model of MobileNetV2 needs 2.40×10^9 Flops less compared with the human-designed generator.

8.4.5. COMPARISON WITH STATE-OF-THE-ART METHODS

On the ImageNet classification dataset, we present the results of additional data-free compression methods as shown in Table 8.5. The comparison is mainly for data-free quantization except that the GDFQ and our methods apply knowledge distillation on the output layer. None of the compared methods apply knowledge distillation on the intermediate layers. The results of DFQ [201] and ZeroQ [94] are cited from the GDFQ paper and have a rather low accuracy for ultra-low precision data-free quantization. The DFC [211] method achieves a moderate accuracy with a combination of BN-Statistics and Inception schemes. Our method achieves better accuracy compared to the GDFQ method since the AutoReCon method discovers an optimized generator architecture for reconstruction.

8.5. CONCLUSION

In this chapter, we present the AutoReCon method, which is the first work to consider network engineering of the reconstruction method to improve the performance of data-free compression. In particular, our AutoReCon method can search for an optimized generator architecture from a stochastic super net with gradient-based neural architecture search for reconstruction. When we plug our discovered generator to replace the human-designed generator, our data-free compression benefits from the optimization of the generator architecture and achieves better accuracy. Specifically, using ResNet50 as the pre-trained model and 5-bit width quantization, the Top-1 accuracy of our data-free compression on ImageNet with an optimized generator surpasses the GDFQ method by 8.43%. The Top-1 accuracy of the DFAD method on CIFAR-100 increases by 2.28% using ResNet34 as the pre-trained teacher model, ResNet18 as the student model, and the generator discovered by the AutoReCon method.

9

CONCLUSION AND FUTURE DIRECTIONS

This chapter summarizes the overall achievements of this thesis and highlights some future research directions. Section 9.1 provides the main conclusions of this thesis. Then, Section 9.2 discusses some future research directions.

9.1. CONCLUSIONS

Throughout this thesis, we have developed solutions to the research questions formulated in 1.3. In this section, we mainly reiterate the research questions and summarize the answers as follows.

Chapter 1 briefly introduced computing Deep Neural Networks (DNNs) on lightweight devices and the research focus of this thesis. It first described the motivation of this thesis. Then, it highlighted the opportunities and challenges facing efficient inference in convolutional neural networks (CNNs). This thesis focused on addressing the challenges in algorithms for efficient inference in CNNs. Afterward, it formulated the research questions and the corresponding contributions to answer these questions.

How to accelerate integer CNNs? This question belongs to acceleration and quantization. Chapter 2 proposed Diminished-1 Fermat Number Transform to accelerate integer CNNs. The computational complexity analysis showed that the proposed algorithm requires fewer multiplications and additions to compute integer convolution than the direct method. Also, the implemented results revealed that the acceleration algorithm is faster and has better parallelism than the direct method. When computing integer convolution with typical parameter configurations, the DFNT achieves a speedup of 2-3x compared to the direct method. Besides, the accuracy of the proposed acceleration algorithm is the same as the direct method since there is no round-off error introduced.

How to improve multiple binary CNNs? This question belongs to quantization. Chapter 3 described the piecewise approximation scheme for multiple binary CNNs. The proposed scheme achieves the best accuracy among current single and multiple binary CNNs. The proposed scheme is applied on full-precision CNNs with various architectures and evaluated on classification and detection tasks to show the generalization. With the proposed binarization scheme, the Top-1 and Top-5 accuracy drop of ResNet with various depths on ImageNet can be reduced to approximately 1.0%. Computational complexity analysis exhibited that the latency of the proposed scheme is almost the same as that of the single binary. The memory usage and FLOPs of the proposed scheme required are less than those needed for current multiple binary CNNs.

How to design neural network architectures automatically for binary CNNs? This question belongs to quantization and efficient neural network architecture design. Chapter 4 presented the NASB strategy for binary CNNs. The search space of the NASB strategy is a combination between a human-designed backbone and a NAS-convolutional cell. The NASB strategy discovered optimized neural network architectures, which are suitable for binarization. Based on the conducted experimental results, the discovered neural network architectures from the NASB strategy achieve a better trade-off between accuracy and efficiency than current single and multiple binary CNNs. Without a significant overhead increase, the Top-1 accuracy of the binary CNNs searched from the NASB strategy is 4.0% and 1.0% better than existing single and multiple binary CNNs, respectively.

How to design neural network architectures manually for binary CNNs? This question belongs to quantization and efficient neural network architecture design. Chapter 5 introduced unified effective depth reduction (EDR) techniques for binary CNNs. The limitation of relying solely on enhancing the shortcut EDR technique is identified. The chapter then introduces the shortcut and fractal architecture EDR techniques to develop unified models. With similar computational complexity, the Top-1 accuracy of the developed UA-ResNet37(41) and UA-DenseNet51(53) on ImageNet surpasses Bi-Real ResNet18(64) and BinaryDenseNet51(32) by 3.29% and 1.41%, respectively. Compared with Bi-Real ResNet and BinaryDenseNet, the quality of gradient paths in the unified models improved, and the accuracy of unified models on the classification datasets increased.

How to reduce feature reuse within the convolution for full-precision CNNs? This question belongs to efficient neural network architecture design. Chapter 6 described the REAF scheme for full-precision CNNs. Under the same computational budget constraint, the proposed scheme on classification datasets achieved better accuracy than standard convolution and group convolution. The proposed scheme with various configurations is better than standard convolution. Under the given computational complexity budget, the Top-1 accuracy of REAF-ResNet50 and REAF+-MobileNetV2 on ImageNet will improve by 0.37% and 0.69% respectively. Compared with standard convolution, feature reuse reduction is observed from the class selectivity analysis in the proposed scheme and group convolution.

How to improve the attention module for full-precision CNNs? This question belongs to efficient neural network architecture design. Chapter 7 presented an attention module for full-precision CNNs. The architecture of calculating the attention maps is refined to reduce the computational complexity. Integrated with the proposed attention module, the models without attention mechanisms achieve better accuracy on classification and detection tasks. The accuracy of the attention-based models boosts further by plugging the proposed attention module. The proposed attention module is demonstrated as a complementary method to current attention-based models. Integrating with the proposed attention module, the previous attentional activations-based models can further improve their Top-1 accuracy on ImageNet classification by 0.57% and COCO-style Average Precision on the COCO object detection by 0.45.

How to improve the reconstruction method for data-free compression? This question belongs to quantization, knowledge distillation, and efficient neural network architecture design. Chapter 8 described the AutoReCon method. The proposed method leverages the neural network architecture for the current reconstruction method. The proposed method consists of a stochastic super net and a gradient-based algorithm. Plugging the discovered generator from the proposed method, the data-free quantization and knowledge distillation on classification datasets achieve better accuracy. In particular, the Top-1 accuracy of our data-free compression on ImageNet with a discovered generator outperforms the GDFQ method by 8.43% when using ResNet50 as the pre-trained model and 5-bit width quantization.

9.2. FUTURE RESEARCH DIRECTIONS

In this thesis, we have developed various algorithms for efficient inference of CNNs. In Chapter 2, using DFNT to accelerate ICNNs is expected to implement DFNT on customized hardware platforms to demonstrate the speedup further. Considering the trade-off between accuracy and efficiency, Chapter 5 is of better practical value than Chapter 4 and Chapter 3. In addition, the efficiency in Chapter 5 can be improved further by exploring different architecture configurations of our proposed architectures. To investigate more efficient convolutions, the feature reuse in Chapter 6 and the attention mechanism in Chapter 7 can be improved further by quantitative analysis, where the reused features and the attention capacity are quantified. For Chapter 8, our proposed method is expected to be extended to detection and segmentation to show generalization.

In addition, there are still lots of other topics that need to be explored to improve the efficiency of DNNs. We list the potential research topics as follows.

Training DNNs on the edge: Computing DNNs in the centralized cloud faces the two main challenges of high data communication cost and privacy concerns. More importantly, training with the new data continuously on the edge is important since DNNs need to adapt to a specific domain. The inference of DNNs on the edge is well-explored from Chapter 2 to Chapter 7 in this thesis, but training DNNs on the edge still needs to be investigated. In particular, the algorithms and hardware for efficient training in DNNs need to be developed. The algorithm and hardware for efficient training are more complex than those for efficient inference since the training process has two extra steps compared to the inference process.

Computer vision tasks beyond classification: We develop algorithms and hardware for efficient computation in CNNs for a classification task. Then, the developed algorithms and hardware without much modification are evaluated on detection and segmentation tasks, such as Chapter 3 and Chapter 7. However, the detection and segmentation tasks require more fine-grained information than the classification task. Thus, we suggest that there should be specific algorithms and hardware for efficient computation in CNNs for the computer vision tasks beyond the classification task. Taking efficient neural network architecture design as an example, the neural network architecture of CNNs for the classification task should be different from that for the detection task or the segmentation task.

Rotation or scaling equivariance: The efficiency of CNNs can be further improved by designing neural network architecture to have more transformation equivariance. Current CNNs have the translation equivariance since the convolution operation adopts weight sharing. However, current CNNs have to memorize all the training samples with rotation or scaling augmentation, which requires redundant weights and is inefficient. The neural network architecture of currently used CNNs needs to be redesigned to enable the property of rotation or scaling equivariance and to improve efficiency.

REFERENCES

[1] Z. Baozhou, N. Ahmed, J. Peltenburg, K. Bertels, and Z. Al-Ars. "Diminished-1 Fermat Number Transform for Integer Convolutional Neural Networks". In: *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*. IEEE. 2019, pp. 47–52.

[2] M. Mathieu, M. Henaff, and Y. LeCun. "Fast training of convolutional networks through ffts". In: *arXiv preprint arXiv:1312.5851* (2013).

[3] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. "Fast convolutional nets with fbfft: A GPU performance evaluation". In: *arXiv preprint arXiv:1412.7580* (2014).

[4] J. Cong and B. Xiao. "Minimizing computation in convolutional neural networks". In: *International conference on artificial neural networks*. Springer. 2014, pp. 281–290.

[5] A. Lavin and S. Gray. "Fast algorithms for convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4013–4021.

[6] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160* (2016).

[7] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. "Deep learning with limited numerical precision". In: *International Conference on Machine Learning*. 2015, pp. 1737–1746.

[8] *cuDNN*. https://developer.nvidia.com/cudnn. Accessed: 2015-11-01.

[9] W. Xu, X. You, and C. Zhang. "Using Fermat number transform to accelerate convolutional neural network". In: *2017 IEEE 12th International Conference on ASIC (ASICON)*. 2017, pp. 1033–1036. DOI: 10.1109/ASICON.2017.8252655.

[10] M. A. Hanif, R. Hafiz, and M. Shafique. "Error resilience analysis for systematically employing approximate computing in convolutional neural networks". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE. 2018, pp. 913–916.

[11] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. Fua. "Learning separable filters". In: *IEEE transactions on pattern analysis and machine intelligence* 37.1 (2015), pp. 94–106.

[12] P. Gysel, M. Motamedi, and S. Ghiasi. "Hardware-oriented approximation of convolutional neural networks". In: *arXiv preprint arXiv:1604.03168* (2016).

[13] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. “Quantization and training of neural networks for efficient integer-arithmetic-only inference”. In: *arXiv preprint arXiv:1712.05877* (2017).

[14] S. Wu, G. Li, F. Chen, and L. Shi. “Training and inference with integers in deep neural networks”. In: *arXiv preprint arXiv:1802.04680* (2018).

[15] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, *et al.* “Mixed Precision Training of Convolutional Neural Networks using Integer Operations”. In: *arXiv preprint arXiv:1802.00930* (2018).

[16] S. Higginbotham. “Google takes unconventional route with homegrown machine learning chips”. In: *Next Platform, May* (2016).

[17] T. Morgan. “Nvidia pushes deep learning inference with new Pascal GPUs”. In: *Next Platform, September* (2016).

[18] *Advanced_Vector_Extensions#AVX-512*. https://en.wikipedia.org/wiki/Advanced_Vector_Extensions#AVX-512. Accessed: 2018-05-12.

[19] R. Agarwal and C. Burrus. “Fast convolution using Fermat number transforms with applications to digital filtering”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 22.2 (1974), pp. 87–97.

[20] L. Leibowitz. “A simplified binary arithmetic for the Fermat number transform”. In: *IEEE Transactions on acoustics, speech, and signal processing* 24.5 (1976), pp. 356–359.

[21] B. Zhu, Z. Al-Ars, and W. Pan. “Towards lossless binary convolutional neural networks using piecewise approximation”. In: *European Conference on Artificial Intelligence (ECAI)*. 2020.

[22] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[23] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[24] 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”. In: *arXiv preprint arXiv:1704.04861* (2017).

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *arXiv preprint arXiv:1801.04381* (2018).

[26] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. “Soft filter pruning for accelerating deep convolutional neural networks”. In: *arXiv preprint arXiv:1808.06866* (2018).

[27] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. “Efficient and accurate approximations of nonlinear convolutional networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1984–1992.

[28] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[29] A. Mishra and D. Marr. "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy". In: *arXiv preprint arXiv:1711.05852* (2017).

[30] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. "Incremental network quantization: Towards lossless cnns with low-precision weights". In: *arXiv preprint arXiv:1702.03044* (2017).

[31] D. Zhang, J. Yang, D. Ye, and G. Hua. "LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks". In: *The European Conference on Computer Vision (ECCV)*. Sept. 2018.

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

[33] 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". In: *arXiv preprint arXiv:1602.02830* (2016).

[34] V. W.-S. Tseng, S. Bhattacharya, J. F. Marqués, M. Alizadeh, C. Tong, and N. D. Lane. "Deterministic binary filters for convolutional neural networks". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. AAAI Press. 2018, pp. 2739–2747.

[35] S. Darabi, M. Belbahri, M. Courbariaux, and V. P. Nia. "BNN+: Improved binary network training". In: *arXiv preprint arXiv:1812.11800* (2018).

[36] C. Liu, W. Ding, X. Xia, B. Zhang, J. Gu, J. Liu, R. Ji, and D. Doermann. "Circulant Binary Convolutional Networks: Enhancing the Performance of 1-bit DCNNs with Circulant Back Propagation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2691–2699.

[37] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[38] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng. "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 722–737.

[39] X. Lin, C. Zhao, and W. Pan. "Towards accurate binary convolutional neural network". In: *Advances in Neural Information Processing Systems*. 2017, pp. 345–353.

[40] J. Fromm, S. Patel, and M. Philipose. "Heterogeneous bitwidth binarization in convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2018, pp. 4006–4015.

[41] Y. Guo, A. Yao, H. Zhao, and Y. Chen. “Network sketching: Exploiting binary structure in deep cnns”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5955–5963.

[42] W. Tang, G. Hua, and L. Wang. “How to train a compact binary neural network with high accuracy?” In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[43] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao. “Performance guaranteed network acceleration via high-order residual quantization”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2584–2592.

[44] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid. “Structured Binary Neural Networks for Accurate Image Classification and Semantic Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 413–422.

[45] M. Shen, K. Han, C. Xu, and Y. Wang. “Searching for accurate binary neural architectures”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019, pp. 116–131.

[46] Y. Bengio, N. Léonard, and A. Courville. “Estimating or propagating gradients through stochastic neurons for conditional computation”. In: *arXiv preprint arXiv:1308.3432* (2013).

[47] F. Li and B. Liu. “Ternary Weight Networks”. In: *CoRR* abs/1605.04711 (2016).

[48] C. Zhu, S. Han, H. Mao, and W. J. Dally. “Trained ternary quantization”. In: *arXiv preprint arXiv:1612.01064* (2016).

[49] D. Wan, F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. Tao Shen. “Tbn: Convolutional neural network with ternary inputs and binary weights”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 315–332.

[50] Z. He and D. Fan. “Simultaneously Optimizing Weight and Quantizer of Ternary Neural Network using Truncated Gaussian Approximation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11438–11446.

[51] 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: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4350–4359.

[52] C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A. M. Bronstein, and A. Mendelson. “Uniq: Uniform noise injection for non-uniform quantization of neural networks”. In: *ACM Transactions on Computer Systems (TOCS)* 37.1–4 (2021), pp. 1–15.

[53] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. “Weight uncertainty in neural network”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1613–1622.

[54] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[55] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[56] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.* “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[57] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. “The Pascal Visual Object Classes Challenge: A Retrospective”. In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136.

[58] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. “SSD: Single Shot MultiBox Detector”. In: (2016). Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling, pp. 21–37.

[59] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[60] B. Zhu, Z. Al-Ars, and H. P. Hofstee. “NASB: Neural Architecture Search for Binary Convolutional Neural Networks”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[61] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size”. In: *arXiv preprint arXiv:1602.07360* (2016).

[62] D. Zhang, J. Yang, D. Ye, and G. Hua. “Lq-nets: Learned quantization for highly accurate and compact deep neural networks”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 365–382.

[63] S. Anwar, K. Hwang, and W. Sung. “Structured pruning of deep convolutional neural networks”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.3 (2017), p. 32.

[64] X. Zhang, X. Zhou, M. Lin, and J. Sun. “Shufflenet: An extremely efficient convolutional neural network for mobile devices”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6848–6856.

[65] Z. Cai, X. He, J. Sun, and N. Vasconcelos. “Deep learning with low precision by half-wave gaussian quantization”. In: *arXiv preprint arXiv:1702.00953* (2017).

[66] A. Polino, R. Pascanu, and D. Alistarh. “Model compression via distillation and quantization”. In: *arXiv preprint arXiv:1802.05668* (2018).

[67] L. Hou, Q. Yao, and J. T. Kwok. “Loss-aware binarization of deep networks”. In: *arXiv preprint arXiv:1611.01600* (2016).

[68] S. Darabi, M. Belbahri, M. Courbariaux, and V. P. Nia. “BNN+: Improved binary network training”. In: *arXiv preprint arXiv:1812.11800* (2018).

[69] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid. “Structured Binary Neural Networks for Accurate Image Classification and Semantic Segmentation”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[70] S. Zhu, X. Dong, and H. Su. “Binary Ensemble Neural Network: More Bits per Network or More Networks per Bit?” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4923–4932.

[71] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[72] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[73] F. Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[74] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger. “Condensenet: An efficient densenet using learned group convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2752–2761.

[75] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.* “Searching for mobilenetv3”. In: *arXiv preprint arXiv:1905.02244* (2019).

[76] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. “Shufflenet v2: Practical guidelines for efficient cnn architecture design”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 116–131.

[77] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi. “Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9190–9200.

[78] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. “Efficient neural architecture search via parameter sharing”. In: *arXiv preprint arXiv:1802.03268* (2018).

[79] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.

[80] H. Cai, L. Zhu, and S. Han. “ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HylVB3AqYm.

[81] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. “Mnasnet: Platform-aware neural architecture search for mobile”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2820–2828.

[82] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.

[83] H. Liu, K. Simonyan, and Y. Yang. “Darts: Differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).

[84] Z. Baozhou, H. P. Hofstee, L. Jinho, and Z. Al-Ars. “Unified Effective Depth Reduction Techniques for Binary Convolutional Neural Networks”. In: *(under review)*.

[85] G. Larsson, M. Maire, and G. Shakhnarovich. “Fractalnet: Ultra-deep neural networks without residuals”. In: *arXiv preprint arXiv:1605.07648* (2016).

[86] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. “Deeply-supervised nets”. In: *Artificial intelligence and statistics*. 2015, pp. 562–570.

[87] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. “Fitnets: Hints for thin deep nets”. In: *arXiv preprint arXiv:1412.6550* (2014).

[88] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. “A survey of the recent architectures of deep convolutional neural networks”. In: *Artificial Intelligence Review* (2020), pp. 1–62.

[89] T. Verelst and T. Tuytelaars. “Dynamic Convolutions: Exploiting Spatial Sparsity for Faster Inference”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[90] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. “Deep learning for generic object detection: A survey”. In: *International journal of computer vision* 128.2 (2020), pp. 261–318.

[91] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. “Rethinking atrous convolution for semantic image segmentation”. In: *arXiv preprint arXiv:1706.05587* (2017).

[92] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang. “Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[93] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han. “APQ: Joint Search for Network Architecture, Pruning and Quantization Policy”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2078–2087.

[94] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer. “Zeroq: A novel zero shot quantization framework”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13169–13178.

[95] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang. “Block-wisely Supervised Neural Architecture Search with Knowledge Distillation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1989–1998.

[96] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr. “WRPN: wide reduced-precision networks”. In: *arXiv preprint arXiv:1709.01134* (2017).

[97] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos. “Training binary neural networks with real-to-binary convolutions”. In: *arXiv preprint arXiv:2003.11535* (2020).

[98] A. Bulat and G. Tzimiropoulos. “XNOR-Net++: Improved Binary Neural Networks”. In: *arXiv* (2019), arXiv–1909.

[99] B. Zhu, Z. Al-Ars, and W. Pan. *Towards Lossless Binary Convolutional Neural Networks Using Piecewise Approximation*. 2020. arXiv: 2008.03520 [cs.CV].

[100] Z. Wang, J. Lu, C. Tao, J. Zhou, and Q. Tian. “Learning channel-wise interactions for binary convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 568–577.

[101] R. Ding, T.-W. Chin, Z. Liu, and D. Marculescu. “Regularizing activation distribution for training binarized deep networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11408–11417.

[102] F. Lahoud, R. Achanta, P. Márquez-Neila, and S. Süsstrunk. “Self-binarizing networks”. In: *arXiv preprint arXiv:1902.00730* (2019).

[103] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua. “Quantization networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7308–7316.

[104] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, and J. Song. “Forward and Backward Information Retention for Accurate Binary Neural Networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2250–2259.

[105] J. Bethge, H. Yang, M. Bornstein, and C. Meinel. “BinaryDenseNet: Developing an Architecture for Binary Neural Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019, pp. 249–256.

[106] Z. Zhang, J. Li, W. Shao, Z. Peng, R. Zhang, X. Wang, and P. Luo. “Differentiable Learning-to-Group Channels via Groupable Convolutional Neural Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 3542–3551.

[107] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer. “Shift: A zero flop, zero parameter alternative to spatial convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9127–9135.

[108] 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: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 4852–4861.

[109] B. Zhuang, L. Liu, M. Tan, C. Shen, and I. Reid. “Training quantized neural networks with a full-precision auxiliary module”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1488–1497.

[110] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer. “Mixed precision quantization of convnets via differentiable neural architecture search”. In: *arXiv preprint arXiv:1812.00090* (2018).

[111] Y. Li, W. Wang, H. Bai, R. Gong, X. Dong, and F. Yu. “Efficient Bitwidth Search for Practical Mixed Precision Neural Network”. In: *arXiv preprint arXiv:2003.07577* (2020).

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

[113] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu. "Residual networks of residual networks: Multilevel residual networks". In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.6 (2017), pp. 1303–1314.

[114] H. Hu, D. Dey, A. Del Giorno, M. Hebert, and J. A. Bagnell. "Log-DenseNet: How to sparsify a densenet". In: *arXiv preprint arXiv:1711.00002* (2017).

[115] L. Zhu, R. Deng, M. Maire, Z. Deng, G. Mori, and P. Tan. "Sparsely aggregated convolutional networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 186–201.

[116] Z. Xu and R. C. Cheung. "Accurate and compact convolutional neural networks with trained binarization". In: *arXiv preprint arXiv:1909.11366* (2019).

[117] Z. Baozhou, Z. Al-Ars, and H. P. Hofstee. "REAF: Reducing Approximation of Channels by Reducing Feature Reuse Within Convolution". In: *IEEE Access* 8 (2020), pp. 169957–169965.

[118] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[119] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei. "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 82–92.

[120] H. Seong, J. Hyun, and E. Kim. "FOSNet: an end-to-end trainable deep neural network for scene recognition". In: *IEEE Access* 8 (2020), pp. 82066–82077.

[121] T. Vu, H. Jang, T. X. Pham, and C. Yoo. "Cascade RPN: Delving into High-Quality Region Proposal Network with Adaptive Convolution". In: *Advances in Neural Information Processing Systems*. 2019, pp. 1430–1440.

[122] S. Kornblith, J. Shlens, and Q. V. Le. "Do better imagenet models transfer better?" In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2661–2671.

[123] A. Dalca, M. Rakic, J. Guttag, and M. Sabuncu. "Learning conditional deformable templates with convolutional networks". In: *Advances in neural information processing systems*. 2019, pp. 804–816.

[124] X. Li, W. Wang, X. Hu, and J. Yang. "Selective kernel networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 510–519.

[125] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. "Momentum contrast for unsupervised visual representation learning". In: *arXiv preprint arXiv:1911.05722* (2019).

[126] L. Jing and Y. Tian. “Self-supervised visual feature learning with deep neural networks: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[127] K. He, X. Zhang, S. Ren, and J. Sun. “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[128] X. Wang, M. Kan, S. Shan, and X. Chen. “Fully learnable group convolution for acceleration of deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 9049–9058.

[129] M. Tan and Q. V. Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *arXiv preprint arXiv:1905.11946* (2019).

[130] Y. Bengio, A. Courville, and P. Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[131] S. Zagoruyko and N. Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[132] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[133] I. Bello, B. Zoph, A. Vaswani, J. Shlens, and Q. V. Le. “Attention augmented convolutional networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 3286–3295.

[134] S. Xie, A. Kirillov, R. Girshick, and K. He. “Exploring randomly wired neural networks for image recognition”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1284–1293.

[135] J. Hu, L. Shen, S. Albanie, G. Sun, and A. Vedaldi. “Gather-excite: Exploiting feature context in convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 9401–9411.

[136] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu. “Residual dense network for image super-resolution”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2472–2481.

[137] A. S. Winoto, M. Kristianus, and C. Premachandra. “Small and slim deep convolutional neural network for mobile device”. In: *IEEE Access* 8 (2020), pp. 125210–125222.

[138] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[139] R. K. Srivastava, K. Greff, and J. Schmidhuber. “Highway networks”. In: *arXiv preprint arXiv:1505.00387* (2015).

[140] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. “Deep networks with stochastic depth”. In: *European conference on computer vision*. Springer. 2016, pp. 646–661.

[141] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. “The lottery ticket hypothesis at scale”. In: *arXiv preprint arXiv:1903.01611* (2019).

[142] J. Jeong and J. Shin. “Training cnns with selective allocation of channels”. In: *arXiv preprint arXiv:1905.04509* (2019).

[143] X. Ding, X. Zhou, Y. Guo, J. Han, J. Liu, *et al.* “Global Sparse Momentum SGD for Pruning Very Deep Neural Networks”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 6379–6391.

[144] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun. “Metapruning: Meta learning for automatic neural network channel pruning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 3296–3305.

[145] X. Gastaldi. “Shake-shake regularization”. In: *arXiv preprint arXiv:1705.07485* (2017).

[146] M. Abdi and S. Nahavandi. “Multi-residual networks: Improving the speed and accuracy of residual networks”. In: *arXiv preprint arXiv:1609.05672* (2016).

[147] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. “Deformable convolutional networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 764–773.

[148] Y. Jeon and J. Kim. “Active convolution: Learning the shape of convolution for image classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4201–4209.

[149] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[150] J. Jin, A. Dundar, and E. Culurciello. “Flattened convolutional neural networks for feedforward acceleration”. In: *arXiv preprint arXiv:1412.5474* (2014).

[151] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, and A. Y. Ng. “Tiled convolutional neural networks”. In: *Advances in neural information processing systems*. 2010, pp. 1279–1287.

[152] Y. Chen, H. Fan, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng. “Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks With Octave Convolution”. In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2019.

[153] 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”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[154] K. Sun, M. Li, D. Liu, and J. Wang. “Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks”. In: *arXiv preprint arXiv:1806.00178* (2018).

[155] M. D. Zeiler and R. Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[156] A. S. Morcos, D. G. Barrett, N. C. Rabinowitz, and M. Botvinick. “On the importance of single directions for generalization”. In: *arXiv preprint arXiv:1803.06959* (2018).

[157] Z. Baozhou, H. P. Hofstee, L. Jinho, and Z. Al-Ars. “An Attention Module for Convolutional Neural Networks”. In: *(under review)*.

[158] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[159] S. Ren, K. He, R. Girshick, and J. Sun. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[160] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.* “Recent advances in convolutional neural networks”. In: *Pattern Recognition* 77 (2018), pp. 354–377.

[161] 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, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1409.1556.

[162] S. Zagoruyko and N. Komodakis. “Wide Residual Networks”. In: *Proceedings of the British Machine Vision Conference (BMVC)*. Ed. by E. R. H. Richard C. Wilson and W. A. P. Smith. BMVA Press, Sept. 2016, pp. 87.1–87.12. ISBN: 1-901725-59-6. DOI: 10.5244/C.30.87. URL: https://dx.doi.org/10.5244/C.30.87.

[163] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.* “Spatial transformer networks”. In: *Advances in neural information processing systems*. 2015, pp. 2017–2025.

[164] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. “DRAW: A Recurrent Neural Network For Image Generation”. In: *International Conference on Machine Learning*. 2015, pp. 1462–1471.

[165] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. “Show, attend and tell: Neural image caption generation with visual attention”. In: *International conference on machine learning*. 2015, pp. 2048–2057.

[166] S. Woo, J. Park, J.-Y. Lee, and I. So Kweon. “Cbam: Convolutional block attention module”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 3–19.

[167] D. Linsley, D. Shiebler, S. Eberhardt, and T. Serre. “Learning what and where to attend with humans in the loop”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=BJgLg3R9KQ.

[168] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool. “Dynamic filter networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 667–675.

[169] B. Klein, L. Wolf, and Y. Afek. “A dynamic convolutional layer for short range weather prediction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 4840–4848.

[170] F. Wu, A. Fan, A. Baevski, Y. Dauphin, and M. Auli. “Pay Less Attention with Lightweight and Dynamic Convolutions”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=SkVhlh09tX.

[171] H. Su, V. Jampani, D. Sun, O. Gallo, E. Learned-Miller, and J. Kautz. “Pixel-adaptive convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11166–11175.

[172] X. Liu, G. Yin, J. Shao, X. Wang, *et al.* “Learning to predict layout-to-image conditional convolutions for semantic image synthesis”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 568–578.

[173] F. Zhao, J. Zhao, S. Yan, and J. Feng. “Dynamic conditional networks for few-shot learning”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 19–35.

[174] I. Bello, B. Zoph, A. Vaswani, J. Shlens, and Q. V. Le. “Attention Augmented Convolutional Networks”. In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2019.

[175] J. Hu, L. Shen, and G. Sun. “Squeeze-and-excitation networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[176] D. Han, J. Kim, and J. Kim. “Deep pyramidal residual networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5927–5935.

[177] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. “Searching for MobileNetV3”. In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2019.

[178] R. A. Rensink. “The dynamic representation of scenes”. In: *Visual cognition* 7.1-3 (2000), pp. 17–42.

[179] M. Corbetta and G. L. Shulman. “Control of goal-directed and stimulus-driven attention in the brain”. In: *Nature reviews neuroscience* 3.3 (2002), p. 201.

[180] Y. Chen, Y. Kalantidis, J. Li, S. Yan, and J. Feng. “Aˆ 2-Nets: Double Attention Networks”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 352–361.

[181] T.-W. Ke, M. Maire, and S. X. Yu. “Multigrid neural architectures”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6665–6673.

[182] I. Bello, S. Kulkarni, S. Jain, C. Boutilier, E. H.-h. Chi, E. Eban, X. Luo, A. Mackey, and O. Meshi. “Seq2Slate: Re-ranking and Slate Optimization with RNNs”. In: *CoRR* abs/1810.02019 (2018). arXiv: 1810.02019. URL: http://arxiv.org/abs/1810.02019.

[183] O. Vinyals, M. Fortunato, and N. Jaitly. “Pointer Networks”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 2692–2700. URL: http://papers.nips.cc/paper/5866-pointer-networks.pdf.

[184] J. Park, S. Woo, J.-Y. Lee, and I.-S. Kweon. “BAM: Bottleneck Attention Module”. In: *British Machine Vision Conference (BMVC)*. British Machine Vision Association (BMVA). 2018.

[185] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang. “Residual attention network for image classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3156–3164.

[186] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[187] X. Wang, R. Girshick, A. Gupta, and K. He. “Non-local neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7794–7803.

[188] H. Li, Y. Liu, W. Ouyang, and X. Wang. “Zoom out-and-in network with map attention decision for region proposal and object detection”. In: *International Journal of Computer Vision* 127.3 (2019), pp. 225–238.

[189] B. Yang, L. Wang, D. F. Wong, L. S. Chao, and Z. Tu. “Convolutional Self-Attention Networks”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 4040–4045.

[190] A. Krizhevsky and G. Hinton. “Learning multiple layers of features from tiny images”. In: *Master’s thesis, Department of Computer Science, University of Toronto* (2009).

[191] 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*. Ieee. 2009, pp. 248–255.

[192] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. “The pascal visual object classes challenge: A retrospective”. In: *International journal of computer vision* 111.1 (2015), pp. 98–136.

[193] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[194] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. “Feature pyramid networks for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[195] Z. Baozhou, H. P. Hofstee, J. Peltenburg, L. Jinho, and Z. Al-Ars. “AutoReCon: Neural Architecture Search-based Reconstruction for Data-free Compression”. In: *IJCAI 2021-30th International Joint Conference on Artificial Intelligence*. 2021.

[196] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang. "Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2009–2018.

[197] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh. "Improved knowledge distillation via teacher assistant". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5191–5198.

[198] H. Chen, Y. Wang, C. Xu, Z. Yang, C. Liu, B. Shi, C. Xu, C. Xu, and Q. Tian. "Data-free learning of student networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 3514–3522.

[199] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry. "ACIQ: analytical clipping for integer quantization of neural networks". In: (2018).

[200] S. Xu, H. Li, B. Zhuang, J. Liu, J. Cao, C. Liang, and M. Tan. "Generative Low-bitwidth Data Free Quantization". In: *arXiv preprint arXiv:2003.03603* (2020).

[201] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling. "Data-free quantization through weight equalization and bias correction". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1325–1334.

[202] R. G. Lopes, S. Fenu, and T. Starner. "Data-free knowledge distillation for deep neural networks". In: *arXiv preprint arXiv:1710.07535* (2017).

[203] Y. Choi, J. Choi, M. El-Khamy, and J. Lee. "Data-Free Network Quantization With Adversarial Knowledge Distillation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 710–711.

[204] G. K. Nayak, K. R. Mopuri, V. Shaj, R. V. Babu, and A. Chakraborty. "Zero-shot knowledge distillation in deep networks". In: *arXiv preprint arXiv:1905.08114* (2019).

[205] G. Fang, J. Song, C. Shen, X. Wang, D. Chen, and M. Song. "Data-Free Adversarial Distillation". In: *arXiv preprint arXiv:1912.11006* (2019).

[206] J. Yoo, M. Cho, T. Kim, and U. Kang. "Knowledge extraction with no observable data". In: *Advances in Neural Information Processing Systems*. 2019, pp. 2705–2714.

[207] P. Micaelli and A. J. Storkey. "Zero-shot knowledge transfer via adversarial belief matching". In: *Advances in Neural Information Processing Systems*. 2019, pp. 9551–9561.

[208] C. Gao, Y. Chen, S. Liu, Z. Tan, and S. Yan. "Adversarialnas: Adversarial neural architecture search for gans". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5680–5689.

[209] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. "Searching for efficient multi-scale architectures for dense image prediction". In: *Advances in neural information processing systems*. 2018, pp. 8699–8710.

[210] J. Peng, M. Sun, Z.-X. ZHANG, T. Tan, and J. Yan. “Efficient neural architecture transformation search in channel-level for object detection”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 14313–14322.

[211] M. Haroush, I. Hubara, E. Hoffer, and D. Soudry. “The knowledge within: Methods for data-free model compression”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8494–8502.

CURRICULUM VITÆ

Baozhou ZHU

Baozhou Zhu was born 1992 in Heze, China. He received his B.S. degree in mechanical engineering and automation from South China University of Technology (SCUT), China, in 2015. He received his M.S. degree in microelectronics and solid-state electronics from the National University of Defense and Technology (NUDT), China, in 2017. In September 2017, he joined as a Ph.D. candidate the Quantum and Computer Engineering Department, Delft University of Technology, The Netherlands. His research interests include the efficient inference in deep neural networks.

List of Publications

7. **Baozhou, Zhu**, N. Ahmed, J. Peltenburg, K. Bertels, and Z. Al-Ars. "Diminished-1 Fermat Number Transform for Integer Convolutional Neural Networks". In: 2019 IEEE 4th International Conference on Big Data Analytics (ICBDA). IEEE. 2019, pp. 47–52.

6. **Zhu, Baozhou**, Z. Al-Ars, and W. Pan. "Towards lossless binary convolutional neural networks using piecewise approximation". In: European Conference on Artificial Intelligence (ECAI). 2020.

5. **Zhu, Baozhou**, Z. Al-Ars, and H. P. Hofstee. "NASB: Neural Architecture Search for Binary Convolutional Neural Networks". In: 2020 International Joint Conference on Neural Networks (IJCNN). IEEE. 2020, pp. 1–8.

4. **Baozhou, Zhu**, Z. Al-Ars, and H. P. Hofstee. "REAF: Reducing Approximation of Channels by Reducing Feature Reuse Within Convolution". In: IEEE Access (2020).

3. **Baozhou, Zhu**, H. P. Hofstee, J. Peltenburg, L. Jinho, and Z. Al-Ars. "AutoReCon: Neural Architecture Search-based Reconstruction for Data-free Compression". In: IJCAI 2021-30th International Joint Conference on Artificial Intelligence. 2021.

2. **Baozhou, Zhu**, H. P. Hofstee, L. Jinho, and Z. Al-Ars. "An Attention Module for Convolutional Neural Networks". In: 2021 International Conference on Artificial Neural Networks (ICANN). IEEE. 2021.

1. **Baozhou, Zhu**, H. P. Hofstee, L. Jinho, and Z. Al-Ars. "Unified Effective Depth Reduction Techniques for Binary Convolutional Neural Networks". (under review)