FASTER CONVOLUTIONAL NEURAL NETWORKS TRAINING

by

Shanshan Jiang

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2021

Approved by:

_____
Dr. Sheng-Guo Wang

_____
Dr. Bei-Tseng Chu

_____
Dr. Aidong Lu

_____
Dr. Shen-En Chen

ABSTRACT

SHANSHAN JIANG. Faster convolutional neural networks training.
(Under the direction of DR. SHENG-GUO WANG)

Convolutional Neural Network (CNN) models have become the mainstream method in Artificial Intelligence (AI) areas for computer vision tasks like image classification and image segmentation. Deep CNNs contain a large volume of convolution calculations. Thus, training a CNN requires powerful GPU resources. Training a large CNN may take days or even weeks, which is time-consuming and costly. When we need multiple runs to search for the optimal CNN hypermeter settings, it would take a couple of months with limited GPUs, which is not acceptable and hinders the development of CNNs. It is essential to train CNN faster.

There are two kinds of methods to train CNN faster when no additional computing resources are available. The first method is to do the model compression, either by reducing parameters or using less storage to represent the models. This method reduces training time by reducing the architecture complexity. The second method is to reduce the input data feed into the network without affecting the network architecture.

Architecture complexity reduction is a popular research area to train CNN faster. Nowadays, mobile devices like smartphones and smart cars rely on deep CNNs to accomplish complex tasks like human body recognition and face recognition. Due to the high real-time demands and the memory constraints for mobile device applications, conventional large CNN is not suitable. CNN model compression is a trend to train a deep CNN model with less computation

cost. Currently, there are many successful networks designed to solve this problem, like ResNeXt, MobileNet, ShuffleNet, and GhostNet. They use 1×1 convolution, depthwise convolution, or group convolution to replace the standard convolution to reduce the computation. However, there are fewer studies on the following questions. First, does the variety of convolution layers (the output channel number is larger or smaller than the input channel number) affect different compression strategies' performance? Second, does the expansion ratio (either the output channel number over the input channel number if the output channel number is larger, or the input channel number over the input channel number if the input channel is larger) of the convolution layers affect different compression strategies' performance? Third, does the compression ratio (the reduced parameter number/FLOPs over the original parameter number/FLOPs) affect the performance of different compression strategies? Current networks tend to use the same convolution strategy inside a basic network block, ignoring the variety of network layers. We have proposed a novel Conditional Reduction (CR) module to compress a single 1×1 convolution layer. Then we have developed a novel three-layer Conditional block (C-block) to compress the CNN bottleneck or inverted bottlenecks. At last we have developed a novel Conditional Network (CRnet) based on the CR module and C-block. We have tested the CRnet on two image classification datasets: CIFAR-10 and CIFAR-100, with multiple network expansion ratios and compression ratios. The experiments verify our methods' correctness with attention to the importance of the input-output pattern when selecting a compression strategy. The experiments show that our proposed CRnet better balances the model complexity and accuracy compared to the state-of-the-art group convolution and Ghost module compression.

Data reduction reduces the training time in a direct and simple way through data dropping. There are works drop data by the sample importance ranking. The ranking process takes extra time when there is a large number of training samples. When we tune the different network settings to search for an optimal setting, we expect a way to reduce a large percentage of training time with tiny or no accuracy loss. There are fewer studies on the following questions. First, what are suitable sampling ratios? Second, should we use the same sampling ratio for each training epoch? Third, does the sampling ratio performs differently on small and large datasets? We have proposed a flat reduced random sampling training strategy and a bottleneck reduced random sampling strategy. We have proposed a three-stage training method based on the bottleneck reduced random sampling with consideration of the distinctiveness of the network early-stage training and end-stage training. Furthermore, we have proved the data visibility of a sample in the whole training process and the theoretical reduced time by four theorems and two corollaries. We have tested the two sampling strategies on three image classification datasets: CIFAR-10, CIFAR-100 and ImageNet. The experiments show that our proposed two sampling strategies effectively reduce a significant training time percentage at a very small accuracy loss.

ACKNOWLEDGEMENTS

I first would like to express my deep thanks and respect to my advisor Dr. Sheng-Guo Wang, who provides me the chance to do my Ph.D. study and participate in exciting research projects related to artificial intelligence. Without his funding support, I would not be able to complete this dissertation. I have learned a lot from him in critical thinking and writing. His passion for work and rigorous attitude in math has taught me how to be a responsible researcher. Especially, I am grateful for his remarkable patience throughout my research and projects, which reminds me to be rational and conscientious.

I would like to thank my dissertation committee members: Dr. Aidong Lu, Dr. Bill Chu, and Dr. Shen-en Chen, for their heuristic questions and comments. The discussions about the latest development of deep learning and visualization applications are insightful and encouraging.

My sincere thanks go to my parents and my husband for giving me strong emotional support. My parents always tell me never to give up. My husband, Mr. Muqi Li, is also my friend and life teacher. He teaches me to be faithful to myself and live earnestly. I am grateful for his selfless love.

I would like to thank my teammates and friends, Dr. Jing Deng, Dr. Yinan He, and Zhongqi Hao. Jing encourages me to challenge myself and try new things. Yinan provides me many helpful suggestions in dissertation writing and project works. Zhongqi Hao provides me with kindly help in my life and accompanies me to get through some challenging moments.

TABLE OF CONTENTS

LIST OF TABLES X

LIST OF FIGURES XI

LIST OF ABBREVIATIONS XIII

CHAPTER 1: INTRODUCTION 1

1.1 ARCHITECTURE COMPLEXITY REDUCTION 2

1.2 DATA REDUCTION 7

1.3 CONTRIBUTIONS 8

1.3.1 Contribution in Architecture Complexity Reduction 8

1.3.2 Contribution in Data Reduction 10

1.4 DISSERTATION ARRANGEMENTS 11

CHAPTER 2: RELATED WORK 12

2.1 ARCHITECTURE COMPLEXITY REDUCTION 12

2.1.1 Model Compression 12

2.1.2 Convolution Layer Input-Output Pattern 14

2.1.3 CNN Bottleneck and Inverted Bottleneck 15

2.1.4 Compression Strategy 17

2.2 DATA REDUCTION 18

2.2.1 Data Sampling 18

2.2.2 Epoch and Batch 19

2.2.3 Data Visibility 20

2.2.4 Data Reduction Strategy 20

CHAPTER 3: METHODOLOGY 22

3.1 Architecture Complexity Reduction 22

3.1.1 Conditional Reduction Module 23

3.1.2 Conditional Block 27

3.1.3 Conditional Reduction Network 30

3.2 Data Reduction 33

3.2.1 Flat Reduced Random Sampling 33

3.2.2 Bottleneck Reduced Random Sampling 36

3.2.3 Three-Stage Training 39

CHAPTER 4: EXPERIMENT 43

4.1 Architecture Complexity Reduction 43

4.1.1 Dataset 43

4.1.2 Evaluation Metrics 43

4.1.3 CRnet-CB Results 45

4.1.4 CRnet-CIB Results 51

4.2 Data Reduction 57

4.2.1 Dataset 57

4.2.2 Evaluation Metrics 58

4.2.3 CIFAR-10 Experiments 58

4.2.4 CIFAR-100 Experiments 61

4.2.5 ImageNet Experiments 64

CHAPTER 5: CONCLUSIONS AND FUTURE WORKS 67

5.1 CONCLUSIONS ON ARCHITECTURE COMPLEXITY REDUCTION 67

5.2 FUTURE WORK ON ARCHITECTURE COMPLEXITY REDUCTION 69

5.3 CONCLUSIONS ON DATA REDUCTION 69

5.4 FUTURE WORK ON DATA REDUCTION 70

REFERENCES 71

LIST OF TABLES

Table 3-1: CRnet for CIFAR-10 and CIFAR-100 (backbone: ResNet50) 31

Table 3-2: CRnet-CIB for CIFAR-10 and CIFAR-100 (backbone: GhostNet)..... 32

Table 4-1: CRnet-CB results on CIFAR-10 with a batch size of 256 46

Table 4-2: CRnet-CB results on CIFAR-10 with a batch size of 64 47

Table 4-3: CRnet-CB results on CIFAR-100 with a batch size of 256 48

Table 4-4: CRnet-CB results on CIFAR-100 with a batch size of 64 49

Table 4-5: CRnet-CIB results on CIFAR-10 with a batch size of 256 52

Table 4-6: CRnet-CIB results on CIFAR-10 with a batch size of 64 53

Table 4-7: CRnet-CIB results on CIFAR-100 with a batch size of 256 54

Table 4-8: CRnet-CIB results on CIFAR-100 with a batch size of 64 55

Table 4-9: Training performances on CIFAR-10 59

Table 4-10: Training performances analysis on CIFAR-10 60

Table 4-11: Training performances on CIFAR-100 62

Table 4-12: Training performances analysis on CIFAR-100 63

Table 4-13: Training performances on ImageNet..... 65

Table 4-14: Training performances analysis on ImageNet..... 65

LIST OF FIGURES

Figure 2.1: Input-output patterns of a convolution layer 15

Figure 2.2: Bottleneck block 16

Figure 2.3: Inverted Bottleneck block 16

Figure 2.4: Ghost module 18

Figure 3.1: Feature maps of the first convolution layer of ResNet56 24

Figure 3.2: Conditional Reduction module to compress the 1×1 convolution 25

Figure 3.3: Conditional block (C-block). 28

Figure 3.4: Conditional Bottleneck block (C-Bneck). 29

Figure 3.5: Conditional Inverted Bottleneck block (C-IBneck). 29

Figure 3.6: Flat reduced sampling. 34

Figure 3.7: Bottleneck sampling block 37

Figure 3.8: Stacking of bottleneck sampling blocks. 38

Figure 3.9: Three-stage training method based on bottleneck sampling method. 39

Figure 4.1: Relative training time reduction vs relative top-1 accuracy change for CIFAR-10 dataset 60

Figure 4.2: Relative training time reduction vs relative top-5 accuracy change for CIFAR-10 dataset 61

Figure 4.3: Relative training time reduction vs relative top-1 accuracy change for CIFAR-100 dataset 63

Figure 4.4: Relative training time reduction vs relative top-5 accuracy change for CIFAR-100 dataset 64

Figure 4.5: Relative training time reduction vs relative top-1 accuracy change for ImageNet dataset 66

Figure 4.6: Relative training time reduction vs relative top-5 accuracy change for ImageNet dataset 66

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AMC	AutoML Model Compression
API	Application Programming Interface
BN	Batch Normalization
c1	1×1 convolution
C-block	Conditional Block
C-Bneck	Conditional Bottleneck
C-IBneck	Conditional Inverted Bottleneck
CNN	Convolutional Neural Network
CR	Conditional Reduction
CRnet	Conditional Reduction Network
CRnet-CB	Conditional Reduction Network with Conditional Bottlenecks
CRnet-CIB	Conditional Reduction Network with Inverted Conditional Bottlenecks
dw	depthwise convolution
FLOPs	floating point operations
gc	group convolution
GPU	Graphics Processing Unit
NIN	Network In Network
RAM	Random Access Memory
SE	Squeeze and Excite
SGD	Stochastic Gradient Descent

CHAPTER 1: INTRODUCTION

The deep CNN methods have outperformed other conventional machine learning methods in AI areas, including image classification [1, 2], image segmentation [3, 4], and image inpainting [5, 6]. Training deep CNN is challenging due to its high demands of CPU and GPU resources. Training a large dataset may even take several days or weeks [7]. CNN's training hyperparameter number setting affects the model performance, e.g., the initial learning rate, learning rate decay, batch size, activation function selection. Tuning the hyperparameters is essential to achieve better performance [7, 8], which will take several rounds and costs far more time than training a single model. The time-consuming training process becomes a hinder for researchers to explore optimal networks. Thus, accelerating the CNN training speed has become a new research area. To train CNN faster is a benefit to saving training time and saving the training cost.

There are many works discussing how to train a faster CNN. Some works focus on training a compact model to reduce the model complexity and to reduce the average training time of a single image. In this case, the model parameters and FLOPs (floating-point operations) are usually reduced. Thus, the inference time will also be reduced correspondingly. Some works focus on reducing the total training time by reducing the training data for some epochs. In this dissertation, we have explored the new methods to train a faster CNN in both reducing single image average training time and total training time. We have implemented our methods on image classification benchmark datasets, and the results show the correctness and efficiency of our methods.

1.1 Architecture Complexity Reduction

With the fast-developing speed of mobile devices like phones or cars, it is necessary to deploy deep learning models to these mobile devices to accomplish complex computer vision tasks like human body recognition, face recognition, and handwriting recognition. As mobile devices do not have strong computing resources as workstations or cloud computing clusters, it is challenging to utilize conventional deep CNN architectures in portable devices or cars due to their high demand for computing resources [9]. These application demands have motivated many researchers to develop more computing-efficient CNN architectures to reduce model complexity with little or no accuracy trade-off [10]. A compact model with a little sacrifice of accuracy is acceptable compared to a large model since compact models are faster and are easier to be deployed to mobile devices, as well as to be applied, especially for real-time identification systems and control systems.

There are three different branches in treating model redundancy from the architecture level in the model compression area.

One branch, known as network pruning [11-18], removes unimportant elements like convolutional filters or channels following some criteria like weight value or L1-norm value. A convolutional filter is a group of $k \times k$ convolutional kernels where k is the kernel size. A channel is a $w \times h$ feature map where w and h stand for image weight and height, respectively. This kind of model compression process is also mentioned as model pruning [14]. The main idea of model pruning is to prune filters or channels that have the least contribution to the model. Weight pruning or connection pruning [12, 19] discards weights with smaller values and gets a sparse connection.

However, the non-structured connections will cause irregular memory access issues for typical hardware [15, 16], hindering real applications. Filter level pruning [15, 18] measures filter importance by the deeper layer or the final response layer's reconstruction error after removing this filter. The main idea of filter pruning is to prune filters that have little effect on the reconstruction error. This method will keep the model structure and keep pruned model training smoothly after removing the filters, without taking special care of the change of hardware and software support. Nevertheless, it is an optimization process for each pruning step and needs fine-tuning. It will iteratively remove filters, following by a fine-tuning process after pruning a filter or a layer [18]. The fine-tuning needs to run a few epochs, which will take a long time for big data training. In addition to this, the pruned optimal network architecture based on one dataset may not be the optimal architecture of another dataset. Some researchers try to learn optimal compression ratio or channel number through reinforcement learning instead of the human setting. AutoML Model Compression (AMC) [20] leverages reinforcement learning to study optimal compression policy for each pre-trained model layer. The output of each layer determination is a sparsity ratio. AMC did not use fine-tuning in getting the reward step. As different mobile devices have different computation restraints, slimmable networks [17, 21] were proposed based on the pruning idea to train an integrated model that contains models to meet different environment requirements. Slimmable neural networks [21] train models of predefined multiple widths at the same time. Sub-networks share weights except for batch normalization parameters. Universally slimmable networks (US-Nets) [22] proposed a generalized slimmable network to generate arbitrary width sub-networks using distillation to transfer knowledge from the entire network to subnetworks.

AutoSlim [17] searches for an optimal channel setting based on a universally slimmable network to meet computation limitation requirements. AutoSlim trains a slimmable model for a few epochs to get model candidates, slims the layer greedily to get optimized channel setting, and then trains them for full epochs [17]. The core sight of pruned networks and slimmable neural networks is to remove network elements following importance measuring criteria like L1-norm or construction error. This process can be seen as reducing network redundancy. There are three steps to build a pruned model: pretraining, pruning, and fine-tuning. However, there are arguments about the necessity of the three-step process. Liu [22] observed that training a pruned architecture directly from scratch can reach comparable results without inheriting the pre-trained weight, which indicates the importance of searching for an efficient architecture. The pretraining process trains a big model with redundancy. The pruning process studies the importance of the target elements to prune the network. The fine-tuning process retrains the pruned network with the pre-trained big model's weight as the initial state. The advantage of pruning is that the pre-trained model can provide a reference for pruning and fine-tuning so that the pruned model searched for an optimal way to remove elements contributing the least to the network. However, the three-step process is time-consuming. For each dataset, a pre-trained model is essential, taking a long time for a large dataset. A study has also shown that the pruned structure itself may be more important than the inherited weights pre-trained [22]. What is more, the pruned network only prunes elements. Thus no new network elements are studied and explored during this process, which may become a constraint for network compression's future development.

Knowledge distillation is another branch of model compression. It is interested in extracting the learned knowledge from trained networks and migrating them to another network. It is a mainstream method in transfer learning. Knowledge distillation can leverage the ensemble multimodal to build a single model and leverage the large model's knowledge to build a small model [23]. A compact student model can be achieved from a large teacher model. A teacher model could provide many kinds of knowledge to the student model, like soft labels [23], spatial attention maps [24] and flow between layers [25]. The advantage of knowledge distillation is that it is an efficient way to broadcast the pre-trained networks' learned knowledge. Thus it has been broadly used in transfer learning. However, it needs to run both teacher and student models during training, which may cause memory shortage when GPU resources are limited during training since it needs memory to store both teacher and student models and results.

The third branch of model compression concentrates on implementing a network using more computation efficient ways, usually called building compact models. It aims to build a network with fewer parameters and FLOPs. Instead of removing network elements to reduce the redundancy, compact models [9, 10, 26-31] tend to keep the redundancy to some extent but replace dense convolution with sparse convolution. There are many efficient methods like using 1×1 convolution to replace some 3×3 convolution [32], using depthwise convolution [26] to replace normal convolution, and reducing the number of 1×1 dense convolution [10]. The advantage of training a compact network is that it studies the nature of the network, exploring different network structures. Additionally, since the compact model does not need pretraining and fine-tuning and computes faster to generate the same number of feature maps, it is time-saving, especially for large

datasets. However, current compact models usually adopt the same compression policies for different convolution environments, ignoring their inner difference. For example, in a bottleneck block, a convolution layer's input and output have two patterns: large input and small output, and small input and large output. Current methods tend to use the same strategies for two patterns, like replacing all 1×1 convolution layers to group convolution or reducing the number of 1×1 convolution channels. Intuitively, the large input and small output convolution can be seen as using fewer feature maps to encode and summarize previous features. The small input and large output can be seen as a brainstorm of current features, putting forward more possible features that would be useful. Thus, these kinds of differences should be considered when building their compact structures. Since the same compression strategy can perform differently in different cases, it is beneficial to study the element patterns in networks and design a flexible compression strategy, leading to a network with better performance and less computation cost.

In this dissertation, we focus on the building compact models branch and answering the following questions. First, does the variety of convolution layers (the output channel number is larger or smaller than the input channel number) affect different compression strategies' performance? Second, does the expansion ratio (either the output channel number over the input channel number if the output channel number is larger, or the input channel number over the input channel number if the input channel is larger) of the convolution layers affect different compression strategies' performance? Third, does the 1×1 channel number compression ratio (the reduced parameter number/FLOPs over the original parameter number/FLOPs) affect the performance of different compression strategies?

1.2 Data Reduction

Section 1.1 introduces a way to reduce the training time by reducing the CNN architecture complexity. The nature of this way is to reduce the average training time of a single image. This means the forward pass of the CNN for one image is reduced since the model has been compressed by different kinds of compression strategies. Correspondingly, this way will also reduce the predicting time.

When a base CNN architecture is given, there are many ways to train CNN faster without modifying the network architecture [7]. Currently, there are works using a larger batch size on more GPUs [33-37], using lower bits to represent the network parameters [38-41], and using dropping techniques[42-49] to reduce the training time without affecting the network architecture.

One branch is to reduce the CNN parameter data storge. Low-bit quantization is a popular branch in model compression. Its main idea is to reduce the bit representation of the network [50, 51]. Usually, the networks' parameters are stored using a 32-bit floating-point [51], and it is a trend to migrate training to 16-bit precision [52]. Singular Value Decomposition compressed the matrices [53] during inference. Vector quantization [50] reduces the parameter storage redundancy from the vector level. Stochastic rounding [51] converts a number to a lower precision fixed-point considering the probability to round x to $\lfloor x \rfloor$ during training. The state-of-the-art work reports the possibility of training network parameters with 8-bit floating-point [52]. Low-bit quantization reduces the model computation resource from the energy-saving perspective. This branch focuses on reducing training or prediction costs from the hardware perspective.

Another branch is to reduce the training sample data. The training sample data can be reduced either in sample number [42-48] or sample image dimension [49]. There are works to calculate the sample losses [43, 44] and set the selection probability [42] or drop this sample based on the calculation. Reference [45] recovers the dropped samples after training for a few cycles. Reference [47] updates the learning rate based on the importance of the samples. These kinds of non-uniform data reduction methods drop data based on data losses and importance rank. When the dataset is large, the ranking process for each epoch will also take extra time and slow the training process.

In this dissertation, we focus on using a simple way to reduce the training time and answering the following questions. First, what are suitable sampling ratios? Second, should we use the same sampling ratio for each training epoch? Third, does the sampling ratio perform differently on small and large datasets?

1.3 Contributions

The objective of this dissertation is to develop new methods to train CNN faster from the network architecture perspective and data reduction perspective. The new methods from the architecture perspective explore the new architecture of CNN to reduce the training time. The new methods from the data reduction perspective reduce the training time without changing the CNN architecture.

1.3.1 Contribution in Architecture Complexity Reduction

Motivated by the issues mentioned in the above compact model design introduction, to better balance the model complexity and the accuracy, we explore a simple way to reduce the

model complexity while maintaining the model accuracy or even achieving higher accuracy. Our contributions are:

(1) To address the CNN layer input-output different pattern issue, we introduce a novel Conditional Reduction (CR) module. The CR module checks the CNN layer input-output pattern and uses different compression strategies [54].

(2) Based on the CR module, we introduce a novel Conditional block (C-block). The C-block is a uniform format of traditional bottleneck block and inverted bottleneck block after adopting the CR module. The C-block can also be classified as Conditional Bottleneck block (C-Bneck) and Conditional Inverted Bottleneck block (C-IBneck) depending on the base bottleneck type [54].

(3) Based on the C-Bneck and C-IBneck, we introduce a Conditional Reduction Network (CRnet). The CRnet is a uniform format of CNN stacked with bottlenecks or inverted bottlenecks. The CRnet can also be classified as CRnet-CBneck (CRnet-CB) and CRnet-CIBneck (CRnet-CIB) depending on the base block type [54].

(4) We evaluate the introduced CRnet built with the CR module and C-block on two image classification benchmark datasets: CIFAR-10 and CIFAR-100 [54].

The experiments show that:

(1) The introduced CRnet better balances the accuracy and model complexity than the networks adopting the state-of-the-art compression strategies. The CRnet can achieve better accuracy than the network with fewer parameters or FLOPs. The CRnet can achieve fewer

parameters or FLOPs but comparable or higher accuracy than the network with more parameters or FLOPs [54].

(2) The introduced CR module is efficient in replacing any traditional 1×1 convolution layer, making it easily applied to any CNN. Correspondingly, the introduced C-Bneck and C-IBneck are seamless to replace the traditional bottleneck and inverted bottleneck in any CNN, respectively. The plug-and-play characteristic shows that our introduced methods are efficient and easy to use [54].

1.3.2 Contribution in Data Reduction

Motivated by the challenging task of balancing the training speed and accuracy, in this dissertation, we explore a simple data reduction way suitable for any deep learning network training. Our contributions are:

(1) We introduce two kinds of fast training strategies for deep learning training: the flat reduced random sampling strategy and the bottleneck reduced random sampling strategy. The flat reduced random sampling simply randomly reduces a fixed ratio of data for each epoch, and the optimal ratio has been studied. The bottleneck sampling divides epochs into multiple blocks. For each block, the beginning and end epochs have a larger sampling ratio, and the intermediate epochs have a lower sampling ratio [7].

(2) We further introduce a three-stage training method based on the bottleneck reduced random sampling strategy. It combines the flat reduced random sampling, and the bottleneck reduced random sampling. The three-stage training method divides the total epochs into three parts. The first and third parts adopt the flat reduced random sampling. The second part adopts the

bottleneck reduced random sampling. The flat reduced sampling is a special case of the three-stage training method by setting the second part epoch over the total epochs to zero [7].

(3) We evaluate the introduced flat reduced random sampling and three-stage training with bottleneck reduced random sampling on three image classification benchmark datasets: CIFAR-10, CIFAR-100, and ImageNet [7].

(4) We give the theoretical analysis for the two data sampling strategies. We have developed and presented four theorems and two corollaries to show the properties and benefits of the presented new methods [7].

The experiments show that:

(1) The introduced two sampling strategies get significant training time percentage reduction at a very small accuracy loss. Therefore, the strategies are effective in reducing the CNN training time [7].

(2) Both sampling strategies are easy to be applied to deep learning networks [7].

1.4 Dissertation Arrangements

Chapter 1 introduces current methods to train CNN faster, the dissertation objectives, and contributions. Chapter 2 introduces the related work to our proposed methods. Chapter 3 presents the methodologies of our proposed methods. Chapter 4 presents the experiment datasets, parameter settings, and results with the analysis. Chapter 5 finally presents conclusions with a summary analysis of the experimental results and points out the future work.

CHAPTER 2: RELATED WORK

In this chapter, typical and the state-of-the-art methods in architecture complexity reduction from the designing compact model aspect and data reduction from the training data level are reviewed.

2.1 Architecture Complexity Reduction

In this section, we first highlight the related methods in model compression. Then we introduce the essential network elements of our proposed methods.

2.1.1 Model Compression

Instead of removing the redundant network layers or filters, many works replace conventional convolution with more computation-saving ways [10, 55]. Conventional networks are usually stacked with standard 3×3 convolution layers. SqueezeNet [29] fully or partially replaced the 3×3 convolutions to 1×1 convolution in a layer. The mainstream networks are usually stacked with residual blocks that contain multiple standard 3×3 layers since the ResNet [32] proposed the shortcut approach. Bottleneck structure [32] and inverted bottleneck structure [9] were proposed to replace a conventional two 3×3 convolution layer block with a 3×3 convolution layer and two 1×1 convolution layers before and after the 3×3 convolution layer. A standard three-layer bottleneck has a big-small-big pattern in dimension, where the input and output layers of this bottleneck have more channels than the inner layer. The inverted three-layer bottleneck has a small-big-small pattern in dimension, which is more memory efficient if we dispose of the inner convolution tensors after computation [9]. Depthwise convolution [26] further

replaces standard 3×3 convolution by using one channel and a kernel to generate one feature map instead of using multiple channels to generate one feature map. Since depthwise is an efficient way to reduce computation while keeping network capacity, it has been used broadly by the state of the art compression networks like ShuffleNet [31], MobileNet [28], and GhostNet [10]. Different from depthwise convolution to generate a feature map using only one input channel, group convolution generates a feature map using a part of the input channels.

Researchers have observed that the dense 1×1 convolution has become the key structure contributing to model parameter number and complexity after the depthwise convolution replaced standard 3×3 convolution [31]. ResNeXt [56] showed that grouped convolution might improve accuracy while maintaining the same model parameter number and similar model complexity, which means that grouped convolution may help to reduce the accuracy loss in model compression. ShuffleNet [31] adopted a three-layer bottleneck structure and replaced the 1×1 convolution with grouped 1×1 convolution and a channel shuffle operation after the first 1×1 grouped convolution layer. ShuffleNet v2 [30] stated that grouped convolution of the 1×1 convolution should be treated carefully since increased group number may increase the memory access cost and slow down the speed. MobileNet v2 [9] proposed a backbone network using inverted residual blocks consisting of 1×1 pointwise convolution and $N \times N$ depthwise convolution. MobileNet v3 [27] adopted the squeeze and excitation module [57] to improve accuracy. To reduce the computation cost of dense 1×1 convolution, GhostNet replaced a part of 3×3 or 1×1 convolution with depthwise convolution, except that the depthwise convolution outputs were generated from remaining 1×1 convolution outputs instead of original inputs. The core idea of

GhostNet is to reduce the channel number generated by 1×1 convolution and recover the model capacity by depthwise convolution. GhostNet used two modules for convolution stride 1 and 2. For stride 1, GhostNet used a two-layer residual module, with each layer containing 1×1 convolution and depthwise 3×3 convolution. For stride 2, GhostNet used a three-layer inverted residual bottleneck, with a depthwise 3×3 convolution as the middle layer. Filter pruning network [55] utilized this idea of recovering model capacity from remaining feature maps.

2.1.2 Convolution Layer Input-Output Pattern

Figure 2.1 shows two input-output patterns for a convolution layer generally used in the bottleneck blocks and inverted bottleneck blocks. The first pattern is a large-small input-output pattern whose output channel number is smaller than the input channel number. It corresponds to the green layer in Fig. 2.2 and Fig. 2.3. The second pattern is a small-large input-output pattern whose output channel number is larger than the input channel number. It corresponds to the orange layer in Fig. 2.2 and Fig. 2.3. We note that both the bottleneck block and the inverted bottleneck block contain convolution layers of these two patterns. The state-of-the-art compact model design methods like the ShuffleNet [31], MobileNet v2 [9], and GhostNet [10] use the same compression strategies for them.

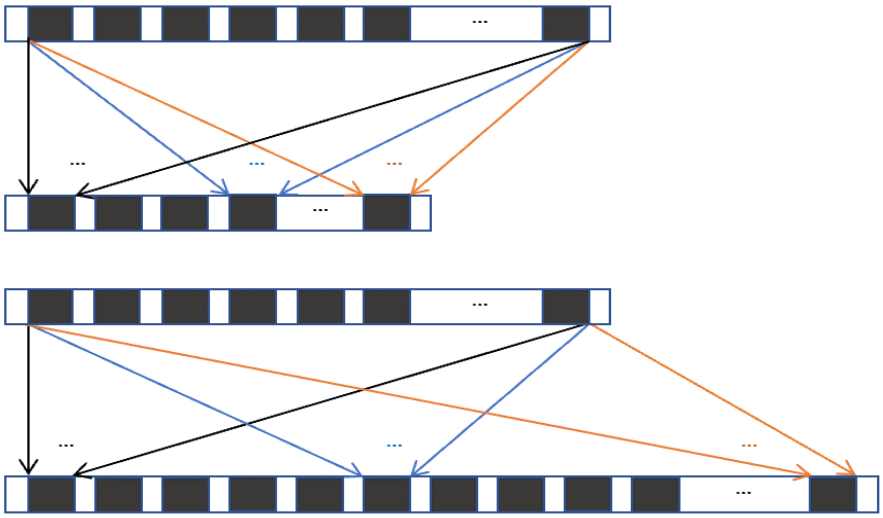

Figure 2.1: Input-output patterns of a convolution layer. Top: large-small pattern. Bottom: small-large pattern.

2.1.3 CNN Bottleneck and Inverted Bottleneck

The bottleneck block is largely used in compact network design methods to reduce the computation cost of the stacking of $N \times N$ convolutions. Figure 2.2 shows the typical 3-layer building of a residual bottleneck proposed in ResNet [32]. The first layer is a 1×1 convolution which will change the channel dimension. The second layer is an $N \times N$ convolution (usually N is 3). The third layer is another 1×1 convolution to restore the channel dimension. So there are two 1×1 convolution layers: the first layer and the third layer. The first 1×1 convolution layer's input-output pattern is large-small, denoted by green color. The second 1×1 convolution layer's input-output pattern is small-large, denoted by orange color. Figure 2.3 shows the typical 3-layer building of a residual inverted bottleneck proposed in MobileNet v2 [9]. Unlike the bottleneck, for the inverted bottleneck, the first 1×1 convolution layer's input-output pattern is small-large, denoted by orange color; and the second 1×1 convolution layer's input-output pattern is large-small, denoted by green color.

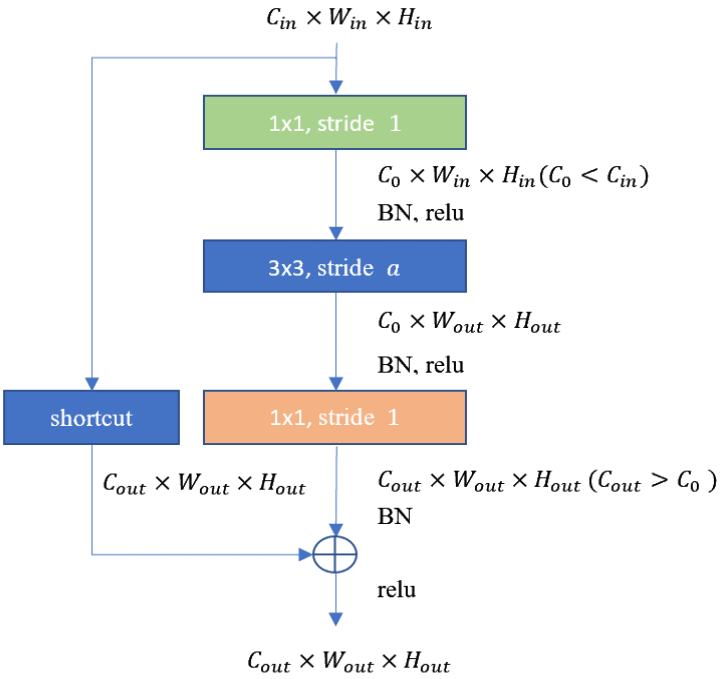

Figure 2.2: Bottleneck block

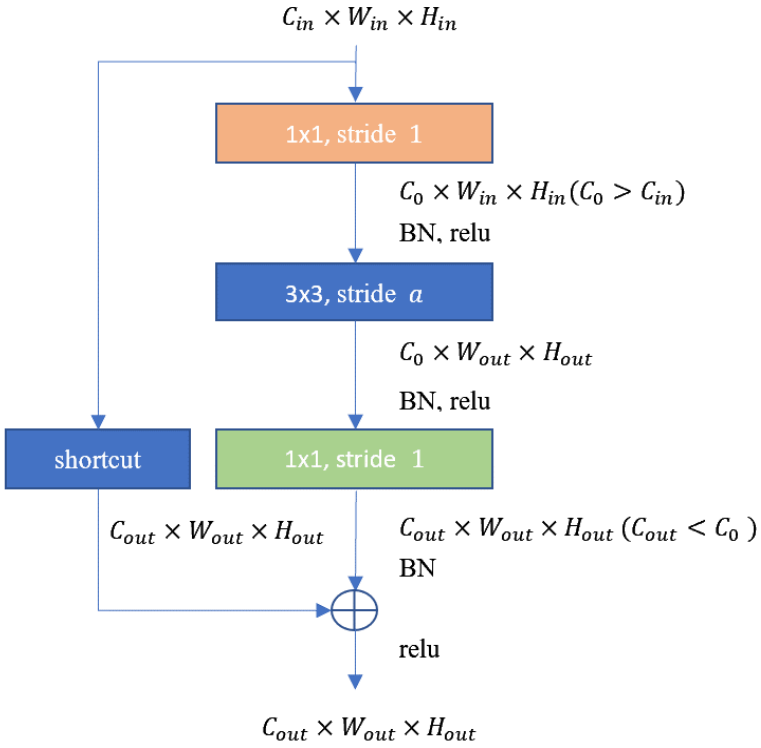

Figure 2.3: Inverted Bottleneck block

2.1.4 Compression Strategy

The 1×1 convolution layer and the 3×3 convolution layer are two important elements to be compressed in the bottleneck blocks or the inverted bottleneck blocks. The mainstream compact networks leverage the depthwise convolution to replace the standard 3×3 convolution, like ShuffleNet [31] and MobileNet v2 [9]. Thus, the dense 1×1 convolutions have become the key elements to be compressed in these CNNs and are the focus of this dissertation.

The intuitive idea to reduce the 1×1 convolution computation is to reduce the input channel number or the output channel number of this 1×1 convolution layer, which means to reduce the number of 1×1 kernels. ShuffleNet [31] uses group convolution to compress the two 1×1 convolution layers in a residual block. GhostNet [10] uses depthwise convolution to generate the output channels that had been cut off from the existing features generated from a reduced number of 1×1 kernels.

Our proposed Conditional Reduction (CR) module is motivated by the Ghost module [10]. Ghost module was proposed in GhostNet [10] for one convolution layer feature map generation. Figure 2.4 shows the basic design of the Ghost module. Ghost module uses normal convolution to generate a reduced number of feature maps called intrinsic features and then uses depthwise convolution to generate other feature maps from intrinsic features. When using Ghost module to compress a 1×1 convolution layer, it contains two parts: normal 1×1 convolution (c1) to get a reduced number of output feature maps and depthwise convolution (dw) to generate the other feature maps from the c1 results. We use the c1dw compression strategy to indicate the two compression steps.

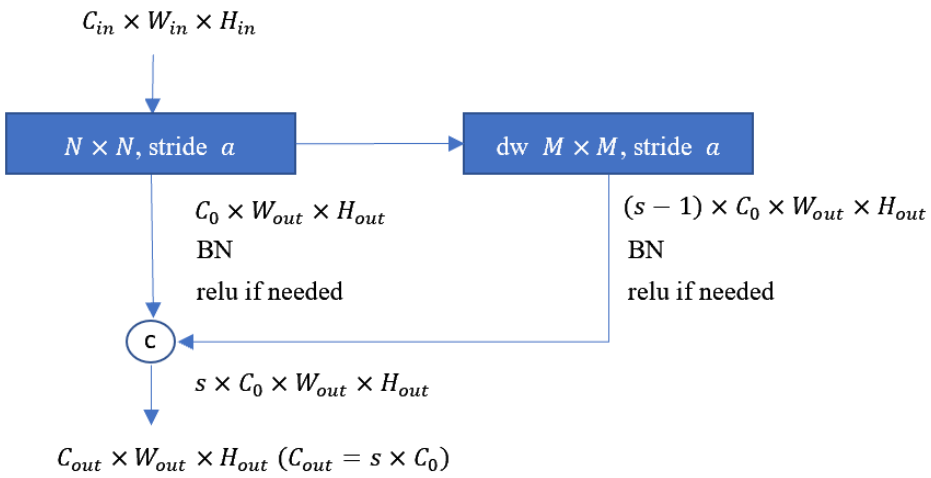

Figure 2.4: Ghost module [10]. Parameter s is the total number of output channel over the output channel generated by normal $N \times N$ convolutions and s > 1. dw stands for depthwise convolution.

2.2 Data Reduction

In this section, we first highlight the related methods in reducing training time without affecting the model architecture. Then we introduce the essential basic knowledge of our proposed methods.

2.2.1 Data Sampling

Some works use the data importance indices ranked by training loss [42, 45, 47] which select the data subsets non-uniformly. The ranking process may take extra time when there is a large number of training samples. Typical deep learning frameworks like Keras, TensorFlow and PyTorch provide users with well-built data access functions (data loaders and samplers) to read images from the local computers or from memory. When using the sample ranking and dropping strategy, we need to refresh the sampler data indices after each dropping, which will take extra work to fit this strategy into the data access interface. Furthermore, the data importance ranking will take extra time when the number of samples is large. Thus, a question arises: is there a more

straightforward way to reduce the training time without affecting existed deep learning data access interface? Inspired by this idea, we propose some novel data reduction strategies which are easily used to fit in any deep learning API (Application Programming Interface) and data access functions.

2.2.2 Epoch and Batch

Traditionally a deep learning training epoch refers to a cycle to complete all training samples. Since the computer is resource-constraint to input all the training samples into the network forward pass, the samples are split into mini-batches. In one iteration, only a batch of images is input to the network forward pass. The most commonly used batch-based gradient descent method updates the weights through the backward propagation of the batch [58]. After the network completes the backward propagation of a batch and updates the parameters, it will read the next batch and repeat the forward and backward processing.

To speed up the training process, we can either reduce the training epochs (full batches-fewer epochs) or reduce the sample number for each epoch (fewer batches-full epochs). It is obvious that we can also use fewer batches-fewer epochs. Enough training epochs are often essential for the deep learning models to converge [58]. It is noted that the model generated from full batches-fewer epochs and fewer batches-full epochs are different. It is due to the learning rate variation related to the epoch number. Take two popular learning rate scheduling methods as examples. The step decay method drops the learning rate at specific epochs defined by the user. The cosine decay method gradually drops the learning rate based on the iteration number and epoch number. Thus, full batches-fewer epochs simply using small epochs to reduce the training time is not expected and will affect the learning rate decay. We use the full epochs-fewer batches to drop

data for each epoch, maintaining the total epoch number. It will fit well with the original training hyperparameters.

2.2.3 Data Visibility

In standard full batches-full epochs training mode, an image sample is used exactly once for each epoch. When the total epoch number is N, the number of an image sample's occurrences is N. In other words, every image sample is visible to the network for each epoch. With shuffling operation before training, the order of an image appearance is different for each epoch. If there is no shuffling operation, the order of an image appearance will be the same for each epoch. Reference [45] proposed a drop-and-refresh training strategy where samples with lower losses will be dropped for a few epochs and reviewed. That means some samples will be invisible to the training process for some epochs and visible again in case the network forgets those samples. The samples are dropped non-uniformly in [45].

2.2.4 Data Reduction Strategy

The training modes can be summarized as four types based on the batch number and epoch number: full batches-full epochs, full batches-fewer epochs, fewer batches-full epochs, and fewer batches-fewer epochs. Our objective is the fewer batches-full epochs mode since it is beneficial to not to interfere with the normal learning rate decay.

We propose two data reduction strategies: flat reduced random sampling and bottleneck reduced random sampling. For the flat reduced random sampling, we are inspired by the popular 80-20 split ratio that is usually used to split training and validation datasets and the 0.618 golden ratio that is a pattern shows in nature. For the bottleneck reduced random sampling, we are inspired

by the ResNet [32] bottleneck pattern for the convolutional layers. We show that all data samples will be used with an extremely high probability during the whole training procedure for both data reduction strategies, and the training time will be significantly reduced in percentage with a comparable modeling accuracy.

CHAPTER 3: METHODOLOGY

In this chapter, we first introduce our novel methods in architecture complexity reduction. The object is to build a compact model to reduce the network parameter number and FLOPs. We propose a Conditional Reduction (CR) module and then propose the convolutional network internal blocks based on the CR module. Based on the CR module and new convolutional blocks, we propose a Conditional Reduction Network (CRnet). Then we introduce our novel methods in data reduction. The object is to reduce the training time without affecting the network architecture. We first propose a flat reduced random sampling method for the deep learning model training. Then we propose a bottleneck reduced sampling strategy as a three-stage training method based on the bottleneck reduced sampling.

3.1 Architecture Complexity Reduction

In Section 2.12, we have introduced the convolution layer's input-output pattern. In order to utilize the convolution layer's input-output pattern information, we propose a novel input-output pattern aware module called Conditional Reduction (CR) module. The CR module is developed to replace a single convolution layer. Then we apply the CR module to two typical blocks that constitute the CNN: bottleneck blocks and inverted bottleneck blocks to get the Conditional block (C-block). At last, we apply the proposed C-block to the popular ResNet50 [32] stacked with bottlenecks and the state-of-the-art network GhostNet [10] stacked with inverted bottlenecks to get our Conditional Reduction Network (CRnet).

3.1.1 Conditional Reduction Module

The dense 1×1 convolution layers have become a constraint of model complexity since the kernel size 1×1 is the smallest among $N \times N$ kernel sizes. Thus, the key to building a more compact network is to reduce the calculation of the dense 1×1 convolution layers. Current compression strategies ignore the convolution layer's input-output patterns. They use the same compression strategy for a small input-large output 1×1 convolution layer and large input-small output 1×1 convolution layer.

Figure 2.4 in section 2.1.4 shows the state-of-the-art Ghost module [10] that inspired this work. The Ghost module uses the c1dw (1×1 convolution and depthwise convolution) to compress a 1×1 convolution layer. It first uses the regular 1×1 convolution (c1) to get a reduced number of output feature maps and then uses the depthwise convolution (dw) to generate the other feature maps from the c1 results. The nature of Ghost module is that the feature map similarity shows the possibility to generate the total feature maps from its subset. Figure 3.1 shows the similarity examples.

Although the feature map similarity exists in both small input-large output 1×1 convolution layer and large input-small output 1×1 convolution layer, the meanings behind those two patterns are different. For the small input-large output layer, its nature is to discover new features that may benefit the training. For the large input-small output layer, its nature is more like to distilling useful features to represent the previous layers. The c1dw compression may work well for the small input-large output layer since it first uses the normal 1×1 convolution to generate some feature maps. Moreover, the other new features or similar features can be realized by a

computation-saving way. However, for the large input-small output layer, a small number of features are extracted from a large feature database, and each one of the extracted features is important. Reducing the number of such features generated by 1×1 convolution may lose important information summarized from previous layers. For example, as Figure 3.1 shows, if we reduce the output features to only two images, e.g., the first two images of the first row of the feature maps, each image is very important and different from the others. In this case, we do not want to generate only one feature map and use it to generate the other. We take the first convolution layer's results to express our idea. In real experiments, the first convolution layer is usually not compressed.

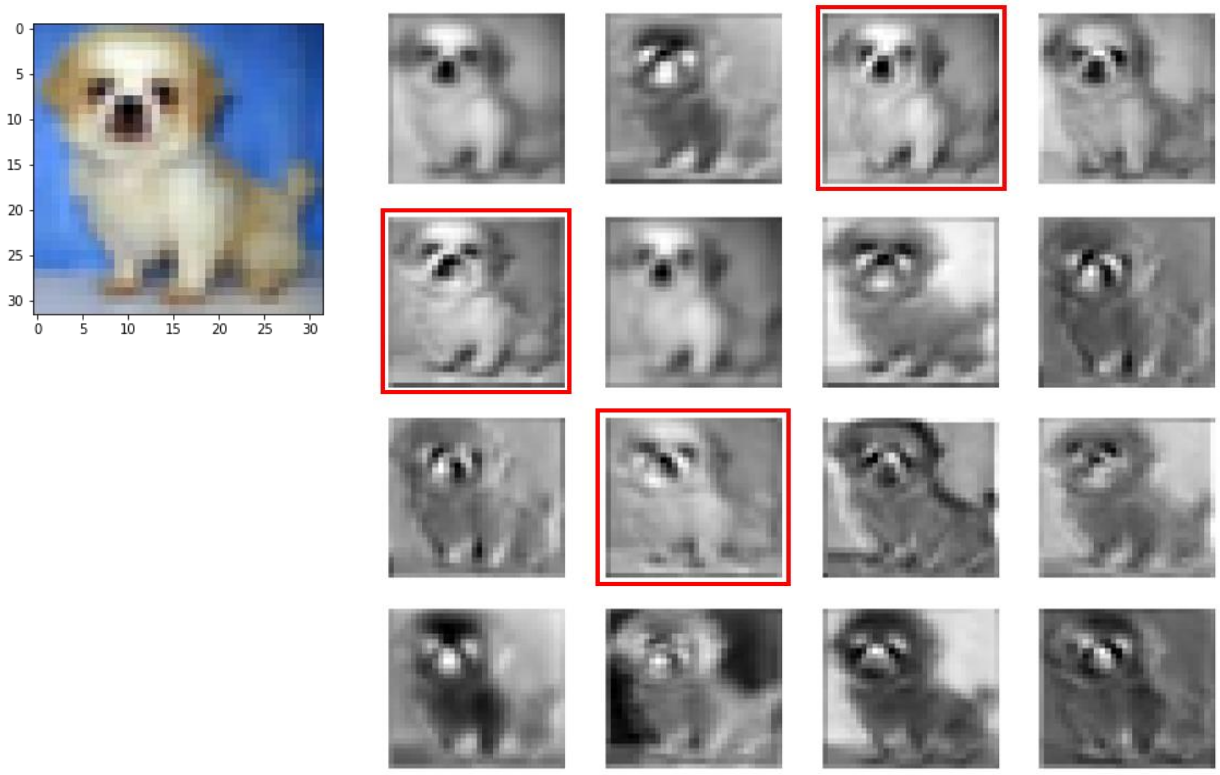

Figure 3.1: Feature maps of the first convolution layer of ResNet56 [32]. The dog image source: CIFAR-10. Red box: similar feature examples.

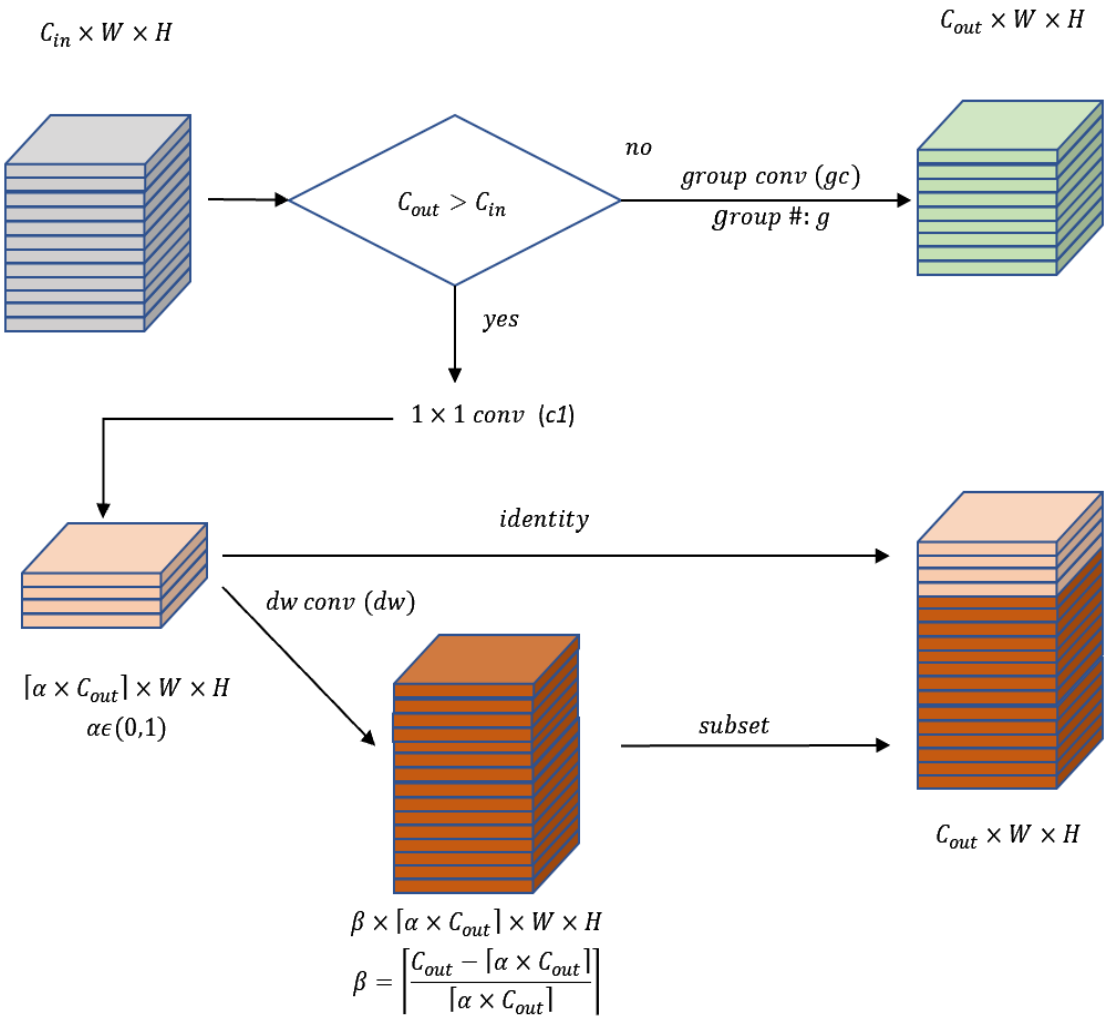

Figure 3.2: Conditional Reduction module to compress the 1×1 convolution. dw conv represents depthwise convolution. group conv represents group convolution.

Based on the analysis above, we propose a Conditional Reduction (CR) module. Figure 3.2 shows the structure of the CR module. Before compressing a 1×1 convolution layer, we check the input-output pattern and choose different compression strategies. When it is the small input-large output mode, we use the c1dw compression strategy as Ghost module [10]. Otherwise, we

use the group convolution. The proposed CR module is aware of the input-output pattern and takes advantage of the c1dw compression and group convolution (gc) compression.

Next, we introduce how we use the CR module to reduce the model complexity, including parameter number and FLOPs [30]. Assume an 1×1 convolution layer's input is $C_{in} \times W \times H$, and output is $C_{out} \times W \times H$. If we use gc compression, the model parameter number after compression over the original parameter number and the FLOPs after compression over the original FLOPs will be as formula (1) and (2), respectively.

$$R_p{}^{gc}(g) = \frac{\frac{C_{in}}{g} \cdot C_{out}}{C_{in} C_{out}} = \frac{1}{g} \tag{1}$$

$$R_{FLOPs}{}^{gc}(g) = \frac{\frac{C_{in}}{g} \cdot C_{out} \cdot W \cdot H}{C_{in} \cdot C_{out} \cdot W \cdot H} = \frac{1}{g} \tag{2}$$

where gc stands for group convolution, g is the group number. If we use c1dw compression, the model parameter number after compression over the original parameter number and the FLOPs after compression over the original FLOPs will be as formula (3) and (4), respectively.

$$R_p{}^{c1dw}(\alpha, \beta) = \frac{C_{in} \cdot \alpha \cdot C_{out} + d \cdot d \cdot \beta \cdot \alpha \cdot C_{out}}{C_{in} \cdot C_{out}} = \alpha + \frac{d \cdot d \cdot \beta \cdot \alpha}{C_{in}} \tag{3}$$

$$R_{FLOPs}{}^{c1dw}(\alpha, \beta) = \frac{C_{in} \cdot \alpha \cdot C_{out} \cdot W \cdot H + d \cdot d \cdot \beta \cdot \alpha \cdot C_{out} \cdot W \cdot H}{C_{in} \cdot C_{out} \cdot W \cdot H} = \alpha + \frac{d \cdot d \cdot \beta \cdot \alpha}{C_{in}} \tag{4}$$

where d is from the kernel size $d \times d$ of the depthwise convolution, α is the ratio of the reduced 1×1 convolution outputs over the total outputs and $\alpha \in (0, 1)$, β is the number of features generated using depthwise convolution over the reduced 1×1 convolution outputs. The number of the reduced 1×1 convolution output feature is $\lceil \alpha \times C_{out} \rceil$. The remaining features are generated using the depthwise convolution with $d \times d$ kernels. As the depthwise convolution can only generate an integer times of input number, the β is calculated as formula (5).

$$\beta = \left\lceil \frac{C_{out} - \lceil \alpha \times C_{out} \rceil}{\lceil \alpha \times C_{out} \rceil} \right\rceil,\ \beta \geq 1 \quad (5)$$

When we set $g = \left\lceil \frac{1}{\alpha} \right\rceil$, the gc compression's parameter number and FLOPs will always be smaller than the c1dw compression. Thus, after we replace the c1dw by gc for the large input-small output pattern, the model complexity is reduced.

3.1.2 Conditional Block

Since He [32] proposed the residual network, the structure of stacking residual convolution blocks has become a mainstream backbone of many state-of-art networks. Networks with fewer layers generally perform worse than deep networks for complex computer vision tasks. Thus, to train a compact work without damaging the network capacity and depth is very important. This section will focus on compression methods without damaging the network depth and node (channel) number as a key approach for the compressed networks. In other words, the compressed network will have the same depth and the same number of channels for each layer but reduced model complexity. Thus, model complexity usually is compared through parameter number and FLOPs (floating-point operations) in this approach.

Section 3.1.1 introduces the CR module, which is suitable to compress a single 1×1 convolution layer. CNN is stacked with bottleneck or inverted bottleneck blocks, which consist of two or three convolution layers. We apply the proposed CR module to the CNN block element and get the Conditional block (C-block). Figure 3.3 shows the structure of the C-block. It is a three-layer block with two compressed 1×1 convolution layers and an intermediate layer where the pooling operation is operated when needed. The intermediate layer is a depthwise 3×3 convolution.

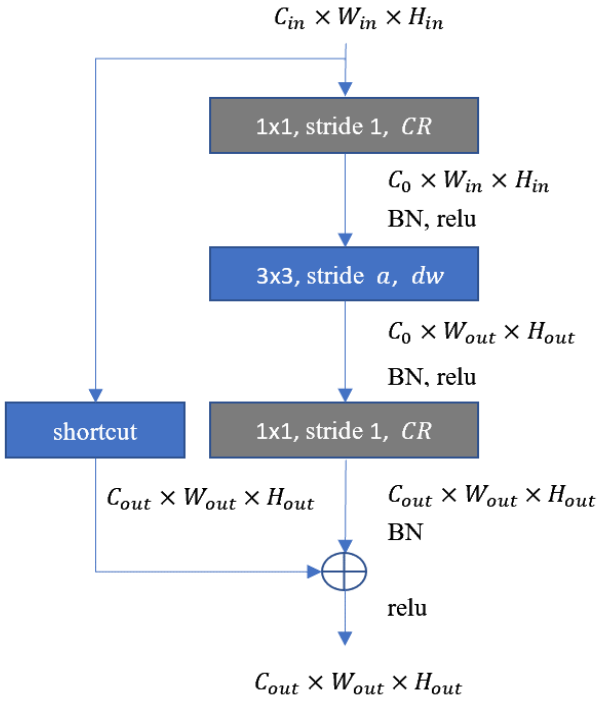

Figure 3.3: Conditional block (C-block).

Since the CR module can tackle either the small input-large output or the large input-small output 1×1 convolution pattern, the C-block is a general format utilizing the CR module in CNN bottleneck blocks and inverted bottleneck blocks. When the block type is a bottleneck, the C-block becomes a Conditional Bottleneck (C-Bneck) as Fig. 3.4. The first layer is a large input-small output 1×1 convolution layer. The second layer is a 3×3 depthwise convolution where the pooling is applied when needed. The third layer is a small input-large output 1×1 convolution layer. When the block type is an inverted bottleneck, the C-block becomes a Conditional Inverted Bottleneck (C-IBneck) as Fig. 3.5. The first layer is a small input-large output 1×1 convolution layer. The second layer is a 3×3 depthwise convolution where the pooling is applied when needed. The third layer is a large input-small output 1×1 convolution layer. For the C-IBneck, the second layer will be skipped as GhostNet [10].

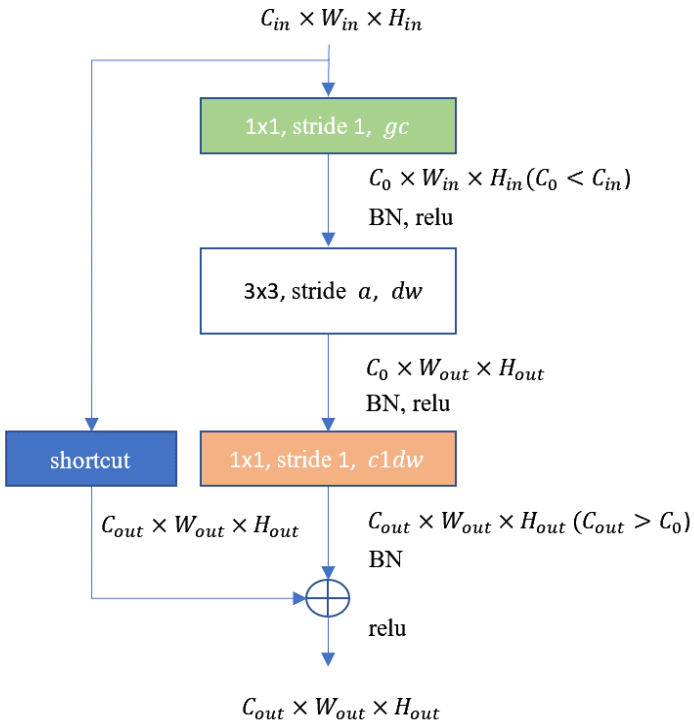

Figure 3.4: Conditional Bottleneck block (C-Bneck).

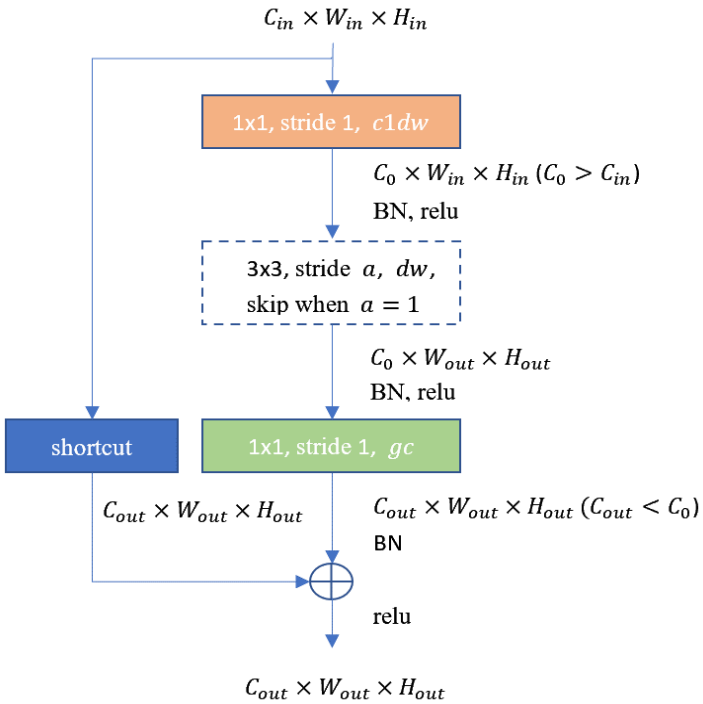

Figure 3.5: Conditional Inverted Bottleneck block (C-IBneck).

3.1.3 Conditional Reduction Network

Section 3.1.2 introduces the C-Bneck and C-IBneck blocks. We replace the conventional bottleneck and inverted bottleneck blocks by the proposed blocks and get the compact Conditional Reduction Network (CRnet). In this dissertation, we take the ResNet50 [32] built based on bottlenecks and the GhostNet [10] built based on inverted bottlenecks as the backbone CNN architectures.

ResNet50 is a baseline residual network architecture with bottlenecks. It is proposed to compress the conventional two-layer 3×3 convolution block. GhostNet is a state-of-the-art compact network with inverted bottlenecks. It is proposed to further compress the dense 1×1 convolutions. The reason that the state-of-the-art compact CNN turns to inverted bottlenecks is that the inverted bottleneck is memory efficient since we can dispose of the inner convolution tensors after computation [9].

To verify the effectiveness of our C-Bneck and C-IBneck, we apply the C-Bneck to replace the bottlenecks of ResNet50 to get the CRnet-CB, and we apply the C-IBneck to replace the inverted bottlenecks of GhostNet to get the CRnet-CIB. We have modified the ResNet50 node number to fit the datasets CIFAR-10 and CIFAR-100.

Table 3.1 shows the architecture of CRnet-CB with the ResNet50 as the backbone for CIFAR-10 and CIFAR-100. Table 3.2 shows the architecture of CRnet-CIB with the GhostNet as the backbone for CIFAR-10 and CIFAR-100.

Table 3-1: CRnet-CB for CIFAR-10 and CIFAR-100 (backbone: ResNet50)

Layer	Input	Operator	Inner	Exp	Out	Str	#block
conv1	$3 \times 32 \times 32$	conv3×3	-	-	16	1	-
conv2-x	$16 \times 32 \times 32$	C-Bneck*	16	4	64	1	3
conv3-x	$64 \times 32 \times 32$	C-Bneck	32	4	128	2	4
conv4-x	$128 \times 16 \times 16$	C-Bneck	32	4	128	1	6
conv5-x	$128 \times 16 \times 16$	C-Bneck	64	4	256	2	3
	$256 \times 8 \times 8$	avg_pool 8×8	-	-	256	-	-
	$256 \times 1 \times 1$	fc	-	-	class	-	-

Input = Input image size, Inner = Inner channel number, Exp = Expansion, Out = Output channel number = $\text{inner} \times$ exp, Str = Stride, #block = number of blocks. The input, inner and output channel numbers are the C_{in}, C_0 and C_{out} in Fig. 3.4, respectively. C-Bneck = Conditional Bottleneck, class = label classes. The stride 2 is only used in conv 3-1 and conv 5-1. The inner channel number is fixed. The output channel number varies along with the expansion ratio.

*The conv 2-1's first 1×1 layer's input channel number and the output channel number are the same. We also use gc for this layer.

Table 3-2: CRnet-CIB for CIFAR-10 and CIFAR-100 (backbone: GhostNet)

Input	Operator	Inner	Exp	Out	Str	SE	DW
3 × 32 × 32	conv 3 × 3	-	-	16	1	-	-
16 × 32 × 32	C-IBneck	64	4	16	1	0	3
16 × 32 × 32	C-IBneck	64	4	16	1	0	3
16 × 32 × 32	C-IBneck	64	4	16	1	0	3
16 × 32 × 32	C-IBneck	64	4	16	1	0.25	5
16 × 32 × 32	C-IBneck	64	4	16	1	0.25	5
16 × 32 × 32	C-IBneck	128	4	32	2	0	3
32 × 16 × 16	C-IBneck	128	4	32	1	0	3
32 × 16 × 16	C-IBneck	128	4	32	1	0	3
32 × 16 × 16	C-IBneck	128	4	32	1	0	3
32 × 16 × 16	C-IBneck	128	4	32	1	0.25	3
32 × 16 × 16	C-IBneck	128	4	32	1	0.25	3
32 × 16 × 16	C-IBneck	256	4	64	2	0.25	5
64 × 8 × 8	C-IBneck	256	4	64	1	0	3
64 × 8 × 8	C-IBneck	256	4	64	1	0.25	3
64 × 8 × 8	C-IBneck	256	4	64	1	0	3
64 × 8 × 8	C-IBneck	256	4	64	1	0.25	3
64 × 8 × 8	conv 1 × 1	-	-	256	1	-	-
256 × 8 × 8	avg_pool 8 × 8	-	-	256	-	-	-
256 × 1 × 1	conv2d 1 × 1	-	-	64	1	-	-
64 × 1 × 1	fc	-	-	class	-	-	-

Input = Input image size, Inner = Inner channel number = out × exp, Exp = Expansion, Out = Output channel number, Str = Stride, SE = Squeeze and excite ratio [57], DW = depthwise kernel size. The input, inner and output channel numbers are the C_{in}, C_0 and C_{out} as shown in Fig. 3.5, respectively. C-IBneck = Conditional Inverted Bottleneck, class = label classes. The output channel number is fixed. The inner channel number varies along with the expansion ratio.

3.2 Data Reduction

Section 2.2 introduces four batch-epoch training types based on the batch number and epoch number: full batches-full epochs, full batches-fewer epochs, fewer batches-full epochs, and fewer batches-fewer epochs. In this dissertation, we focus on the fewer batches-full epochs mode since it is beneficial to not interfere with the normal learning rate decay and can be easily applied to any CNN training. We propose two compression strategies: the flat reduced random sampling and the bottleneck reduced random sampling. Then we propose a three-stage training method to control the bottleneck reduced sampling epochs [7].

3.2.1 Flat Reduced Random Sampling

For the flat reduced random sampling, the word flat means that we use the same fixed sampling ratio for each epoch. A question naturally arises: what sampling ratio should we choose? Since the sampling ratio belongs to (0%,100%], it is not possible to try every sampling ratio. Inspired by the popular 80-20 split ratio that is usually used to split training and validation datasets and the 0.618 golden ratio that is a pattern shows in nature, we take two flat ratios: 0.8 and 0.618. The flat random sampling ratio of 0.8 or 0.618 means that we randomly sample 80% or 61.8% data to train for each epoch.

We maintain the same batch size (the number of images feeds into the network forward pass once a time) but reduce the number of batches for each epoch based on the flat reduction ratio. The randomness is assured by shuffling the dataset before running the next epoch. In this way, the proposed method can be easily applied to current deep learning frameworks.

The flat reduced sampling method is shown in Fig. 3.6.

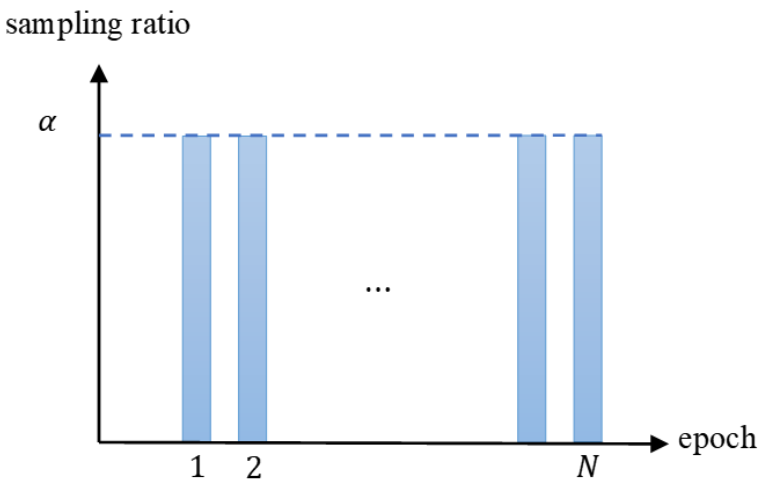

Figure 3.6: Flat reduced sampling. α is the flat reduced random sampling ratio.

There are similar images in the training dataset. We do not expect the model to learn similar images again and again in one epoch. The nature of the flat reduced training is to use a randomly selected subset with a flat ratio. With the same batch size, the training time in an epoch is positively correlated with the number of batches in an epoch. Thus, the total training time is positively correlated with the total number of batches in the CNN training process. But for accuracy, this is not the exact case. The verification/test accuracy may be higher, lower, or comparable with different sampling ratios, i.e., with a variation, but it is limited as shown late in the proposed methods. The deep CNN models need enough training samples and epochs to converge. If we set the flat ratio too low, e.g., 30%, the model will not train well, and the training accuracy will be low, not to mention the verification accuracy. If we set the flat ratio too high, e.g., 95%, the training time will not be reduced much comparing to 100% training.

An optimal flat ratio should balance the training time and accuracy well. At the same time, we do not want to miss any image samples during the whole training process. Thus, we calculate

the probability that an image is invisible to all epochs for our proposed flat random sampling ratio 0.8 and 0.618 as Theorem 1.

Theorem 1. If the deep learning network is trained by *N* epochs, and the flat reduced random sampling ratio is α as 0.8 or 0.618, then the probability that a sample has been missed by the whole training process is as formula (6).

$$p = (1-\alpha)^N \tag{6}$$

In the experiments as commonly used, $N = 120$. By Theorem 1, we have the probability value p for $\alpha = 0.8$ or 0.618 respectively as formula (7a) or (7b).

$$p = (1-0.8)^{120} = 1.3292 \cdot 10^{-84} \tag{7a}$$

$$p = (1-0.618)^{120} = 7.0405 \cdot 10^{-51} \tag{7b}$$

It shows that each sample in the training dataset will be used for the model training almost certainly with a missing probability less than 10^{-51}. This method can also be viewed as a special and simple case of the second proposed bottleneck sampling method.

The relationship between the flat reduced random sampling training time and the total training time is calculated as Theorem 2. Based on Theorem 2, we derive the Corollary 1.

Theorem 2. Consider a fast training method with a flat reduced random sampling ratio α. Its average training time T_α and the average training time T of a regular method satisfy formula (8)-(9):

$$\frac{T_\alpha}{T} = \frac{T_{0\alpha} \cdot M_\alpha \cdot N}{(T_0 \cdot (M-1) + T_1) \cdot N} = \frac{M_\alpha T_{0\alpha}/T_0}{(M-1) + T_1/T_0} \tag{8}$$

$$\frac{M_\alpha T_{0\alpha}/T_0}{M} \leq \frac{T_\alpha}{T} < \frac{M_\alpha\, T_{0\alpha}/T_0}{M-1} \tag{9}$$

where N is the total training epochs number, $M_\alpha = \lfloor \alpha M \rfloor$ and $M = \lceil D/B \rceil$ are the batch numbers of each epoch in the flat α reduction method and regular method respectively, D is the training data set sample size, and B is the batch sample size, T_0 and $T_{0\alpha}$ denote the average training times of a full sample batch with the regular method and α-program method, respectively.

Corollary 1. The fast training method with a flat reduced sampling ratio α has an average training time reduction ratio

$$\frac{T_\alpha}{T} \approx \alpha \tag{10}$$

if $\alpha M \gg 1$ and $T_{0\alpha}/T_0 \approx 1$.

From Theorem 2 and Corollary 1, if we only consider the training time reduction, it is obvious that the flat fast training method with the flat reduced random sampling rate $\alpha = 0.618$ saves more time than $\alpha = 0.8$.

3.2.2 Bottleneck Reduced Random Sampling

We note that the variation between neighboring two epochs is quite small, except for the epoch where the learning rate drops to a small percentage of the previous value. For example, ResNet [32] initial learning rate 0.1 multiples by 0.1 after training several epochs and becomes 0.01. A question arises: is it essential to use a large sampling rate for each epoch? Motivated by this question, we propose the bottleneck reduced random sampling strategy.

The proposed bottleneck random sampling is inspired by the image convolutional bottleneck block design of ResNet [32]. The difference is, the ResNet bottleneck block is for the image convolution. The number of feature maps input to the block is large, becoming small at the

intermediate layers and then get large again at the output layer. In this dissertation, the bottleneck design is for the sampling rate. We take several epochs as a sampling block. Within each block, the start and the end epoch use a larger sampling ratio, and the middle epochs use a smaller ratio. The ratio pattern of the sampling ratio block is large-small-large. Thus, we named the sampling ratio block as a sampling bottleneck, and the method using the sampling bottleneck as the bottleneck reduced random sampling.

The bottleneck sampling strategy is shown in Fig. 3.7. A bottleneck block is composed of k+2 epochs as shown in Fig. 3.7, and each epoch (each bar) has its sampling ratio, where different bar height size corresponds to a different number of samples. The internal epochs correspond to a squeezed batch number in the bottleneck stage.

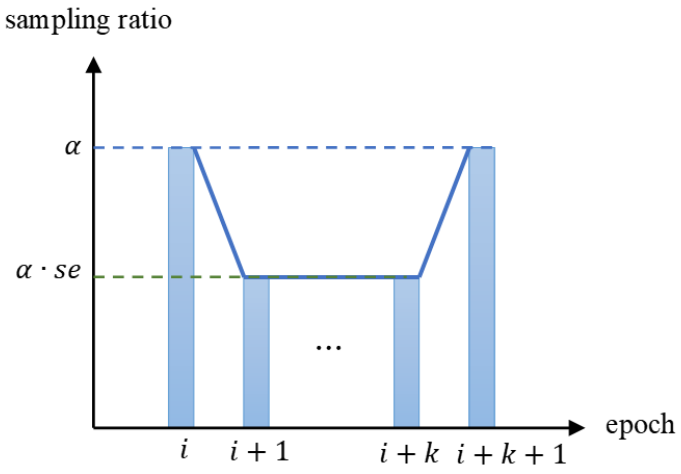

Figure 3.7: Bottleneck sampling block. α is the sampling ratio for the input and output epoch of a bottleneck. se is the squeeze ratio for the internal epochs. k is the number of internal bottleneck epochs. In our experiments, $k = 4$.

Figure 3.8 shows the stacking method of the bottleneck sampling block. There are two blocks in Fig. 3.8. The second block uses the first block's end epoch as its start epoch. For more bottleneck blocks, they also stack using the same way.

The bottleneck sampling has the following characteristics.

(1) The samples are selected randomly as the original methods and the flat random sampling method. The randomness is fulfilled by shuffling the dataset at the beginning of each epoch.

(2) For all data samples, they have the same probabilities to be selected as uniformly random selection. Thus, there are no extra calculations of sample importance. Therefore, it is also easily implemented.

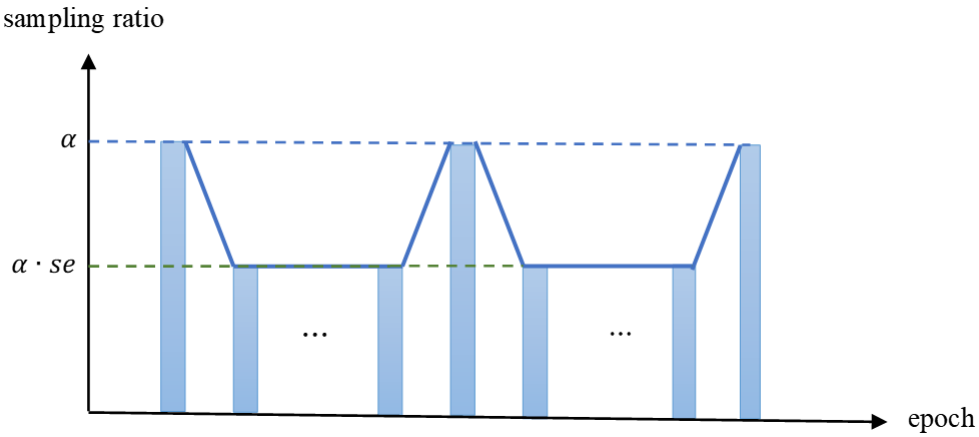

Figure 3.8: Stacking of bottleneck sampling blocks.

3.2.3 Three-Stage Training

Section 3.2.2 introduces the new bottleneck sampling strategy. We develop a three-stage training method based on the bottleneck sampling due to the following two reasons.

(1) The network is randomly initialized and knows nothing about the data at the very beginning. Enough training data is essential at the early stage of the training to get a model with some knowledge of the data. Thus, if we would reduce the sampling rate to a smaller value, we should give the model a high sampling rate at the early stage.

(2) When the model trains several epochs, the model starts to converge, and the accuracy increase is quite small. In this case, a high sampling ratio is not essential. We may use a small sampling ratio.

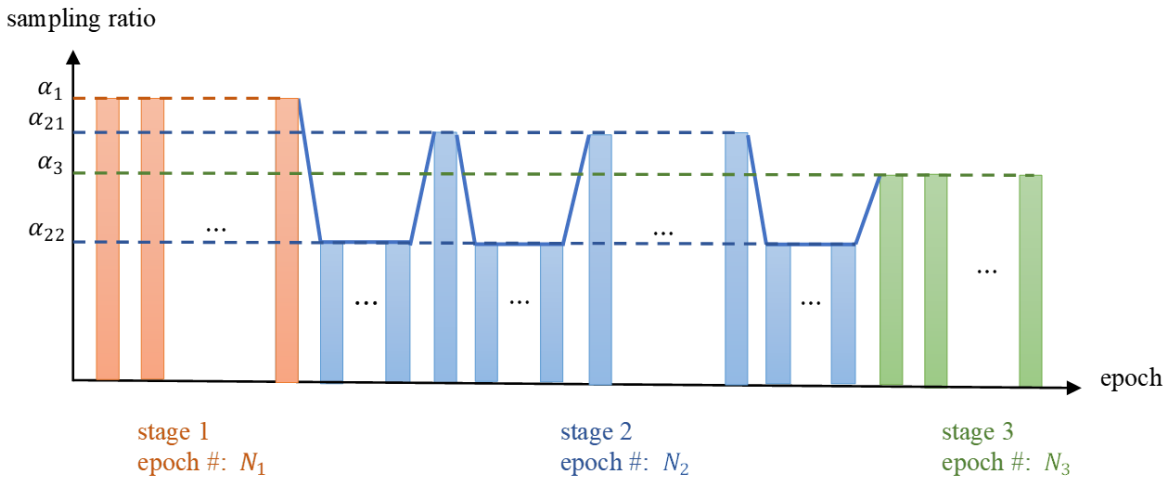

Figure 3.9: Three-stage training method based on bottleneck sampling method. The bottleneck sampling is used in stage 2. α_i is the sampling ratio for stage i. N_i is the number of epochs for stage i.

We propose a new three-stage training method, as shown in Fig. 3.9. The first stage contains the early epochs of the training procedure. The middle stage contains the middle epochs

after the first stage and before the third stage. The third stage contains the last remaining epochs in the whole training process. We applied the bottleneck sampling in the middle stage. The first block and the last block of the middle stage are special as they connect with the first stage and the third stage. The first bottleneck of stage 2 is not a full bottleneck. It uses the last epoch of stage 1 to replace the start epoch. The last epoch of stage 2 may be full or not full depending on the $N_1 + N_3$ because $N_2 = N - (N_1 + N_3)$. Here the internal squeezed epoch number is fixed as 4. If N_2 is exactly divisible by 5 (internal 4 squeezed ratios α_{22} and 1 normal ratio α_{21}), the last bottleneck is full, and the last epoch of it will be α_{21}. Otherwise, the last bottleneck is not full, and the last epoch of it will be α_{22}. The meanings of the above math symbols are defined in Theorem 3 below.

Next, we analyze the probability of a sample get missed by the whole training process and give the training time calculation.

Theorem 3. The probability p of a sample missing in the bottleneck random sampling method is shown in formula (11).

$$p = (1 - \alpha_1)^{N_1}(1 - \alpha_{21})^{N_{21}}(1 - \alpha_{22})^{N_{22}}\ (1 - \alpha_3)^{N_3} \tag{11}$$

where each epoch has its independent random sampling from the whole training dataset, α_1 and α_3 are two flat sampling rates for each epoch in stages 1 and 3 respectively; $\alpha_{21}$and α_{22} are respective sampling rates for different bottleneck epochs in stage 2, se is a squeeze ratio as $se = \alpha_{22}/\alpha_{21}$; N_1 and N_3 are the total epoch numbers in stages 1 and 3 with their random sampling rates α_1 and α_3, respectively; N_{21} and N_{22} are the total epoch numbers with a sampling rate α_{21} and its squeezed rate α_{22}, respectively; and $N_2 = N_{21} + N_{22}$ represents the total epoch numbers in

stage 2.

Again, the probability of any sample missing in the whole three-stage bottleneck training process is extremely low for deep learning where the total training epoch number $N = N_1 + N_{21} + N_{22} + N_3$ is high as shown in (11).

Let T_B be the average training time of the three-stage bottleneck reduced random sampling strategy. Similarly, we have the following Theorem 4.

Theorem 4. Consider a three-stage fast training method with a bottleneck random sampling ratio set of $\{\alpha_1, \alpha_{21}, \alpha_{22}, \alpha_3\}$ and its respective epoch number set of $\{N_1, N_{21}, N_{22}, N_3\}$. The training data set has the total sample number *D*, and the batch sample size is *B*. In stages {1, 21, 22, 3}, where stages 21 and 22 are in stage 2, each epoch may have different batch numbers $\{M_1, M_{21}, M_{22}, M_3\} = \{\lfloor(\alpha_1, \alpha_{21}, \alpha_{22}, \alpha_3) \cdot M\rfloor\}$ with $M = \lceil D/B \rceil$, respectively. The average bottleneck training time T_B and the average training time *T* of a regular method satisfy formula (12):

$$\frac{T_B}{T} = \frac{(T_{0\alpha 1} M_1 N_1 + T_{0\alpha 21} M_{21} N_{21} + T_{0\alpha 22} M_{22} N_{22} + T_{0\alpha 3} M_3 N_3)}{[T_0(M-1) + T_1]N} \tag{12}$$

where $N = (N_1 + N_{21} + N_{22} + N_3)$ is the total training epochs number, T_0 and $T_{0\alpha i}$ denote the average training times of a full sample batch with *B* samples under a regular method and an α_i-reduced method respectively, and T_1 is the average training time of the last batch in each epoch of the regular method, which may not have full *B* samples.

Corollary 2. If each member of $\{\alpha_1 M, \alpha_{21} M, \alpha_{22} M, \ \alpha_3 M\}$ is much larger than 1, and $T_{0\alpha i} \approx T_0$, $i = 1, 21, 22, 3$, then the bottleneck fast training method has a simplified training time reduction

as in (13)

$$\frac{T_B}{T} \approx \frac{M_1N_1+M_{21}N_{21}+M_{22}N_{22}+M_3N_3}{MN} = \frac{\alpha_1N_1+\alpha_{21}N_{21}+\alpha_{22}N_{22}+\alpha_3N_3}{N} \tag{13}$$

where the parameters are as defined in Theorem 4, respectively.

From Theorems 2 and 4, and Corollaries 1–2, it is observed that the flat fast training method with $\alpha = 0.618$ is better than the bottleneck fast training method in view of training time reduction and the process complexity.

CHAPTER 4: EXPERIMENT

4.1 Architecture Complexity Reduction

We propose a novel CR module to compress the 1×1 convolution. Based on the CR module, we have developed a C-Bneck and a C-IBneck to compress the CNN bottleneck block and inverted bottleneck block, respectively. Then we use the C-Bneck to replace the bottlenecks in a backbone network to get the CRnet-CB. We use the C-IBneck to replace the inverted bottlenecks in a backbone network to get the CRnet-CIB. In this dissertation, we take the state-of-the-art network ResNet50 stacked with bottlenecks as our backbone architecture to test the C-Bneck block. We take the state-of-the-art network GhostNet stacked with inverted bottlenecks as our backbone to test the C-IBneck block.

4.1.1 Dataset

We use two benchmark datasets: CIFAR-10 [59] and CIFAR-100 [59], to conduct the experiments. The CIFAR-10 contains 50k $3 \times 32 \times 32$ training images and 10k $3 \times 32 \times 32$ testing images for 10 classes. The 10 classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The CIFAR-100 contains 50k $3 \times 32 \times 32$ training images and 10k $3 \times 32 \times 32$ testing images for 100 classes. The 100 classes include beaver, dolphin, ray, shark, roses, sunflowers, bottles, cups, pears, clock, etc.

4.1.2 Evaluation Metrics

Model complexity is an important factor in measuring model performance. A compact model with a little sacrifice of accuracy is acceptable compared to a large model since compact

models are faster and are easier to be deployed to mobile devices, as well as to be applied, especially for real-time identification systems and control systems.

There are two basic metrics to measure the model complexity: parameter number and FLOPs. The parameter number is the number of trainable elements in the CNN network. For example, a $N \times N$ convolution kernel contains $N \times N$ parameters without considering bias. The key of deep learning is that it can be trained through backpropagation, which makes it obviously different from conventional machine learning methods. After a batch of images sent into the feedforward network to get the prediction, we will get the predicted outputs for this batch which can be used to calculate the loss. The loss will be propagated back to the network to update the trainable parameters. Thus, the number of parameters is a key metric to reflect the model capacity and complexity. It describes the RAM needs of the model to store its architectures and parameters. FLOPs (floating-point operations) is another important metric to measure model complexity. It is often defined as the number of floating-point multiplication-adds [31]. It indicates the computation amount of one forward pass for a single image. As the whole image contributes to the multiplication-adds operations, FLOPs number is related to image size. With the same convolution kernel, the larger the image is, the more the FLOPs are. It describes the hardware needs of the model.

Besides model complexity, accuracy is a conventional measurement of model performance. No or a little sacrifice is acceptable when building compact models. A compact model containing very few parameters and requiring small FLOPs but performing poorly in accuracy is also a failed and unacceptable model. Here the accuracy is also mentioned as Top 1 accuracy. The top 1

accuracy is the accuracy of general understanding. For each sample, the one with the highest probability is the predicted type. If it matches the sample label, the prediction matches the label. For the whole dataset, top 1 accuracy is the number of top 1 matched predictions over the total number of samples.

4.1.3 CRnet-CB Results

We first test the performance of the CR module compression strategy in the CNN stacked with bottlenecks. We use ResNet50 [32] as the backbone network and test different compression strategies. We use C-Bneck to compress the bottlenecks and get the network CRnet-CB.

For each bottleneck, there are two 1×1 convolution layers. We use the proposed CR module to compress each 1×1 convolution layer and get a CRnet-CB with ResNet50 as backbone, which is shown in Table 3-1. We conduct the CRnet-CB on CIFAR-10 and CIFAR-100. The compact model has been run for 160 epochs. We use SGD optimization with a momentum 0.9 and a decay 0.0001. The learning rate starts at 0.1, divided by 10 at epoch 80 and 120. We have run two batch sizes of 256 and 64 on two Nvidia Quadro RTX 5000 GPUs in the experiments**.** We use PyTorch as our deep learning API.

The experimental results of the CRnet-CB on the CIFAR-10 testing set are shown in Table 4-1 and Table 4-2. The experimental results of the CRnet-CB on the CIFAR-10 testing set are shown in Table 4-3 and Table 4-4. The number of parameters and FLOPs are for the whole network.

Table 4-1: CRnet-CB results on CIFAR-10 with a batch size of 256

Setting	1st -2nd 1 × 1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$exp = 4$ $g = 2$ $\alpha * = 0.5$	*c1dw-c1dw*	0.16	28.17	90.72	Ghost module [10]
	gc-c1dw	**0.16**	**27.36**	**91.46**	**CR module**
	gc-gc	0.15	24.78	89.61	Group Convolution
	c1dw-gc	0.15	25.59	90.35	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.27	44.00	90.90	Ghost module
	gc-c1dw	**0.26**	**43.19**	**91.80**	**CR module**
	gc-gc	0.25	39.31	90.81	Group Convolution
	c1dw-gc	0.25	40.13	90.42	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.12	21.15	89.21	Ghost module
	gc-c1dw	**0.12**	**19.94**	**90.67**	**CR module**
	gc-gc	0.10	16.07	89.42	Group Convolution
	c1dw-gc	0.11	17.28	88.81	Inverse CR

$*\alpha = \frac{the\ number\ of\ 1\times1\ output\ channel\ generated\ by\ c1}{the\ total\ 1\times1\ output\ channels}$.

Compression ratio variation: larger g indicates higher compression ratio for gc, smaller α indicates higher compression ratio for $c1dw$.

Table 4-2: CRnet-CB results on CIFAR-10 with a batch size of 64

Setting	1st -2nd 1 × 1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$exp = 4$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.16	28.17	90.88	Ghost module
	gc-c1dw	0.16	27.36	90.73	CR module
	gc-gc	0.15	24.78	89.22	Group Convolution
	c1dw-gc	0.15	25.59	90.08	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.27	44.00	90.52	Ghost module
	gc-c1dw	**0.26**	**43.19**	**91.11**	**CR module**
	gc-gc	0.25	39.31	88.88	Group Convolution
	c1dw-gc	0.25	40.13	90.49	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.12	21.15	90.17	Ghost module
	gc-c1dw	**0.12**	**19.94**	**90.34**	**CR module**
	gc-gc	0.10	16.07	88.21	Group Convolution
	c1dw-gc	0.11	17.28	89.37	Inverse CR

Table 4-3: CRnet-CB results on CIFAR-100 with a batch size of 256

Setting	1st -2nd 1×1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$exp = 4$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.18	28.20	67.04	Ghost module
	gc-c1dw	**0.18**	**27.39**	**68.40**	**CR module**
	gc-gc	0.17	24.81	66.96	Group Convolution
	c1dw-gc	0.17	25.62	67.11	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.30	44.03	67.61	Ghost module
	gc-c1dw	**0.30**	**43.22**	**68.05**	**CR module**
	gc-gc	0.28	39.35	66.70	Group Convolution
	c1dw-gc	0.29	40.16	67.17	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.15	21.18	65.75	Ghost module
	gc-c1dw	**0.14**	**19.96**	**67.29**	**CR module**
	gc-gc	0.13	16.09	66.31	Group Convolution
	c1dw-gc	0.13	17.31	65.15	Inverse CR

Table 4-4: CRnet-CB results on CIFAR-100 with a batch size of 64

Setting	1st -2nd 1×1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$exp = 4$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.18	28.20	68.36	Ghost module
	gc-c1dw	**0.18**	**27.39**	**68.76**	**CR module**
	gc-gc	0.17	24.81	66.82	group convolution
	c1dw-gc	0.17	25.62	67.61	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.30	44.03	68.85	Ghost module
	gc-c1dw	**0.30**	**43.22**	**69.91**	**CR module**
	gc-gc	0.28	39.35	67.17	group convolution
	c1dw-gc	0.29	40.16	66.95	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.15	21.18	67.51	Ghost module
	gc-c1dw	**0.14**	**19.96**	**68.62**	**CR module**
	gc-gc	0.13	16.09	66.24	group convolution
	c1dw-gc	0.13	17.31	65.57	Inverse CR

The experimental observations of the CRnet-CB on CIFAR-10 and CIFAR-100 are as follows.

(1) The $*$-*c1dw* (*gc-c1dw* and *c1dw-c1dw*) compression for the two 1×1 layers performs better than $*$-*gc* (*gc-gc* and *c1dw-gc*) for most experiments.

(2) Among the $*$-*c1dw* compression strategies, the *gc-c1dw* (the proposed CR module compression for bottlenecks) achieves better accuracy than *c1dw-c1dw* for most experiments.

Furthermore, the network compression using the CR module has fewer parameters and FLOPs than *c1dw-c1dw*.

(3) The *gc-c1dw* (the proposed CR module compression) performs much better than *c1dw-gc* (the inverse version of the CR module compression). For some experiments on the CIFAR-100 with a batch size of 64, the CR module even achieves about 3% accuracy increase than the inverse version of the CR module under the same experiment settings.

(4) The CR module achieves better accuracy than the strategies with fewer parameters and FLOPs. The CR module has fewer parameters and FLOPs comparing to the method with comparable accuracy on average. Thus the CR module better balances the model complexity and accuracy.

The observations above show that:

(1) For the bottlenecks, the performances of the different compression strategies are affected by the input-output patterns.

(2) A compression strategy that does not perform well for one input-output pattern may perform well for another input-output pattern, i.e., it is related to the input-output pattern. Thus, it is essential to treat the compression strategies carefully and select the compression strategies as aware of network architectures as we propose.

(3) The proposed CR module performs well in compressing the bottleneck structures to balance the model complexity and accuracy.

(4) The expansion ratio and compression ratio importance is less important than the input-output pattern.

4.1.4 CRnet-CIB Results

We test the performance of our CR module compression strategy in the CNN stacked with the inverted bottlenecks. We use GhostNet [10] as the backbone network and test different compression strategies.

For each inverted bottleneck, there are two 1×1 convolution layers. We use the proposed CR module to compress each 1×1 convolution layer and get a CRnet-CIB with GhostNet as the backbone, and that structure is shown in Table 3-2. We conduct this experiment on CIFAR-10 and CIFAR-100. The model has been run for 160 epochs. We use SGD optimization with a momentum 0.9 and a decay 0.0001. The learning rate starts at 0.1, divided by 10 at epoch 80 and epoch 120. We have run two batch sizes of 256 and 64 on two Nvidia Quadro RTX 5000 GPUs in our experiments**.** We use PyTorch as our deep learning API.

The experimental results of the CRnet-CIB on CIFAR-10 are shown in Table 4-5 and Table 4-6. The experimental results of the CRnet-CIB on CIFAR-100 are shown in Table 4-7 and Table 4-8. The number of parameters and FLOPs are for the whole network.

Table 4-5: CRnet-CIB results on CIFAR-10 with a batch size of 256

Setting	1st -2nd 1×1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$exp = 4$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.29	24.58	92.66	Ghost module [10]
	c1dw-gc	**0.29**	**23.90**	**93.11**	**CR module**
	gc-gc	0.28	20.51	88.77	Group Convolution
	gc-c1dw	0.28	21.19	91.16	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.52	36.23	92.90	Ghost module
	c1dw-gc	**0.52**	**35.55**	**93.23**	**CR module**
	gc-gc	0.50	30.46	89.46	Group Convolution
	gc-c1dw	0.51	31.14	92.28	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.25	17.71	91.49	Ghost module
	c1dw-gc	**0.24**	**16.69**	**91.88**	**CR module**
	gc-gc	0.23	11.60	87.80	Group Convolution
	gc-c1dw	0.23	12.62	90.37	Inverse CR

Table 4-6: CRnet-CIB results on CIFAR-10 with a batch size of 64

Setting	1^{st} -2^{nd} 1×1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$\exp = 4$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.29	24.58	92.12	Ghost module
	c1dw-gc	**0.29**	**23.90**	**92.60**	**CR module**
	gc-gc	0.28	20.51	89.87	Group Convolution
	gc-c1dw	0.28	21.19	91.99	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.52	36.23	93.20	Ghost module
	c1dw-gc	0.52	35.55	92.94	CR module
	gc-gc	0.50	30.46	89.26	Group Convolution
	gc-c1dw	0.51	31.14	92.02	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.25	17.71	92.32	Ghost module
	c1dw-gc	**0.24**	**16.69**	**92.33**	**CR module**
	gc-gc	0.23	11.60	88.58	Group Convolution
	gc-c1dw	0.23	12.62	91.08	Inverse CR

Table 4-7: CRnet-CIB results on CIFAR-100 with a batch size of 256

Setting	1^{st} -2^{nd} 1×1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$exp = 4$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.30	24.59	70.06	Ghost module
	c1dw-gc	**0.30**	**23.91**	**70.78**	**CR module**
	gc-gc	0.29	20.52	64.78	Group Convolution
	gc-c1dw	0.29	21.20	67.65	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.53	36.23	70.81	Ghost module
	c1dw-gc	0.53	35.55	70.58	CR module
	gc-gc	0.51	30.46	64.96	Group Convolution
	gc-c1dw	0.51	31.15	69.05	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.25	17.71	67.63	Ghost module
	c1dw-gc	**0.25**	**16.69**	**69.26**	**CR module**
	gc-gc	0.23	11.60	62.18	Group Convolution
	gc-c1dw	0.24	12.63	66.22	Inverse CR

Table 4-8: CRnet-CIB results on CIFAR-100 with a batch size of 64

Setting	1st -2nd 1×1 compression	Parameter (M)	FLOPs (M)	Acc. (%)	Compression Strategy
$exp = 4$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.30	24.59	71.76	Ghost module
	c1dw-gc	**0.30**	**23.91**	**72.10**	**CR module**
	gc-gc	0.29	20.52	64.48	Group Convolution
	gc-c1dw	0.29	21.20	69.59	Inverse CR
$exp = 6$ $g = 2$ $\alpha = 0.5$	*c1dw-c1dw*	0.53	36.23	70.79	Ghost module
	c1dw-gc	**0.53**	**35.55**	**72.70**	**CR module**
	gc-gc	0.51	30.46	65.68	Group Convolution
	gc-c1dw	0.51	31.15	70.08	Inverse CR
$exp = 4$ $g = 4$ $\alpha = 0.25$	*c1dw-c1dw*	0.25	17.71	69.01	Ghost module
	c1dw-gc	**0.25**	**16.69**	**70.41**	**CR module**
	gc-gc	0.23	11.60	61.87	Group Convolution
	gc-c1dw	0.24	12.63	68.05	Inverse CR

The experimental observations of CRnet-CIB for CIFAR-10 and CIFAR-100 are as follows.

(1) The *c1dw-*$*$ (*c1dw-gc* and *c1dw-c1dw*) compression for the two 1×1 layers performs better than *gc-*$*$ (*gc-gc* and *c1dw- gc*) for most experiments.

(2) Among the *c1dw-*$*$ compression strategies, the *c1dw-gc* (the proposed CR module compression for the inverted bottlenecks) achieves better accuracy than *c1dw-c1dw* for most experiments. And the network compression using the CR module has fewer parameters and FLOPs

than *c1dw-c1dw*.

(3) The *c1dw-gc* (the proposed CR module) performs much better than *gc-c1dw* (the inverse version of the CR module). For some experiments on the CIFAR-100 with batch sizes of 256 and 64, the CR module achieves about a 3% accuracy increase than the inverse version of the CR module under the same experiment settings.

(4) The CR module achieves better accuracy than the strategies with fewer parameters and FLOPs. The CR module has fewer parameters and FLOPs comparing to the method with comparable accuracy on average. Thus the CR module better balances the model complexity and accuracy.

The observations above show that:

(1) For the inverted bottlenecks, the performances of the different compression strategies are affected by the input-output patterns.

(2) A compression strategy that does not perform well for one input-output pattern may perform well for another input-output pattern, i.e., it is related to the input-output pattern. It is important to develop an architecture-aware compression method.

(3) The proposed CR module performs well in compressing the inverted bottleneck structures to balance the model complexity and accuracy.

(4) The expansion ratio and compression ratio importance is less important than the input-output pattern.

4.2 Data Reduction

We propose two compression strategies: flat reduced random sampling and bottleneck reduced random sampling (B-neck sampling). Based on the bottleneck reduced random sampling, we propose a three-stage training method. The first and third stages use two fixed ratios. The second stage uses the bottleneck reduced random sampling ratio. The flat reduced random sampling is a special case of the three-stage training method, by setting the second and third stage total epoch number to 0.

4.2.1 Dataset

We use three benchmark datasets: CIFAR-10 [59], CIFAR-100 [59] and ImageNet (ILSVRC 2012) [60] to conduct the experiments. CIFAR-10 and CIFAR-100 have been introduced in section 4.1.1. The ImageNet 2012 classification dataset contains 1.28 million training images, 50k validation images, and 100k testing images for 1000 classes. Since the labels of testing images are not available, the training images and validation images are often used to compare the algorithms for convenience. The ImageNet image size is not fixed. The 1000 classes include goldfish, great white shark, great grey owl, snowbird, pillow, pencil sharpener, purse, shopping basket, sliding door, school bus, etc.

We apply the same random seed number for each run on the same dataset. As our aim for this experiment is to study the effect of the proposed fast random sampling methods for model training, we do not apply the variations that can enhance the model performance, like tuning the batch size, learning rate, etc.

4.2.2 Evaluation Metrics

We use Top 1 accuracy and Top 5 accuracy metrics to measure model accuracy. For the whole dataset, top 1 accuracy is the number of top 1 matched predictions over the total number of samples. When calculating the top 5 accuracy, a prediction is considered correct if its predicted top 5 highest probabilities contain the label. For the whole dataset, top 5 accuracy is the number of top 5 matched predictions over the total number of samples. In addition to the Top 1 accuracy and Top 5 accuracy, we also evaluate the model using training time.

4.2.3 CIFAR-10 Experiments

For CIFAR-10, we have trained the network for 120 epochs with a batch size of 128 [32]. We use the stochastic gradient descent optimizer (batch gradient descent) with a momentum of 0.9 and a weight decay of 1e-4 [32]. The learning rate starts at 0.1 and drops to 0.01 at epoch 60 and to 0.001 at epoch 90. The ResNet56 [32] is used as the backbone network. For fair comparison, all experiments are implemented on the same workstation with two Nvidia Quadro RTX 5000 GPUs. We use PyTorch as the deep learning API.

We have tested the developed flat reduced random sampling method and three-stage training method on the CIFAR-10 dataset. Table 4-9 shows the details of the experiments, where the accuracy is on the test set. Table 4-10 lists the relative percentage improvement by comparison with the ResNet56 reimplementation results for training time, top-1 accuracy, and top-5 accuracy. Figure 4.1 shows the relative training time reduction vs. the relative top-1 accuracy change for the CIFAR-10 dataset. Figure 4.2 shows the relative training time reduction vs. relative top-5 accuracy change for the CIFAR-10 dataset. The flat reduced random sampling method result is indicated by

red color, and the three-stage training method result using B-neck sampling is indicated by green color.

The results show that both proposed methods get significant training time percentage reduction with a very small accuracy cost for CIFAR-10. For this dataset, the three-stage training method with bottleneck sampling strategy achieves better results compared to the flat reduced sampling method with comparable accuracy but less time.

Table 4-9: Training performances on CIFAR-10

Method	Setting	Top1 %	Top5 %	Time min/sec
ResNet56 [32]	-	93.03	-	-
ResNet56 Re-impl. *	-	92.53	99.78	42/47
ResNet56 R1_0.8	$\alpha = 0.8$	92.20	99.77	33/47
ResNet56 R1_0.618	$\alpha = 0.618$	91.52	99.78	26/30
ResNet56 R2	$\alpha = 0.8 \cdot [1,1,1]$ $\gamma = [1/3\,, 1/3\,, 1/3]$ $se = 0.8$	92.41	99.76	32/26
ResNet56 R2_1	$\alpha = [0.8,1.0,0.8]$ $\gamma = [1/3\,, 1/3\,, 1/3]$ $se = 0.8$	91.89	99.73	34/38
ResNet56 R3	$\alpha = [0.8,0.8,0.25]$ $\gamma = [1/3\,, 1/2\,, 1/6]$ $se = 0.8$	92.34	99.75	29/09

*Re-impl.: Re-implementation.

Table 4-10: Training performances analysis on CIFAR-10

Method	Top1 %	Top5 %	Time min/sec	Top1↑ %	Top5↑ %	Time↑ %	Flat/ B-neck
ResNet56 Re-impl.	92.53	99.78	42/47	-	-	-	-
ResNet56 R1_0.8	92.20	99.77	33/47	-0.36	-0.01	-21.04	Flat
ResNet56 R1_0.618	91.52	99.78	26/30	-1.09	0	-38.06	Flat
ResNet56 R2	92.41	99.76	32/26	-0.13	-0.02	-24.19	B-neck
ResNet56 R2_1	91.89	99.73	34/38	-0.69	-0.05	-19.05	B-neck
ResNet56 R3	92.34	99.75	29/09	-0.21	-0.03	-31.87	B-neck

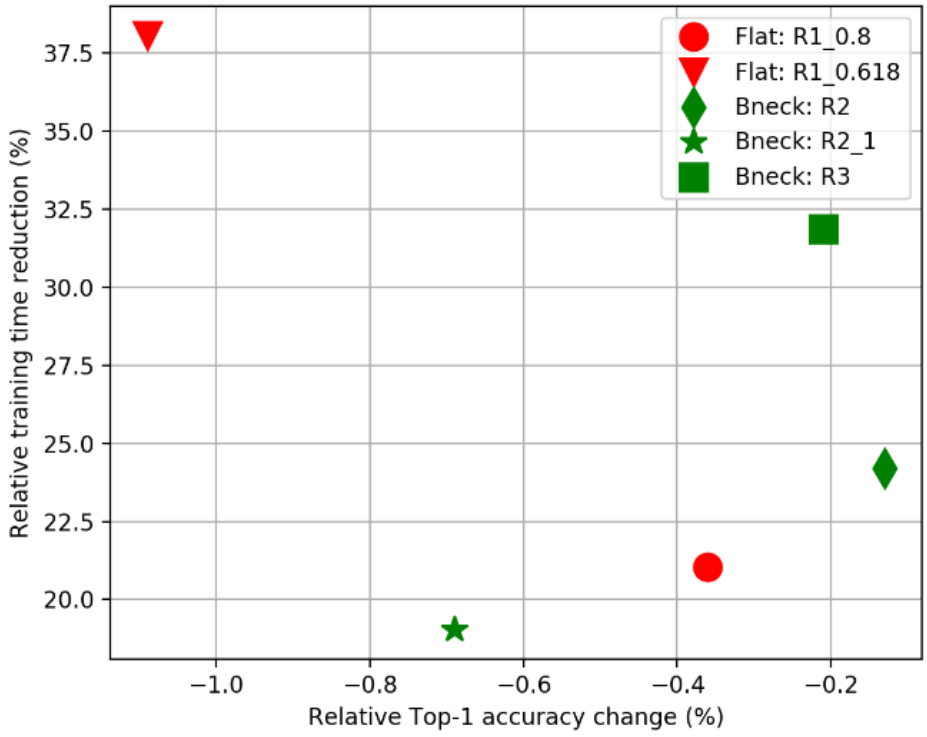

Figure 4.1: Relative training time reduction vs relative top-1 accuracy change for CIFAR-10 dataset.

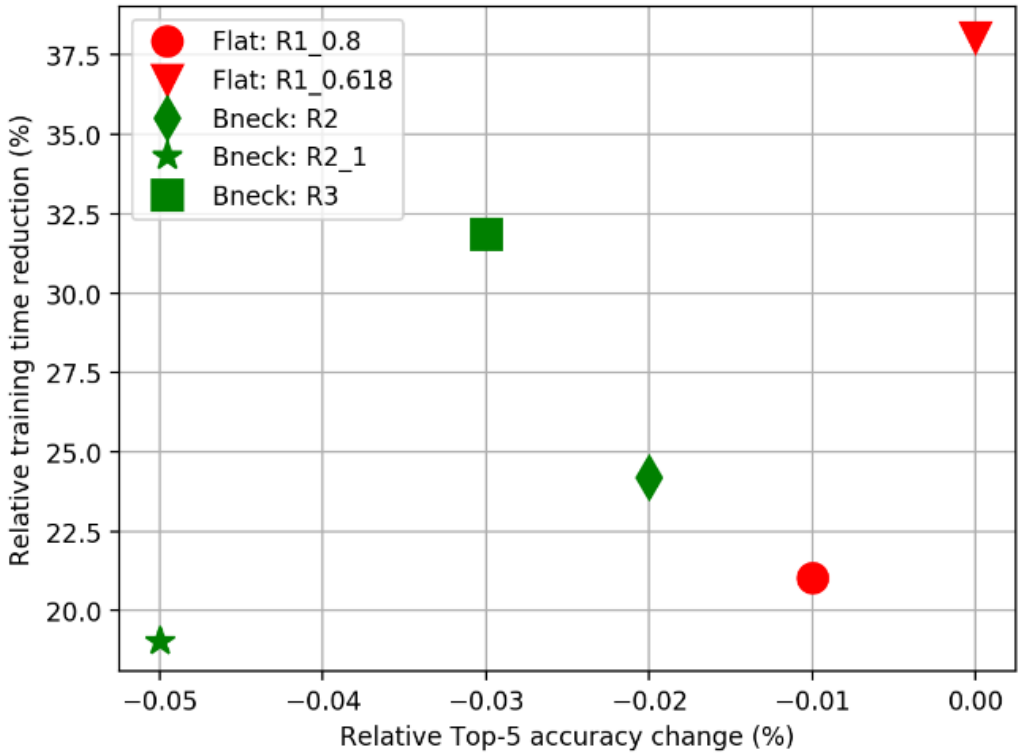

Figure 4.2: Relative training time reduction vs relative top-5 accuracy change for CIFAR-10 dataset.

4.2.4 CIFAR-100 Experiments

The experiment settings for CIFAR-100 are the same as CIFAR-10.

We have tested the proposed methods on the CIFAR-100 dataset. Table 4-11 shows the details of the experiments. Table 4-12 lists the relative percentage improvement by comparison with the ResNet56 reimplementation results for training time, top-1 accuracy, and top-5 accuracy.

Figures 4.3 and 4.4 show the relative training time reduction vs. the relative top-1 accuracy change and the relative top-5 accuracy change for the CIFAR-100 dataset, respectively.

The results for CIFAR-100 show that both proposed methods get significant training time percentage reduction with a very small top-1 accuracy cost and a slight top-5 accuracy increase for some settings. For this dataset, the three-stage training method performs better than the flat

reduced sampling method, with comparable top-1 accuracy as the reimplemented ResNet56 and less training time. We also notice that some settings of the three-stage sampling method have slightly increased the top-5 accuracy.

Table 4-11: Training performances on CIFAR-100

Method	Setting	Top1 %	Top5 %	Time min/sec
ResNet56 [1]		-	-	-
ResNet56 Re-impl.	-	70.58	91.65	42/47
ResNet56 R1_0.8	$\alpha = 0.8$	70.02	91.49	34/53
ResNet56 R1_0.618	$\alpha = 0.618$	69.41	91.41	26/36
ResNet56 R2	$\alpha = 0.8 \cdot [1,1,1]$ $\gamma = [1/3, 1/3, 1/3]$ $se = 0.8$	69.76	91.66	32/39
ResNet56 R2_1	$\alpha = [0.8,1.0,0.8]$ $\gamma = [1/3, 1/3, 1/3]$ $se = 0.8$	70.07	91.55	34/26
ResNet56 R3	$\alpha = [0.8,0.8,0.25]$ $\gamma = [1/3, 1/2, 1/6]$ $se = 0.8$	70.45	91.69	28/13

Table 4-12: Training performances analysis on CIFAR-100

Method	Top-1 %	Top-5 %	Time min/sec	Top1↑ %	Top5↑ %	Time↑ %	Flat/ B-neck
ResNet56 Re-impl.	70.58	91.65	42/47	-	-	-	-
ResNet56 R1_0.8	70.02	91.49	34/53	-0.79	-0.17	-18.47	Flat
ResNet56 R1_0.618	69.41	91.41	26/36	-1.66	-0.26	-37.83	Flat
ResNet56 R2	69.76	91.66	32/39	-1.16	+0.01	-23.69	B-neck
ResNet56 R2_1	70.07	91.55	34/26	-0.72	-0.11	-19.52	B-neck
ResNet56 R3	70.45	91.69	28/13	-0.18	+0.04	-34.05	B-neck

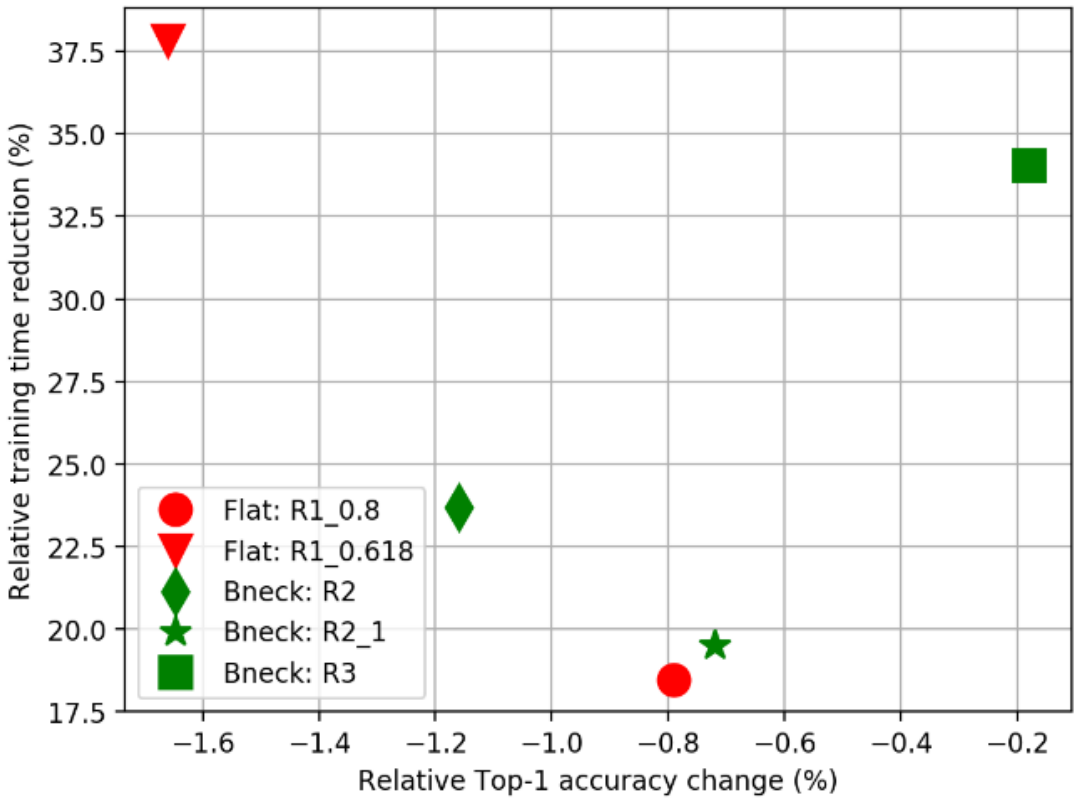

Figure 4.3: Relative training time reduction vs relative top-1 accuracy change for CIFAR-100 dataset.

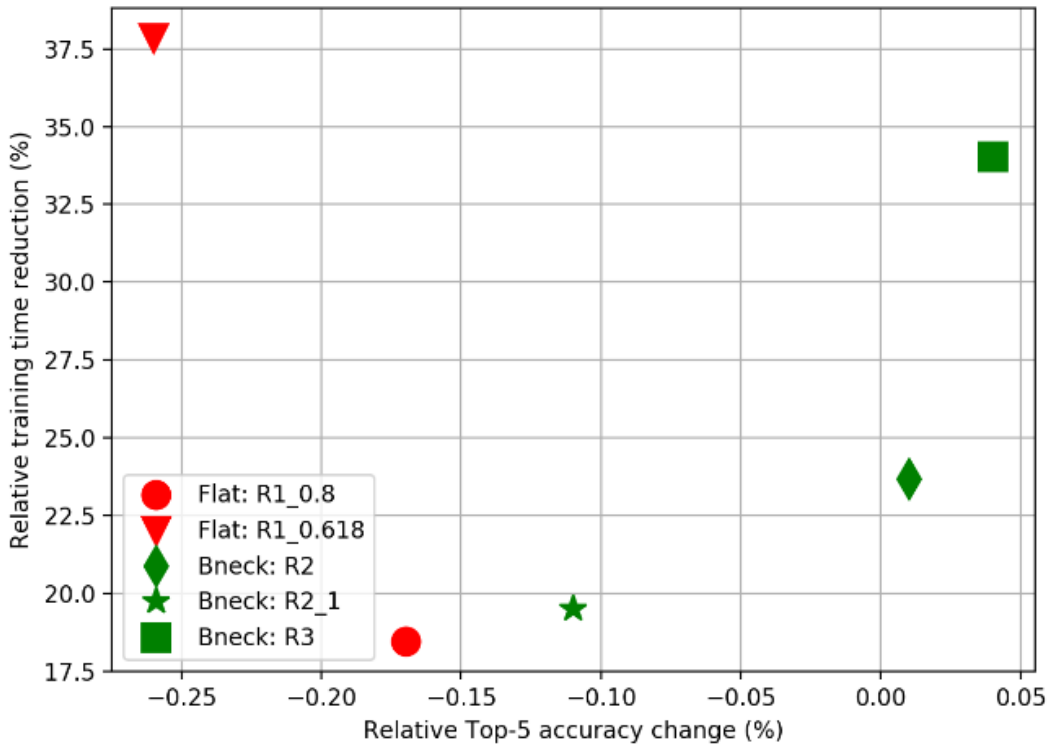

Figure 4.4: Relative training time reduction vs relative top-5 accuracy change for CIFAR-100 dataset.

4.2.5 ImageNet Experiments

For the ImageNet dataet, we have trained the network for 150 epochs with a batch size of 256. We use the stochastic gradient descent optimizer with a momentum of 0.9 and a weight decay of 4e-5 [61]. The learning rate starts at 0.05, and a cosine decay strategy is applied to the learning rate [61]. The MobileNet v2 (M-Net v2) [9] is used as the backbone network.

Table 4-13 shows the details of the experiments. Table 4-14 lists the relative percentage improvement by comparison with the MobileNet v2 reimplementation results for training time, top-1 accuracy, and top-5 accuracy. We use the results on the ImageNet validation set for the test accuracy comparison.

Similarly, Figures 4-5 and 4-6 show the relative training time reduction vs. the relative top-

1 accuracy change and top-5 accuracy change, respectively, for the ImageNet dataset.

The results show that for the ImageNet dataset, both proposed methods show significant training time percentage reduction with a small accuracy cost. For this dataset, the flat reduced sampling method is comparable with the three-stage training method in the training time reduction and accuracy change.

Table 4-13: Training performances on ImageNet

Method	Setting	Top1 %	Top5 %	Time hour/min
M-Net v2* [31]		72.00		
M-Net v2 Re-impl.	-	72.15	90.42	180/09
M-Net v2 R1_0.8	$\alpha = 0.8$	71.62	90.23	144/45
M-Net v2 R1_0.618	$\alpha = 0.618$	71.20	90.21	112/04
M-Net v2 R2_2	$\alpha = [0.8,0.8,0.5]$ $\gamma = [1/3\,, 1/3\,, 1/3]$ $se = 0.625$	71.29	90.15	112/18

*M-Net v2: MobileNet v2

Table 4-14: Training performances analysis on ImageNet

Method	Top1 %	Top5 %	Time hour/min	Top1↑ %	Top5↑ %	Time↑ %	Flat/ B-neck
M-Net v2 Re-impl.	72.15	90.42	180/09	-	-	-	-
M-Net v2 R1_0.8	71.62	90.23	144/45	-0.74	-0.21	-19.65	Flat
M-Net v2 R1_0.618	71.20	90.21	112/04	-1.32	-0.23	-37.79	Flat
M-Net v2 R2_2	71.29	90.15	112/18	-1.19	-0.30	-37.66	B-neck

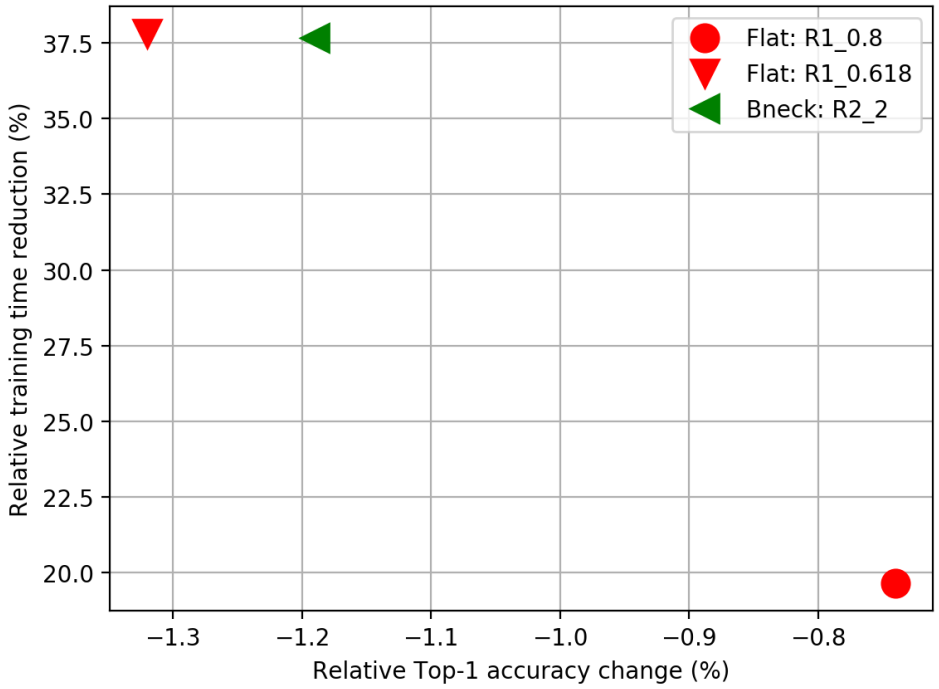

Figure 4.5: Relative training time reduction vs relative top-1 accuracy change for ImageNet dataset.

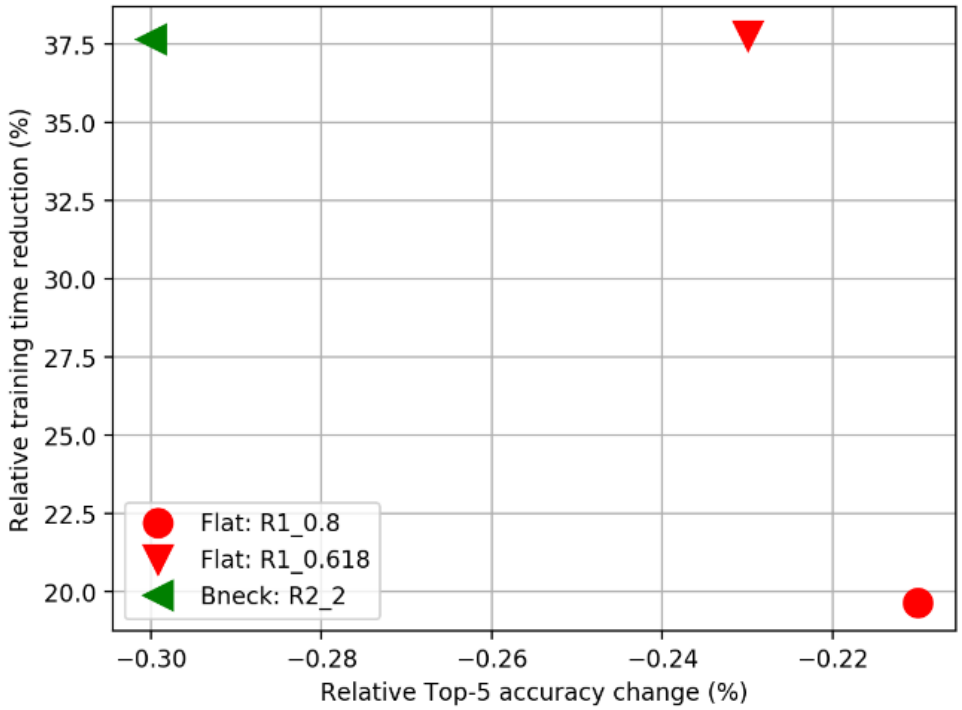

Figure 4.6: Relative training time reduction vs relative top-5 accuracy change for ImageNet dataset.

CHAPTER 5: CONCLUSIONS AND FUTURE WORKS

5.1 Conclusions on Architecture Complexity Reduction

Model compression is a popular deep CNN research area to train faster networks by reducing the network architecture complexity. We note that the CNN layer input-output patterns are ignored in current compression strategies. Based on this finding, we have proposed the following methods.

(1) For single 1×1 convolution layer compression, we propose a Conditional Reduction (CR) module. The CR module checks the CNN layer input-output pattern and uses different compression strategies.

(2) For image convolution blocks: bottleneck convolution block and inverted bottleneck block, we apply the CR module to them and get the Conditional block (C-block) which can be classified as C-Bneck and C-IBneck based on the original block types to be compressed.

(3) For network architecture, we apply the CR module to two types of CNN: one is stacked with bottlenecks, and the other is stacked with inverted bottlenecks. We implement this by replace the bottlenecks/inverted bottlenecks with C-Bneck/C-IBneck and get the Conditional Reduction Network (CRnet). The CRnet can be classified as CRnet-CBneck (CRnet-CB) and CRnet-CIBneck (CRnet-CIB).

We have tested the CRnet on two image classification datasets: CIFAR-10 and CIFAR-100. For a fair and comprehensive comparison, we test the methods with multiple network expansion ratio and compression ratio settings.

The experiments show that:

(1) If we switch the compression strategy related to the input-output pattern condition, that is, using the inverse CR compression, the accuracies are all lower than our CR module compression for our verified datasets. This verifies our method correctness with the attention to the importance of the input-output pattern when selecting a compression strategy.

(2) If we use group convolution to compress the network, although it largely reduces the parameter number and the FLOPs, it has low accuracy compared to other compression strategies. However, after we combine the group convolution with the input-output patterns and use it conditionally, the accuracy is the highest on average for all experiments. What is more, the use of group convolution helps to reduce the parameter number and the FLOPs comparing to the state-of-the-art Ghost module compression. That again verifies our method correctness and assumption/suggestion that the performance of a compression strategy may relate to the network architecture.

(3) The expansion ratio and compression ratio importance is less important than the input-output pattern.

(4) Our proposed CRnet with CR module compression better balances the model complexity and accuracy compared to the popular group convolution and the state-of-the-art Ghost module compression.

(5) The CR module and the C-block are simple, effective, and plug-and-play, so they can be easily utilized in any deep CNN.

5.2 Future Work on Architecture Complexity Reduction

We have tested our proposed methods using two benchmark image classification datasets: CIFAR-10 and CIFAR-100, with various parameter settings to verify the methods' effectiveness. We will study the method using larger datasets in the future.

In addition, we note that the CNN intrusion arouses researchers' interests. CNNs can be cheated by adversarial samples with few pixel modifications. These kinds of modifications usually are hard for humans to recognize. However, the CNNs can output a distinctive different image class. Thus, current CNNs are at risk of cyber-attacks. We are interested in further study on the model performance difference caused by model compressions for the security issues.

5.3 Conclusions on Data Reduction

Most of the deep CNN training uses full batch training. Full batch training will train all the samples at each epoch. Although there are methods to drop samples based on the sample importance rank calculated from the training loss or validation loss, their sorting calculations take extra time, especially for large datasets. When we search for the optimal hyperparameter setting, we expect to train CNN much faster, saving the multiple run's time, by our proposed random training data reduction methods.

In this case, a little accuracy drop is acceptable. Based on this object, we have proposed the following methods.

(1) We propose a flat reduced random sampling training strategy and a bottleneck reduced random sampling strategy.

(2) We propose a three-stage training method based on the bottleneck reduced random sampling with consideration of the distinctiveness of the network early-stage training and end-stage training.

(3) We prove the data visibility of a sample in the whole training process and the theoretical reduced time by four theorems and two corollaries.

We have tested the flat reduced random sampling and the three-stage training on three image classification datasets: CIFAR-10, CIFAR-100, and ImageNet. The experiments show that:

(1) The introduced two sampling strategies get significant training time percentage reduction at a very small accuracy loss. Therefore, the strategies are effective in reducing the CNN training time.

(2) We have proved that with our sampling setting, an image that gets missed by the whole training process is extremely low. That means each image will be studied by the network.

(3) For large dataset training, the flat golden ratio $\alpha = 0.618$ and the three stage-ratio R2-2 are good choices, with little accuracy drop but a large amount of training time percentage reduction.

5.4 Future Work on Data Reduction

The proposed methods show their benefit in reducing the training time with tiny accuracy drop. We expect to apply the proposed methods to various deep learning applications to help reduce the training time and get an optimal network setting.

REFERENCES

[1] B. M. Garlapati and S. R. Chalamala, A system for handwritten and printed text classification, in *Proceedings of the 19th International Conference on Computer Modelling & Simulation*, 2017: 50-54.

[2] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, End-to-end encrypted traffic classification with one-dimensional convolution neural networks, in *Proceedings of the 15th IEEE International Conference on Intelligence and Security Informatics*, 2017: 43-48.

[3] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, Unet++: A nested u-net architecture for medical image segmentation, in *Proceedings of the 4th International Workshop on Deep Learning in Medical Image Analysis and the 8th International Workshop on Multimodal Learning for Clinical Decision Support*, 2018: 3-11.

[4] F. Milletari, N. Navab, and S.-A. Ahmadi, V-net: Fully convolutional neural networks for volumetric medical image segmentation, in *Proceedings of the 4th International Conference on 3D vision*, 2016: 565-571.

[5] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, Image inpainting for irregular holes using partial convolutions, in *Proceedings of the 15th European Conference on Computer Vision*, 2018: 85-100.

[6] R. A. Yeh, C. Chen, T. Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, Semantic image inpainting with deep generative models, in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017: 5485-5493.

[7] S. Jiang and S.-G. Wang, Fast Training Methods and Their Experiments for Deep Learning CNN Models, accepted by the *40th IEEE Chinese Control Conference*, 2021.

[8] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, Bag of tricks for image classification with convolutional neural networks, in *Proceedings of the 32nd IEEE Conference on Computer Vision and Pattern Recognition*, 2019: 558-567.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in *Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition*, 2018: 4510-4520.

[10] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, GhostNet: More features from cheap operations, in *Proceedings of the 33rd IEEE Conference on Computer Vision and Pattern Recognition*, 2020: 1580-1589.

[11] S. Han, H. Mao, and W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, *arXiv:1510.00149,* 2015.

[12] S. Han, J. Pool, J. Tran, and W. Dally, Learning both weights and connections for efficient neural network, in *Proceedings of the 29th Conference on Neural Information Processing Systems*, 2015: 1135-1143.

[13] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, *arXiv:1607.03250,* 2016.

[14] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, Pruning filters for efficient convnets, *arXiv:1608.08710,* 2016.

[15] J.-H. Luo, J. Wu, and W. Lin, Thinet: A filter level pruning method for deep neural network compression, in *Proceedings of the 16th IEEE International Conference on Computer Vision*, 2017: 5058-5066.

[16] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, Learning structured sparsity in deep neural networks, in *Proceedings of the 30th Conference on Neural Information Processing Systems*, 2016: 2074-2082.

[17] J. Yu and T. Huang, AutoSlim: Towards one-shot architecture search for channel numbers, *arXiv:1903.11728,* 2019.

[18] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, *et al.*, Nisp: Pruning networks using neuron importance score propagation, in *Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition*, 2018: 9194-9203.

[19] M. Zhu and S. Gupta, To prune, or not to prune: exploring the efficacy of pruning for model compression, *arXiv:1710.01878,* 2017.

[20] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, Amc: Automl for model compression and acceleration on mobile devices, in *Proceedings of the 15th European Conference on Computer Vision*, 2018: 784-800.

[21] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, Slimmable neural networks, *arXiv:1812.08928,* 2018.

[22] J. Yu and T. S. Huang, Universally slimmable networks and improved training techniques, in *Proceedings of the 17th IEEE International Conference on Computer Vision*, 2019: 1803-1811.

[23] G. Hinton, O. Vinyals, and J. Dean, Distilling the knowledge in a neural network, *arXiv:1503.02531,* 2015.

[24] S. Zagoruyko and N. Komodakis, Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer, *arXiv:1612.03928,* 2016.

[25] J. Yim, D. Joo, J. Bae, and J. Kim, A gift from knowledge distillation: Fast optimization, network minimization and transfer learning, in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017: 4133-4141.

[26] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017: 1251-1258.

[27] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, *et al.*, Searching for MobileNetV3, in *Proceedings of the 17th IEEE International Conference on Computer Vision*, 2019: 1314-1324.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, *et al.*, Mobilenets: Efficient convolutional neural networks for mobile vision applications, *arXiv:1704.04861,* 2017.

[29] 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, *arXiv:1602.07360,* 2016.

[30] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, ShuffleNet V2: Practical guidelines for efficient CNN architecture design, in *Proceedings of the 15th European Conference on Computer Vision*, 2018: 116-131.

[31] X. Zhang, X. Zhou, M. Lin, and J. Sun, ShuffleNet: An extremely efficient convolutional neural network for mobile devices, in *Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition*, 2018: 6848-6856.

[32] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016: 770-778.

[33] T. Akiba, S. Suzuki, and K. Fukuda, Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes, *arXiv:1711.04325,* 2017.

[34] H. Mikami, H. Suganuma, Y. Tanaka, and Y. Kageyama, Massively distributed SGD: ImageNet/ResNet-50 training in a flash, *arXiv:1811.05233,* 2018.

[35] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota, and S. Matsuoka, Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks, in *Proceedings of the 32nd IEEE Conference on Computer Vision and Pattern Recognition*, 2019: 12359-12367.

[36] Y. Ueno, K. Osawa, Y. Tsuji, A. Naruse, and R. Yokota, Rich information is affordable: A systematic performance analysis of second-order optimization using K-FAC, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020: 2145-2153.

[37] M. Yamazaki, A. Kasagi, A. Tabuchi, T. Honda, M. Miwa, N. Fukumoto, *et al.*, Yet another accelerated SGD: ResNet-50 training on ImageNet in 74.7 seconds," *arXiv:1903.12650,* 2019.

[38] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, *et al.*, Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minutes, *arXiv:1807.11205,* 2018.

[39] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, Training deep neural networks with 8-bit floating point numbers, *arXiv:1812.08011,* 2018.

[40] N. J. Higham and S. Pranesh, Simulating low precision floating-point arithmetic, *SIAM Journal on Scientific Computing,* 41(5): C585-C602, 2019.

[41] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. V. Srinivasan, *et al.*, Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks, in *Proceedings of the 33rd Conference on Neural Information Processing Systems,* 2019: 4900-4909.

[42] I. Loshchilov and F. Hutter, Online batch selection for faster training of neural networks, *arXiv:1511.06343,* 2015.

[43] M. J. van Grinsven, B. van Ginneken, C. B. Hoyng, T. Theelen, and C. I. Sánchez, Fast convolutional neural network training using selective data sampling: Application to hemorrhage detection in color fundus images, *IEEE Transactions on Medical Imaging,* 35 (5): 1273-1284, 2016.

[44] L. Berger, H. Eoin, M. J. Cardoso, and S. Ourselin, An adaptive sampling scheme to efficiently train fully convolutional networks for semantic segmentation, in *Proceedings of the 22nd Annual Conference on Medical Image Understanding and Analysis*, 2018: 277-286.

[45] B. Cheng, Y. Wei, J. Yu, S. Chang, J. Xiong, W.-M. Hwu, *et al.*, A Simple Non-iid Sampling Approach for Efficient Training and Better Generalization, *arXiv:1811.09347,* 2018.

[46] W. Huang, T. Zhang, Y. Rong, and J. Huang, Adaptive sampling towards fast graph representation learning, *arXiv:1809.05343,* 2018.

[47] T. B. Johnson and C. Guestrin, Training deep models faster with robust, approximate importance sampling, in *Proceedings of the 32nd Conference on Neural Information Processing Systems, 2018:* 7265-7275.

[48] H. Inoue, Multi-sample dropout for accelerated training and better generalization, *arXiv:1905.09788,* 2019.

[49] S. Ramjee, S. Ju, D. Yang, X. Liu, A. E. Gamal, and Y. C. Eldar, Fast deep learning for automatic modulation classification, *arXiv:1901.05850,* 2019.

[50] Y. Gong, L. Liu, M. Yang, and L. Bourdev, Compressing deep convolutional networks using vector quantization, *arXiv:1412.6115,* 2014.

[51] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, Deep learning with limited numerical precision, in *Proceedings of the 32nd International Conference on Machine Learning*, 2015: 1737-1746.

[52] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul, Mixed precision training with 8-bit floating point, *arXiv:1905.12334,* 2019.

[53] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in *Proceedings of the 28th Conference on Neural Information Processing Systems*, 2014: 1269-1277.

[54] S. Jiang and S.-G. Wang, Architecture-aware Compact Convolutional Neural Network Design, *in review*, 2021.

[55] Y. Zuo, B. Chen, T. Shi, and M. Sun, Filter Pruning Without Damaging Networks Capacity, *IEEE Access*, 8: 90924-90930, 2020.

[56] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, Aggregated residual transformations for deep neural networks, in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1492-1500.

[57] J. Hu, L. Shen, and G. Sun, Squeeze-and-excitation networks, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018: 7132-7141.

[58] Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in *Neural networks: Tricks of the trade*, G. Montavon, G. B. Orr, KR. Müller, Eds. Berlin, Heidelberg: Springer, 2012: 437-478.

[59] A. Krizhevsky and G. Hinton, Learning multiple layers of features from tiny images, *Technical Report*. 2009.

[60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, Imagenet: A large-scale hierarchical image database, in *Proceedings of the 22nd IEEE Conference on Computer Vision and Pattern Recognition*, 2009: 248-255.

[61] D. Li, A. Zhou, and A. Yao, HBONet: Harmonious bottleneck on two orthogonal dimensions, in *Proceedings of the 17th IEEE International Conference on Computer Vision*, 2019: 3316-3325.