



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

공학박사 학위논문

Quantization Algorithm and Methodology for Efficient Deep Neural Network

효율적인 심층 신경망을 위한 양자화 알고리즘 및 방법론

2020 년 2 월

서울대학교 대학원

컴퓨터공학부

박 은 혁

Quantization Algorithm and Methodology for Efficient Deep Neural Network

효율적인 심층 신경망을 위한 양자화 알고리즘 및
방법론

지도교수 유승주

이 논문을 공학박사 학위논문으로 제출함

2019년 11월





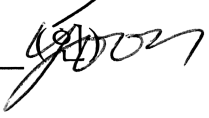
서울대학교 대학원

컴퓨터공학부

박은혁

박은혁의 공학박사 학위论문을 인준함

2019년 12월

위원장	이재욱	
부위원장	유승주	
위원	김장우	
위원	유민수	
위원	윤성로	

Abstract

Quantization Algorithm and Methodology for Efficient Deep Neural Network

Eunhyeok Park

Department of Computer Science and Engineering

The Graduate School

Seoul National University

Deep neural networks (DNN) are becoming increasingly popular and widely adopted for various applications. Energy efficiency of neural networks is critically important for both edge devices and servers. It is imperative to optimize neural networks in terms of both speed and energy consumption while maintaining the accuracy of the network.

Quantization is one of the most effective optimization techniques. By reducing the bit-width of activations and weights, both the speed and energy can be improved by executing more computations using the same amount of memory access and computational resources (e.g. silicon chip area and battery). It is expected that computations with 4-bit and lower precision will contribute to the energy efficient and real-time characteristics of future deep learning applications.

One major drawback of quantization is the drop in accuracy, resulting from the reduction in the degree of freedom of data representation. Recently, there have been several studies that demonstrated that the inference of DNNs can be accurately done by using 8-bit precision. However, many studies show that the network quantized into 4-bit or less precision suffers from significant quality degradation. Especially, the state-of-the-art networks cannot be quantized easily due to their optimized structure.

In this dissertation, several methods are proposed that use different approaches to minimize the reduction in the accuracy of the quantized DNNs. Weighted-entropy-based quantization is designed to fully utilize the limited number of quantization levels by maximizing the weighted information of the quantized data. This work shows the potential of multi-bit quantization for both activation and weight. Value-aware quantization, or outlier-aware quantization is designed to support sub-4-bit quantization, while allowing a small amount (1 ~ 3 %) of large values in high precision. This helps the quantized data to maintain the statistics, e.g. mean and variance corresponding to the full-precision, thus minimizing the accuracy drop after quantization. The dedicated hardware accelerator, called OLAcel, is also proposed to maximize the performance of the network quantized by the outlier-aware quantization. The hardware takes advantage of the benefit of reduced precision, i.e. 4-bit, with minimal accuracy drop by the proposed quantization algorithm. Precision-highway is the structural concept that forms an end-to-end high-precision information flow while performing ultra-low-precision computations. This minimizes the accu-

mulated quantization error, which helps to improve the accuracy of the network even with extremely low precision. BLast, the training methodology, and differentiable and unified quantization (DuQ), a novel quantization algorithm, are designed to support sub-4-bit quantization for the optimized mobile networks, i.e. MobileNet-v3. These methods allow the MobileNet-v3 network to be quantized into 4-bit for both activation and weight with negligible accuracy loss.

Keywords: Deep neural network, optimization, quantization, hardware architecture, accelerator

Student Number: 2015-31050

Contents

Abstract	i
Contents	ix
List of Tables	xiii
List of Figures	xvii
Chapter 1 Introduction	1
Chapter 2 Background and Related Work	4
Chapter 3 Weighted-entropy-based Quantization	15
3.1 Introduction	15
3.2 Motivation	17
3.3 Quantization	
based on Weighted Entropy	20
3.3.1 Weight Quantization	20
3.3.2 Activation Quantization	24

3.3.3	Integrating Weight/Activation Quantization into the Training Algorithm	27
3.4	Experiment	28
3.4.1	Image Classification: AlexNet, GoogLeNet and ResNet-50/101	28
3.4.2	Object Detection: R-FCN with ResNet-50	35
3.4.3	Language Modeling: An LSTM	37
3.5	Conclusion	38
 Chapter 4 Value-aware Quantization for Training and Inference of Neural Networks		40
4.1	Introduction	40
4.2	Motivation	41
4.3	Proposed Method	43
4.3.1	Quantized Back-Propagation	44
4.3.2	Back-Propagation of Full-Precision Loss	46
4.3.3	Potential of Further Reduction in Computation Cost	47
4.3.4	Local Sorting in Data Parallel Training	48
4.3.5	ReLU and Value-aware Quantization (RV-Quant)	49
4.3.6	Activation Annealing	50
4.3.7	Quantized Inference	50
4.4	Experiments	51
4.4.1	Training Results	52

4.4.2	Inference Results	59
4.4.3	LSTM Language Model	61
4.5	Conclusions	62
Chapter 5	Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation	63
5.1	Introduction	63
5.2	Proposed Architecture	65
5.2.1	Overall Structure	65
5.2.2	Dataflow	68
5.2.3	PE Cluster	72
5.2.4	Normal PE Group	72
5.2.5	Outlier PE Group and Cluster Output Tri-buffer	75
5.3	Evaluation Methodology	78
5.4	Experimental Results	80
5.5	Conclusion	90
Chapter 6	Precision Highway for Ultra Low-Precision Quantiza- tion	92
6.1	Introduction	92
6.2	Proposed Method	93
6.2.1	Precision Highway on Residual Network	94
6.2.2	Precision Highway on Recurrent Neural Network	96
6.2.3	Practical Issues with Precision Highway	98

6.3	Training	99
6.3.1	Linear Weight Quantization based on Laplace Distribution Model	99
6.3.2	Fine-tuning for Weight/Activation Quantization	100
6.4	Experiments	101
6.4.1	Experimental Setup	101
6.4.2	Analysis of Accumulated Quantization Error	101
6.4.3	Loss Surface Analysis of Quantized Model Training	103
6.4.4	Evaluating the Accuracy of Quantized Model	103
6.4.5	Hardware Cost Evaluation of Quantized Model	108
6.5	Conclusion	109
Chapter 7	Towards Sub-4-bit Quantization of Optimized Mobile Networks	114
7.1	Introduction	114
7.2	BLast Training	117
7.2.1	Notation	118
7.2.2	Observation	118
7.2.3	Activation Instability Metric	120
7.2.4	BLast Training	122
7.3	Differentiable and Unified Quantization	124
7.3.1	Rounding and Truncation Errors	124
7.3.2	Limitations of State-of-the-Art Methods	124

7.3.3	Proposed Method: DuQ	126
7.3.4	Handling Negative Values	128
7.4	Experiments	131
7.4.1	Accuracy on ImageNet Dataset	131
7.4.2	Discussion on Fused-BatchNorm	133
7.4.3	Ablation Study	134
7.5	Conclusion	137
Chapter 8	Conclusion	138
	Bibliography	141
	국문초록	154
	Acknowledgements	157

List of Tables

Table 3.1	Memory requirement comparison with AlexNet (P: Pruning ratio, Q: Quantization, H: Huffman encoding).	34
Table 3.2	Accuracy comparison of our approach under different styles of layer-wise quantization.	35
Table 3.3	Impact of quantization on word-level perplexity of an LSTM for language modeling.	38
Table 4.1	Top-1/top-5 accuracy [%] of ResNet-50 with various bitwidth & AR configurations. The full precision network gives the accuracy of 75.92 / 92.90%.	53
Table 4.2	Top-1/top-5 accuracy [%] of ResNet-50 under RV-Quant. The full-precision network gives the accuracy of 75.92 / 92.90%.	54
Table 4.3	Training results. Full means the results of conventional full-precision training, while 3-bit 2% and 8-bit 0% correspond to RV-Quant. The full-precision network gives the accuracy of 75.92 / 92.90%.	55
Table 4.4	Comparison of memory cost (in GB).	56

Table 4.5	Sensitivity analysis of RV-Quant configurations (bitwidth and AR [%]) across training phases. The full-precision network gives the accuracy of 75.92 / 92.90%.	57
Table 4.6	Impact of quantization on word-level perplexity of an LSTM for language modeling.	61
Table 5.1	Configurations of Eyeriss, ZeNA, and OLAcel.	79
Table 6.1	2-bit quantization results. Top-1 / Top-5 accuracy [%].	103
Table 6.2	Comparison of accuracy loss in 2-bit activation / weight quantization. Bi-real applies 1-bit activation / weight quantization.	105
Table 6.3	Impact of highway precision (y-axis: low precision and x-axis: highway precision). Top-1 / Top-5 accuracy [%].	106
Table 6.4	Accuracy of wide ResNet-18 and ResNet-50 with quantization. Top-1/Top-5 accuracy [%].	106
Table 6.5	Perplexity of quantized LSTM. (x,y) means x-bit weight / y-bit activation.	107
Table 6.6	Number of operations. * denotes the high-precision operation.	109
Table 7.1	Top-1 / Top-5 accuracy [%] of the quantized networks on ImageNet	132
Table 7.2	Top-1 accuracy [%] comparison of existing works on MobileNet-v1 and MobileNet-v2.	133

List of Figures

Figure 2.1	The representative examples of the quantized data. The weights are extracted from the second 3×3 convolution layer of GoogLeNet [1].	5
Figure 2.2	Computation with quantization operator	9
Figure 2.3	Straight-through estimator for quantized-weight training [2]	10
Figure 3.1	Comparison of various quantization schemes. The weights are extracted from the second 3×3 convolution layer of GoogLeNet [1]. Each quantization scheme is given to assign 24 levels. We use $2^{0.5}$ as the base of LogQuant and optimize both linear quantization and LogQuant towards minimizing the L2 norm of overall activations.	18
Figure 3.2	Top-1 and top-5 accuracy of quantized CNNs after fine-tuning. The dashed lines represent the accuracy of the baseline networks, which use full-precision arithmetic.	30

Figure 3.3	Accuracy comparison of quantization methods applied to AlexNet. ‘Weighted Quantization’ represents our approach, while ‘X’ and ‘D’ are for XNOR-Net and DoReFa-Net, respectively. The dashed lines indicate the accuracy of the baseline full-precision network.	32
Figure 3.4	mAP results of R-FCN. The dashed line represents the accuracy of the baseline full-precision network.	36
Figure 4.1	Activation and weight distributions of second convolutional layer in GoogLeNet.	42
Figure 4.2	Value-aware quantization in training pipeline.	44
Figure 4.3	Training loss of ResNet-50 with various RV-Quant configurations.	58
Figure 4.4	V-Quant results. The dashed lines and black solid bar represent full-precision accuracy. Legend: bitwidth / AR [%] / fine-tuning or not.	59
Figure 4.5	PCA analysis of the input activations on the last fully-connected layer of AlexNet.	60
Figure 5.1	Overall structure of OLAcel.	66
Figure 5.2	Data structure associated with normal / outlier weight and normal activation.	69
Figure 5.3	Operation of PE cluster.	71

Figure 5.4	PE group operation in case that there is an outlier weight among 16 weights.	73
Figure 5.5	PE group in the case of more than one outlier weight.	74
Figure 5.6	Data structure associated with outlier activation.	75
Figure 5.7	Cluster output tri-buffer.	77
Figure 5.8	AlexNet cycle and energy breakdown.	81
Figure 5.9	VGG-16 cycle and energy breakdown.	82
Figure 5.10	ResNet-18 cycle and energy breakdown.	83
Figure 5.11	Normalized energy and cycle vs. outlier ratio: AlexNet on OLAcel16.	85
Figure 5.12	Scalability analysis on AlexNet: speedup (y-axis) vs. number of NPUs.	86
Figure 5.13	Histogram of outlier activation: outlier ratio of 3%.	87
Figure 5.14	Probability of multiple outlier weights (y-axis) vs. outlier ratio.	88
Figure 5.15	Utilization breakdown: AlexNet.	89
Figure 5.16	Execution cycles when processing a chunk of $A_{1 \times 1 \times 16}$ input activations.	90
Figure 6.1	Comparison of conventional quantization and our proposed idea on residual network.	110
Figure 6.2	Comparison on residual network.	111

Figure 6.3	Weight histogram and Laplace approximation (dashed line) of the convolutional layer of a trained full-precision ResNet-50.	111
Figure 6.4	Quantization error accumulation across residual blocks in ResNet-50.	112
Figure 6.5	Loss surface of ResNet-18 on Cifar-10: (a) full-precision model (FP), (b) 1-bit activation and 2-bit weight quantized model (1A2W) without precision highway (PH), (c) 1A2W with precision highway, and (d) cross-section of loss surface.	112
Figure 6.6	Comparison of chip area and energy consumption on the hardware accelerator.	113
Figure 7.1	Comparison of accuracy and complexity (GBOPs, [3]) of the quantized network. The tuple (a,w) represents the bit-width of activation and weight, respectively.	114
Figure 7.2	Top-1 accuracy [%], AIWQ metric, and running mean/variance during the fine-tuning for quantization where N_{lv} represents the number of available quantization levels. Running mean/variance are extracted from the arbitrary channels of batch normalization layer after depth-wise convolution in the 2nd inverted residual module.	117

Figure 7.3	BLast training method.	122
Figure 7.4	Two error sources of quantization.	124
Figure 7.5	PACT algorithm.	125
Figure 7.6	QIL algorithm.	126
Figure 7.7	Proposed DuQ algorithm.	127
Figure 7.8	Negative padding for h-swish function.	129
Figure 7.9	Comparison of quantization algorithms under BLast. .	135
Figure 7.10	Evaluation of BLast(+).	136

Chapter 1

Introduction

Deep neural networks (DNN) are becoming more and more popular and widely adopted for various applications. On edge devices, these applications have tight constraints regarding the latency for real-time operation and energy consumption due to the battery. On servers, fast training and inference are critical in order to support fast deployment and on-line training. Therefore, it is imperative to optimize neural networks in terms of speed and energy consumption while maintaining the accuracy of the network.

Quantization is one of the most effective optimization techniques. By reducing the bit-width of the activations and weights, both the speed and energy can be improved by executing more computations with the same amount of memory and computational resources (e.g. silicon chip area and battery). Recently, there have been active studies that demonstrated that the inference of DNNs can be accurately done by using 8-bits [4,5], and most existing hardware exploits the benefit of quantization by supporting an 8-bit integer arithmetic computation [4,6–8]. Some studies also show the potential of sub-4-bit quantization [9–12], and it is expected that 4-bit and lower precision computation can

contribute significantly to the energy efficiency and real-time characteristics of deep learning applications in the future [13].

One major drawback of quantization is the decrease in accuracy resulting from the reduction in the degree of freedom of data representation. In this dissertation, several methods are proposed that utilize various approaches to minimize the decrease in the accuracy of the quantized DNNs. Weighted-entropy-based quantization [14] is designed to fully utilize the limited number of quantization levels by maximizing the weighted information of the quantized data. Value-aware quantization, or outlier-aware quantization [15] is designed to support sub-4-bit quantization while allowing a small amount (1 ~ 3 %) of large values to have high precision. This helps the quantized data to maintain the statistical characteristics, such as the mean and variance, of full-precision data, thus minimizing the reduction in accuracy after the quantization. A dedicated hardware accelerator [16] is also proposed, thus maximizing the performance of the quantized network based on the proposed algorithm. Precision-highway [17] is the structural concept that forms an end-to-end high-precision information flow while performing ultra-low-precision computations. BLast is the training pipeline for the quantized network, and differentiable and unified quantization [18] is a novel quantization algorithm that is universally applicable to various networks. Both methods are designed to support the quantization of the optimized mobile networks.

This dissertation is organized as follows: Chapter 2 introduces the background of the quantization concept and reviews the previous studies. In chap-

ter 3, the weighted-entropy-based quantization is introduced. Chapter 4 explains the value-aware quantization and the corresponding dedicated hardware accelerator called OLAccel is proposed in Chapter 5. Chapter 6 describes the precision-highway, and the concepts of BLast and DuQ are introduced in Chapter 7 in order to quantize the state-of-the-art optimized mobile networks. Chapter 8 concludes the dissertation.

Chapter 2

Background and Related Work

Quantization refers to the task of restricting data representation. It maps floating point data belonging to a wide range to one of the limited number of discontinuous values, i.e. quantization levels, through clustering. In addition, an integer index is assigned to each cluster sequentially in the ascending order in order to distinguish the different clusters. The range of the index is proportional to the number of discontinuous values (= the number of quantization levels N_{lv}), and thus, the bit-width representing the index data is also related to N_{lv} ; typically the bit-width is equal to $\log_2(N_{lv})$. The quantized data can be represented by the combination of a mapping table and an index tensor; the mapping table maps the quantization level index to the quantization value, and the index tensor consists of element-wise integer indices that are identical in shape to the original data before quantization. Because the bit-width of an index is generally much smaller than the floating-point data, quantization allows the data to be stored in a smaller space.

The major advantage of quantization results from the reduction in the storage space of the data, which improves the performance and energy consump-

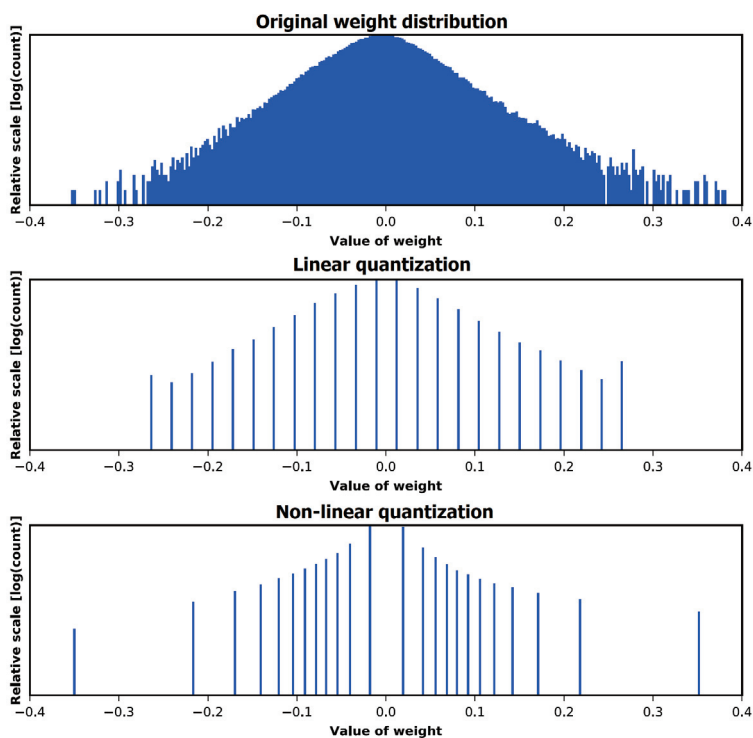


Figure 2.1 The representative examples of the quantized data. The weights are extracted from the second 3×3 convolution layer of GoogLeNet [1].

tion of deep neural networks. More data can be stored in the same memory footprint and the data can be loaded quickly from the same bandwidth. These advantages results in faster operation with less energy. Meanwhile, quantization also has a major drawback, namely, limited capability for expression. Quantized data has lower degrees of freedom when compared to full-precision data. Thus, quantization algorithms should be applied carefully in order to maintain the desired quality of the neural network with limited expressiveness.

Quantization algorithms are classified into two types, namely, linear or non-

linear quantization, based on their representations. Linear quantization utilizes the uniform spacing between quantization levels, while non-linear quantization has non-uniform spacing so that the quantization levels can have arbitrary value. In the case of linear quantization, as the quantization levels are separated evenly and the index is assigned according to the value of the quantization level, a linear relationship is established between the index and the quantization level. In other words, the mapped quantization value can be calculated from the index tensor using a scale and shift operation. This characteristic helps to accelerate the computation of the quantized data at a lower hardware cost. On the other hand, the non-linear quantization has a higher degree of freedom than linear quantization, which is more stable and has a higher bit efficiency. Therefore, non-linear quantization is preferred over linear quantization when the bit-width needs to be reduced extremely while maintaining the accuracy.

Before proceeding, it is necessary to explain in more detail how linear quantization can help hardware acceleration. In general, in order to perform the convolution or matrix multiplication computation, it is necessary to convert the quantized data, which is stored as a low-precision index tensor, into high-precision data using the mapping table. Subsequently, high precision computation is performed. However, when the data is quantized linearly, we can take advantage of the relationship between the quantization index and quantization value for convolution or matrix multiplication computations; those functions are linear operators, and thus, the desired output can be calculated by scaling the outcome of the computation with the integer index alone. It can be

expressed as follows:

$$O = (\alpha \cdot I) \otimes (\beta \cdot W) \quad (2.1)$$

$$= (\alpha \cdot \beta) \cdot (I \otimes W) \quad (2.2)$$

The convolution and matrix multiplication computations can be performed with the index tensor having small bits of integer data. It helps to greatly reduce the cost of computation because the number of operations of the convolution and matrix multiplication are much larger than that of scaling in general.

Quantization of neural networks can be applied to the three types of data, namely, the activation, weight, and gradient. Since each data type has different characteristics, it is necessary to select the quantization algorithm carefully depending on the application.

The weight is held constant after training, while it requires constant storage space even when the network is not activated. In addition, the weight has different characteristics depending on the operator used. In the case of matrix multiplication, it is essential to reduce the bit-width of the weight because it is not reused when the batch-size is 1, and the footprint of the weight is proportional to the square of the feature map size. Meanwhile, in the case of convolution, computation optimization is more important because the weight can be re-used several times and convolution is a computationally intensive operation. Considering those characteristics, different quantization algorithms are selected. Non-linear quantization having high bit-efficiency is preferred in order to minimize the storage overhead while linear quantization is often used

in order to accelerate the computation.

Meanwhile, the activation is related to input data, making it difficult to apply quantization with arbitrary mapping. Instead, the quantization algorithm having regular expression like linear quantization or logarithm quantization is utilized frequently. Those quantization methods enable the calculation of the quantization value from the index without a mapping table, and the reverse calculation is also possible. The purpose of activation quantization is to speed up the computation in both inference and training and to reduce memory consumption in training. In the former case, linear quantization is applied to activation in parallel with the weight, which makes the convolution or matrix computation possible with simple integer operations. On the other hand, in the latter case, because intermediate activations need to be stored during training, a large memory space should be reserved for the activation. Activation quantization is used in order to minimize memory consumption by reducing the bit-width of activation.

Gradient quantization is mainly applied to minimize the computation overhead. As with activation, the gradient is input-dependent, and the linear quantization is often utilized. It has a relatively large bit-width compared to the activation and weight, because it has a profound impact on the accuracy of the network and the distribution varies greatly at each layer and at each stage of learning.

Figure 2.2 shows the typical applications quantization on deep neural network. Convolution and matrix multiplication operations always produce high-

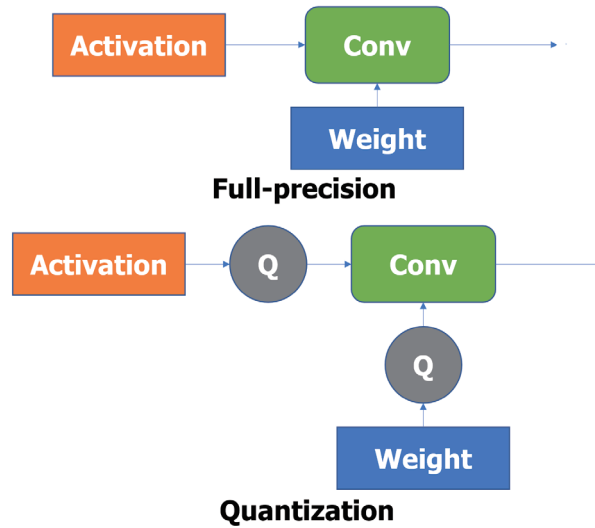


Figure 2.2 Computation with quantization operator

precision output because both operations accumulate the products of activation and weight. In order to reduce the bit-width of high-precision data, the quantization operator is inserted in the middle of the network. The quantization operator can be different depending on the quantization target and position. In addition, the convolution or matrix multiplication function can be optimized for the quantized data. If the conventional high-precision operator is used, then there is no performance gain. Meanwhile, if the optimized operator is used, there can be an improvement in the performance. For instance, if the integer convolution with scaling operator is used for the linearly quantized data, the computation overhead, i.e. the ALU area, power consumption, and performance, can be improved significantly. Likewise, additional improvement can be expected from the combination of quantization with structural and operation optimization.

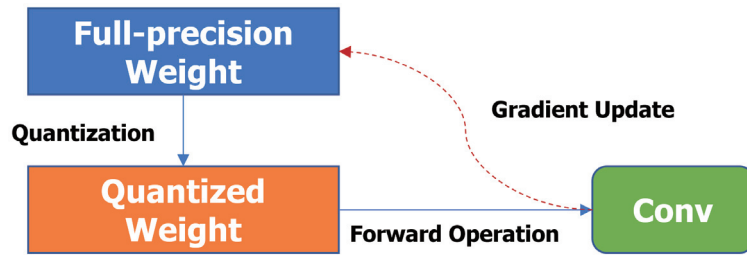


Figure 2.3 Straight-through estimator for quantized-weight training [2]

Typically, the pre-trained weight of full-precision is used as initialization weight for the quantized network because it works as a good initial point. However, there is significant decrease in the accuracy due to quantization, making it inevitable to apply fine-tuning after quantization in order to recover the accuracy loss especially for sub-4-bit quantization. However, the quantized weight has discontinuous values and so, updating the weight through the accumulation of gradient. In order to fine-tune the quantized weight, there is an alternative solution called the straight-through estimator [2]. As shown in the figure 2.3, original floating-point data is held during fine-tuning. In the forward phase, the quantized weight is used to perform the computation. In the backward phase, the quantization operator bypasses the gradient, and the gradient is accumulated on the original full-precision data. By repeating this process for several iterations, the accuracy loss can be recovered. STE is used in all cases where finetuning is required after quantization.

In order to exploit the benefit of quantization, there were several studies that endeavored to minimize the bit-width of deep neural networks while minimiz-

ing the accuracy loss. [9] is one of the early studies that showed the potential for quantization. They proposed the optimal quantization method for weight binarization and also proposed the quantization method for both activation and weight binarization. The fully-binarized network is called XNOR-Net because the multiplication-accumulation operation of the convolution operation can be replaced by the element-wise xnor with a popcount operation. According to their analysis, a weight-binarized AlexNet gives the same accuracy as a full-precision one, and XNOR-Net also shows promising results.

More recently, [19] was introduced in order to increase the accuracy of the binarized network. This study proposed a structural improvement to prevent error accumulation when applying quantization to the network having an identity path, such as ResNet. This idea limits the effect of quantization error to within a single module, thereby improving the accuracy significantly. The precision-highway, introduced in chapter 6, generalizes the structural improvement to a more general network structure including LSTM.

[10] presented DoReFA-Net, which applies *tanh*-based weight quantization and bounded activation quantization. It is a representative study for multi-bit linear quantization that allows the bit-width of activation and weight to have an arbitrary positive integer value. The trade-off relationship between accuracy and bit-width was introduced.

There were also other quantization algorithms based on linear quantization, e.g. [20], [11] and [12]. [20] proposed a balanced quantization that attempts to balance the population of values on quantization levels. By recursively parti-

tioning the parameters into balanced bins, they try to maximize the information (entropy) of the quantized data. They not only showed the quantization results of AlexNet but also presented the quantization result of deeper networks, ResNet-18 and GoogLeNet.

[11] and [12] are the state-of-the-art studies that present the loss-less quantization results of the 4-bit quantization of AlexNet and ResNet-50. Both algorithms are designed to optimize not only the network parameters but also the quantization parameters, e.g. clustering range and the value of quantization level. Because the quantization parameters are converted in order to minimize target loss, it shows better quantization result.

On the other hand, there are also interesting studies based on nonlinear quantization. [21] proposes logarithm-based quantization. This algorithm only allows the value of the quantization level to have an exponent value of two. It can give significant performance benefits because multiplication can be substituted by the shift operation. The study showed that AlexNet can be quantized with 4-bit weights and 5-bit activation at 1.7 % additional loss of top-5 accuracy.

[22] show that deep models can be quantized with separately scaled ternary weights while utilizing full-precision activations. In the case of [23], they quantize the networks using clustering-based methods. Those non-linear quantization methods show the potential to further reduce the precision, but they could not be mapped to optimized computation operation on the existing ALU, which makes them less hardware-friendly.

[24] improves the accuracy of the quantized network by improving the learning methods. It is based on the quantization method proposed by DoReFA-Net, but the quantized network gives higher accuracy by adopting the proposed training technique, a progressive quantization and knowledge distillation. The progressive quantization quantizes the network by lowering the bit-width sequentially from high-precision to lowprecision. Knowledge distillation transfers the knowledge of the teacher network to the quantized network that allows the quantized network to learn better. Those training techniques help to recover the accuracy drop of quantization.

[25] takes a different approach to improve the accuracy of the quantized network. It increases the number of filters of the network while maintaining the connection structure. The increased filters grow the capacity of the network, which helps the network to approximate the target function even with the limited representation.

In order to fully utilize the benefit of the quantized model, there have been active studies on the implementation of quantization on the CPU and GPU [4,6]. Current hardware supports 8-bit integer operations, and thus, those functions are mainly leveraged to accelerate the network with 8-bit quantization.

Besides the existing hardware, several dedicated accelerators are proposed [26–29]. [26,27] adopt high-precision computation in order to accelerate computation without the loss of accuracy. Instead, they perform the optimization by minimizing access to the external memory and maximizing data re-use. [28,29]

are the large-scale accelerators that have tens of thousands of processing elements. They show more than ten times higher energy efficiency than GPU based on their optimized structure for DNN.

Zero skipping is crucial for performance as well as energy efficiency. In [30], Albericio et al. proposed Cnvlutin that skips multiplications with zero-input activations. Only non-zero input activations are broadcast to MAC units while skipping zero-input activations. In [31], Zhang et al. proposed the Cambricon-X, which skips multiplications with zero weights obtained by pruning [23]. In [32], Parashar et al. proposed a sparse CNN (SCNN) that exploits both zero weights and activations by calculating the Cartesian products of non-zero weights and activations, and adding the results to the corresponding partial sums. SCNN suffers from low resource utilization in the case of high sparsity and high area/power overhead due to accumulator buffers and the crossbar. In [33], Kim et al. proposed a zero-aware neural network accelerator (ZeNA) that also skips the computation with both zero weights and activations. They reported that ZeNA provides an acceleration of 4.4x on AlexNet.

Chapter 3

Weighted-entropy-based Quantization

3.1 Introduction

This work was published in CVPR'2017 conference [14]. Existing quantization techniques have two limitations that can hinder practical application of such techniques into mobile and embedded systems. First, existing methods lack in supporting flexible trade-off between output quality and inference performance. Mobile and embedded systems often have stringent constraints in both resource and inference accuracy, which requires design space exploration for trade-off between output quality and inference performance. However, some of the existing approaches are not flexible enough to exploit such trade-off relationship. For example, techniques that binarize weights [9, 10, 34] suffer from a significant loss of output quality for deep networks, which cannot be applied if the target system allows a very small accuracy loss, e.g. 1%.

Second, even if existing quantization techniques support such trade-off, they require modifications to the target network to achieve good quantization

quality and/or apply quantization to only part of the network. Due to this, such techniques may require significant effort at design time, which may eventually prevent widespread adoption of them. In addition, existing methods such as XNOR-Net [9] and DoReFa-Net [10] do not apply quantization to the first and the last layer to avoid excessive accuracy loss, which may limit the benefits of reduced precision.

In order to address these two limitations, we propose a new quantization scheme based on the concept of weighted entropy. Our approach addresses both of the aforementioned limitations while quantizing weights and activations. Our contributions can be summarized as follows:

1. We propose a new multi-bit quantization method for both weights and activations. Unlike binary quantization approaches, our scheme is able to produce quantization results for any number of bits per weight/activation, thereby realizing much more flexibility for exploiting accuracy and performance trade-off.
2. Our scheme facilitates automated quantization of the entire neural network. It does not require any modifications to the network except for activation quantization, and thus, it can be easily integrated into conventional training algorithms for neural networks.
3. We demonstrate the effectiveness of our method based on various practical neural network designs, including AlexNet [35], GoogLeNet [1], ResNet50/101 [36], R-FCN [37], and an LSTM for language model-

ing [38].

3.2 Motivation

Recent studies have shown that most of the weights in convolutional or fully-connected layers are concentrated near zero, resulting in a bell-shaped distribution [23]. The distribution of activation values are similar, except that activation values are always non-negative due to a ReLU layer. Existing quantization schemes are based on such characteristics to judiciously assign quantization levels. For example, logarithm-based quantization (or LogQuant) exploits denser distribution of weights near zero by assigning more quantization levels to near-zero values.

In addition to the distribution of weight/activation values, we make a key observation that *the impact of each weight/activation value on the final result should also be considered during the quantization*. Since the objective of a quantization method is to minimize the accuracy degradation with the fewest quantization levels, taking the actual impact of quantizing each value into account allows us to develop a new scheme that uses each quantization level more effectively. More specifically, our insight can be summarized as follows:

1. **Near-zero values** dominate the total frequency of values in both weight and activation distribution; however, their impact on the output is small (e.g., errors in a very small weight may not affect much to the result of convolution). Thus, it is desirable to assign fewer quantization levels (in

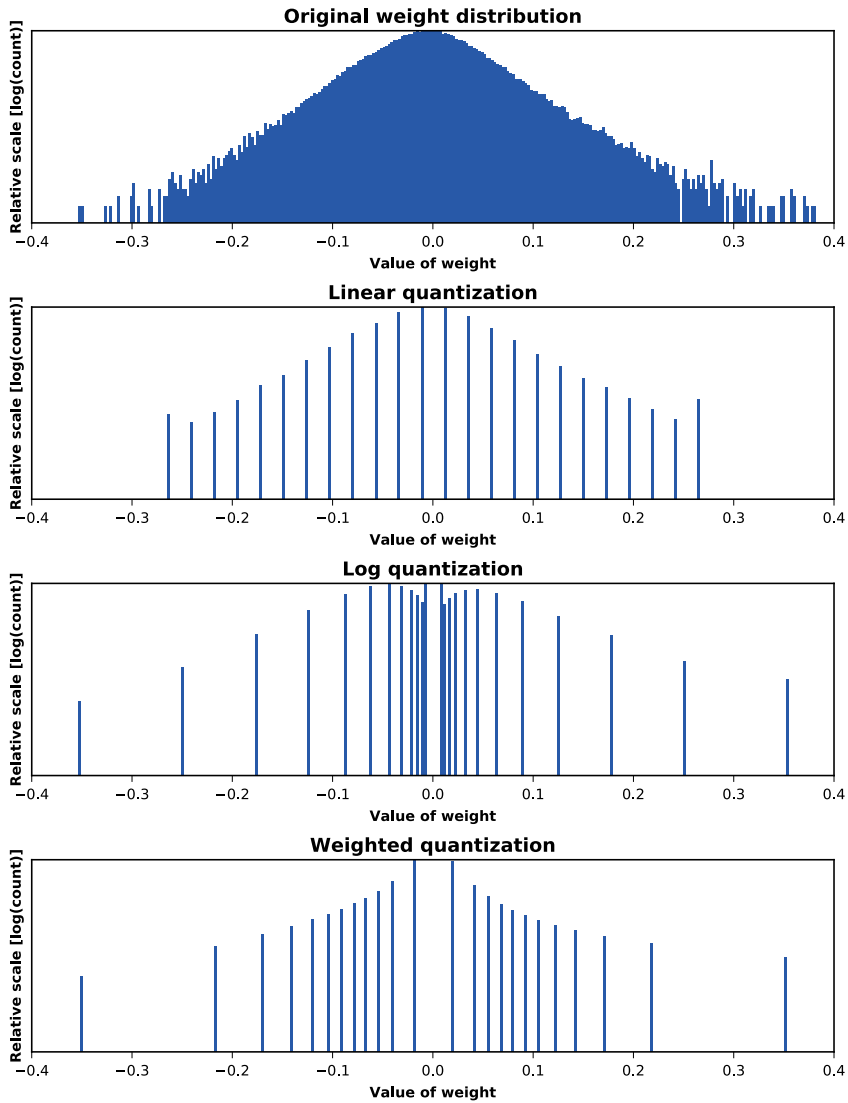


Figure 3.1 Comparison of various quantization schemes. The weights are extracted from the second 3×3 convolution layer of GoogLeNet [1]. Each quantization scheme is given to assign 24 levels. We use $2^{0.5}$ as the base of LogQuant and optimize both linear quantization and LogQuant towards minimizing the L2 norm of overall activations.

short, *levels* throughout this work) to near-zero values than in a typical linear or logarithm-based quantization.

2. **Large weights and activations** have significant impact on the quality of output, but they are infrequent. Thus, it is also desirable to assign a small number of levels to those values in order to maximize the utility of each quantization level.

3. **Values that do not belong to neither of the two aforementioned categories** have a relatively large number of population with noticeable impacts on the output quality. Thus, it makes sense to assign more levels to those values than in conventional quantization methods.

Figure 3.1 illustrates how existing and proposed methods assign levels to the given weight distribution. While the linear quantization does not consider the weight distribution at all and LogQuant assigns too many levels to near-zero values, our approach shows distribution that is more concentrated on the values that are neither too small nor too large. Through quantitative evaluations, we will show later that this style of quantization achieves higher efficiency than conventional schemes.

3.3 Quantization

based on Weighted Entropy

3.3.1 Weight Quantization

The high-level idea of our weight quantization approach is to group weights into N clusters in a way to have more clusters for important ranges of weights, assign a representative value to each cluster, and quantize all weights in each cluster into the representative value of the cluster. For this purpose, we have to be able to evaluate the clustering quality and find a set of clusters optimizing such quality metric.

As the first step, we define a quantitative metric for evaluating the importance of a single weight (or the impact of a weight on output quality). Since larger weights have a higher impact on the output quality, we empirically define the importance $i_{(n,m)}$ of m -th weight in n -th cluster, i.e., $w_{(n,m)}$ to be quadratically proportional to the magnitude of the weight, i.e., $i_{(n,m)} = w_{(n,m)}^2$.

Based on this importance value of each weight, we derive a metric for evaluating the quality of a clustering result (i.e., quantization result) based on *weighted entropy* [39, 40]. Weighted entropy is originated from the concept of entropy in physics and is designed to take the importance of data into account. For a set of clusters C_0, \dots, C_{N-1} , weighted entropy S is defined as

$$S = - \sum_n I_n P_n \log P_n \quad (3.1)$$

where

$$P_n = \frac{|C_n|}{\sum_k |C_k|} \quad (\text{relative frequency}) \quad (3.2)$$

$$I_n = \frac{\sum_m i_{(n,m)}}{|C_n|} \quad (\text{representative importance}) \quad (3.3)$$

In this equation, P_n represents how many weights are in the range of values for cluster C_n , while I_n is the average importance of all weights in cluster C_n . Roughly speaking, clusters for large weights will generally have high I_n but low P_n (i.e., high importance but low frequency), while clusters for small weights will have high P_n but low I_n (i.e., high frequency but low importance). According to our experiments, finding a clustering result that maximizes S yields quantization whose levels are assigned sparsely for too small or too large values, just as we showed in Figure 3.1. Therefore, we define our weight quantization problem as follows:

Problem 1 (Weight Quantization). *Given the training data (i.e., mini-batch input) and the desired $\log N$ -bit precision (i.e., the number of clusters N), our method aims at finding N weight clusters that maximize the weighted entropy. The representative value of a cluster corresponds to a level in the weight quantization.*

Our solution to this problem is shown in Algorithm 1. Note that the algorithm shows the weighted quantization for non-negative weights only. This is because, due to the limitation of the weighted entropy theory, we cannot obtain a clustering result that has both negative and non-negative representative val-

Algorithm 1 Weight Quantization

```
1: function OPTSEARCH( $N, w$ )
2:   for  $k = 0$  to  $N_w - 1$  do
3:      $i_k \leftarrow f_i(w_k)$ 
4:    $s \leftarrow \text{sort}([i_0, \dots, i_{N_w-1}])$ 
5:    $c_0, \dots, c_N \leftarrow$  initial cluster boundary
6:   while  $S$  is increased do
7:     for  $k = 1$  to  $N - 1$  do
8:       for  $c'_k \in [c_{k-1}, c_{k+1}]$  do
9:          $S' \leftarrow S$  with  $c_0, \dots, c'_k, \dots, c_N$ 
10:        if  $S' > S$  then
11:           $c_k \leftarrow c'_k$ 
12:     for  $k = 0$  to  $N - 1$  do
13:        $I_k \leftarrow \sum_{i=c_k}^{c_{k+1}-1} s[i] / (c_{k+1} - c_k)$ 
14:        $r_k \leftarrow f_i^{-1}(I_k)$ 
15:        $b_k \leftarrow f_i^{-1}(s[c_k])$ 
16:      $b_N \leftarrow \infty$ 
17:     return  $[r_0 : r_{N-1}], [b_0 : b_N]$ 
18: function QUANTIZE( $w_n, [r_0 : r_{N-1}], [b_0 : b_N]$ )
19:   return  $r_k$  for  $k$  s.t.  $b_k \leq w_n < b_{k+1}$ 
```

- N : The number of levels
 - N_w : The number of weights
 - w_n : Value of n -th weight
 - i_n : Importance of n -th weight
 - f_i : Importance mapping function
 - c_i : Cluster boundary index
 - S : Overall weighted entropy
-

ues. Thus, we separate the weights into two negative and non-negative groups, and apply our algorithm to each group with $N/2$ levels each.

At the beginning of the algorithm, we calculate the importance of each weight (lines 2 and 3). This is done by an importance mapping function f_i , which calculates the importance i_k from weight w_k . In this work, we empirically choose a square function $f_i(w) = w^2$ to compute the importance of each weight. After obtaining the importance values of all weights, they are sorted in the increasing order of their magnitude (line 4).

Based on the sorted importance values, the algorithm initializes cluster boundary indexes c_0 to c_N ¹ (line 5) such that (1) each cluster has the same number of weights and (2) weights in C_{i+1} have higher importance than weights in C_i . This is achieved simply by partitioning the sorted array s into N pieces and assign each piece to each cluster. For example, if $s = [1, 2, 3, 4]$ and $N = 2$, we set $c_0 = 0$, $c_1 = 2$, and $c_2 = 4$ so that $C_0 = \{1, 2\}$ and $C_1 = \{3, 4\}$.

Starting from the initial cluster boundaries, we iteratively perform incremental search on the new cluster boundaries (lines 6–11). At each iteration, for each cluster C_i and its boundaries c_i and c_{i+1} , we sweep c_i from c_{i-1} to c_{i+1} by using bisection method. For each cluster boundary candidate c'_i , we recalculate the weighted entropy of cluster C_{i-1} and C_i , which are the only ones affected by the new boundary, and update the boundary to c'_i only if the new overall weighted entropy S' is higher than the current one.

¹Cluster boundary indexes determine which weights belong to which clusters. Precisely, cluster C_i is defined as containing c_i -th weight to $(c_{i+1} - 1)$ -th weight (zero-based indexing) in array s .

After obtaining the new cluster boundaries, we calculate the representative importance I_k of each cluster C_k (line 13). We obtain the representative weight value r_k for cluster C_k (line 14). In order to identify which weights belong to which cluster, weight values at cluster boundaries, b_k , are identified as well for weight quantization (line 15), i.e., cluster C_i contains weights w that satisfy $b_k \leq w < b_{k+1}$. Function QUANTIZE implements this quantization method. That is, given a weight w_n , it produces the representative weight value r_k of the associated cluster c_k .

The weighted-entropy-based clustering can provide levels that satisfy our requirements on quantization in Section 3.2. Maximizing the weighted entropy optimizes the quantization result towards maximizing entropy while considering the importance of data. Thus, our method groups many near-zero values into a large cluster by considering their lower importance. Large, but infrequent values are also grouped into a cluster that covers a wide range of weight values.

3.3.2 Activation Quantization

Activation quantization needs a different approach from weight quantization. While weights are fixed after the training, activations change at inference time according to the input data. This makes activations less suitable to be quantized by clustering-based approaches, which require a stable distribution of values.

According to our investigation, logarithm-based quantization (LogQuant) can be effective for activation quantization. LogQuant is also beneficial to minimize the cost of implementation (e.g., dedicated hardware accelerators) as it

can transform multiplications into inexpensive bitwise shift operations (i.e., $w \times 2^x = w \ll x$). However, the original LogQuant method does not provide an effective search strategy for exploring the best LogQuant parameters (i.e., base and offset) for each layer of the network.

Our approach to activation quantization consists of two parts: a modified version of LogQuant and a fast search strategy for LogQuant parameters. Algorithm 2 shows key functions used in our modified LogQuant method.

First, we modify the the original LogQuant method to improve overall accuracy and stability. Unlike the conventional LogQuant, we adopt smaller log bases ($1/8$ and its multiples) and offsets ($1/16$ and its multiples), which correspond to ‘step’ and ‘fsr’ in Algorithm 2, respectively. We assign the first quantization level to zero activation and the other levels to the corresponding log scale. For example, when we perform 3-bit quantization of activations, the first level is assigned to value 0, the second one to $2^{\frac{\text{fsr}}{16}}$, the third one to $2^{\frac{\text{fsr}+\text{step}}{16}}$, and so on. For simplicity, we integrate our activation quantization as part of the rectified linear unit (ReLU) activation function, which is described as Function WEIGHTEDLOGQUANTRELU in Algorithm 2.

Second, we propose a novel parameter search method for our LogQuant variant, which determines the base and the offset in a way to minimize the loss of output quality. Our idea is to take advantage of the concept of weighted entropy maximization in our weight quantization. Algorithm 2 shows functions that calculate the representative importance I (REPRIMPORTANCE) and the relative frequency P (RELATIVEFREQUENCY), which are the two ingredients for

Algorithm 2 Activation Quantization

```
function BINARYTOLOGQUANT( $a_n$ )
    return round( $\frac{16 \times \log_2 a_n - \text{fsr}}{\text{step}}$ ) + 1

function LOGQUANTTOBINARY(index)
    if index = 0 then
        return 0
    else
        return  $2^{\frac{1}{16} \times (\text{fsr} + \text{step} \cdot (\text{index} - 1))}$ 

function WEIGHTEDLOGQUANTRELU( $a_n$ )
    if  $a_n < 0$  then
        return 0
    level_idx  $\leftarrow$  BINARYTOLOGQUANT( $a_n$ )
    if level_idx  $\leq 0$  then
        return 0
    else if level_idx  $\geq N - 1$  then
        return LOGQUANTTOBINARY( $N - 1$ )
    else
        return LOGQUANTTOBINARY(level_idx)

function REPRIMPORTANCE(index)
    return LOGQUANTTOBINARY(index)

function RELATIVEFREQUENCY(index,  $a$ )
    for  $k = 0$  to  $N_a - 1$  do
        level_idx $_k$   $\leftarrow$  BINARYTOLOGQUANT( $a_n$ )
    if index = 0 then
        return  $|\{a_n \mid \text{level\_idx}_n \leq 0\}|$ 
    else if index =  $N - 1$  then
        return  $|\{a_n \mid \text{level\_idx}_n \geq N - 1\}|$ 
    else
        return  $|\{a_n \mid \text{level\_idx}_n = \text{index}\}|$ 
```

- N : The number of levels
 - N_a : Total number of activations
 - a_n : Value of n -th activation
 - fsr: Optimal fsr value (integer)
 - step: Optimal step value (a multiple of 2)
-

computing the weighted entropy. During training, in order to maximize the weighted entropy of the given per-layer activations under LogQuant, we apply an exhaustive search for ‘fsr’ and ‘step’ since the numbers of possible bases and offsets are usually small (e.g., 16 for bases and around 500 for offsets in our experiments).

3.3.3 Integrating Weight/Activation Quantization into the Training Algorithm

We integrate the proposed weight/activation quantization into the conventional training algorithm for neural networks. Since weights do not change during each mini-batch, weight quantization can be simply applied by quantizing the weights at the end of each mini-batch after the weight update. Note that we use full-precision weights during the weight update as in other previous work [9, 10].

On the other hand, activation quantization has to be applied to every forward/backward pass as each pass has its own set of activations. For each layer, we first perform the forward pass and apply the ordinary ReLU (without LogQuant). The resulting activations are fed into our algorithm for LogQuant parameter search. The best base/offset combination from the algorithm is then used to quantize the activations by using WEIGHTEDLOGQUANTRELU. The quantized activations are passed to the next layer to perform the same process for the rest of the layers in the network.

Under our training framework, any network can automatically benefit from

our quantization schemes without modifications to the network. This makes it much easier to apply aggressive quantization to the entire neural network, which contributes to greatly reducing the inference cost of the network. Existing approaches are less practical in this regard, considering that they require network modification and/or significant manual effort at design time.

3.4 Experiment

We evaluate our approach in three representative domains of neural network applications: image classification, object detection, and language modeling. We modify Caffe [41] to implement our technique on top of all networks², except for language modeling, in which we use TensorFlow [42] to implement an LSTM. We constrain the accuracy loss to 1% and aim at finding the quantization configuration that gives the minimum bitwidth while satisfying the accuracy constraint. For brevity, we introduce a notation (x,y) to represent the bitwidth of weights x and that of activations y in a quantization configuration. In this notation, ‘f’ represents full precision. For example, $(1,f)$ indicates 1-bit weights and full-precision activations.

3.4.1 Image Classification: AlexNet, GoogLeNet and ResNet-50/101

For image classification tasks, we evaluate the proposed method by quantizing two widely used CNNs for ImageNet tasks [43]: AlexNet [35], GoogLeNet [1]

²Modified caffe code is available at https://github.com/EunhyeokPark/script_for_WQ

(both from Caffe framework [41]), and ResNet³ [36]. In order to apply our quantization scheme into these networks, we perform fine-tuning combined with our weight/activation quantization schemes under the batch size of 256 (for AlexNet), 64 (for GoogLeNet), or 16 (for ResNet-50/101). In the cases of GoogLeNet and ResNet, the batch size is limited due to insufficient GPU memory capacity; this may increase overall accuracy loss. We use ILSVRC2012 data set, which contains 1.28M images for training and 50K images for testing. During the six epochs of fine-tuning, we first set the initial learning rate to 0.001 and decrease it by 10 times every two epochs.

In the following subsections, we present two styles of evaluation results. First, we demonstrate the effectiveness of our approach by quantizing the entire networks (whole network quantization), which was not possible in prior work. Second, we apply our scheme to all layers except the first and the last one (partial network comparison) and compare ours against previous quantization approaches that use the full precision at the first/last layer of a network.

Whole Network Quantization

Figure 3.2 compares the test accuracy of CNNs quantized by our techniques. As shown in the figure, the quantized CNNs achieve higher accuracy under less restrictive bitwidth constraint.

For AlexNet, the best quantization configurations that use the fewest bits while satisfying the 1% top-5 accuracy loss constraint are (3,6), (4,4), (4,5) and

³Base models are available at <https://github.com/KaimingHe/deep-residual-net-works>

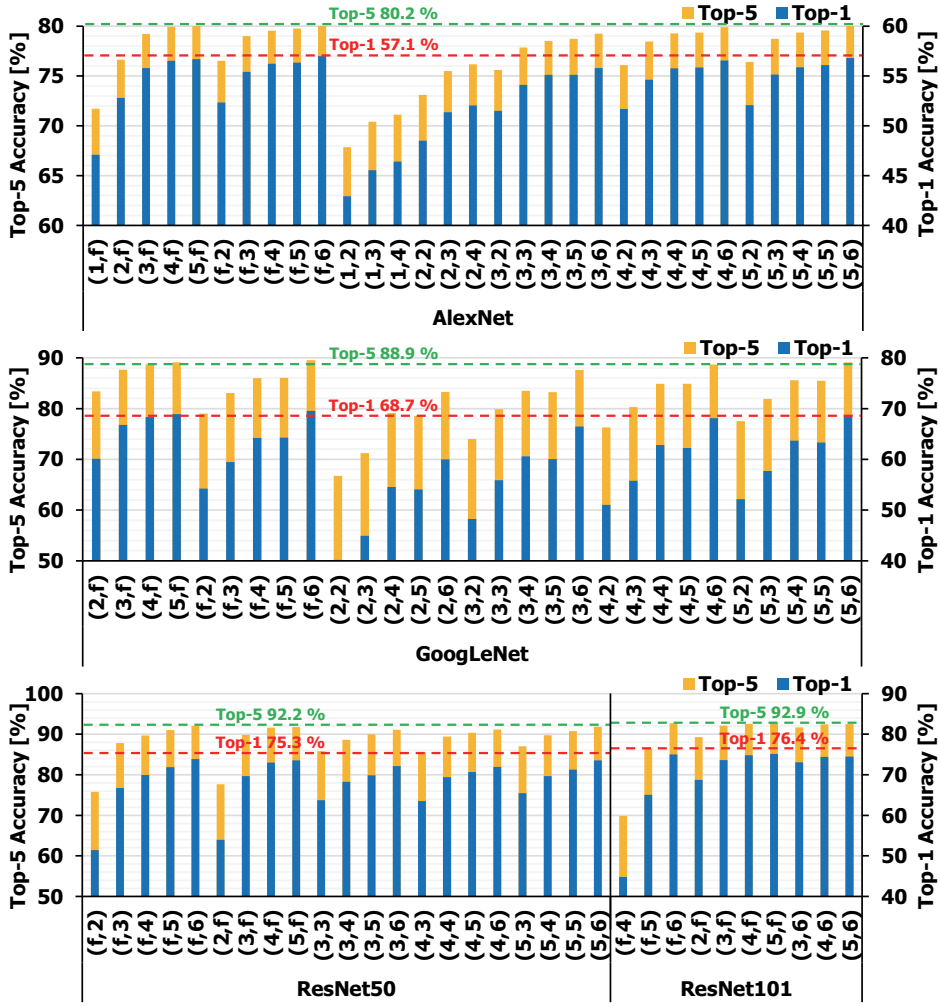


Figure 3.2 Top-1 and top-5 accuracy of quantized CNNs after fine-tuning. The dashed lines represent the accuracy of the baseline networks, which use full-precision arithmetic.

(4,6). For example, (4,4) reduces the bitwidths of both weights and activations by 87.5% ($= 1 - 4/32$) with less than 1% loss of top-5 accuracy. Moreover, our approach provides much lower iso-accuracy bitwidth compared to previous work. For example, LogQuant [21] achieves 75.1% top-5 accuracy with 4-bit weights and 5-bit activations; Qiu et al. [44] used 8-bit weights and activations and showed 76.6% (53.0%) of top-5 (top-1) accuracy. Our approach achieves a similar level of top-5/top-1 accuracy (i.e., 75.49%/51.37%) with only 2-bit weights and 3-bit activations.

For GoogLeNet, under the 1% accuracy loss constraint, our approach can quantize weights and activations down to only 4–5 bits and 6 bits, respectively, as shown in Figure 3.2. We also observe that GoogLeNet suffers more from accuracy loss than AlexNet under the same level of bitwidth constraint. We believe that this is because the model size of GoogLeNet is more compact than AlexNet, yet the former performs more computation than the latter. In other words, GoogLeNet reuses each weight more frequently during the computation than AlexNet, which makes the impact of reduced weight precision more pronounced in GoogLeNet. Even so, our approach still achieves a significant (more than 5x) reduction in both the model size and the amount of computation (in bits) compared to the full-precision implementation.

For ResNet, to the best of our knowledge, this study is the first to report the result of quantizing the entire networks whose depth is as much as 50 and 101 layers. Both networks maintain similar levels of accuracy even after aggressive quantization of weights, e.g., 3 bits. However, we observe that the deeper

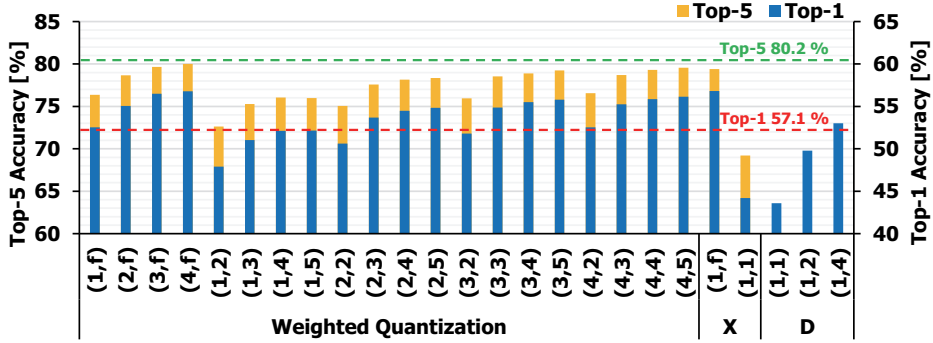


Figure 3.3 Accuracy comparison of quantization methods applied to AlexNet. ‘Weighted Quantization’ represents our approach, while ‘X’ and ‘D’ are for XNOR-Net and DoReFa-Net, respectively. The dashed lines indicate the accuracy of the baseline full-precision network.

network demands more bits for activations, e.g., 6 bits, possibly because quantization errors of activations get accumulated over deeper layers.

Partial Network Quantization

In this subsection, we compare the performance of our quantization method against two state-of-the-art approaches: XNOR-Net [9] and DoReFa-Net [10]. For fair comparison, we apply our quantization scheme to all layers but the first and last ones, just as in our comparison targets. Note that the comparison is still not apple-to-apple in that (1) the (best available) results from previous work are limited to 1-bit weights and k -bit activations, while ours include k -bit weights and activations, and (2) both XNOR-Net and DoReFa-Net modify the network, whereas ours does not except ReLU layers (in activation quantization).

Figure 3.3 shows the comparison of our approach against XNOR-Net and

DoReFa-Net. While XNOR-Net with binary weights, i.e., (1,f), shows very small accuracy drop with 1-bit weights, it is limited to binary quantization and full-precision activation, which is not flexible enough to exploit accuracy-performance trade-off under a stringent accuracy loss constraint. XNOR-Net with binary weight and activation quantization, i.e., (1,1), degrades the accuracy too much, whereas our method provides multi-bit quantization meeting the accuracy constraint. DoReFa-Net alleviates some of such limitations by allowing multi-bit quantization of activations. However, under the similar configurations, our scheme with 2-bit weights and 3-bit activations outperforms DoReFa-Net with 1-bit weights and 4-bit activations by 0.69% in terms of top-1 accuracy. In summary, our method facilitates more flexible choice of quantization configurations with smaller iso-accuracy bitwidth than previous work, which is extremely useful for systems that require efficient inference under a tight accuracy constraint.

Compression Analysis

Table 3.1 compares existing methods and ours in the context of compression. From this, we observe the followings.

First, both XNOR-Net [9] and DoReFa-Net [10] show larger weights than our method (WQ) since they do not quantize the first and the last layers. Moreover, Huffman encoding is not helpful since they use binary and full-precision weights, respectively.

Second, when WQ is applied on top of pruning [23], it achieves 5.4x smaller

	Weights			Activations	Top-1
	P [%]	Q [MB]	+H	Q [MB]	[%]
WQ(4,4)	-	30.5	18.1	0.47	55.8
WQ(2,3)	-	15.3	12.5	0.35	53.7
XNOR-Net [9]	-	23.7	-	0.72	44.2
DoReFa-Net [10]	-	23.6	-	0.47	53.0
Deep Compression [23]	11	8.9	6.9	3.75	57.2
[23] + WQ(4,6)	11	8.3	6.5	0.70	56.3

Table 3.1 Memory requirement comparison with AlexNet (P: Pruning ratio, Q: Quantization, H: Huffman encoding).

activations and slightly smaller weights (8.9 MB vs. 8.3 MB) at an additional accuracy loss of 0.9 %. Ours achieves larger bitwidth reductions in activations than in weights because [23] utilizes full-precision activations while ours uses 6-bit activations.

Layer-wise Quantization: A Feasibility Study

In the previous subsections, we use the same bitwidth constraint for all layers in the network. However, according to our observation, different layers have different levels of sensitivity to the quantization bitwidth. Thus, we perform a feasibility study of the potential of *layer-wise* quantization, where different layers may have different bitwidths. In this study, we evaluate the following four styles of per-layer bitwidth assignment based on AlexNet: monotonically decreasing (DEC), monotonically increasing (INC), concave (CONCAVE), and convex (CONVEX). All four schemes are designed to have the same number of bitwidth in total. For example, DEC assigns 6 bits to each weight/activation in

the first convolution layer, while it uses only 2 bits for weights/activations in the last fully-connected layer.

	DEC	INC	CONCAVE	CONVEX
Top-1 [%]	53.79	50.35	54.45	54.33
Top-5 [%]	77.59	74.89	76.43	78.20

Table 3.2 Accuracy comparison of our approach under different styles of layer-wise quantization.

As shown in Table 3.2, we observe that using less bits in intermediate layers (i.e., CONVEX) achieves the highest accuracy, while assigning fewer bits to near-input layers (i.e., INC) shows the lowest. A similar phenomenon to this was observed by Zhou et al. [10]. We believe that even more aggressive bitwidth optimization could be possible by taking this layer-wise sensitivity to bitwidths into account during quantization. Exhaustive search of all possible combinations of bitwidths is impractical as there are too many of them even for small networks (e.g., AlexNet has $5^{15} \approx 3 \times 10^{10}$ possible configurations that use two to six bits for each layer). Algorithms for fast design space exploration of layer-wise quantization are left for future work.

3.4.2 Object Detection: R-FCN with ResNet-50

In order to evaluate the effectiveness of our quantization method on more complex vision tasks, we use a state-of-the-art 50-layer R-FCN model for object detection [37]. The R-FCN model combines a residual network (ResNet) [36] (for representation) and Faster R-CNN [45] (for region proposal, object classi-

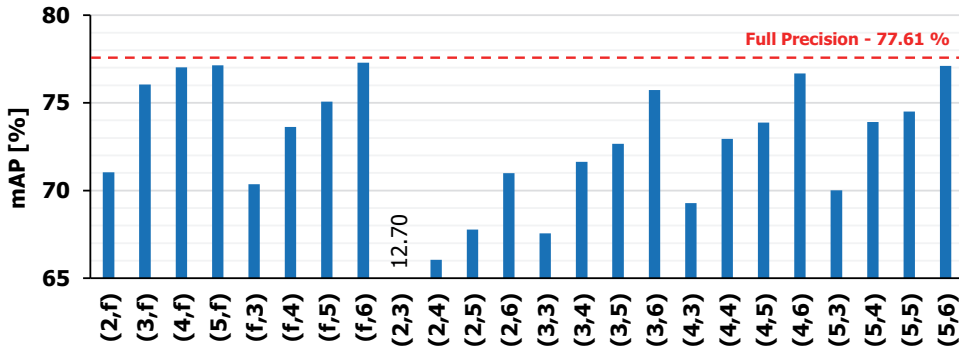


Figure 3.4 mAP results of R-FCN. The dashed line represents the accuracy of the baseline full-precision network.

fication, and box localization). On top of the existing full-precision model, we perform fine-tuning with our quantization method.

Even though deep models are known to be difficult to quantize since quantization errors are accumulated over deep layers, our method successfully quantizes the 50-layer model for object detection with very small accuracy loss. Figure 3.4 shows that the configuration of 5-bit weights and 6-bit activations loses only 0.51% of mAP, while reducing the model size and the amount of computation by more than 5x. We also observe that activations typically require more bits than weights in our quantization method (e.g., 6-bit activations or 4-/5-bit weights are needed for stable and satisfactory results, as shown in Figure 3.4). We believe that this is because the bounding box regression mechanism of R-FCN is simple (obtaining boxes directly from the region proposal network), and thus, is sensitive to activation accuracy. We will perform further investigation on this in our future work.

In our future work, we will also study the feasibility of our method with deeper models. According to our preliminary study with R-FCN based on ResNet-101, we failed to obtain quantization with reasonable accuracy when fine-tuning the model with the PASCAL VOC data set. We believe that this problem is due to the mixed effects of transfer learning and quantization on a very deep network, which requires further investigation into quantization on the very deep networks.

3.4.3 Language Modeling: An LSTM

In order to evaluate our scheme on recurrent neural networks, we perform a preliminary analysis by applying our method to an LSTM network for language modeling [38], provided along with the Tensorflow framework [42]. We evaluate three sizes of RNNs, small (200 hidden units and 20 time steps), medium (650 hidden units and 35 time steps), and large (1500 hidden units and 35 time steps), all of which have two layers each. We measure the word-level perplexity of these three RNNs before/after quantization with the Penn Tree Bank dataset [46]. We apply only the weight quantization to the LSTM network since our activation quantization is currently incompatible with bounded gates and linear outputs in RNNs.

Table 3.3 compares the word-level perplexity of the LSTM network between full-precision (float) and quantization cases. The result shows that 4-bit weights achieve comparable results to the full-precision implementation. Also, our scheme provides options to further reduce the model size and the amount

	Large		Medium		Small	
	Valid	Test	Valid	Test	Valid	Test
float	82.77	78.63	87.69	83.54	119.19	114.46
1-bit	92.20	88.48	104.0	100.7	147.19	141.07
2-bit	86.73	82.90	92.49	89.24	137.34	131.15
3-bit	85.59	81.57	86.73	83.50	121.21	117.00
4-bit	81.83	78.09	88.01	83.84	121.84	114.95

Table 3.3 Impact of quantization on word-level perplexity of an LSTM for language modeling.

of computation by using fewer bits at a cost of lower output quality (i.e., higher perplexity).

3.5 Conclusion

In this work, we proposed a novel weight/activation quantization method based on the concept of weighted entropy. The key benefits of our approach are twofold: (1) flexible multi-bit quantization, which allows us to optimize the neural network design under the tight accuracy loss constraint and (2) automated quantization, which does not require modifications to the input networks. According to our extensive evaluation results based on practical neural networks including AlexNet, GoogLeNet, ResNet-50/101, R-FCN, and an LSTM, our approach achieves 1% accuracy loss (top-5 or mAP) with 4-bit weights/activations (AlexNet), 4/5-bit weights and 6-bit activations (GoogLeNet, ResNet and R-FCN). Our future work includes investigating the effectiveness of our method on very deep neural network models (e.g., ResNet-152) and

devising activation quantization for RNN models.

Chapter 4

Value-aware Quantization for Training and Inference of Neural Networks

4.1 Introduction

This work was published in ECCV'2018 conference [15]. Reduced precision has potential to resolve the problems of runtime, energy consumption, and memory cost by reducing the data size thereby enabling more parallel and energy-efficient computation, e.g., four int8 operations instead of a single fp32 operation, at a smaller memory footprint. The state-of-the-art techniques of quantization are 16-bit training [47] and 8-bit inference [6]. Considering the trend of ever-increasing demand for training and inference on both servers and edge devices, further optimizations in quantization, e.g., 4 bits, will be more and more required.

In this work, we propose a novel quantization method based on the fact that the distributions of weights and activations have the majority of data concentrated in narrow regions while having a small number of large values scattered

in large regions. By exploiting the fact, we apply reduced precision only to the narrow regions thereby reducing quantization errors for the majority of data while separately handling large values in high precision. For very deep networks such as ResNet-152 and DenseNet-201, our proposed quantization method enables training with 3-bit activations (2% large values). Our method also offers low-precision inference with 4 to 5-bit weights and activations (1% large values) even for optimized networks such as SqueezeNet-1.1 and MobileNet-v2 as well as deeper networks.

4.2 Motivation

Figure 4.1 (a) and (b) illustrate the distributions (y-axis in log scale) of activations and weights in the second convolutional layer of GoogLeNet. As the figures show, both distributions are wide due to a small number of large values. Given a bitwidth for low precision, e.g., 3 bits, the wider the distribution is, the larger quantization errors we obtain. Figure 4.1 (c) exemplifies the conventional 3-bit linear quantization applied to the distribution of activations in Figure 4.1 (a). As the figure shows, the spacing between quantization levels (vertical bars) is large due to the wide distribution, which incurs large quantization errors.

When comparing Figure 4.1 (a) and (c), it is clear that the majority of quantization levels is not fully utilized. Especially, the levels assigned to large values have much fewer data than those assigned to small values, which motivates our idea. Figure 4.1 (d) illustrates our idea. We propose applying low precision only to small values, i.e., the majority of data, not all. As the figure shows, the

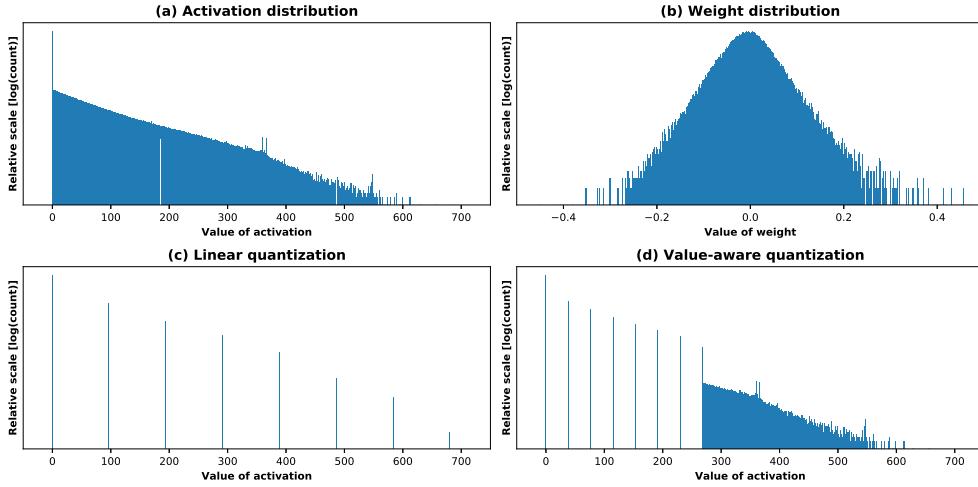


Figure 4.1 Activation and weight distributions of second convolutional layer in GoogLeNet.

spacing between quantization levels gets much smaller than that in the conventional linear quantization in Figure 4.1 (c). Such a small spacing can significantly reduce the quantization error for the majority of data. Large values have the more significant impact on the quality of network output. Thus, we propose handling the remaining large values in high precision, e.g., in 32 or 16 bits. The computation and memory overhead of handling high-precision data is small because their frequency, which is called the ratio of large activations, in short, *activation ratio* (AR), is small, e.g., 1-3% of total activation data.¹

¹We use two ratios of large values, one for large weights and the other for large activations. We use AR to denote the ratio of large activations.

4.3 Proposed Method

Our basic approach is first to perform value profiling to identify large values during training and inference. Then, we apply reduced precision to the majority of data, i.e., small ones while keeping high precision for the large values. We call this method value-aware quantization (V-Quant).

We apply V-Quant to training to reduce the memory cost of activations. We also apply it to inference to reduce the bitwidth of weights and activations of the trained neural network. To do that, we address new problems as follows.

- (Sections 4.3.1 and 4.3.2) To prevent the quality degradation of training results due to quantization, we propose a novel scheme called *quantized activation back-propagation*, in short, quantized back-propagation. We apply our quantization only to the activations used in the backward pass of training and perform forward pass with full-precision activations.
- (Sections 4.3.4 and 4.3.7) Identifying large values requires sorting which is expensive. To avoid the overhead of global communication between GPUs for sorting during training, we propose performing sorting and identifying large values locally on each GPU.
- (Sections 4.3.5 and 4.3.6) We present new methods for further reduction in memory cost of training. To reduce the overhead of mask information required for ReLU function during back-propagation, we propose ReLU and value-aware quantization. For further reduction in memory

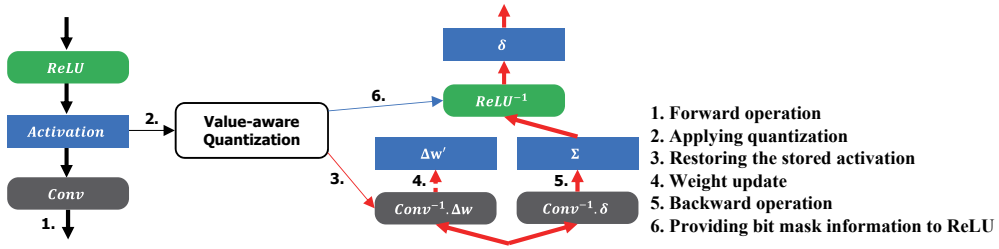


Figure 4.2 Value-aware quantization in training pipeline.

cost, we also propose exploiting the fact that, as training continues, the less amount of large activations is required.

4.3.1 Quantized Back-Propagation

Figure 4.2 shows how to integrate the proposed method with the existing training pipeline. As the figure shows, we add a new component of value-aware quantization to the existing training flow. In the figure, thick arrows represent the flow of full-precision activations (in black) and gradients (in red).

First, we perform the forward pass with full-precision activations and weights, which gives the same loss as that of the existing full-precision forward pass (step 1 in the figure). During the forward pass, after obtaining the output activations of each layer, e.g., layer l , the next layer (layer $l + 1$) of network takes as input the full-precision activations. Then, we apply our quantization method, RV-Quant to them (those of layer l) in order to reduce their size (step 2). As the result of the forward pass, we obtain the loss and the quantized activations.

During the backward pass, when the activations of a layer are required for

weight update, we convert the quantized, mostly low-precision, activations, which are stored in the forward pass, into full-precision ones (step 3). Note that this step only converts the data type from low to high precision, e.g., from 3 to 32 bits. Then, we perform the weight update with back-propagated error (thick red arrow) and the activations (step 4).

Note that there is no modification in the computation of the existing forward and backward passes. Especially, as will be explained in the next subsection, when ReLU is used as activation function, the backward error propagation (step 5 in the figure) keeps full-precision accuracy. The added component of value-aware quantization performs conversions between full-precision and reduced-precision activations and compresses a small number of remaining large high-precision activations, which are sparse, utilizing a conventional sparse data representation, e.g., compressed sparse row (CSR).

The conversion from full to reduced precision (step 2) reduces memory cost while that from reduced to full precision (step 3) changes data type back to full precision one thereby increasing memory cost back to that of full precision. Note that the full-precision activations, obtained from the quantized ones, are discarded after weight update for their associated layer. Thus, we need memory resource for the stored quantized activations of the entire network and the full-precision input/output activations of only one layer, which we call working activations, for the forward/backward computation.

As will be explained later in this section, for further reduction in memory cost, the ReLU function consults the value-aware quantization component for

the mask information which is required to determine to which neuron to back-propagate the error (step 6).

4.3.2 Back-Propagation of Full-Precision Loss

Our proposed method can suffer from quantization error in weight update since we utilize quantized activations. We try to reduce the quantization error by applying reduced precision only to narrow regions, determined by AR, having the majority of data while separately handling the large values in high precision.

Moreover, in state-of-the-art networks where ReLU is utilized as activation function, the back-propagated error is not affected by our quantization of activations as is explained below. Equation (1) shows how we calculate weight update during back-propagation for a multilayer perceptron (MLP).

$$\Delta w_{ji} = \eta \delta_j y_i \quad (4.1)$$

where Δw_{ji} represents the update of weight from neuron i (of layer l) to neuron j (of layer $l + 1$), η learning rate, δ_j the local gradient of neuron j (back-propagated error to this neuron), and y_i the activation of neuron i . Equation (1) shows that the quantization error of activation y_i can affect the weight update. In order to reduce the quantization error in Equation (1), we apply V-Quant to activations y_i .

The local gradient δ_j is calculated as follows.

$$\delta_j = \varphi'(v_j) \cdot (\sum_k \delta_k w_{kj}) \quad (4.2)$$

where $\phi'()$ represents the derivative of activation function, v_j the input to neuron j and w_{kj} the weight between neuron j (of layer $l + 1$) to neuron k (of layer $l + 2$). Equation (2) shows that the local gradient is a function of the input to neuron, v_j which is the weighted sum of activations. However, if ReLU is used as the activation function, then $\phi'()$ becomes 1 yielding $\delta_j = \phi'(v_j) \cdot (\sum_k \delta_k w_{kj}) = \sum_k \delta_k w_{kj}$, which means *the local gradient becomes independent of activations*. Thus, aggressive quantizations of intermediate activations, e.g., 3-bit activations can hurt only the weight update in Equation (1), not the local gradient in Equation (2). This is the main reason why our proposed method can offer full-precision training accuracy even under aggressive quantization of intermediate activations as will be shown in the experiments.

4.3.3 Potential of Further Reduction in Computation Cost

Compared with the existing methods of low memory cost in training [48, 49], our proposed method reduces computation cost by avoiding re-computation during back-propagation. More importantly, our proposed method has a potential for further reduction in computation cost especially in Equation (1), though we have not yet realized the speedup potential in this study. It is because the activation y_i is mostly in low precision in our method. Thus, utilizing the capability of 8-bit multiplication on GPUs, our method can transform a single 16-bit x 16-bit multiplication in Equation (1) into an 8-bit x 16-bit multiplication. In state-of-the-art GPUs, we can perform two 8-bit x 16-bit multiplications at

the same computation cost, i.e., execution cycle, of one 16-bit x 16-bit multiplication, which means our proposed method can double the performance of Equation (1) on the existing GPUs.

Assuming that the forward pass takes M multiplications, the backward pass takes $2M$ multiplications while each of Equations (1) and (2) taking M multiplications, respectively. Thus, the 2x improvement in computation cost of Equation (1) can reduce by up to 1/6 total computation cost of training. In order to realize the potential, further study is needed to prove that our proposed method enables 8-bit low-precision activations (with a small number of 16-bit high-precision activations) without losing the accuracy of 16-bit training [47].

Although our method can currently reduce computation cost utilizing only 8-bit multiplications on GPUs, its reduced-precision computation, e.g., 3-bit multiplications, offers opportunities of further reduction in computation cost for training in future hardware platforms supporting aggressively low precision, e.g., [50].

4.3.4 Local Sorting in Data Parallel Training

V-Quant requires sorting activations. Assuming that we adopt data parallelism in multi-GPU training, the sorting can incur significant overhead in training runtime since it requires exchanging the activations of each layer between GPUs. What is worse, in reality, such a communication is not easily supported in some training environments, e.g., PyTorch, and it restrict the scalability in any training environments including TensorFlow , Caffe2 and others as well as

PyTorch. In order to address the problem of activation exchange, we propose performing sorting locally on each GPU, which eliminates inter-GPU communication for activation exchange. Then, each GPU performs V-Quant locally by applying the same AR, i.e., the same ratio of large activations. Compared with the global solution that collects all the activations and applies the AR to the global distribution of activations, the proposed local solution can lose accuracy in selecting large values. However, our experiments show that the proposed method of local sorting works well, which means that the selection of large values does not need to be accurate.

4.3.5 ReLU and Value-aware Quantization (RV-Quant)

The error is back-propagated through the neurons the output activations of which are non-zero. The zero activations result from quantization (called *quantization-induced zero*) as well as ReLU activation function. In order to realize the same back-propagation as the full-precision training, it is required to back-propagate errors in the case of quantization-induced zero. To identify the neurons having quantization-induced zero, we would need a bit mask, i.e., 1-bit memory cost for a neuron. In case that the activations are quantized at a minimal number of bits, e.g., 3 bits, the overhead of the mask bit is significant, e.g., one additional bit for 3-bit activation on each neuron. To reduce the overhead of the mask bit, we exploit the fact that the mask bit is needed only for the case of zero activation. We allocate two states to represent two different types of zero values, i.e., original zero and quantization-induced zero. Thus, given K

bits for low precision, we allocate two of 2^K quantization levels to the zero values while representing the positive activation values with $2^K - 2$ levels. We call this quantization ReLU and value-aware quantization (RV-Quant). As will be shown in the experiments, RV-Quant removes the overhead of mask bit while keeping training accuracy.

4.3.6 Activation Annealing

According to our investigation, the required amount of large activations varies across training phases. To be specific, the early stage of training tends to require more large activations while the later stage tends to need less large activations. We propose exploiting the fact and adjusting AR in a gradual manner from large to small AR across training phases, which we call *activation annealing*. As will be shown in the experiments, activation annealing can maintain training quality while reducing the average memory cost across the entire training phases.

4.3.7 Quantized Inference

In order to obtain quantized neural networks for inference, we perform V-Quant as a post-processing of training, i.e., we apply V-Quant to the weights and activations of trained networks. To recover from the accuracy loss due to quantization, we perform fine-tuning as follows. We perform forward pass while utilizing the quantized network, i.e., applying V-Quant to weights and activations. During back-propagation, we update full-precision weights. As will be

shown in the experiments, the fine-tuning incurs a minimal overhead in training time, i.e., only a few additional epochs of training. Note that we apply local sorting in Section 4.3.4 to avoid communication overhead when multiple GPUs are utilized in fine-tuning.

During fine-tuning, we evaluate candidate ratios for large weights and activations and, among those candidates, select the best configuration which minimizes the bitwidth while meeting accuracy requirements. Note that, as will be explained in the experiments, the total number of candidate combinations is small.

In order to identify large activations meeting the AR, we need to sort activations, which can be expensive in inference. In order to avoid the sorting overhead, we need low-cost sorting solutions, e.g., sampling activations to obtain an approximate distribution of activations. Detailed implementations of quantized models including the low-cost sorting are beyond the scope of this work and left for further study.

4.4 Experiments

We evaluate our proposed method on ImageNet classification networks, AlexNet, VGG-16, SqueezeNet-1.1, MobileNet-v2, Inception-v3, ResNet-18/50/101/152 and DenseNet-121/201. We test the trained/quantized networks with ILSVRC2012 validation set (50k images) utilizing a single center crop of 256x256 resized image. We also use an LSTM for word-level language modeling [38, 51, 52]. We implemented our method on PyTorch framework [53] and

use the training data at Torchvision [54].

The initial learning rate is set to 0.1 (ResNet-18/50/152 and DenseNet-201), or 0.01 (AlexNet and VGG-16). The learning rate is decreased by 10x at every multiple of 30 epochs and the training stops at 90 epochs. In SqueezeNet-1.1, MobileNet-v2, and Inception-v3, we use the same parameters in the papers except that we use a mini-batch of 256 and SGD instead of RMSprop. In addition, we replace ReLU6 in MobileNet-v2 with ReLU to apply V-Quant.

We apply V-Quant and RV-Quant to training to minimize memory cost. During training, in order to compress the sparse large activations on GPU, we use the existing work in [55]. To obtain quantized networks for inference, we perform fine-tuning with V-Quant for a small number of additional epochs, e.g., 1-3 epochs after total 90 epochs of original training. All networks are initialized in the same condition.

We compare classification accuracy between full-precision models and those under RV-Quant (training) and V-Quant (training/inference). For each network, we use the same randomly initialized condition and perform training for different RV-Quant and V-Quant configurations.

4.4.1 Training Results

Table 4.1 shows top-1/top-5 accuracy of ResNet-50 obtained, under V-Quant, varying the bitwidth of low-precision activation and the ratio of large activation, AR. The table shows that the configuration of 3-bit activations with the AR of 2% (in bold) gives training results equivalent to the full-precision (32-

AR [%]	0	1	2	3	4	5
1-bit	5.30 / 15.23	74.51 / 92.05	75.17 / 92.50	75.21 / 92.48	75.70 / 92.66	75.57 / 92.66
2-bit	65.75 / 86.72	75.65 / 92.66	75.64 / 92.70	75.66 / 92.51	75.34 / 92.66	75.58 / 92.62
3-bit	75.49 / 92.61	75.71 / 92.59	75.92 / 92.86	75.93 / 92.96	75.89 / 92.94	75.73 / 92.63
4-bit	75.70 / 92.75	75.78 / 92.67	75.88 / 92.93	75.79 / 92.71	75.85 / 92.69	75.92 / 92.86
5-bit with AR 0 %		75.60 / 92.61	6-bit with AR 0 %		75.92 / 92.83	
7-bit with AR 0 %		75.89 / 92.79	8-bit with AR 0 %		75.67 / 92.85	

Table 4.1 Top-1/top-5 accuracy [%] of ResNet-50 with various bitwidth & AR configurations. The full precision network gives the accuracy of 75.92 / 92.90%.

bit) training in terms of top-1 accuracy, which corresponds to 6.1X ($=1/((3+1)/32 + 0.04)$) reduction in the memory cost of stored activation at the same quality of training.² The table also shows that a very aggressive quantization of 2-bit activation and 1% AR loses only 0.27%/0.24% in top-1/top-5 accuracy, which is comparable to the case of 5-bit quantization without large values (5-bit with AR 0% in the table).

Note that the total memory cost of activations includes that of stored activations of the entire network and that of full-precision working activations (input to the associated layer) required for weight update. Thus, the above-mentioned reduction of 6.1X is only for the memory cost of stored activations. We will give the comparison of the total memory cost of activations later in this section. In addition, we do not compare the accuracy of the state-of-the-art method [48] since it provides the same accuracy as full-precision training. However, we provide the memory consumption of this method later in this sec-

²Note that V-Quant still requires 1-bit mask information for each neuron. In addition, the sparse data representation of large values, e.g., CSR doubles the size of the original sparse data yielding the memory cost of 4% with the AR of 2%.

tion, including that of the conventional linear quantization method (8-bit with AR 0%).

AR [%]	0	1	2	3	4	5
2-bit	35.52 / 60.86	75.34 / 92.56	75.41 / 92.49	75.67 / 92.59	75.50 / 92.46	75.27 / 92.65
3-bit	75.16 / 92.55	75.88 / 92.80	75.93 / 92.70	75.66 / 92.74	75.91 / 92.75	75.49 / 92.58

Table 4.2 Top-1/top-5 accuracy [%] of ResNet-50 under RV-Quant. The full-precision network gives the accuracy of 75.92 / 92.90%.

Table 4.2 shows top-1/top-5 accuracy of ResNet-50 under RV-Quant. As the table shows, RV-Quant gives similar results to V-Quant, e.g., top-1 accuracy of 3-bit 2% RV-Quant gives an equivalent result to full precision. Compared with V-Quant, RV-Quant reduces the memory cost by 1 bit per neuron. Thus, the configuration of 3-bit 2% RV-Quant gives 7.5X ($=1/(3/32 + 0.04)$) reduction in the memory cost of stored activations. In addition, we can further reduce the memory cost of stored activations by applying traditional compression techniques to the reduced-precision activations. In the case of 3-bit 2% RV-Quant for ResNet-50, by applying Lempel-Ziv compression, we can further reduce the memory cost of the 3-bit data by 24.4%, which corresponds to 9.0x reduction in the memory cost of the whole stored activations.

Table 4.3 compares the accuracy of neural networks under full-precision training and two RV-Quant configurations. As the table shows, 3-bit 2% RV-Quant gives almost the same training accuracy as full-precision training for all the networks.

Table 4.4 compares the total memory cost of activations (both stored quan-

	AlexNet	ResNet-18	SqueezeNet-1.1	MobileNet-v2
Full	56.35 / 79.02	69.91 / 89.38	58.67 / 81.05	70.10 / 89.74
3-bit 2%	56.14 / 78.99	69.92 / 89.23	58.53 / 80.94	70.12 / 89.76
8-bit 0%	56.24 / 78.95	70.01 / 89.28	58.75 / 81.29	70.29 / 89.64
	VGG-16	Inception-v3	ResNet-152	DenseNet-201
Full	71.86 / 90.48	74.19 / 91.92	77.95 / 94.02	77.42 / 93.59
3-bit 2%	71.74 / 90.46	74.14 / 91.92	77.76 / 93.89	77.28 / 93.44
8-bit 0%	71.77 / 90.66	74.22 / 92.08	78.35 / 93.95	77.32 / 93.51

Table 4.3 Training results. Full means the results of conventional full-precision training, while 3-bit 2% and 8-bit 0% correspond to RV-Quant. The full-precision network gives the accuracy of 75.92 / 92.90%.

tized and full-precision working activations) in training with 256 mini-batch sizes. We compare two existing methods and three RV-Quant configurations. ‘Full’ represents the memory cost of conventional training with full-precision activation. As a baseline, we use the checkpointing method of Chen et al. [48] since it is superior to others including [49], especially for deep neural networks. We calculate the memory cost of the checkpointing method to account for the minimum amount of intermediate activations to re-compute correct activations while having the memory cost of $O(\sqrt{N})$ where N is the number of layers [48].

The table shows that, compared with the checkpointing method, RV-Quant gives significant reductions in the total memory cost of activations. For instance, in the case of ResNet-152 which is favorable to the checkpointing method due to the simple structure as well as a large number of layers, ours reduces the memory cost by 41.6% (from 5.29GB to 3.09GB). In networks having more complex sub-networks, e.g., Inception modules, ours gives more reductions. In the case of Inception-v3, ours gives a reduction of 53.7% (3.87GB to 1.79GB). Note that in the case of AlexNet, the reduction is not significant. It

	AlexNet	ResNet-18	SqueezeNet-1.1	MobileNet-v2	ResNet-50
Full	0.35	1.86	1.58	7.34	9.27
Chen et al. [48]	x	0.98 (52.1 %)	1.05 (66.9 %)	4.21 (52.1 %)	3.70 (39.9 %)
(2,0)	0.23 (66.4 %)	0.42 (22.6 %)	0.59 (37.5 %)	0.74 (10.0 %)	1.22 (13.2 %)
(3,0)	0.23 (67.8 %)	0.46 (24.3 %)	0.61 (38.8 %)	0.84 (11.4 %)	1.34 (14.5 %)
(3,2)	0.24 (69.5 %)	0.50 (26.5 %)	0.64 (40.4 %)	1.13 (15.4 %)	1.52 (16.4 %)

	VGG-16	Inception-v3	ResNet-152	DenseNet-201
Full	9.30	9.75	20.99	24.53
Chen et al. [48]	x	3.87 (39.8 %)	5.29 (25.2 %)	6.62 (27.0 %)
(2,0)	3.65 (39.2 %)	1.16 (11.9 %)	1.64 (7.78 %)	2.09 (8.51 %)
(3,0)	3.75 (40.3 %)	1.43 (14.8 %)	2.27 (10.8 %)	2.85 (11.6 %)
(3,2)	3.88 (41.7 %)	1.79 (18.4 %)	3.09 (14.7 %)	3.83 (15.6 %)

Table 4.4 Comparison of memory cost (in GB).

is because the input data occupy the majority of stored activations and we store them in full precision. However, the impact of input data storage diminishes in deep networks.

We also measured the training runtime of ResNet-50 with mini-batch of 64 on NVIDIA Tesla M40 GPU. Compared to the runtime of existing full-precision training, our method requires a small additional runtime, 8.8% while the checkpointing method has much larger runtime overhead, 32.4%. Note that as mentioned in Section 4.3.3, our method has a potential for further reduction in training time on hardware platforms supporting reduced-precision computa-

tion.

Configuration	Accuracy	Configuration	Accuracy	Configuration	Accuracy
(3,2)-(2,1)-(2,0)	75.01 / 92.42	(2,0)-(2,1)-(3,2)	47.35 / 72.31	(3,2)-(3,1)-(3,0)	75.72 / 92.69
(F)-(3,2)-(2,0)	75.45 / 92.63	(2,0)-(3,2)-(F)	50.36 / 75.02	(3,2)-(3,1)-(2,0)	75.34 / 92.55
(F)-(2,1)-(2,0)	75.38 / 92.44	(2,0)-(2,1)-(F)	52.72 / 76.76	(3,0)-(3,1)-(3,2)	75.60 / 92.77
(2,0)-(3,1)-(3,2)	48.67 / 73.54				

Table 4.5 Sensitivity analysis of RV-Quant configurations (bitwidth and AR [%]) across training phases. The full-precision network gives the accuracy of 75.92 / 92.90%.

Table 4.5 shows the impact of RV-Quant configurations on training accuracy of ResNet-50. We change the configurations when the learning rate changes (with the initial value of 0.1) at 0.01 and 0.001. For instance, (F)-(3,2)-(2,0) represents the case that, as the initial configuration, we use full-precision activation (F) during back-propagation. After 30 epochs, the configuration is changed to 3-bit 2% RV-Quant. Then, after 60 epochs, it is changed to 2-bit 0% RV-Quant.

In Table 4.5, the key observation is that it is important to have high precision at the beginning of training. Compared with the case that training starts with full-precision activations and ends with aggressively reduced precision, (F)-(3,2)-(2,0), the opposite case, (2,0)-(3,2)-(F) gives significantly lower accuracy, 75.45% vs. 50.36%. Another important observation is that activation annealing works. For instance, (3,2)-(3,1)-(3,0) gives almost the same result to (3,2)-(3,2)-(3,2) in Table 4.2 and, a more aggressive case, (3,2)-(3,1)-(2,0) gives only by 0.58% smaller top-1 accuracy. Thus, as training advances, we need the smaller amount of large values, which means we can have smaller

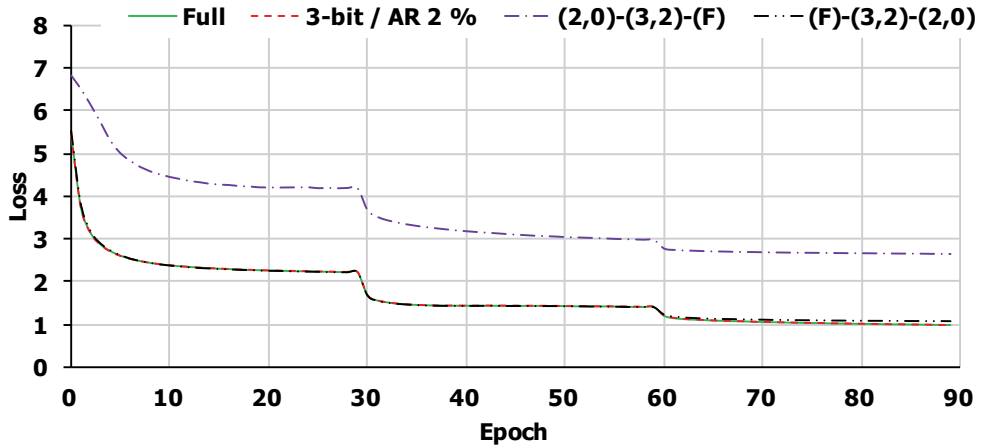


Figure 4.3 Training loss of ResNet-50 with various RV-Quant configurations.

memory cost of activations. This can be exploited for memory management in servers. We expect it can also be utilized in memory-efficient server-mobile co-training in federated learning [56] where the later stage of training requiring smaller memory cost can be performed on memory-limited mobile devices while meeting the requirements of user-specific adaptation using private data.

Figure 4.3 shows the training loss of different RV-Quant configurations during training. First, the figure shows that too aggressive quantization at the beginning of training, i.e., (2,0)-(3,2)-(F), does not catch up with the loss of full-precision training (Full in the figure). The figure also shows that the configuration of 3-bit 2% RV-Quant gives almost the same loss as the full-precision training.

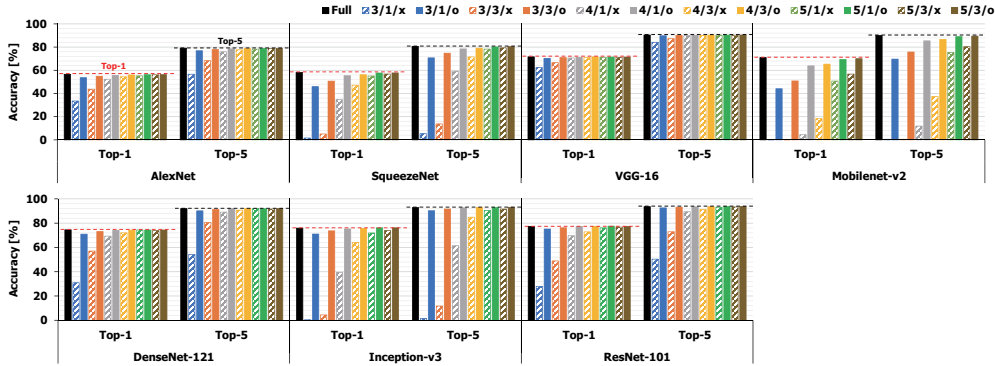


Figure 4.4 V-Quant results. The dashed lines and black solid bar represent full-precision accuracy. Legend: $\text{bitwidth} / \text{AR} [\%] / \text{fine-tuning or not}$.

4.4.2 Inference Results

Figure 4.4 shows the accuracy of quantized models across different configurations of bitwidth and AR. We apply the same bitwidth of low precision to both weights and activations and 16 bits to large values of weights and activations. In addition, we quantize all the layers including the first (quantized weights) and last convolutional layers. As the figure shows, V-Quant with fine-tuning, at 4 bits and an AR of 1%, gives accuracy comparable to full precision in all the networks within 1% of top-1 accuracy. If V-Quant is applied without fine-tuning, the larger AR needs to be used to compensate for accuracy drop due to quantization. However, the figure shows that fine-tuning successfully closes the accuracy gap between V-Quant and full-precision networks.

Figure 4.5 illustrates the effect of large values on the classification ability. The figure shows the principal component analysis (PCA) results of the last convolutional layer of AlexNet for four classes. Figure 4.5 (a) shows the

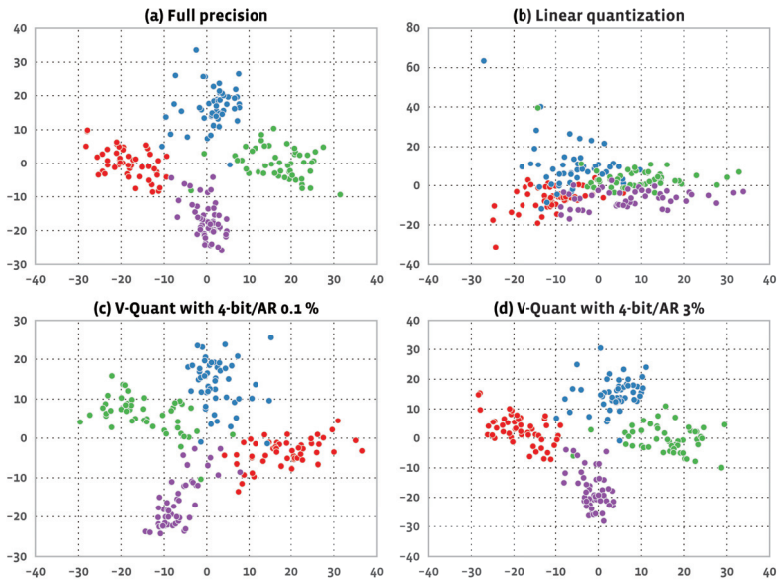


Figure 4.5 PCA analysis of the input activations on the last fully-connected layer of AlexNet.

PCA result of full-precision network. As Figure 4.5 (b) shows, when the conventional 4-bit linear quantization, or 4-bit 0% V-Quant is applied to weight/activations, it is difficult to classify four groups of data successfully. This is because the quantization errors are accumulated across layers thereby deteriorating the quality of activations. However, as Figure 4.5 (c) shows, only a very small amount (0.1%) of large values can improve the situation. As more large values are utilized, the classification ability continues to improve (3% in Figure 4.5 (d)). The figure demonstrates that our idea of reducing quantization errors for the majority of data by separately handling large values is effective in keeping good representations.

4.4.3 LSTM Language Model

	Large-1%		Large-3%		Small-1%		Small-3%	
	Valid	Test	Valid	Test	Valid	Test	Valid	Test
float	75.34	72.31	75.34	72.31	103.64	99.24	103.64	99.24
2-bit	79.92	77.31	77.87	74.99	140.70	135.11	122.25	117.76
3-bit	76.19	73.22	75.79	72.72	107.60	102.82	105.99	101.44
4-bit	75.46	72.48	75.44	72.44	104.22	99.83	103.95	99.57

Table 4.6 Impact of quantization on word-level perplexity of an LSTM for language modeling.

We apply V-Quant to an LSTM for word-level language modeling [38, 51, 52]. Table 4.6 shows the results of the models. Each of the large and small models has two layers. The large model has 1,500 hidden units and the small one 200 units. We measure word-level perplexity on Penn Tree Bank data [46]. We apply V-Quant only to the weights of the models since clipping is applied to the activation.³

As Table 4.6 shows, we evaluate three cases of bitwidth, 2, 3 and 4 bits and two ratios of large weights, 1%, and 3%. As the table shows, for the large model, the 4-bit 1% V-Quant preserves the accuracy of the full-precision model. However, the small model requires the larger ratio of large weights (3%) in order to keep the accuracy.

³The distribution of activations obtained by clipping tends to have the large population near the maximum/minimum values. Considering that clipped activation functions like ReLU6 are useful, it will be interesting to further investigate clipping-aware quantization.

4.5 Conclusions

We presented a novel value-aware quantization to reduce memory cost in training and computation/memory cost in inference. To realize aggressively low precision, we proposed separately handling a small number of large values and applying reduced precision to the majority of small values, which contributes to reducing quantization errors. In order to apply our idea to training, we proposed quantized back-propagation which utilizes quantized activations only during back-propagation. For inference, we proposed applying fine-tuning to quantized networks to recover from accuracy loss due to quantization. Our experiments show that our proposed method outperforms the state-of-the-art method of low-cost memory in training in deep networks, e.g., 41.6% and 53.7% smaller memory cost in ResNet-152 and Inception-v3, respectively. It also enables 4-bit inference (with 1% large values) for deep networks such as ResNet-101 and DenseNet-121, and 5-bit inference for efficient networks such as SqueezeNet-1.1 and MobileNet-v2 within 1% of additional top-1 accuracy loss.

Chapter 5

Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation

5.1 Introduction

This work was published in ISCA'2018 conference [16]. Our proposed accelerator is based a quantization method called value-aware quantization, or *outlier-aware quantization*, which divides the distribution of data (weights or activations) into two regions, of low and high precision. It applies reduced precision, e.g., 4-bit representation, to the low-precision region that contains a majority of the data. The high-precision region contains only a small portion (e.g., 3%) of the total data, and maintains the original, high precision, e.g., 16-bit representation. We call the data in the high-precision region *outliers*, as they are far fewer in number and larger in size than data in the low-precision region. Outlier-aware quantization enables low precision, e.g., four bits, for very deep models, such as ResNet-101 and DenseNet-121, with a very small (3%) ratio of outliers at a negligible ($<1\%$) loss of accuracy.

In this study, we propose a hardware accelerator that performs dense and reduced precision computations on a majority of data in the low-precision region while performing sparse and high-precision computation on outliers. The proposed accelerator, called the outlier-aware accelerator (*OLAccel*), achieves a significant reduction in energy consumption by reducing the amount of memory access, and by using smaller units of computation.

The contributions of this work are as follows:

- We propose an accelerator called *OLAccel* that implements 4-bit computations on very deep neural networks, e.g., ResNet-101 and DenseNet-121.
- *OLAccel* differently handles outlier activations and weights for computational efficiency.
- *OLAccel* performs sparse high-precision computation for outlier activations in parallel with dense low-precision computation for a majority of activations.
- *OLAccel* reduces the number of additional execution cycles due to outlier weights by equipping SIMD lanes with an outlier MAC unit that runs in parallel with them to reduce latency in most cases of outlier weight occurrence.
- *OLAccel* skips computations with zero-input activation, thereby further improving energy efficiency.

- We designed OLAcel in Verilog and compare it with prevalent accelerators, Eyeriss [26] and ZeNA [33] at reduced precision.

5.2 Proposed Architecture

5.2.1 Overall Structure

The computational micro-architecture is important, especially for performance. We use 4-bit data that reduce buffer size and MAC unit overhead significantly. This emphasizes the control overhead of the accelerator, and thus a SIMD structure is needed to minimize overhead. However, the conventional SIMD structure is not appropriate for outlier-aware quantization. For instance, if we use 4-bit precision in 16-way SIMD accelerators like DianNao [27] and SCNN [32], the performance overhead for outlier handling can be significant owing to the high probability of outlier occurrence, e.g., 27.5% ($= 1 - 0.99^{32}$), even with 1% of outliers. OLAcel addresses this with two following novel solutions: outlier PEs and outlier PE groups, and by supporting data structures for sparse and dense data.

Figure 5.1 shows the overall structure of OLAcel. At the top of the hierarchy is a PE swarm consisting of multiple PE clusters, a swarm buffer, and a controller. As the figure shows, each PE cluster has cluster activation/weight buffers (left), PE groups (center), and a cluster output buffer (right). Each PE group consists of multiple MAC units and group activation/weight/output buffers. OLAcel is based on 4-bit MAC units¹ (and a small number of full

¹In this work, we explain OLAcel based on 4-bit MAC units. However, it is not limited to

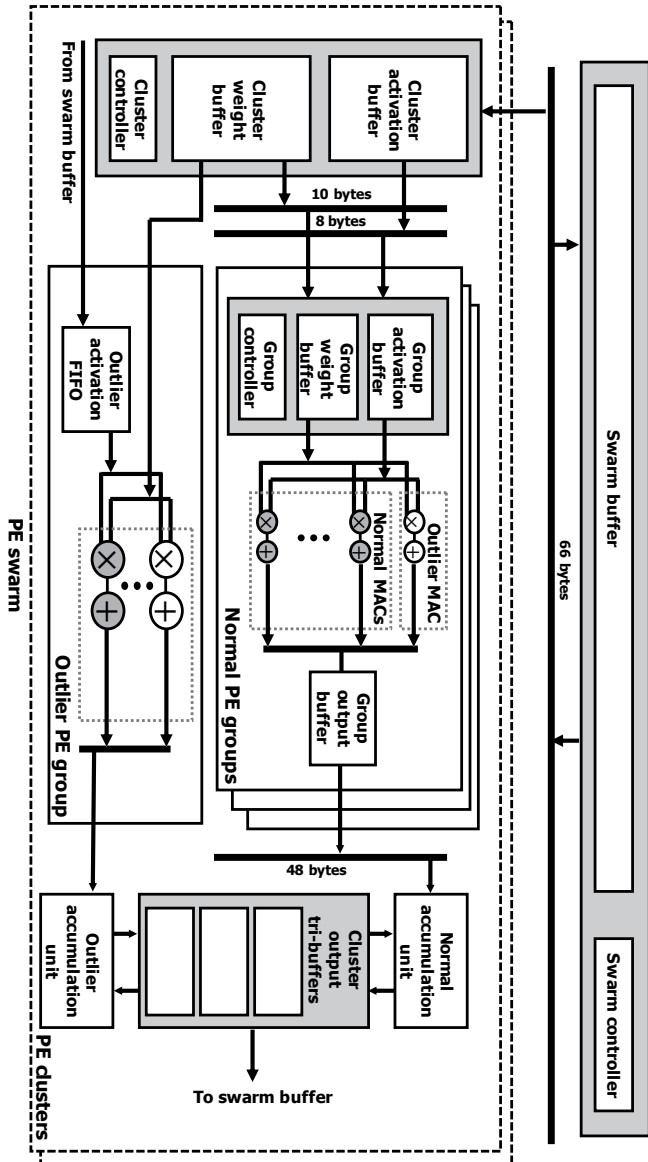


Figure 5.1 Overall structure of OLAaccel.

precision MAC units), and supports computation with 4-/8-/16-bit activations and 4-/8-bit weights.

As the figure shows, there are two types of PE groups, normal and outlier. The normal PE group contains 17 (16 normal and one outlier) 4-bit MAC units while the outlier PE group contains 17 mixed-precision MAC units that support $16\text{-bit} \times 4\text{-bit}$ operations. Note that each normal 4-bit or 16-bit MAC unit is in charge of producing a partial sum of its associated output channel. Thus, on each cycle, it receives its associated weight and a broadcast activation and produces a partial sum. As will be explained in Section 5.2.4, the additional outlier MAC unit in the PE group is involved in multiplication using outlier weights. Thus, the group weight buffer contains both normal (4-bit) and outlier (8-bit) weights while the group activation buffer contains only 4-bit activations. As the figure shows, the outlier PE group reads 4-bit or 8-bit weights from the cluster weight buffer, and 8-bit or 16-bit outlier activations from the swarm buffer. It then performs computations using outlier activations. Note that the outlier activations are stored only in the swarm buffer while outlier weights can be stored in the swarm buffer and the cluster/group weight buffers.

The figure shows two (normal and outlier) accumulation units connected to the tri-buffer containing the output of the cluster. Each accumulation unit adds the results of the associated (normal or outlier) PE group with the associated partial sum stored in the tri-buffer. To resolve the issue of coherence between the accumulation units, their execution is pipelined. The outlier ac-

4-bit precision, and can be easily extended to other base bitwidths, e.g., 2 or 8 bits.

cumulation unit accesses partial sums only once the normal accumulation unit finishes adding the partial sums. Thus, both accumulation units run in parallel in a pipeline. To support the required bandwidth for the normal and outlier accumulation units, the tri-buffer containing the cluster output consists of three buffers, two for the normal accumulation unit and one for the outlier accumulation unit.

5.2.2 Dataflow

Figure 5.2 shows the data structures of the normal/outlier weights and normal activations. In the figure, we use the following notation for the sake of a clear and concise description. In the case of activation, the width, height, and the number of channels of the 3D tensor are expressed as $A_{w \times h \times c}$. The kernel weight uses a similar notation, $K_{w \times h \times i \times o}$, which represents the width, height, and the numbers of input and output channels of a 4D tensor. As the figure shows, the cluster weight buffer stores a subset of kernel weights, e.g., $K_{w \times h \times 16 \times 16}$. As shown in the box of the cluster weight buffer in the figure, the weights are stored at a granularity of 80-bit weight chunks (entries in the table). The weight chunk consists of 16 4-bit weights ($= 4b \times 16$), an 8-bit pointer (OL_{ptr}), a 4-bit pointer (OL_{idx}), and the most significant four bits of an outlier weight (OL_{MSB}). As the figure shows, the cluster weight buffer contains 200 weight chunks. The PE group receives these chunks and stores them in the group weight buffer for its execution.

The weight chunk represents a set of 16 kernel weights, each of which

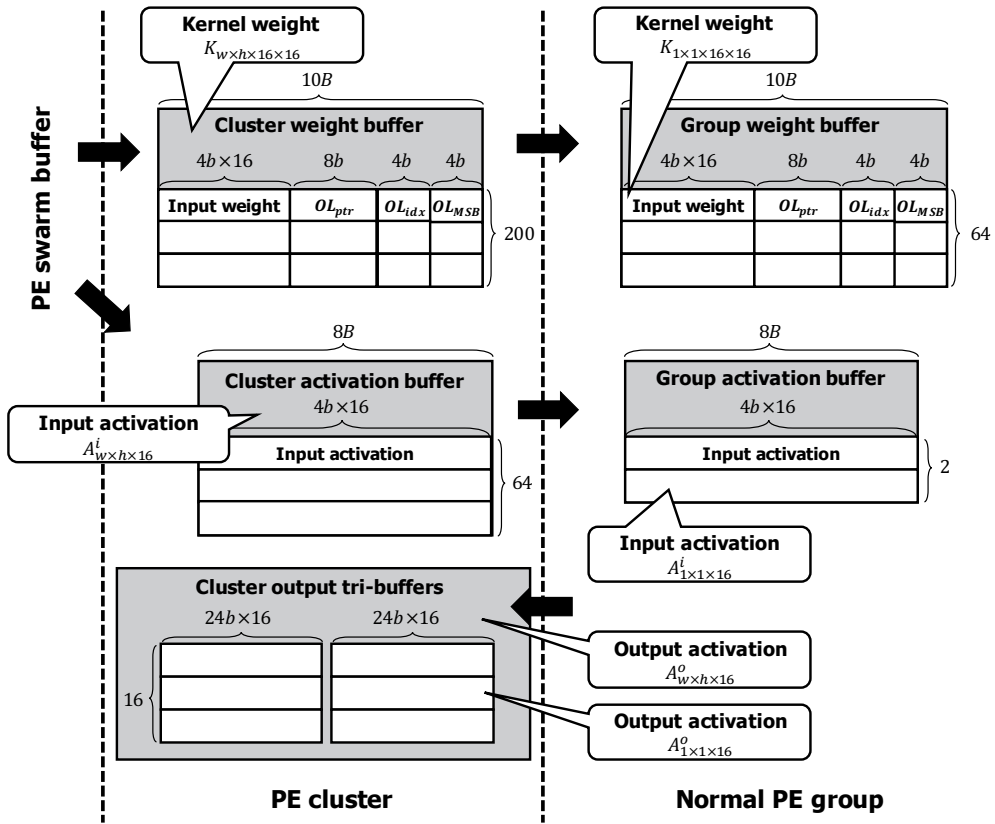


Figure 5.2 Data structure associated with normal / outlier weight and normal activation.

belongs to one of 16 output channels. In case there is no outlier weight among the 16 weights, the two pointers, OL_{ptr} , and OL_{idx} , and the 4-bit field OL_{MSB} are all zero. When an outlier weight exists, OL_{idx} points to the index (of the 16) of the weight in the given chunk and OL_{MSB} contains the most significant four bits of the 8-bit outlier weight. The remaining least significant three bits and a sign bit of the outlier weight are stored in the associated position of 4×16 weight bits in the chunk. In case there is more than one outlier weight among the 16 weights, their most significant four bits are stored in (the 64b field of the input weight of) another weight chunk, and OL_{ptr} points to that chunk in the cluster weight buffer (or group weight buffer). Thus, the PE group checks OL_{ptr} to see if there are more than one outlier weights in the weight chunk. We explain this in detail in Section 5.2.4.

Figure 5.2 also shows how low-precision input activations are stored in the PE cluster and group. The cluster activation buffer stores a subset of input activations of $A_{w \times h \times 16}^i$, which consist of activation chunks each of which has 16×4 -bit input activations. The PE group receives input activations at the granularity of the chunk. As the figure shows, the cluster activation buffer stores 64 activation chunks while the group activation buffer stores two chunks. As shown at the bottom of the figure, the cluster output tri-buffer manages output partial sums with a larger chunk size, $24b \times 16$, as a partial sum requires higher precision (24 bits) than that of the input (4/8/16 bits).

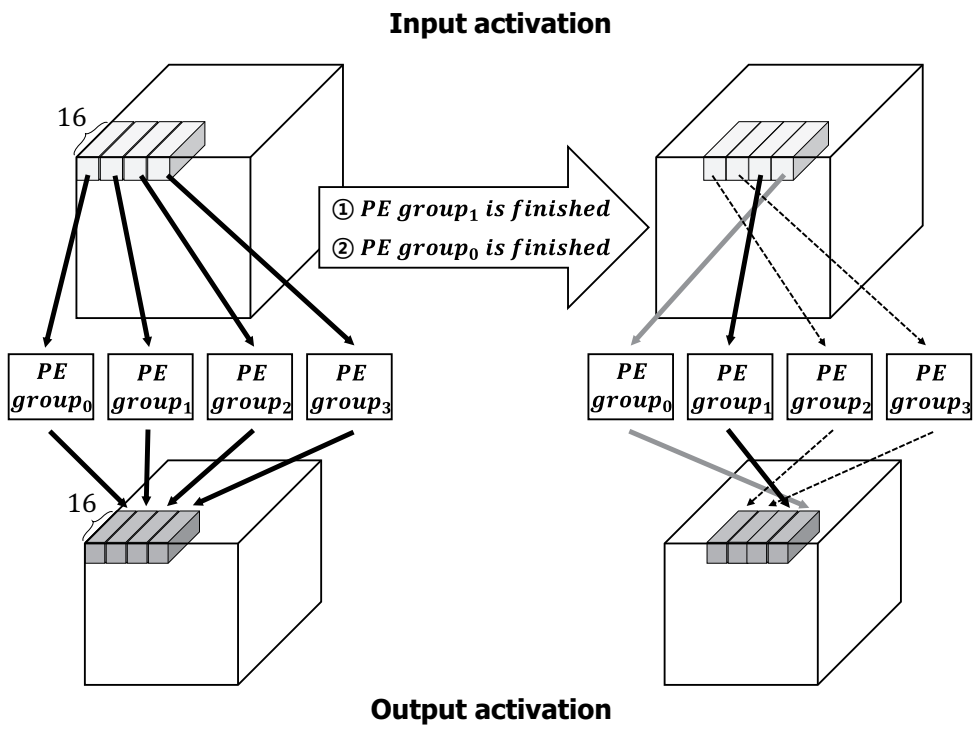


Figure 5.3 Operation of PE cluster.

5.2.3 PE Cluster

Figure 5.3 shows how the PE cluster controls the execution of its PE groups. As the figure shows, a PE group takes as input an activation chunk $A_{1 \times 1 \times 16}^i$ and generates as output a partial sum chunk of $A_{1 \times 1 \times 16}^o$. The figure illustrates a scenario where four PE groups run in parallel.

The PE group skips computations with zero-input activations. Thus, owing to the different numbers of non-zero values in the input activation chunks, some PE groups can finish earlier than others. In the figure, we assume that PE group $_1$ finishes followed by group $_0$, before the other PE groups. In such a case, as shown on the right-hand side of the figure, the PE cluster allocates new input activation chunks to the PE groups that are ready, which keeps them busy as far as there are available input activations in the cluster activation buffer. This mechanism is important in OLA_{ccel} as it enables high utilization of MAC units.

5.2.4 Normal PE Group

Figure 5.4 shows how the PE group functions when there is one outlier among the 16 4-bit weights. The group activation buffer shows that the input activation chunk has only three non-zero activations (a_0 , a_3 , and a_{15}) among the 16 activations. The PE group selects, from the input activation chunk, a non-zero activation, e.g., a_0 , and broadcasts it to the 16 normal and one outlier MAC units. The PE group also selects a weight chunk from the group weight buffer and provides the 16 normal MAC units with their associated kernel weights.

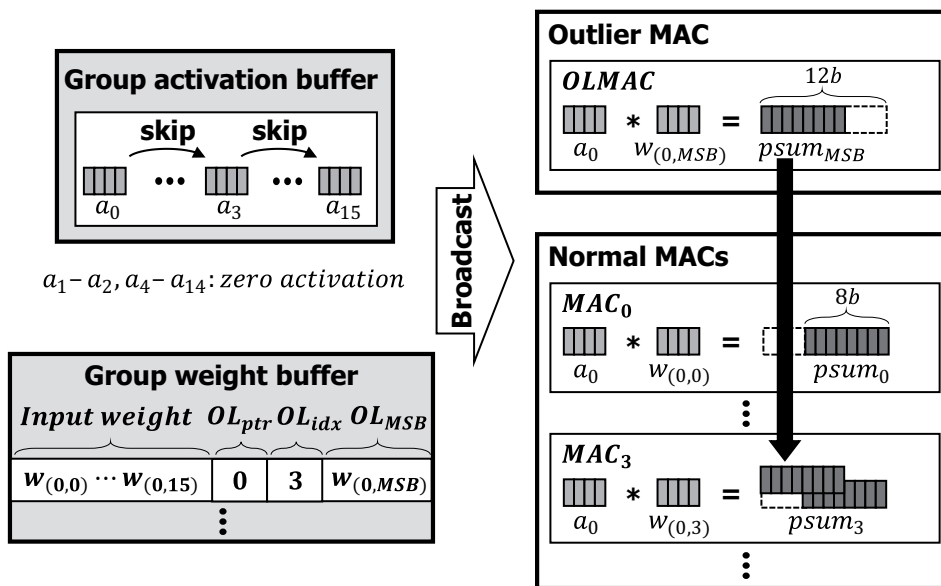


Figure 5.4 PE group operation in case that there is an outlier weight among 16 weights.

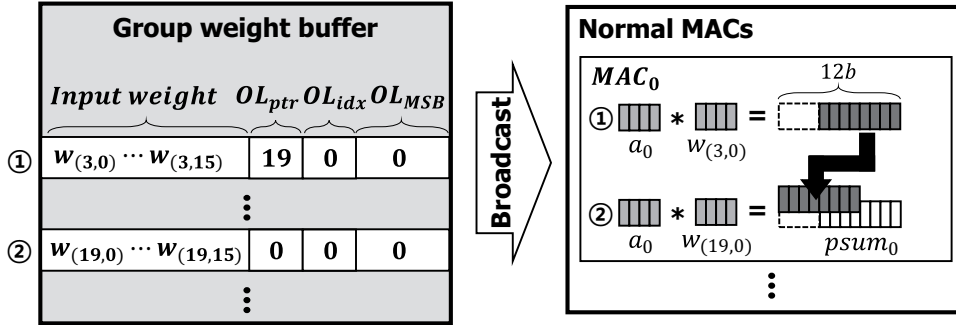


Figure 5.5 PE group in the case of more than one outlier weight.

The figure shows that the selected weight chunk contains an outlier weight, the index of which, OL_{idx} is three. Thus, the outlier MAC unit takes as input the most significant four bits of the outlier weight OL_{MSB} from the weight chunk and multiplies it with the broadcast activation a_0 . Then, the result is broadcast to all normal MAC units and the normal MAC unit pointed by OL_{idx} receives it. As the figure shows, the MAC unit, MAC_3 , which performs computation with the least significant four bits of the outlier weight (in the input weight field of the weight chunk), receives and adds it to its partial sum. The entire operation of 17 MAC units can be carried out in one clock cycle. When there is no outlier weight, the PE group runs in the same manner as above, but OL_{MSB} stores the value zero. Thus, the outlier MAC unit generates a zero result, which does not affect the partial sum of the normal MAC units.

Figure 5.5 shows how the PE group performs computation when there are more than one outlier weights in the weight chunk. As mentioned above, in such a case, two weight chunks are used. In the figure, the first weight chunk

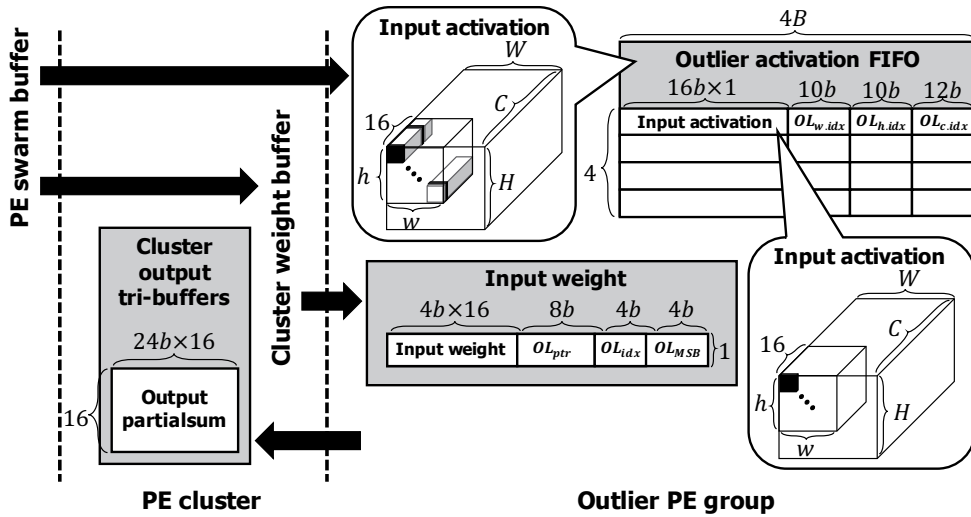


Figure 5.6 Data structure associated with outlier activation.

sets the pointer, OL_{ptr} , to the index of the second weight chunk, e.g., 19. In this case, the MAC operation takes two clock cycles to perform an 8-bit weight \times 4-bit activation. Thus, in the first clock cycle, the 16 4-bit weights (the least significant four bits) of the first chunk are provided to the 16 normal MAC units. In the next clock cycle, the 4-bit weights of the second weight chunk are provided. Each normal MAC units adds the two results. Note that the outlier MAC is not used in this scenario. Moreover, MAC units with normal 4-bit weights become idle in the second clock cycle.

5.2.5 Outlier PE Group and Cluster Output Tri-buffer

Figure 5.6 shows the data structure and data flow in the outlier PE group. As the figure shows, input outlier activations are fetched from the swarm buffer to

the outlier PE group. The activations are sparse data. Thus, as shown in the figure (Outlier activation FIFO), each outlier activation is represented by an outlier chunk consisting of a 16-bit activation and the three coordinates ($OL_{w.idx}$, $OL_{h.idx}$, and $OL_{c.idx}$) of the outlier in the tensor of the input activation. The outlier PE group also fetches weight chunks from the cluster weight buffer. As mentioned above, the outlier PE group consists of 17 mixed-precision MAC units and runs in the same way as the normal PE group. Thus, the non-zero activation is broadcast to the 17 MAC units and multiplied by the associated weights to produce partial sums for the 16 output channels. The partial sums (16 24-bit data items) are stored in the cluster output tri-buffer as shown in the figure.

Figure 5.7 shows how the normal and outlier accumulation units accumulate the partial sums of the normal and outlier PE groups in a pipelined manner while accessing the three buffers in the cluster output tri-buffer. The figure illustrates a scenario of the pipeline at times t_0 and t_1 . At t_0 , the normal accumulation unit calculates 16 partial sums by accessing two buffers, $buffer_0$ and $buffer_1$, in the cluster output tri-buffers. Then, at t_1 , the normal accumulation unit accesses two buffers, $buffer_1$ and $buffer_2$, while the outlier accumulation unit accesses $buffer_0$. Thus, there is no problem of coherence when the outlier accumulation unit accesses the buffer. To complete partial sum calculations for a convolution, e.g., a 3×3 convolution in the figure, multiple stages of the pipeline operation are needed.

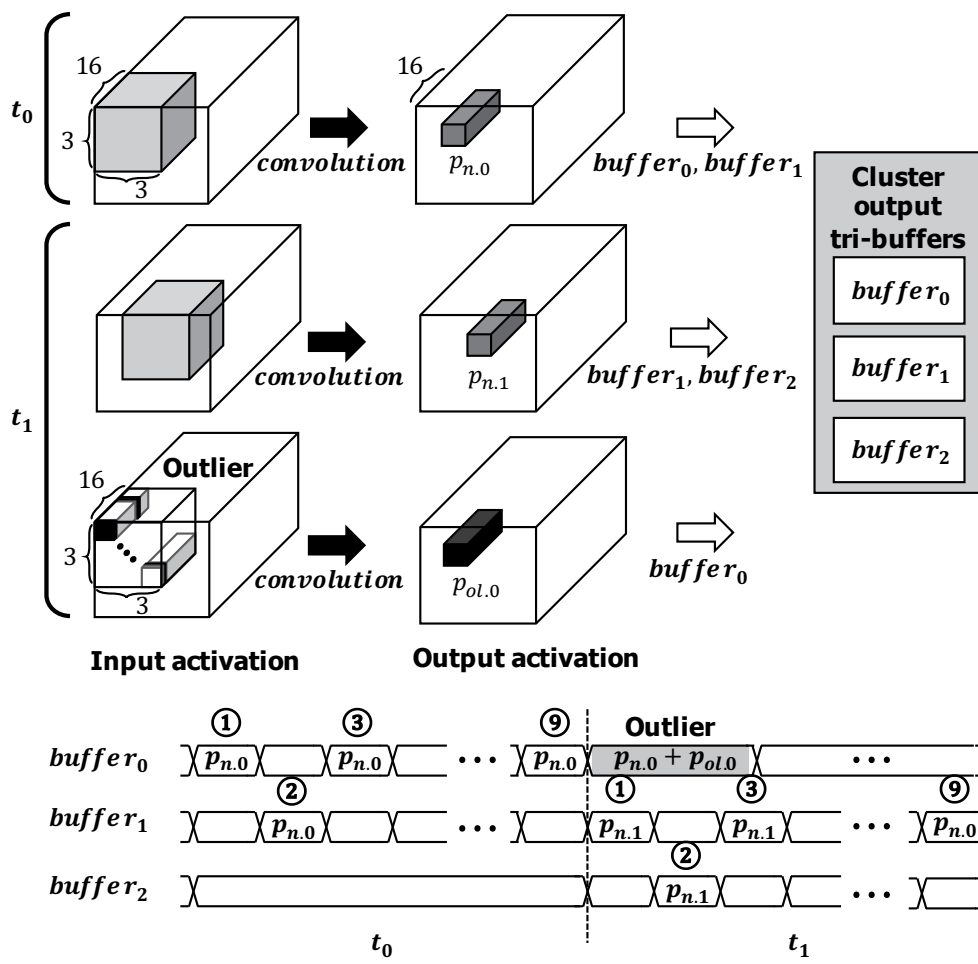


Figure 5.7 Cluster output tri-buffer.

5.3 Evaluation Methodology

We implemented outlier-aware quantization in the PyTorch and Caffe environments [41, 53]. We used randomly selected data in TorchVision [54] and Caffe [41] to calculate average energy consumption and execution cycles by running neural networks on our hardware accelerator models. In case of AlexNet and VGG-16, we used pruned models [23] to evaluate the capability of zero skipping while we pruned ResNet-18 on our own.

For the baseline and our architectures, we developed cycle-accurate models. We did a Verilog design and synthesized them using Design Compiler with a commercial 65-nm LP library (typical) at 250 MHz and 1.0 V. We used CACTI to estimate SRAM area/power/latency [57] and the DRAM power model from Micron [58].

We compare our 4-bit OLAcel with the following baselines:

- 16-bit and 8-bit Eyeriss (denoted by Eyeriss16 and Eyeriss8, respectively), which, for zero input, do not reduce the number of execution cycles but clock-gate computations [26].
- 16-bit and 8-bit ZeNAs that reduce the number of execution cycles by skipping computations with zero input weights and activations [33].²

In 16/8-bit comparison, we utilize 16/8-bit raw input activation for all the

²We chose ZeNA as the baseline for the zero-skipping accelerator as it provides the best speedup for AlexNet by skipping both zero weights and activations, compared with other zero-skipping baselines.

networks. In [5], Migacz reported that the minimum bitwidth for linear quantization (without loss of accuracy) is eight bits for deep networks like ResNet-101/152 and GoogLeNet. In case of 8-bit Eyeriss and ZeNA, we assume that existing accelerators support eight bits and there is no degradation in accuracy by using the aforementioned quantization. In 8-bit comparison, we use 8-bit outlier activations for OLAcel based on the same assumption.

	Eyeriss		ZeNA		OLAcel	
	8-bit	16-bit	8-bit	16-bit	8-bit	16-bit
# PEs	165	165	168	168	576	768
area (mm^2)	0.96	1.53	1.01	1.66	0.93	1.67
On-chip memory (AlexNet)		Act Weight	393 kB (16-bit) / 196 kB (8-bit) 16 kB (16-bit) / 8 kB (8-bit)			
On-chip memory (VGG-16 & ResNet-18)		Act Weight	4.8 MB (16-bit) / 2.4 MB (8-bit) 16 kB (16-bit) / 8 kB(8-bit)			

Table 5.1 Configurations of Eyeriss, ZeNA, and OLAcel.

Table 5.1 shows the architectural configurations used in our experiments. In case of 16 (8)-bit comparison, OLAcel has 16 (8)-bit outlier activations with 4-bit MAC units and 8-bit outlier weights. We perform an ISO-area comparison between the baselines and OLAcel. Thus, given the same amount of on-chip memory, which keeps all the data required for a layer to avoid off-chip memory accesses, each architecture is allocated the same chip area for logic and buffer implementations as Eyeriss. In case of 8-bit Eyeriss, we first obtain the area

of 165 8-bit processing elements (PEs) [26]³. We then determine the configuration (number of MAC units) of OLAcel to meet the given area target. In 16 (8)-bit comparisons, Eyeriss, ZeNA, and OLAcel occupy 1.53 (0.96) mm^2 , 1.66 (1.01) mm^2 , and 1.67 (0.93) mm^2 , respectively. ZeNA has 168 PEs in both the 16- and 8-bit cases. In these cases, we use 393 kB (16-bit comparison) and 196 kB (8-bit) of on-chip memory for AlexNet, and 4.8 MB and 2.4 MB for VGG-16 and ResNet-18, respectively. We use the same memory size for the three accelerators on each network for fair comparison. A single large memory for different networks is advantageous for OLAcel, especially on AlexNet as OLAcel benefits from reduced memory access, and the larger memory renders the memory more dominant in terms of energy consumption. In case of OLAcel, the on-chip memory is used as the swarm buffer.

In case of 16-bit comparison, OLAcel is equipped with eight PE clusters, six PE groups/cluster, and a total of 768 ($= 8 \times 6 \times 16$) 4-bit MAC units. However, in case of 8-bit comparison, OLAcel has a smaller number of 4-bit MAC units, 576 (in six PE clusters), to satisfy the constraint of the reduced area, 0.96 mm^2 .

5.4 Experimental Results

Figure 5.8 compares the number of execution cycles and energy consumption (normalized to Eyeriss16) on AlexNet. Compared with Eyeriss16, OLAcel16

³In the case of Eyeriss and ZeNA, the PE consists of a MAC unit and internal buffers.

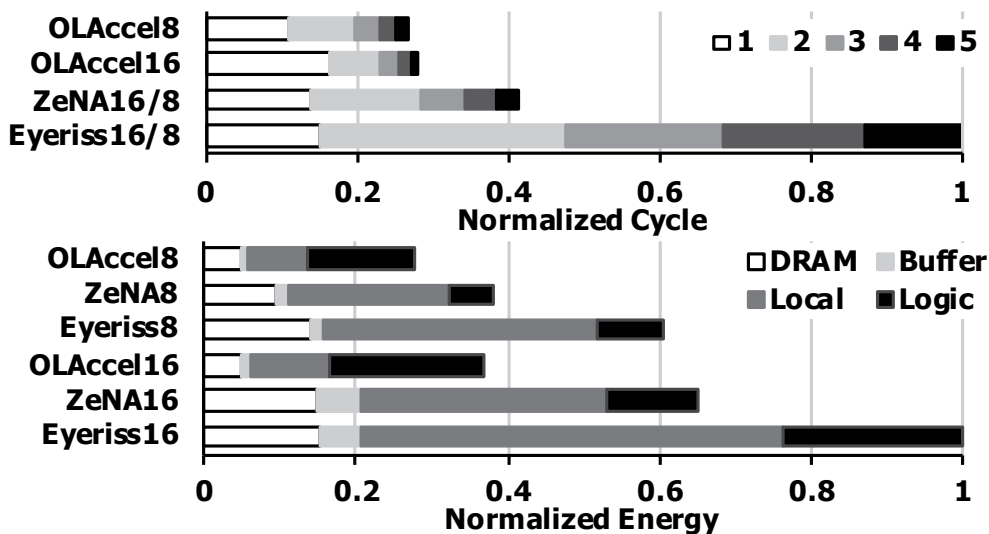


Figure 5.8 AlexNet cycle and energy breakdown.

yields a 71.8% reduction in execution cycles.⁴ This is mainly due to a large number of MAC units (768 in OLAcel versus 165 in Eyeriss), which is enabled by reduced precision (from 16 to 4 bits). Compared with ZeNA16 containing 168 PEs, OLAcel16 gives a 31.5% reduction in execution cycles. The reduction is smaller than in Eyeriss because ZeNA skips computations for zero weights and activations, whereas OLAcel skips computations for only zero activations. In addition, OLAcel16 spends a long execution cycle for the first convolutional layer due to the handling of 16-bit raw input activation on 4-bit MAC units.

OLAcel8 gives an 35.1% reduction in execution cycles compared with ZeNA8. Due to the smaller number (576) of 4-bit MAC units, OLAcel8 slows

⁴Note that both 16-bit and 8-bit Eyeriss (ZeNAs) yield the same number of execution cycles as we maintain the same number of PEs, 165 (168 in ZeNA), in both cases.

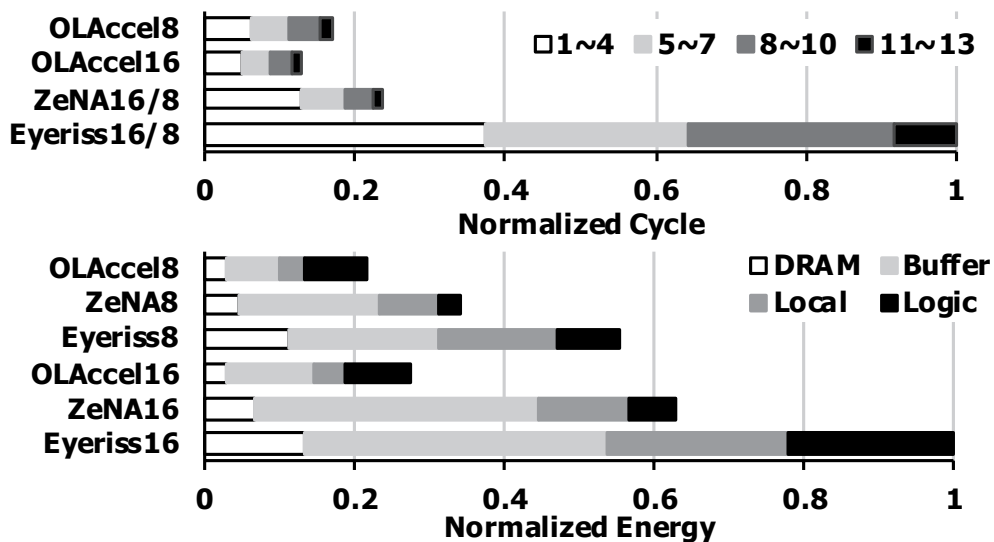


Figure 5.9 VGG-16 cycle and energy breakdown.

down from the second to fifth convolutional layer compared to OLAccel16. However, the input activation of the first layer has 8 bits, which reduces the execution cycle of the first convolutional layer significantly compared with OLAccel16 thereby enabling OLAccel8 to outperform OLAccel16.

Figure 5.8 also compares energy consumption (normalized to Eyeriss16) decomposed into DRAM, on-chip memory or swarm buffer (Buffer), the local buffer of the PE or PE clusters/groups (local), and logic circuits, e.g., MAC unit and bus (logic). OLAccel16 (OLAccel8) yields a 43.5% (27.0%) reduction compared with ZeNA16 (ZeNA8). As the figure shows, the reduction is mainly due to energy reduction in the memory components (DRAM and SRAM buffers) enabled by reduced precision.

Figure 5.9 shows comparisons for VGG-16. Compared with ZeNA8/16,

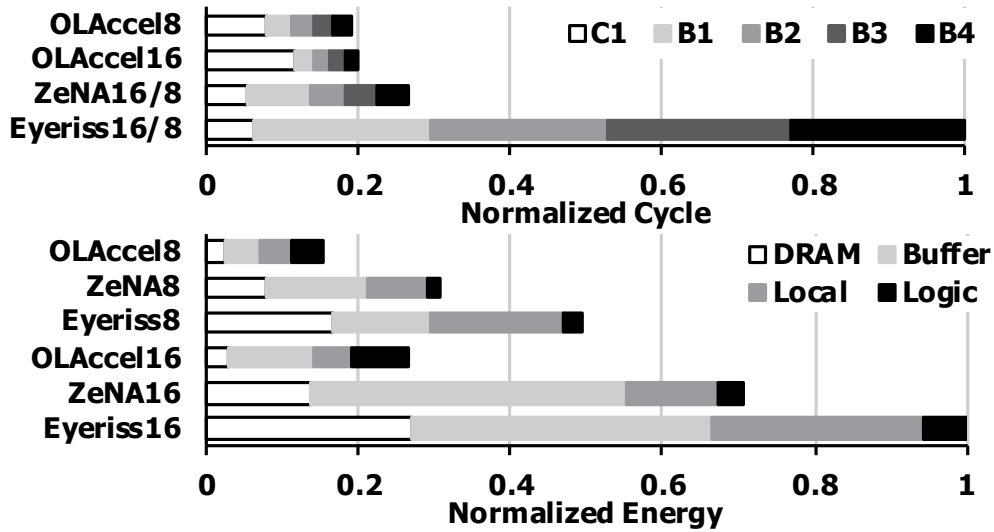


Figure 5.10 ResNet-18 cycle and energy breakdown.

OLAccel16 and OLAccel8 reduce by 45.3% and 28.3% execution cycles, respectively. OLAccel16 delivers the best performance owing to a large number (768) of MAC units. The performance improvement of OLAccel16 is greater than in the case of AlexNet because the effect of the first convolutional layer is mitigated due to the speedup of the other layers. Meanwhile, OLAccel8 shows worse performance than OLAccel16 because of the small number (576) of PEs. The comparison of energy consumption shows that OLAccel16 and OLAccel8 give a reduction of 56.7% and 36.3% compared with ZeNA16 and ZeNA8, respectively. As in the case of AlexNet, the reduced precision enables OLAccel to significantly reduce energy consumption in the memory components.

In the case of ResNet-18, as shown in Figure 5.10, OLAccel significantly reduces the number of execution cycles compared with Eyeriss. As in the

case of AlexNet, ResNet-18 has a long execution cycle for the first convolutional layer. Especially, as shown in Figure 5.10, the first convolutional layer of ResNet-18 requires dense computation with 8-bit weights and 16/8-bit input activations in 16/8-bit comparison. Thus, in 16 (8)-bit comparison, the first convolutional layer takes 8 (4) times longer execution cycle than a simple 4-bit dense computation, which makes the first convolutional layer (C1 in Figure 5.10) occupy half the total execution cycle of OLAcel16. However, owing to the speedups of the other layers where 4-bit dense computation is performed, OLAcel is able to reduce execution cycles more significantly than in the case of AlexNet. Specifically, compared with Eyeriss16 (Eyeriss8), OLAcel16 (OLAcel8) reduces the numbers of cycles by 80.1% (81.1%) in ResNet-18 while it gives a reduction of 71.8% (73.2%) in AlexNet.

Compared with ZeNA, OLAcel16 and OLAcel8 reduce by 25.3% and 29.0% execution cycles, respectively. Considering the fact that, as the decomposition of execution cycle in OLAcel and ZeNA shows in Figure 5.10, OLAcel is superior to ZeNA in the other layers except the first one, we expect that OLAcel can give much better performance than ZeNA in deeper networks, e.g., ResNet-101.⁵

OLAcel significantly reduces energy consumption in ResNet-18. Compared with ZeNA, OLAcel16 and OLAcel8 give energy reductions of 62.2% and 49.5%, respectively. Note that, aside from the effect of the first layer,

⁵When fine-tuning is adopted, OLAcel can give further speedup in ResNet-18 since fine-tuning can reduce the bitwidth of weights from 8 to 4 bits for the dense computation of the first convolutional layer as mentioned in Section II.

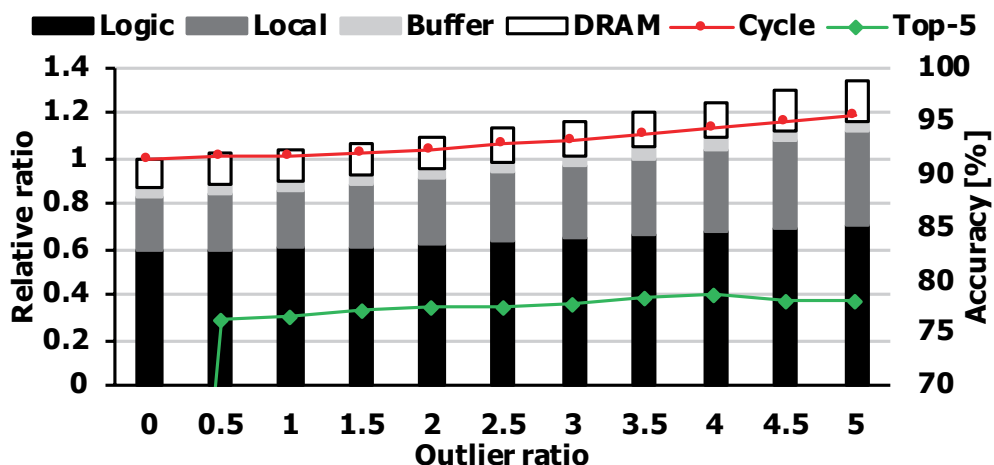


Figure 5.11 Normalized energy and cycle vs. outlier ratio: AlexNet on OLAcell16.

OLAcell gives more reduction in energy consumption for VGG-16 and ResNet-18 than AlexNet because the larger on-chip memory used for VGG-16 and ResNet-18 (Table 5.1) makes the energy consumption of on-chip memory more dominant thereby amplifying the effects of reduced precision, i.e., reduction in on-chip memory traffics.

Figure 5.11 shows the impact of outliers on the number of execution cycles, energy consumption, and accuracy when running AlexNet on OLAcell16. As the figure shows, as outlier ratio increases, both energy consumption and the number of execution cycles increase while accuracy improves. In the case where there are 3.5% outliers, compared with the case of 0% outlier, the total energy consumption and the number of execution cycles increase by 20.6% and 10.6%, respectively, while yielding much better accuracy, only 0.8% drop

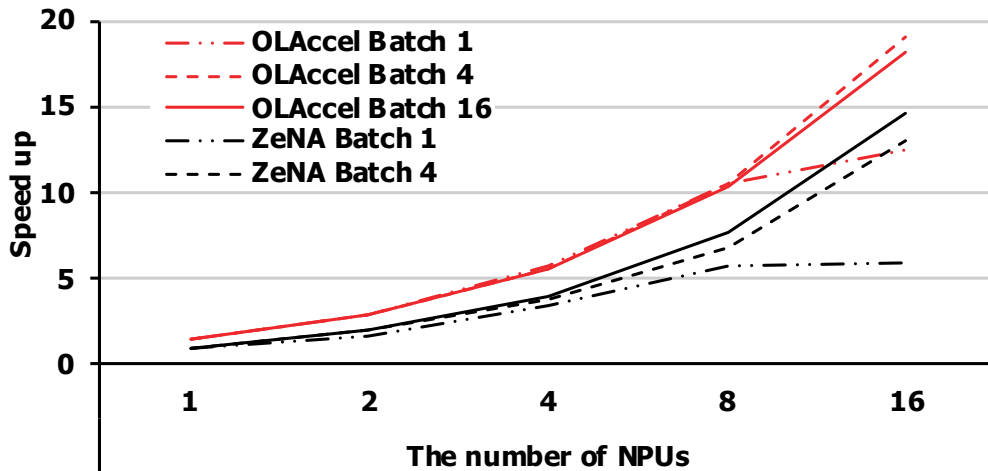


Figure 5.12 Scalability analysis on AlexNet: speedup (y-axis) vs. number of NPUs.

from the top-5 accuracy of full precision.

Figure 5.12 shows scalability on AlexNet. One instance of neural processing unit (NPU) is equipped with 768 4-bit MAC units in OLAcel (16-bit outliers) and 168 16-bit PEs in ZeNA. As the figure shows, we increase both the number of NPUs and batch size. The speedup (y-axis) is normalized to ZeNA with a batch size of one. The figure shows that both OLAcel and ZeNA exhibit good scalability when the batch size is 4 and 16. In case of a single batch, speedup tends to saturate in both cases when the number of NPUs reaches 16. This is due to low resource utilization. OLAcel yields a slightly better speedup in batch 4 than batch 16, mainly due to the off-chip bandwidth limit because batch 16 requires more off-chip memory bandwidth than batch 4.

Figure 5.13 shows the histogram of outlier activations in AlexNet when

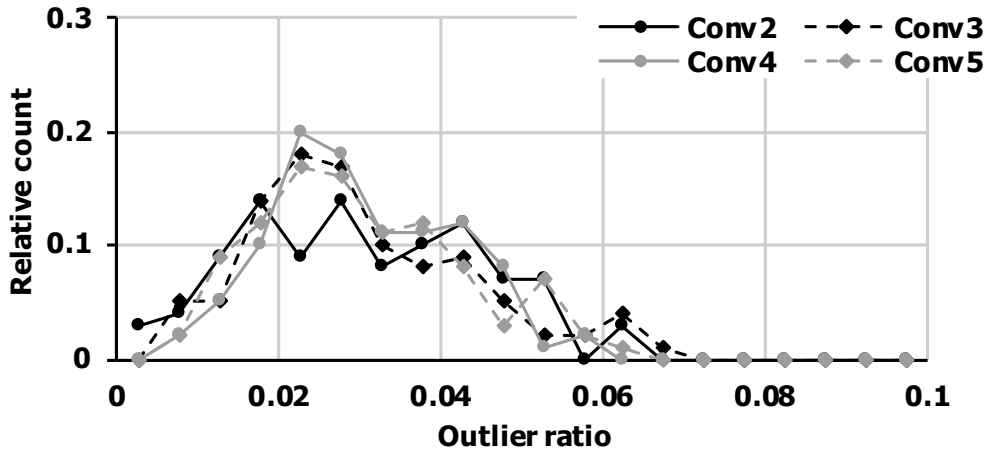


Figure 5.13 Histogram of outlier activation: outlier ratio of 3%.

the outlier ratio is 3%. As explained above, to avoid computing histograms at runtime, we obtain a static threshold for each layer by utilizing sample inputs during design time and compare the activation with the threshold at runtime to identify outlier activations. The figure shows that the distributions have mean values near 0.03, meaning that our implementation works relatively well. Although not shown in the work, the neural networks adopting batch normalization layers, e.g., ResNet-101, tend to provide better distributions with sharp peaks near the target outlier ratio.

Figure 5.14 shows how we determined 16 MAC units in a single PE group. A large number of MAC units in a PE group can improve performance by better exploiting activation broadcast. However, as the figure shows, in case of 32 and 64 MAC units, the probability of multiple outlier occurrences on 32 and 64 weights is higher than 50% at an outlier ratio of 5%. In case of

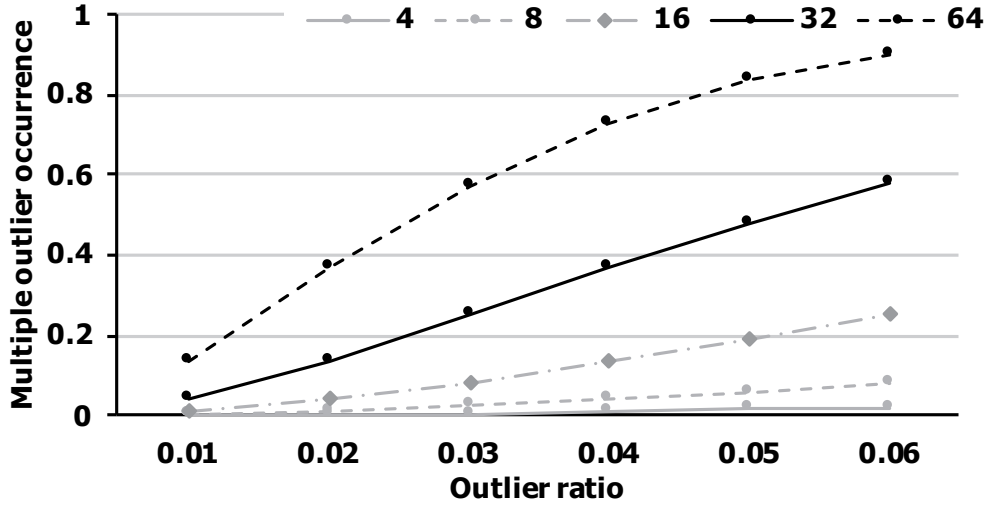


Figure 5.14 Probability of multiple outlier weights (y-axis) vs. outlier ratio.

multiple outlier weights in a PE group, OLAcel suffers from long execution cycles. Thus, we set the number of MAC units in a PE group to 16, which yields a smaller probability, e.g., 20%, even at an outlier ratio of 5%. Another reason for our choice is recent trends in neural network architectures. In state-of-the-art architectures like ResNext [59] and Deep Roots [60], the number of channels (on a branch) tends to decrease, e.g., to 32 or 16. Thus, channel-level parallelism can be limited. In such a case, a PE group with a large number of MAC units can suffer from low resource utilization.

Figure 5.15 shows the utilization breakdown of the convolutional layers in AlexNet. As the figure shows, the active period (Run) is relatively proportional to the ratio of non-zero activations (Non-zero). The figure also shows that the overhead of zero-skip operation (Skip) increases in proportion to the ratio. This

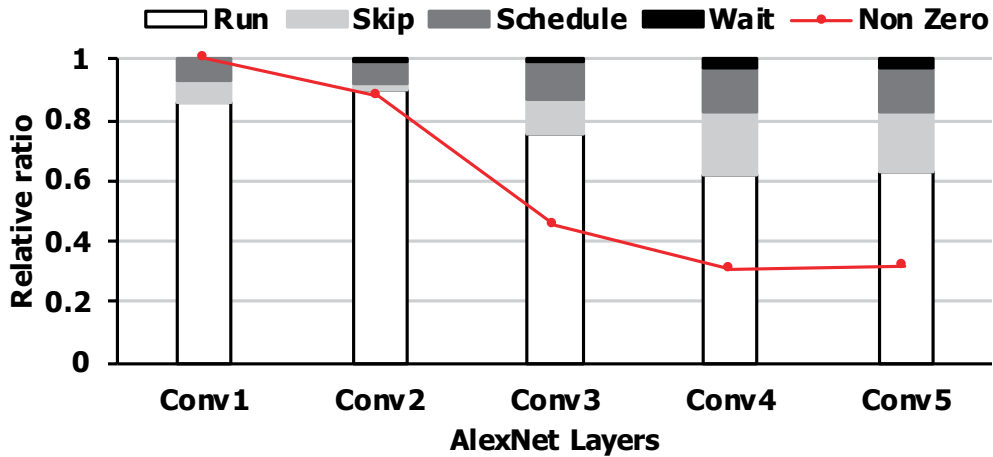


Figure 5.15 Utilization breakdown: AlexNet.

is because in the PE group, the zero-activation skip operation is performed every four activations while consuming a constant overhead of a clock cycle. Thus, when there is a large number of zero activations, as in Conv4 or Conv5, it is often the case that zero skipping, without computations, spends cycles only to skip four consecutive zero activations. As the figure shows, the cycle overhead can amount to approximately 20%. In future work, we will work to reduce these overhead cycles.

Figure 5.16 shows the average number of cycles a PE group spends to process a $A_{1 \times 1 \times 16}$ input activation chunk in AlexNet. Due to the difference in the number of non-zero activations, convolutional layers yield different distributions. For instance, Conv2 has a peak near 15 and 16 cycles owing to the high ratio of non-zero activations shown in Figure 5.15. On the contrary, Conv4 and Conv5 show similar distributions with peaks near five cycles, which can also

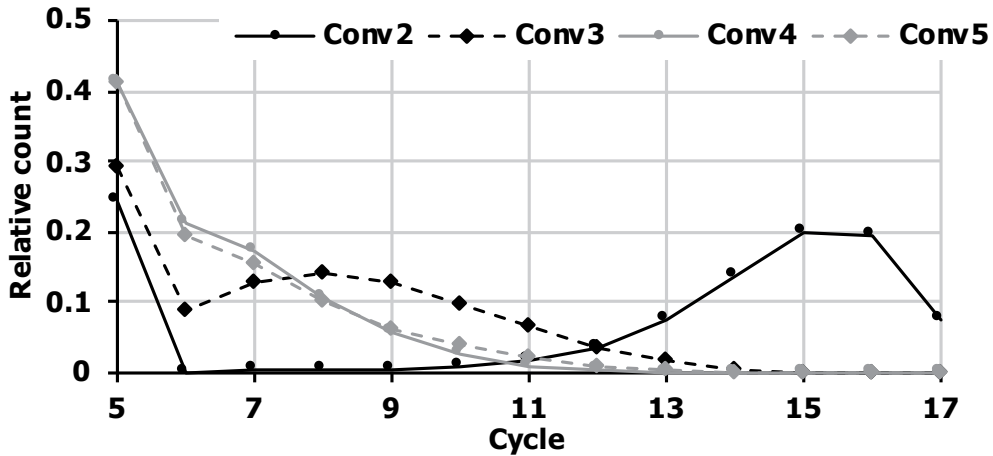


Figure 5.16 Execution cycles when processing a chunk of $A_{1 \times 1 \times 16}$ input activations.

be explained by the low ratio of non-zero activations in Figure 5.15.

5.5 Conclusion

In this work, we proposed a hardware accelerator called OLAcel. It implements outlier-aware quantization, which provides a majority of data with fine-grained quantization while maintaining the precision of important outliers. OLAcel is based on 4-bit MAC units, and performs 4-bit dense computations on a majority of the data. To efficiently handle high-precision outliers, it has two mechanisms at the levels of the PE group and cluster. The PE group, equipped with an outlier MAC unit, performs computations with a single outlier weight without additional cycles, incurring a cycle overhead only when multiple outlier weights need to be handled. The outlier PE group performs

computations with outlier activations using high-precision MAC units. The accumulation of partial sums from the normal and outlier PE groups is performed in a pipelined manner to avoid a coherence problem. Our experiments show that OLAcel reduces energy consumption by 43.5% and 27.0% on AlexNet compared with the state-of-the-art 16-bit and 8-bit zero-aware accelerators, respectively. It provides further energy reduction in VGG-16 (56.7%/36.3% with 16/8 bits) and ResNet-18 (62.2%/49.5%), where the performance degradation due to the first convolutional layer is mitigated and the energy benefit of reduced precision is amplified due to large on-chip memory. The experiments also show that OLAcel has the potential for scalability on large-scale problems.

Chapter 6

Precision Highway for Ultra Low-Precision Quantization

6.1 Introduction

This work is available at [arXiv:1812.09818](https://arxiv.org/abs/1812.09818). The existing quantization methods suffer from a problem called *accumulated quantization error* where large quantization errors get accumulated across layers, making it difficult to enable ultra-low precision in deep neural networks.

In order to address this problem, we propose a novel concept called *precision highway* where an end-to-end path of high-precision information reduces the accumulated quantization error thereby enabling ultra-low-precision computation. Our proposed work is similar to recent studies [19, 61] which propose utilizing pre-activation residual networks, where skip connections are kept in full precision while the residual path performs low-precision computation. Compared with these works, our proposed method offers a generalized concept of high-precision information flow, namely, precision highway, which can be applied to not only the pre-activation convolutional networks but also both the

post-activation convolutional and recurrent neural networks. Our contributions are as follows.

- We propose a novel idea of network-level approach to quantization, called *precision highway* and quantitatively analyze its benefits in terms of the propagation of quantization errors and the difficulty of convergence in training based on the shape of loss surface.
- We provide the detailed analysis of the energy and memory overhead of precision highway based on the state-of-the-art hardware accelerator model. According to our experiments, the overhead is negligible while offering significant improvements in accuracy.
- We apply precision highway to both convolution and recurrent networks. We report a 3-bit quantization of ResNet-50 without accuracy loss and a 2-bit quantization with a very small accuracy loss. We also provide the sub 4-bit quantization results of long short-term memory (LSTM) for language modeling.

6.2 Proposed Method

In *precision highway*, we build a path from the input to output of a network to enable the end-to-end flow of high-precision activation, while performing low-precision computation. Our proposed method was motivated (1) by a residual network where the signal, i.e., the activation/gradient in a forward/backward pass, can be directly propagated from one block to another [62] and (2) by the

LSTM, which provides an uninterrupted gradient flow across time steps via the inter-cell state path [63]. Our proposed method focuses instead on improving the accuracy of quantized network by providing an end-to-end high-precision information flow.

In this section, we first describe the precision highway in the cases of residual network (section 6.2.1) and recurrent neural network (section 6.2.2). Then, we discuss practical issues to be addressed before application to other networks in section 6.2.3.

6.2.1 Precision Highway on Residual Network

In the case of a residual network, we can form a precision highway by making high-precision skip connections. In this subsection, we explain how high-precision skip connections can be constructed to reduce the accumulated quantization error.

In the conventional residual block shown in Figure 6.1 (a), quantization (denoted as $Q_k^{[0,1]}$, k -bit linear quantization in range from 0 to 1) is applied to all of the activations after the activation function. In the figure, thick (thin) arrows represent high-precision (low-precision) activations. As the figure shows, the input of a residual block is first quantized, and the quantized input ($x + e$ in the figure), which contains the quantization error e , enters both the skip connection and residual path. The output of a residual block, y , is calculated as follows:

$$y = F(x + e) + x + e = F(x) + x + e_r + e, \quad (6.1)$$

where $F()$ represents a residual function (typically, 2 or 3 consecutive convo-

lutional layers). For simplicity of explanation, we assume that $F(x + e)$ can be decomposed into $F(x) + e_r$, where e_r represents the resulting quantization error of the residual path incurred by the quantization operations on the residual path as well as the quantization error in the input, e . As the equation shows, output y has two quantization error terms, that of residual path, e_r , and that of the skip connection, e .

Figure 6.1 (b) shows our idea of high-precision skip connection. Compared with Figure 6.1 (a), the difference is the location of the first quantization operation in the residual block. In Figure 6.1 (b), quantization is applied only to the residual path after the bifurcation to the residual path and skip connection. As shown in the figure, the skip connection now becomes a thick arrow, i.e., a high-precision path. The proposed idea gives the output of the residual block as follows:

$$y = F(x + e) + x = F(x) + x + e_r. \quad (6.2)$$

As Equation 6.2 shows, the proposed idea eliminates the quantization error of skip connection e . Thus, only the quantization error of the residual path e_r remains in the output of the residual block. Note that all of the input activations of the residual path are kept in low precision. It enables us to perform low-precision convolution operations in the residual path. We keep high-precision activation only on the skip connection and utilize it only for the element-wise addition. As will be shown in our experiments, the overhead of computation and memory access cost is small since the element-wise addition is much less expensive than the convolution on the residual path, and the low-precision ac-

tivation is accessed for the computation on the residual path.

As will be shown later, our method gives a smaller quantization error, and the gap between the quantization error of the existing method and that of ours becomes wider across layers. Because of the reduction of the accumulated quantization error, the proposed method offers much better accuracy than the state-of-the-art methods with an ultra-low precision of 2 and 3 bits.

Note also that, as shown in figure 6.1 (c), our idea can be applied to other types of residual blocks, including the full pre-activation residual block [62] as proposed in some recent works [11, 19]. However, our idea is general in that it is applicable to recurrent networks as well as post-activation convolutional networks. Especially, our proposed idea is advantageous over the existing ones since hardware accelerators tend to be designed assuming as the input non-negative input activations enabled by ReLU activation functions [16, 33]. Contrary to the existing works [11, 19], we provide a detailed analysis of the effect of precision highway.

6.2.2 Precision Highway on Recurrent Neural Network

Figure 6.2 illustrates how the precision highway can be constructed on the LSTM [63]. In time step t , the LSTM cell takes, as an input, new input x_t , along with the results of the previous time step, output h_{t-1} and cell state c_{t-1} . First, it calculates four intermediate signals: i (input gate), f (forget gate), g (gate gate), and o (output gate). Then, it produces two results, c_t and h_t , as

follows:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}), \quad (6.3a)$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}), \quad (6.3b)$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}), \quad (6.3c)$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}), \quad (6.3d)$$

$$c_t = f_t \odot c_{(t-1)} + i_t \odot g_t, \quad (6.3e)$$

$$h_t = o_t \odot \tanh(c_t), \quad (6.3f)$$

where σ represents a sigmoid function, \odot the element-wise multiplication, W the weight matrix, and b the bias.

In the conventional LSTM operation, as Figure 6.2 (a) shows, the quantization (gray box denoted by Q_k with the output value range as the superscript) is applied to all of the activations before computation. The results of a time step, c_t and h_t , are calculated based on such inputs with quantization errors. More specifically, cell state c_t is calculated with the quantized, i.e., low-precision, inputs of c_{t-1} , f , i , and g . Thus, cell state c_t accumulates the quantization errors of those inputs. In addition, output h_t also accumulates the quantization errors from its inputs, c_t and o . Then, they are propagated to the next time steps. Thus, we have the problem of accumulated quantization error across the time steps. Such an accumulation of quantization error will prevent us from achieving ultra-low precision.

Figure 6.2 (b) shows how we can build the precision highway in the LSTM cell. The figure shows that the quantization operation is applied only to the in-

puts of matrix multiplication (a circle denoted with \times in the figure). Thus, all of the other operations and their input activations are in high precision. Specifically, when calculating c_t , the inputs are not quantized, which reduces the accumulation of quantization error on c_t . The computation of h_t can also reduce the accumulation of quantization error by utilizing high-precision inputs. The construction of such a precision highway allows us to propagate high-precision information, i.e., cell states c_t and outputs h_t , across time steps.

Note that we benefit from low-precision computation by performing low-precision matrix multiplications (in Equations 6.3a-6.3d), which dominate the total computation cost. In our proposed method, all of the element-wise multiplications in Equations 6.3e and 6.3f are performed in high precision. However, the overhead of this high-precision element-wise multiplications is negligible compared with the matrix multiplication in Equations 6.3a-6.3d. In addition, this method can be applied to other types of recurrent neural networks. For instance, the GRU [64] can be equipped with a precision highway, in a way similar to that shown in Figure 6.2 (b), by keeping high-precision output h_t while performing low-precision matrix multiplications and high-precision element-wise multiplications.

6.2.3 Practical Issues with Precision Highway

In order to generalize our proposed idea to other networks in real applications, we need to address the following issues. First, in the case of feed-forward networks with identity path, our precision highway idea is applicable regardless

of pre-activation or post-activation structure. We can exploit the benefit of reduced precision by applying quantization in front of matrix multiplications, while maintain the accuracy by handing the identity path in high precision. Second, in the case of non-residual feed-forward networks, the precision highway can be constructed by equipping them with additional skip connections. In the case of networks with multiple candidates for the precision highway, e.g., DenseNet, which has multiple parallel skip connections [65], we need to address a new problem of selecting skip connections to form a precision highway, which is left for future work.

6.3 Training

In this section, we describe weight quantization and fine-tuning for weight/activation quantization.

6.3.1 Linear Weight Quantization based on Laplace Distribution Model

Figure 6.3 illustrates that a Laplace distribution can well fit the distributions of weights in full-precision trained networks. Thus, we propose modeling the weight distribution with Laplace distribution and selecting quantization levels for weights based on a Laplace distribution.

Given a distribution of weights and a target precision of k bits, e.g., 2 bits, the quantization levels are determined as follows. First, the quantization levels for k bits are pre-computed for the normalized Laplace distribution. We de-

termine quantization levels that minimize L2 error on the normalized Laplace distribution. For instance, in case of the 2-bit quantization, the error is minimized when four quantization levels are placed evenly with a spacing of $1.53 \times \mu$, where μ is the mean of the absolute value of weights. Given the distribution of weights and the pre-calculated quantization levels on the normalized Laplace distribution for the given k bits, we determine the real quantization levels by multiplying the pre-computed quantization levels and the mean of the absolute value of weights.

Our proposed weight quantization is similar to the one in [11]. Compared to it, ours is simpler in that only Laplace distribution model is utilized, and our experiments show that the precision highway together with the proposed simple weight quantization gives outstanding results.

6.3.2 Fine-tuning for Weight/Activation Quantization

Our quantization is applied during fine-tuning after training a full-precision network. As the baseline, we adopt the fine-tuning procedure in [24], where we perform incremental/progressive quantization. In contrast to [24], we first quantize activations and then weights in an incremental quantization. In addition, for each precision configuration, we perform teacher-student training to improve the quantized network [24, 66]. As the teacher network, we utilize a deeper full-precision network, e.g., ResNet-101, compared to the student network, e.g., quantized ResNet-50. Note that, during fine-tuning, we apply quantization in forward pass while updating full-precision weights during backward

pass.

6.4 Experiments

6.4.1 Experimental Setup

We implemented the proposed method in PyTorch and Caffe2. We use two types of trained neural networks, ResNet-18/50 for ImageNet and an LSTM for language modeling [38, 51, 52]. We evaluate 4-, 3-, and 2-bit quantizations for the networks.

For ResNet, we did test with single center crop of 256x256 resized image. We compare our proposed method with the state-of-the-art methods [10, 11, 19, 24, 61]. Note that, for the teacher-student training, we use the same teacher network for both the baseline method (our implementation) [24] and ours. We also evaluate the effects of increasing the number of channels [25] to recover from accuracy loss due to quantization. As in the previous works [11, 19, 20, 24, 25, 61, 67], we do not apply quantization to the first and last layers.

The LSTM has 2 layers and 300 cells on each layer. We used the Penn Tree-bank dataset and evaluated the perplexity per word. We compared the state-of-the-art method in [68] and our proposed method.

6.4.2 Analysis of Accumulated Quantization Error

Figure 6.4 shows the quantization errors across layers in ResNet-50 when applying the state-of-the-art 4-bit quantization to activations. We prepared, from the same initial condition, two activation-quantized networks (one with preci-

sion highway and the other with low precision skip connection) where weights are not modified and only activations are quantized to 4 bits. As the metric of the quantization error, we utilize a metric based on the cosine similarity between the activation tensor of corresponding layer in the full-precision and quantized networks, respectively.

As the figure shows, in the existing method, the quantization errors become larger for deeper layers. It is because the quantization error generated in each layer is propagated and accumulated across layers. We call this *accumulated quantization error*. The accumulated errors become larger with more aggressive quantization, e.g., 2 bits, and cause poor performance, i.e., 4.8 % drop [24] from the top-1 accuracy of the full-precision ResNet-50 for ImageNet classification.

The accumulation of quantization errors is an inherent characteristic of a quantized network in both feed-forward and feed-back networks. In the case of a recurrent neural network, the quantization errors are propagated across time steps. As shown in Figure 6.4, our proposed precision highway significantly reduces the accumulated quantization errors, which enables 3-bit quantization without accuracy drop and much better accuracy in 2-bit quantization than the existing methods.

6.4.3 Loss Surface Analysis of Quantized Model Training

Figure 6.5 visualizes the complexity of loss surface depending on the existence of precision highway. We obtained the figures by applying the method proposed by Li et al. [69]. Each figure represents loss surface seen from the local minimum we obtained from the training, i.e., the weight vector of the final trained model. The origin of the figure at (0, 0) corresponds to the weight vector of the local minimum. As shown in the figure 6.5 (d), the precision highway gives better loss surface (having lower and smoother surface near the minimum point and steep and simple surface elsewhere) than the existing quantization method. This characteristic helps stochastic gradient descent (SGD) method to quickly converge to a good local minimum offering better accuracy than the existing method.

6.4.4 Evaluating the Accuracy of Quantized Model

Laplace	Teacher	Highway	ResNet-18	ResNet-50
✓			61.66 / 84.28	70.50 / 89.84
✓	✓		62.66 / 85.00	71.70 / 90.39
✓		✓	65.83 / 86.71	72.99 / 91.19
✓	✓	✓	66.71 / 87.40	73.55 / 91.40
Full-precision			70.15 / 89.27	76.00 / 92.98
Zhuang’s (ours)			60.06 / 83.34	69.04 / 89.14
Zhuang’s (ours) + Teacher			61.21 / 84.36	70.48 / 89.83

Table 6.1 2-bit quantization results. Top-1 / Top-5 accuracy [%].

Table 6.1 shows the accuracy of 2-bit quantization for ResNet-18/50. We

evaluate each of our proposed methods, Laplace, teacher, and highway, as shown in the table. When the highway box is unchecked the skip connection is branched after the quantization and when the teacher box is unchecked, we use the conventional cross-entropy loss. Compared with the full-precision accuracy, our 2-bit quantization (when all the methods were applied) gives a top-1 accuracy of 73.55 %, which is within 2.45 % of the full-precision accuracy and much better than the state-of-the-art method (Zhuang’s 70.8 %) having a top-1 accuracy loss of 4.8 %. Note that Zhuang’s implemented all the methods, incremental/progressive quantization and teacher-student training, in [24]. We presents the accuracy results of our own implementations of Zhuang’s method under the same amount of training time. Zhuang’s (ours) implemented only incremental and progressive methods while Zhuang’s + Teacher utilized our teacher network.

Table 6.1 shows the effects of the precision highway. Compared with our solution supporting only Laplace and teacher, the highway provides an additional gain of 1.85 % (71.70 % to 73.55 %) in the top-1 accuracy of ResNet-50. The effects of the Laplace method can be evaluated by comparing the result of our implementation of Zhuang’s (69.04 % of top-1 accuracy in ResNet-50) and that of our solution adopting the Laplace model (70.50 %) because these are the same except for the weight quantization method, i.e., *tanh* vs. Laplace based model. The Laplace method gives 1.46 % better accuracy. The table indicates that ResNet-18 also benefits from our proposed methods like ResNet-50.

Table 6.2 compares the additional accuracy loss of quantization methods

	ResNet-18	ResNet-50
Ours	3.44	2.45
DoReFa [10]	7.6	9.8
Zhuang’s [24]	-	4.8
PACT [61]	5.8	4.7
PACT_new [11]	3.4	2.7
Bi-Real [19]	12.9	-

Table 6.2 Comparison of accuracy loss in 2-bit activation / weight quantization. Bi-real applies 1-bit activation / weight quantization.

with respect to full-precision accuracy. The table shows that ours significantly outperform the methods without precision-highway (DoReFa, Zhuang’s, and PACT). PACT_new and Bi-Real utilize high-precision skip connections on pre-activation residual networks. Thus, they show comparable results to ours¹. Note that our results in the table are obtained from the conventional post-activation residual network, which demonstrates the generality of our proposed precision highway. As will be shown below for the LSTM, our proposed method is generally applied to recurrent networks as well as feed forward ones.

Table 6.3 shows the impact of the precision of the precision highway. We obtained the results by varying the highway precision (without retraining) after obtaining the results with the full-precision highway. The table shows that 2-bit quantization with the 8-bit highway gives only 0.09 % and 0.40 % drops in the top-1 accuracy for ResNet-18 and ResNet-50, respectively, from that of

¹We performed 1-bit activation/weight quantization for the post-act style ResNet-18. For a fair comparison, we didn’t apply the teacher-student and progressive quantization method and instead adopted BN-retraining proposed in Bi-Real Net. Our 1-bit activation/weight ResNet-18 gives 56.73 / 80.11 % of Top-1/Top-5 accuracy, which is by 0.33 / 0.61 % higher than the result of Bi-Real Net, respectively.

		Full	8-bit	6-bit
ResNet-18	4-bit	71.05 / 90.16	71.07 / 90.20	70.54 / 89.77
	3-bit	70.29 / 89.54	70.08 / 89.51	69.39 / 88.95
	2-bit	66.71 / 87.40	66.62 / 87.33	65.26 / 86.47

		Full	8-bit	6-bit
ResNet-50	4-bit	76.92 / 93.44	76.69 / 93.27	76.25 / 93.13
	3-bit	76.20 / 93.09	76.08 / 93.03	75.33 / 92.63
	2-bit	73.55 / 91.40	73.15 / 91.34	72.79 / 91.20

Table 6.3 Impact of highway precision (y-axis: low precision and x-axis: highway precision). Top-1 / Top-5 accuracy [%].

the 2-bit quantization with the full-precision highway. Most importantly, our 3-bit quantization (with the 8-bit highway) gives the same accuracy as the full-precision network, i.e., 76.08 % in ResNet-50, which means that our proposed method reduces the precision of the ResNet-50 from 4 bits with [24] down to 3 bits even with the 8-bit highway.

	Full	3-bit	2-bit	Zhuang’s (ours) 2-bit
wResNet-18	74.60 / 91.85	75.49 / 92.45	73.80 / 91.56	70.81 / 90.02
wResNet-50	77.78 / 93.87	78.45 / 94.28	77.35 / 93.69	75.54 / 92.71

Table 6.4 Accuracy of wide ResNet-18 and ResNet-50 with quantization. Top-1/Top-5 accuracy [%].

Table 6.4 shows the effects of two times wider channel under 2-bit quantization. We first doubled the number of channels in ResNet-18 and ResNet-50, and then quantized them with our methods. As the table shows, the wide ResNets give better accuracy than the full-precision ones even for 2-bit quantization, i.e., 73.80 % (77.35 %) in Table 6.4 vs. 70.15 % (76.00 %) of the

full precision in Table 6.1 for ResNet-18 (ResNet-50). It would be worth investigating how to minimize the channel size while meeting the full-precision accuracy with ultra-low precision, which is left for future work.

	(4,4)	(3,4)	(3,3)	(2,3)	(2,2)
Without Highway	97.21	98.14	105.33	107.45	133.25
With Highway	95.94	96.29	100.77	102.55	114.44

Table 6.5 Perplexity of quantized LSTM. (x,y) means x-bit weight / y-bit activation.

Table 6.5 lists the quantization results for the LSTM. We varied the bit configuration (weight, activation) and obtained the perplexity results (the lower, the better). The table shows that our proposed method significantly reduces the perplexity. Compared with the perplexity of full-precision model (92.84), our 4-bit quantization gives a very small increase of 3.3 % (92.84 to 95.94). The precision highway provides more gain for a more aggressive quantization. Specifically, it reduces the perplexity by 14.1 % (from 133.25 to 114.44) in the 2-bit quantization, (2,2). Compared with the state-of-the-art quantization of a similar LSTM [68]², ours offers much better results, i.e., a much smaller increase in perplexity, e.g., a 23.3 % increase (92.84 to 114.44 in Table 6.5) vs. a 39.4 % increase (109 to 152 in [68]) in perplexity for 2-bit quantization.

6.4.5 Hardware Cost Evaluation of Quantized Model

Figure 6.6 shows the chip area cost and energy consumption of ResNet-18 at different levels of precision on the state-of-the-art hardware accelerator [26]. The accelerator is synthesized at 65 nm, 250 MHz, and 1.0 V. Each processing element (PE) consists of a multiply-accumulate (MAC) unit and local buffers. The PEs share global on-chip 2 MB static random access memory (SRAM) at 16-bit precision and the size of which is adjusted proportional to the precision. As the figure shows, the reduced precision offers significant reduction in chip area, e.g., 82.3 % reduction from 16 to 3 bits and energy consumption, e.g., 73.1 % from 16 to 3 bits. In the 2-bit case where the overhead of precision highway is the largest, the precision highway incurs only 3.9 % additional energy consumption due to the high-precision data while offering 4.1 % better accuracy than the case that precision highway is not adopted. The accelerator is already equipped large internal buffer for partial sum accumulation. Thus, precision highway incurs additional energy consumption mainly on the accesses to on-chip SRAM and main memory (dynamic random access memory, DRAM).

Table 6.6 compares the number of operations in three neural networks used in our experiments. The table explains why the high-precision operations incur such a small overhead in energy consumption. As the table shows, it is because the frequency of high-precision operations is much smaller than that of low-precision operations. For instance, the 2-bit LSTM network has one high-

²Note that we compared their relative change from the full-precision perplexity because the full-precision perplexity of the state-of-the-art method (109) is different from that of ours (92.84).

	LSTM (300)	ResNet-18	ResNet-50
Low-precision MAC	720 K	6.89 G	15.1 G
High-precision Add	0.3 K	9.68 M	62.0 M
Non-linear Op*	1.5 K	7.48 M	39.9 M
Elt-wise Multi*	0.9 K	-	-

Table 6.6 Number of operations. * denotes the high-precision operation.

precision (in 32 bits) element-wise multiplication for every 800 2-bit multiplications.

6.5 Conclusion

In this work, we proposed the concept of end-to-end precision highway which can be applied to both feedforward and feedback networks and enable ultra-low precision in deep neural networks. The proposed precision highway reduces quantization errors by keeping high-precision activation from the input to output of the network with small computation costs. We described how it reduces the accumulated quantization error and presented quantitative analyses in terms of accuracy and hardware cost as well as training characteristics. Our experiments showed that the proposed method outperforms the state-of-the-art methods in the 3- and 2-bit quantizations of ResNet-18/50 and 2-bit quantization of an LSTM model. We believe that our work will serve as a step toward mixed precision networks for computational efficiency.

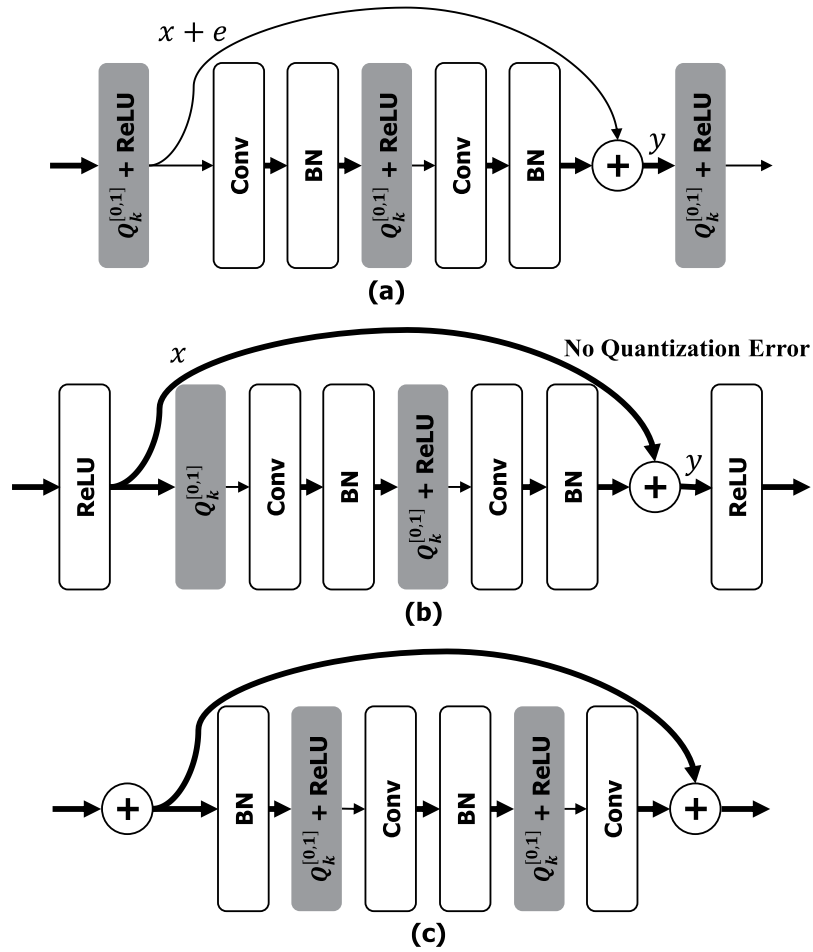


Figure 6.1 Comparison of conventional quantization and our proposed idea on residual network.

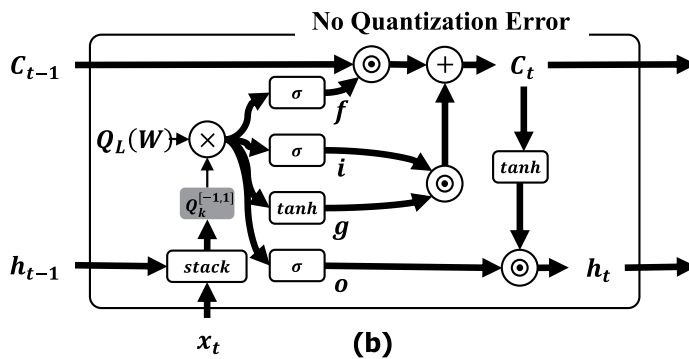
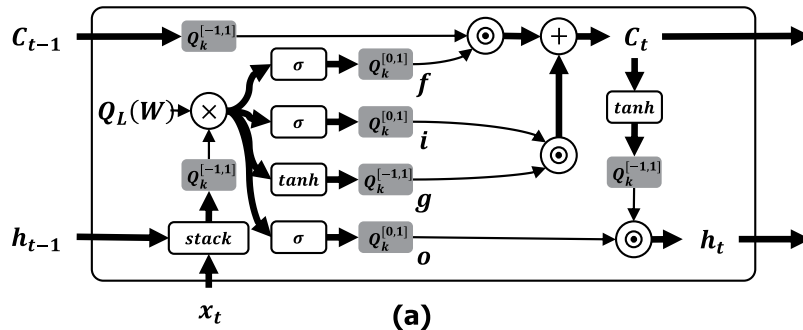


Figure 6.2 Comparison on residual network.

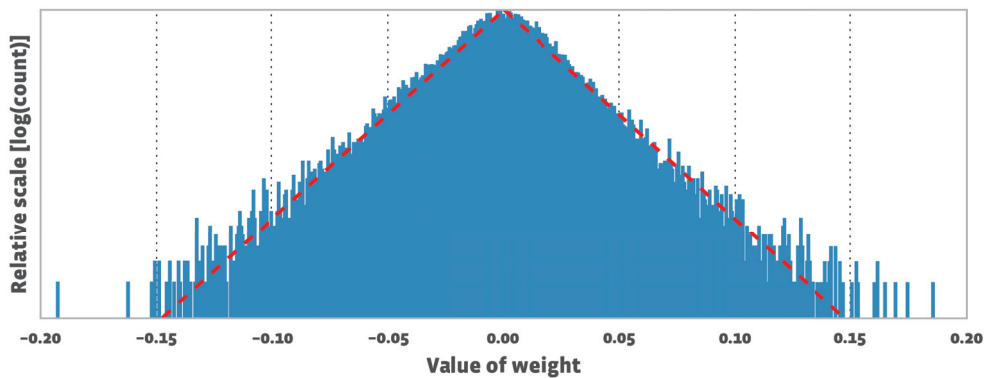


Figure 6.3 Weight histogram and Laplace approximation (dashed line) of the convolutional layer of a trained full-precision ResNet-50.

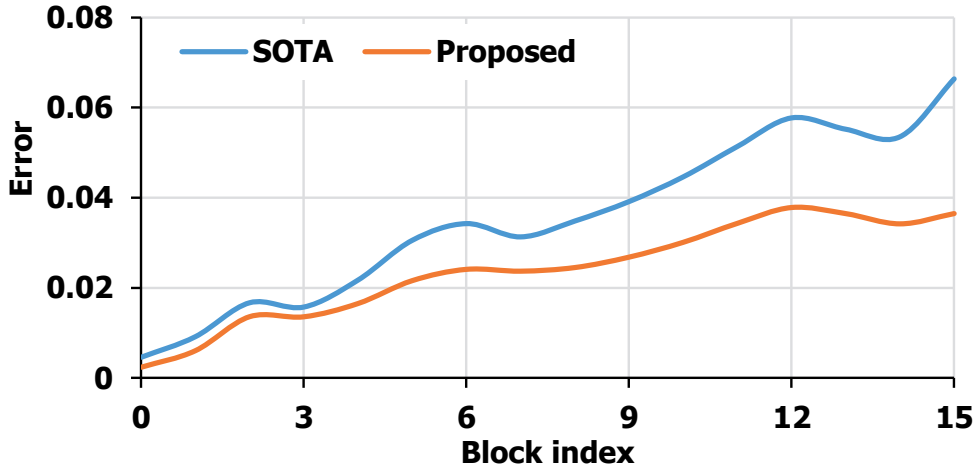


Figure 6.4 Quantization error accumulation across residual blocks in ResNet-50.

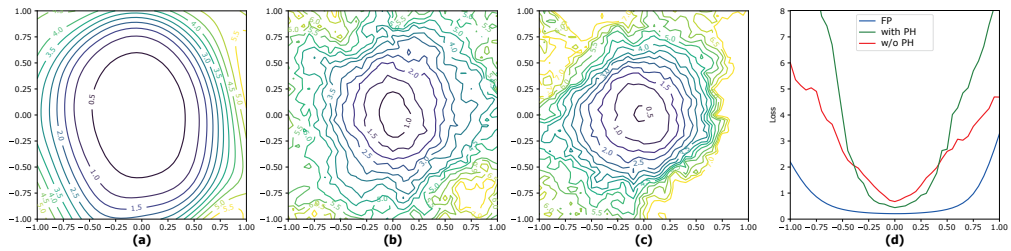


Figure 6.5 Loss surface of ResNet-18 on Cifar-10: (a) full-precision model (FP), (b) 1-bit activation and 2-bit weight quantized model (1A2W) without precision highway (PH), (c) 1A2W with precision highway, and (d) cross-section of loss surface.

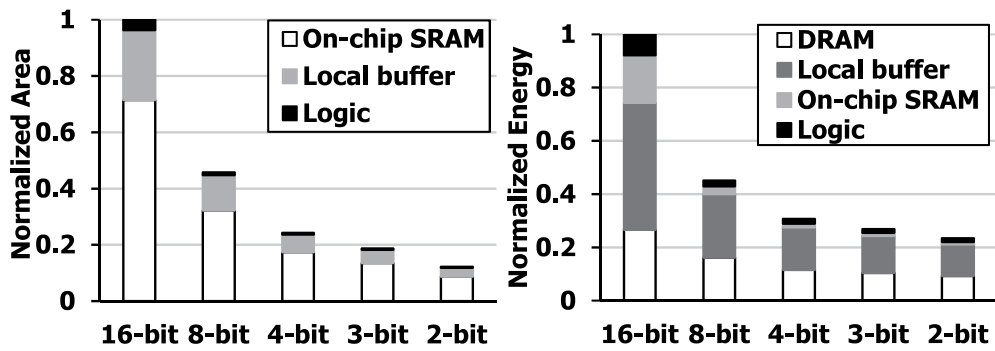


Figure 6.6 Comparison of chip area and energy consumption on the hardware accelerator.

Chapter 7

Towards Sub-4-bit Quantization of Optimized Mobile Networks

7.1 Introduction

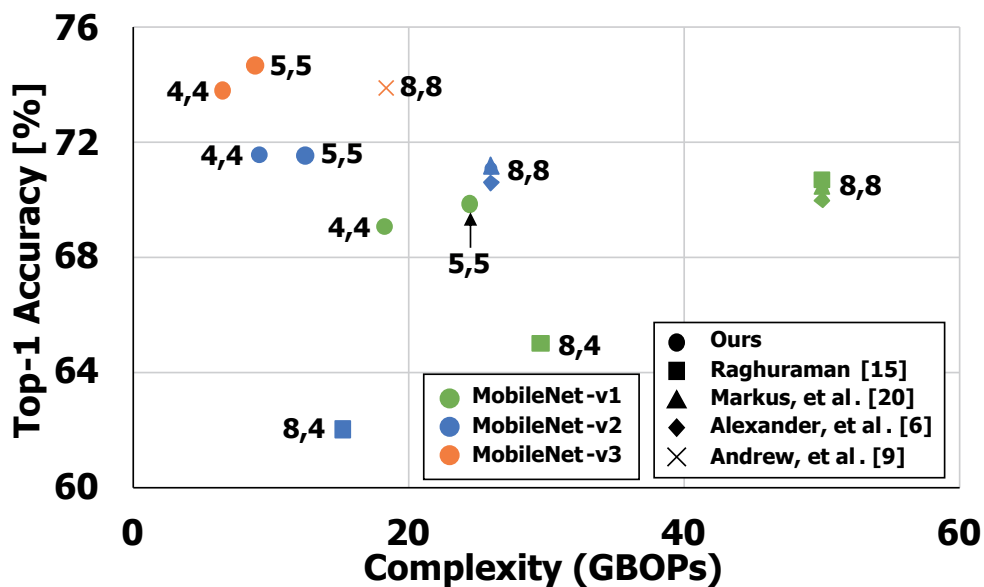


Figure 7.1 Comparison of accuracy and complexity (GBOPs, [3]) of the quantized network. The tuple (a,w) represents the bit-width of activation and weight, respectively.

This work is currently under review in CVPR’2020 conference [18]. Unfortunately, the previous quantization methods do not work on state-of-the-art optimized networks like MobileNet-v3 [70]. It is because they focused on simple neural network architectures having ReLU activation and conventional convolution. Meanwhile, recent state-of-the-art optimized networks adopt novel structures like depth-wise separable convolution [71], inverted residual block with linear expansion layer [72], squeeze-excitation module, and h-swish activation function [70]. As a result, recent optimized mobile models outperform the previous ones by a large margin, e.g., MobileNet-v3 has only 38.5 % of MACs compared to MobileNet-v1 [70, 71]. However, according to our experiments, those novel structures make 4-bit and lower precision quantization challenging in the existing quantization methods.

In this work, we propose two novel ideas that enable 4-bit quantization for the optimized mobile networks. First, we report that the primary reason of accuracy loss in sub-4-bit quantization is the poor running mean and variance. In training time, we obtain running mean and variance and, in test time, use them in the batch normalization layer. Our analysis shows that the running mean and variance are hurt by the activation instability induced by weight quantization (AIWQ).

Fine-tuning is essential to recover from the accuracy loss of sub-4-bit quantization [11, 12]. However, the fine-tuning tends to increase the frequency of weights near the quantization thresholds. In such a case, a small change of the full-precision weight (by back-propagation) near the quantization threshold is

amplified by the weight quantization, i.e., the rounding operation. This causes the convolution output to significantly change across batches, which we call the AIWQ problem.

The activation instability perturbs the activation distribution, which finally prevents us from obtaining good running mean and variance in training time. Such poor running mean and variance make batch normalization less effective in test time. In order to address this problem, we propose a novel training method called BLast (BN Last) that tries to minimize the effects of AIWQ by judiciously performing the training of batch normalization (BN) layer while progressively freezing the weights sensitive to the AIWQ problem.

Second, we identified the limitations of the state-of-the-art trainable methods of linear quantization in terms of rounding/truncation errors and negative activation support and propose a novel quantization method called differentiable and unified quantization (DuQ). The state-of-the-art methods try to minimize either rounding error [12] or truncation error [11]. It is desired to minimize both errors for further reduction in precision. In addition, existing methods like PACT and QIL support only a limited value range of output activation, e.g., non-negative or $[0,1]$. Thus, they cannot be applied to novel activation functions, e.g., h-swish function, which utilizes negative activations and novel structures like squeeze-and-excitation module and expansion layer which both require producing both positive and negative activations. Our proposed DuQ method resolves the above problems without limiting the value range while minimizing both rounding and truncation errors in a differentiable manner.

As Figure 7.1 shows, our proposed methods enable 4-bit quantization of optimized networks at high accuracy thereby pushing mobile networks towards more resource-efficient regime compared with the state-of-the-art quantization solutions.

7.2 BLast Training

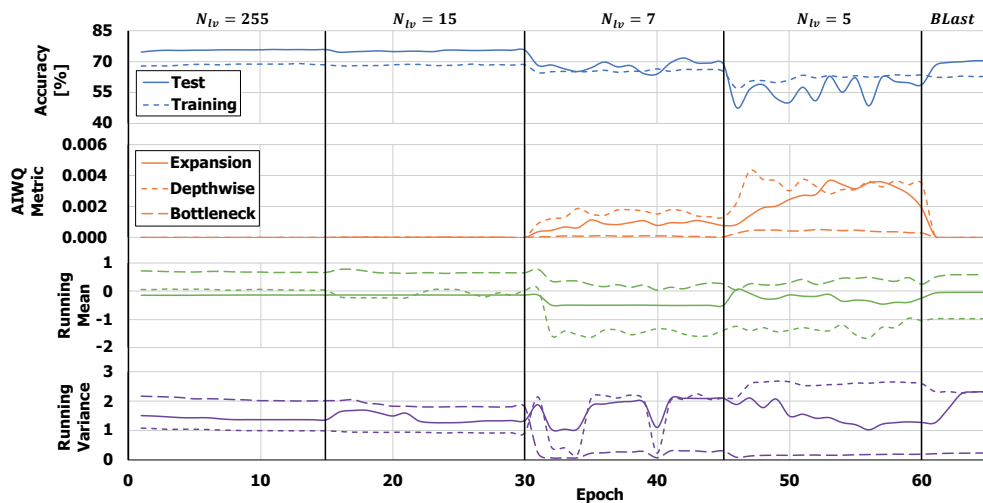


Figure 7.2 Top-1 accuracy [%], AIWQ metric, and running mean/variance during the fine-tuning for quantization where N_{lv} represents the number of available quantization levels. Running mean/variance are extracted from the arbitrary channels of batch normalization layer after depth-wise convolution in the 2nd inverted residual module.

In this section, we first demonstrate the AIWQ problem is strongly correlated with accuracy degradation in reduced precision. Then, we present a metric to measure activation instability. We also propose a training method called BLast which, based on the activation instability metric, controls the training

of each layer to minimize the effect of activation instability thereby offering better test accuracy.

7.2.1 Notation

First, we explain the notation used in the work. The subscripts l, i , and o represent the layer index, input channel index, and output channel index, respectively, and W, I , and O represent weight, input activation, and output activation, respectively. In addition, the superscript t denotes the iteration index. For instance, $W_{l,o,i}^t$ represents the weights updated at t -th iteration on l -th layer, o -th output channel and i -th input channel. In the case of weight-quantized convolution layer, the output activation can be expressed as follows:

$$O_{l,o}^t = \sum_i Q(W_{l,o,i}^t) \otimes I_{l,i}^t, \quad (7.1)$$

where \otimes is the convolution operation and Q is the quantization function. Note that, for simplicity, activation quantization, which is performed after BN layer, is not shown in the equation.

7.2.2 Observation

Figure 7.2 (Accuracy) shows the accuracy of MobileNet-v3 for Cifar-100 dataset [73]. The accuracy is measured during fine-tuning in progressive weight quantization [24] where the number of quantization levels of weights gets gradually reduced from 255 to 5 while using full-precision activation. The orange line, denoted by Training represents the accuracy obtained by using the mean and variance of the current training batch at the batch normalization layers.

Meanwhile, the blue line, denoted by Test represents the accuracy measured by using the running mean and variance which are obtained during training, i.e., continuously updated at each batch for later test time usage.

The accuracy curves in Figure 7.2 show that, at each case of levels, e.g., $N_{lv} = 15$, both training and test accuracy are recovered gradually as fine-tuning advances. However, when the number of available levels gets down to 5, the test accuracy significantly oscillates and fails to converge while the training accuracy can be recovered as usual with small oscillations.

According to our analysis which will be given in the next subsection, this is mainly due to poor running mean and variance obtained during training. The problem is that the effects of weight update, due to back-propagation, could be amplified by the quantization operation, i.e., rounding operation. Specifically, if a weight near the quantization threshold is updated to change its value cross the threshold, then the quantization will result in different quantized weight values before and after weight update. Thus, the results of convolution operation will change due to weight update and quantization, in fact, perturbing the statistics of output activation, which finally yields poor running mean and variance.

In reduced precision, the above activation instability induced by weight quantization (AIWQ) becomes more significant because the two quantized weight values will become farther apart due to the larger spacing between quantization levels which is proportional to the inverse of $2^{\text{bit-width}}$. Thus, the lower the precision gets, the more activation instability can be incurred.

The perturbed activation prevents us from obtaining good running mean and variance during training. If those poor statistics of mean and variance are used in test time, then it is likely to obtain poor test accuracy as demonstrated in Figure 7.2. Note that the effects of perturbed activation on the next layer tends to be small due to the batch normalization layer. Only the running mean and variance are mainly affected by the activation instability since they are obtained by exponential moving average of mean and variance of the un-normalized activation. This is why the test accuracy in Figure 7.2 heavily oscillates while the training accuracy gives much smaller oscillations.

7.2.3 Activation Instability Metric

We present a metric to quantify per-layer activation instability and use it in order to (1) prove that AIWQ is strongly correlated with the test accuracy of reduced precision model (in Figure 7.2) and (2) evaluate the per-layer sensitivity when determining the order of freezing the weights during training (to be explained in Section 7.2.4).

In order to measure per-layer activation instability, we obtain KL divergence between the output of weight-quantized convolution before and after weight update as follows. First, we approximate the output of weight-quantized convolution before and after weight update as uni-variate Gaussian random

variables as follows.

$$p_o^t \approx N\left(\mu_o, \sigma_o \parallel \sum_i Q(W_{l,o,i}^t) \otimes I_{l,i}^t\right), \quad (7.2)$$

$$q_o^t \approx N\left(\mu'_o, \sigma'_o \parallel \sum_i Q(W_{l,o,i}^{t-1}) \otimes I_{l,i}^t\right). \quad (7.3)$$

where p_o^t and q_o^t represent the approximate distribution of convolution output on a channel after (at iteration index t) and before (at index $t - 1$) weight update, respectively. Note that the same input activation $I_{l,i}^t$ is used to evaluate the effect of weight quantization on the convolution output.

We first calculate per-output channel KL divergence between p_o^t and q_o^t . Then, as shown below, we compute the layer-wise AIWQ metric D^l by averaging the per-output channel KL divergence across all the output channels of the current layer in the current training batch.

$$D^l = E_o^t [D_{KL}(p_o^t \parallel q_o^t)]. \quad (7.4)$$

The AIWQ metric in Figure 7.2 illustrates how the AIWQ metric varies during the fine-tuning in the case of the 2nd inverted residual block of MobileNet-v3. As shown in the figure, the instability gets increased in 7- and 5-level quantization, which empirically proves that weight quantization in low precision can incur significant perturbation on the convolution output, i.e., make output activation unstable. As Figure 7.2 shows, such an instability causes running mean and variance¹ to fluctuate, which prevents us from obtaining good running mean and variance during training. When comparing the accuracy, AIWQ

¹We show mean/variance on three sampled output channels.

metric and mean/variance in Figure 7.2, they are closely correlated in low precision, i.e., $N_{lv} = 5$, which shows the poor running mean and variance due to the activation instability can degrade test accuracy.

7.2.4 BLast Training

In order to improve the accuracy of reduced precision networks, we propose a novel training method which aims at minimizing the AIWQ effect. As shown previously, the running mean and variance are affected by AIWQ. Our basic idea is to additionally train the batch normalization and quantization layer, after freezing all the other weights in the last training step of training, which is called BN last training, in short, BLast. In the last training step, there is no AIWQ since the weights are frozen. Thus, the additional training can offer better running mean and variance.

Learning Rate

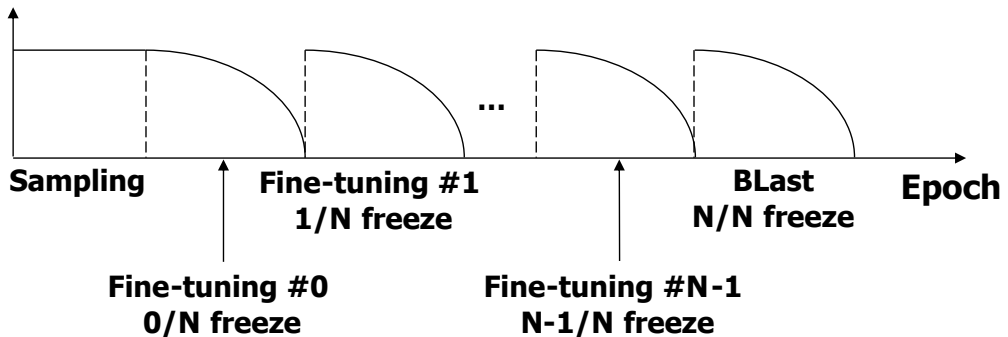


Figure 7.3 BLast training method.

In order to further reduce the impact of AIWQ, we determine the layer-wise order of weight freezing considering the per-layer AIWQ metric in Eqn.

7.4. Figure 7.3 shows how our method works. When BLast is triggered, as shown in the figure, we start a sampling stage where we evaluate the per-layer AIWQ metric for each layer. After the sampling stage, we perform fine-tuning in an initial stage (typically, 10-15 epochs) without freezing weights. Then, after sorting all the weight layers in terms of the per-layer metric, we perform weight freezing stages by selecting the most sensitive layers (the ones having the largest AIWQ metric values) and freezing their weights. As shown in the figure, we perform N freezing stages. Thus, in each stage, $1/N$ of all the layers (not weight-frozen ones) are selected from the sorted layer list and their weights are frozen. Then, we perform BN layer training for all the frozen layers while also training the other un-frozen layers. After finishing a freezing stage, we select the next set of sensitive layers (another $1/N$ of all the layers) and repeat the same procedure until there is no more un-frozen layer left. We call the last freezing stage BLast stage since only BN layers are trained in the stage.

Figure 7.3 shows that there are $N+2$ stages including one sampling, one initial and N freezing ones. We call this method of freezing weights in the order of AIWQ metric and training BN layer in a layer-wise manner, BLast+. BLast+ helps to reduce AIWQ noise for not only running mean and variance but also the weights of subsequent layers. As will be shown in our experimental results, BLast+ helps to improve the accuracy of reduced precision networks.

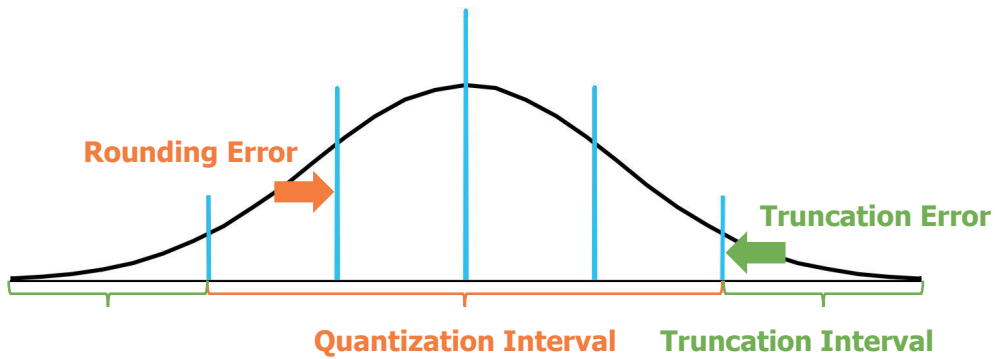


Figure 7.4 Two error sources of quantization.

7.3 Differentiable and Unified Quantization

7.3.1 Rounding and Truncation Errors

Figure 7.4 shows that quantization has two error sources, rounding and truncation. As shown in the figure, depending on whether a data is located inside/outside of quantization interval, rounding/truncation error is incurred. Both errors are closely related with each other. For instance, the smaller truncation error (by increasing the truncation threshold) can incur the larger rounding error (due to the larger quantization interval). Thus, the quantization method needs to find the best trade-off between the two errors in minimizing the training loss.

7.3.2 Limitations of State-of-the-Art Methods

Our goal is to realize a differentiable quantization which minimizes the task loss. There are two representative differentiable quantization methods, parameterized clipping activation function (PACT) [11] and quantization interval learning (QIL) [12]. In both methods, differentiable parameters and quantiza-

tion interval are updated through back-propagation to minimize the task loss.

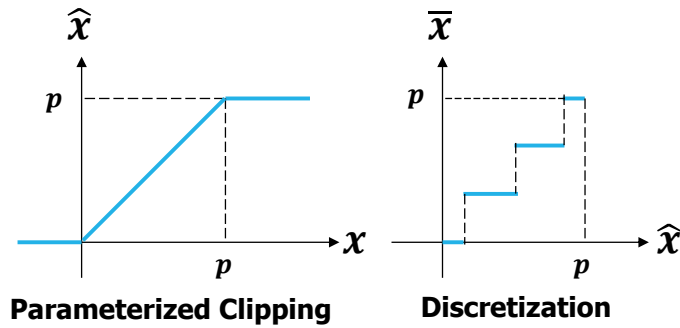


Figure 7.5 PACT algorithm.

Figure 7.5 shows that PACT has two steps, parameterized clipping to determine the truncation threshold and discretization to assign quantization levels. As the figure shows, PACT has a trainable parameter p for the truncation threshold. During back-propagation, the gradient of data in the truncation interval is transferred to p to update it. However, the gradient of data in quantization interval $[0, p]$ is bypassed by straight-through-estimator (STE). Thus, the gradient in the quantization interval is not utilized to update the trainable parameter p . Due to this limitation, p tends to keep increasing to minimize the truncation error. PACT tries to mitigate this problem by introducing an L2 regularization term p^2 to the training loss.

As Figure 7.6 shows, QIL has two stages, transformation and discretization. In transformation, the input data is linearly transformed based on slope ($= 1/2d$) and offset ($= (1 - c)/d$). The transformed input is clipped between 0 and 1. The clipped data are quantized in discretization stage. Since c and d are differentiable, quantization interval $[c - d, c + d]$ can be learned through

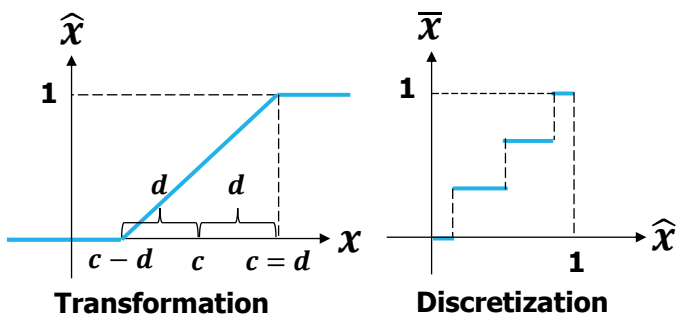


Figure 7.6 QIL algorithm.

back-propagation. In contrast to PACT, QIL ignores the gradient outside of quantization interval. In other words, QIL tries to reduce only rounding error in the quantization interval.

In addition, both PACT and QIL have critical limitations in supporting new activation functions, e.g., h-swish and new structures, e.g., squeeze-and-excitation used on state-of-the-art optimized networks. It is because transformation stage forces the data to be mapped to $[0, 1]$ in [12] or $[0, p]$ in [11]. Thus, it is not applicable to those activation functions and structures requiring negative activations. For instance, h-swish activation, squeeze-and-excitation module and linear expansion layer require representing negative activations. Since both methods assume only non-negative activations, they do not provide competitive results for optimized networks in the reduced precision like 4 bits.

7.3.3 Proposed Method: DuQ

Our proposed method, called differentiable and unified quantization (DuQ), learns quantization and truncation intervals through back-propagation exploit-

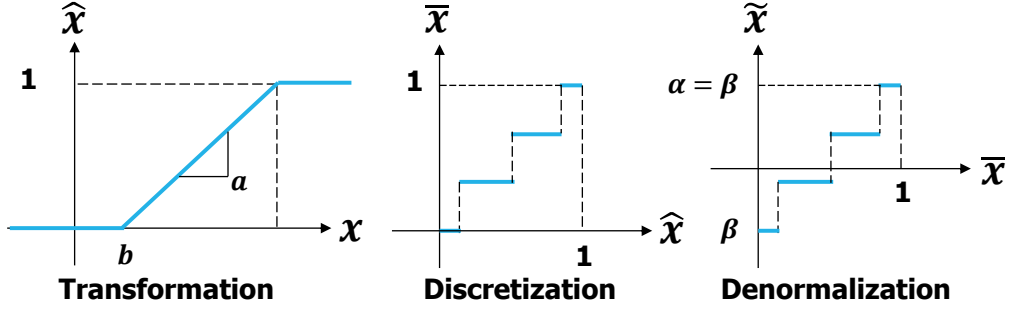


Figure 7.7 Proposed DuQ algorithm.

ing all the gradients over the entire value range of activation. Figure 7.7 shows that DuQ has three stages, transformation, discretization, and denormalization. As shown in the figure, it has four parameters; transform scale (a), transform offset (b), denormalization scale (α), and denormalization offset (β).

Our proposed DuQ method improves upon QIL by additionally performing denormalization and utilizing gradients on truncation interval. Thus, we use the same task loss as QIL. The two stages of transformation (Eqn. 7.5) and discretization (Eqn. 7.6) are identical to those of QIL except that scale a and offset b are used in transformation stage instead of center c and width d in QIL. As Eqns. 7.5 and 7.7 show, we use softplus function for a and $alpha$ in order to make them positive values, which proves effective in improving the stability of transformation stage. Eqn. 7.6 represents the discretization stage where N_{lv} is the number of quantization levels.

$$\hat{x} = clip\left(\frac{x-b}{a'}, 0, 1\right), \quad a' = softplus(a), \quad (7.5)$$

$$\bar{x} = \frac{1}{N_{lv} - 1} \cdot \text{round}((N_{lv} - 1) \cdot \hat{x}), \quad (7.6)$$

$$\tilde{x} = \alpha' \cdot \bar{x} + \beta, \quad \alpha' = \text{softplus}(\alpha). \quad (7.7)$$

Our proposed DuQ method allows us to utilize full value range of activation including negative ones. To do that, in denormalization stage, the discretized data can be mapped to an arbitrary range through scale α and offset β as shown in Eqn. 7.7. Unlike PACT or QIL, DuQ utilizes all the gradients across the entire activation data including not only data in truncation interval but also those in quantization interval. Thus, a good quantization interval can be learned via back-propagation considering the trade-off between rounding and truncation errors.

7.3.4 Handling Negative Values

We apply the proposed DuQ to the quantization of weights and activations. In case of weights, we apply DuQ to the absolute value of weight and multiply sign after the quantization. Thus, the quantization levels are assigned symmetrically without using one level. For instance, in case of 4-bit precision, we use 15 integer levels in $[-7, 7]$. In addition, we fix β to zero. Otherwise, each of quantized weights cannot be mapped to a single integer operation with scaling [4].

There are two cases of encountering negative activations: the activation of linear layer without activation function, e.g., expansion layer in the inverted

residual block, and the output activation of h-swish function [70]. For the output activation of linear layer, we apply DuQ to activations as in the case of weight quantization to obtain an odd number of symmetrical quantization levels. On the other hand, for the output activation of h-swish function whose distribution is not symmetrical, in order to avoid wasting quantization levels for large negative values, we propose an additional technique called negative padding.

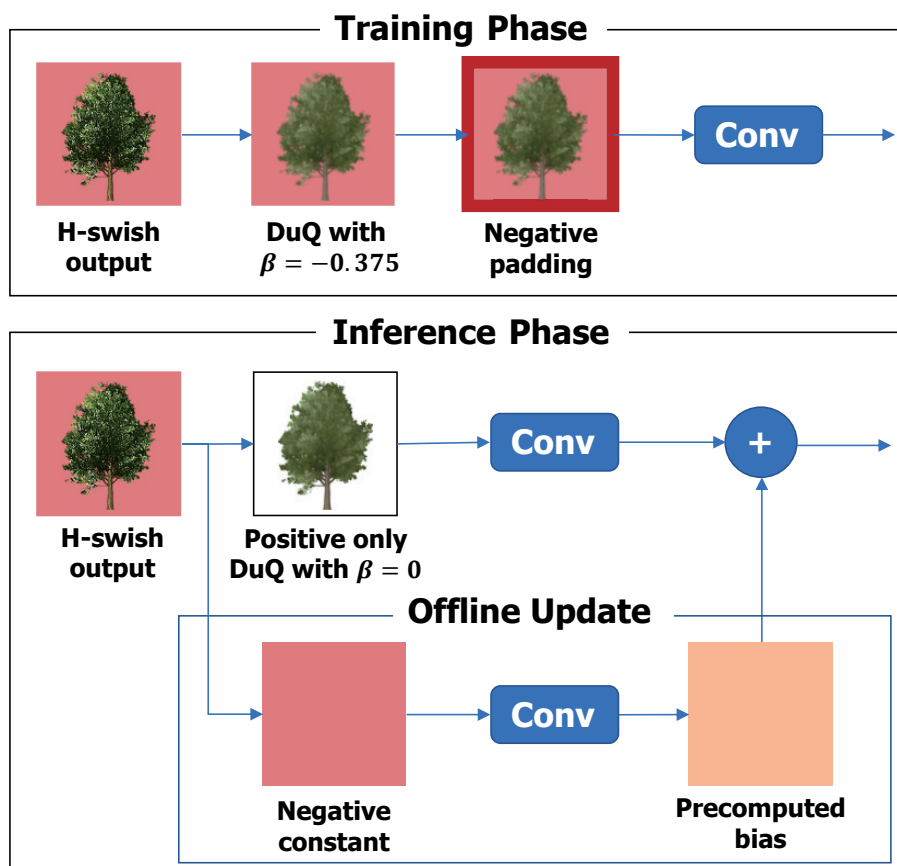


Figure 7.8 Negative padding for h-swish function.

H-swish function gives an asymmetrical distribution of activations having small negative and large positive value ranges. Thus, a symmetrical allocation of quantization levels will waste quantization levels allocated to un-used large negative values. In order to avoid this waste, we propose shifting input activations to the convolution by the minimal value of h-swish function, -0.375^2 and, instead of zero padding, padding the negative value, -0.375 .

Figure 7.8 (Inference Phase) shows how convolution is performed in inference under this method. As the figure shows, the original input (h-swish output) can be decomposed into the negative constant ($=-0.375$) and non-negative quantized input ($=Q(I + 0.375)$ where I is the original activation before quantization). Note that we apply DuQ to the non-negative input thereby avoiding the waste of quantization levels for large negative values. As shown in the bottom of the figure, the convolution output of the negative constant input can be computed offline and absorbed to the bias of convolution or to the batch normalization layer. Note also that our proposed method yields only non-negative activations during inference. Thus, as the figure shows, we set β to 0 in inference while setting it to -0.375 in training.

Quantization of h-swish function output may not guarantee the usage of zero as one of quantization levels. However, our proposed negative padding makes the minimum quantization level of input activation zero. Thus, the proposed method can also be beneficial to zero skipping solutions like zero skip-

²Hard swish function, h-swish is given as $x*\text{ReLU6}(x+3)$ which gives the minimum output of -0.375 [70].

ping hardware accelerators [30, 32] to improve inference speed.

7.4 Experiments

We implemented the proposed methods in PyTorch 1.2.0, and demonstrate their effectiveness by measuring the accuracy of the quantized networks.

7.4.1 Accuracy on ImageNet Dataset

We apply quantization to the well-known optimized CNNs, MobileNet-v1 to v3 and MNasNet. We also quantize ResNet-18 as a representative example of conventional CNN. The networks are trained on 4-GPU with 256 batch, SGD with momentum and cosine learning rate decay with warmup [74, 75]. In order to improve the accuracy, we adopt the progressive quantization method [24] that gradually decreases bit-width to 8, 5, and 4 bits during fine-tuning, and use knowledge distillation [76] using ResNet-101 as teacher. We use exponential moving average of parameters with momentum 0.9997 [77], and all networks are trained using BLast+ and DuQ (with negative padding if applicable). We train the model for 15 epochs every progressive quantization and BLast+ fine-tuning step. Note that all the layers of the networks are quantized including the first and last layers. We used 8-bit data only for the input image of the first convolution layer and the activation of squeeze-excitation module.

Table 7.1 shows the accuracy of quantized networks under our proposed methods. Compared to the full precision (Full in the table), our 4-bit models give comparable accuracy with less than 1% of accuracy loss. To the best of

Table 7.1 Top-1 / Top-5 accuracy [%] of the quantized networks on ImageNet

	MobileNet-v1	MobileNet-v2	MobileNet-v3	MNASNet	ResNet-18
Full	68.848 / 88.740	71.328 / 90.016	74.728 / 92.136	73.130 / 91.276	69.546 / 89.090
8-bit	70.164 / 89.370	72.352 / 90.636	75.166 / 92.498	73.742 / 91.756	71.246 / 89.988
5-bit	69.866 / 89.058	71.540 / 90.058	74.690 / 92.092	73.378 / 91.244	71.672 / 90.168
4-bit	69.056 / 88.412	71.564 / 90.398	73.812 / 91.588	72.244 / 90.584	70.968 / 89.914

the authors’ knowledge, this is the first study to achieve less than 1% top-1 loss on 4-bit MobileNet-v3.

When we train the network without BLast+, the 4-bit accuracy of MobileNet-v3 gives only 71.720 % / 90.386 % for top-1/top-5 accuracy, having more than 2 % accuracy loss compared to the accuracy obtained with BLast+. The table also shows that, compared with full-precision, our methods do not lose accuracy on 4-bit MobileNet-v2 and has less than 1% loss on 4-bit MobileNet-v1 and MNASNet, and better accuracy on 4-bit ResNet-18.

Our 8-bit model accuracy, 75.166 % is superior to that (73.8 %) in [70]. We think it is mainly because fused-batch normalization, adopted in [70], has an adverse effect on the accuracy of low precision model, which will be elaborated in more detail in the next subsection.

We compare the accuracy of MobileNet v1 and v2 with the existing works in Table 7.2 where aN and wM represent N-bit activation and M-bit weight quantization and c and l channel-wise and layer-wise quantization, respectively. * represents the post-training quantization. Compared to [78], our 4-bit layer-wise quantization gives comparable accuracy to the 8-bit models of the previous work, and better accuracy than the channel-wise 8/4-bit models. In ad-

Table 7.2 Top-1 accuracy [%] comparison of existing works on MobileNet-v1 and MobileNet-v2.

	MobileNet-v1	MobileNet-v2
[78], a8,w8,c	70.7	71.1
[78], a8,w8,l	70.0	70.9
[78], a8,w4,c	64.0	58.0
[78], a4,w8,c	65.0	62.0
[79]*, a8,w8	69.99	70.60
[80]*, a8,w8	70.5	71.2
Our, a5,w5	69.866	71.540
Our, a4,w4	69.056	71.564

dition, our 4-bit MobileNet-v2 outperforms even the 8-bit models of existing works.

7.4.2 Discussion on Fused-BatchNorm

Batch normalization layer normalizes input activation using batch statistics, i.e., mean and variance, and applies scale and shift (Eqn. 7.8) [81]. After training, the running mean and variance of the batch normalization layer can be absorbed to scale and shift. In addition, the combined scale and shift terms can be absorbed by scaling convolution kernel weights and adding to the bias of the prior convolution layer (Eqn. 7.9). This technique, called fused-batchnorm [4], was proposed to remove the overhead of batch normalization layer in inference.

$$\hat{x} = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta. \quad (7.8)$$

$$x = W \otimes a, \hat{x} = \left(\frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} W \right) \otimes a + \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \epsilon}}. \quad (7.9)$$

$$\hat{x}' = Q\left(\frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} W \right) \otimes a + \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \epsilon}}. \quad (7.10)$$

When we quantize weights under fused-batchnorm, we need to apply quantization to the weight with batch norm scaling, as shown in Eqn. 7.10. However, according to our observation, all models under fused-batchnorm failed to converge when 4-bit quantization is applied to the weights with fused-batchnorm. It is because the weights under fused-batchnorm tend to have wider value ranges, due to the additional scaling, than the original weights. We think that, in order to exploit fused-batchnorm in 4-bit and lower precision, it is desirable to apply channel-wise quantization, which is the beyond of the scope of this work and left as future work.

7.4.3 Ablation Study

In order to evaluate the proposed method in detail, we perform ablation study on CIFAR-100 dataset with ResNet-18, MobileNet-v2 and MobileNet-v3 models. Each network has the full-precision top-1 accuracy of 74.39 %, 75.21 % and 76.22 %, respectively.

Figure 7.9 compares the top-1 accuracy of PACT, QIL and DuQ. Note that, in order to evaluate the performance of DuQ, all the three methods run under BLAST and we extend PACT and QIL to support symmetric quantization for the

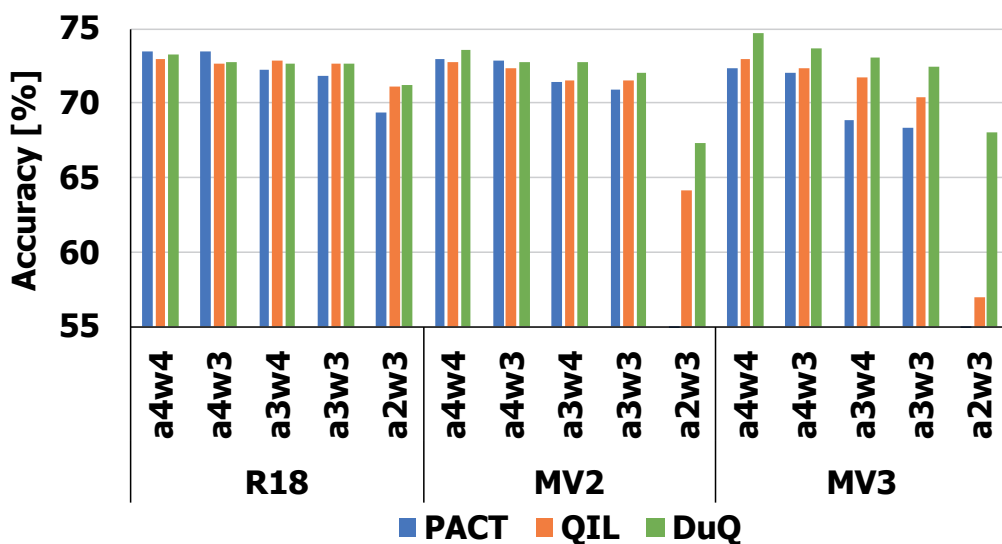


Figure 7.9 Comparison of quantization algorithms under BLast.

activation of linear output as explained in Section 7.3.4. As the figure shows, ResNet-18 gives higher accuracy than MobileNet-v2 and v3 because ResNet-18 is simpler than MobileNets and thus easier to train in low precision. As shown in the figure, PACT and QIL under BLast give comparable accuracy to DuQ on ResNet-18. PACT under BLast offers slightly better accuracy in 3-bit weights due to the weight quantization called statistics-aware weight binning (SAWB). Meanwhile, QIL gives better activation quantization than PACT. In all the three networks, DuQ gives better accuracy when the bit-width is small. Moreover, it significantly outperforms the other two methods in MobileNet-v3 because, as explained in Section 7.3.4, DuQ with negative padding can support the full value range of h-swish function without wasting quantization levels.

In order to evaluate the effects of BLast(+), we measure the test accu-

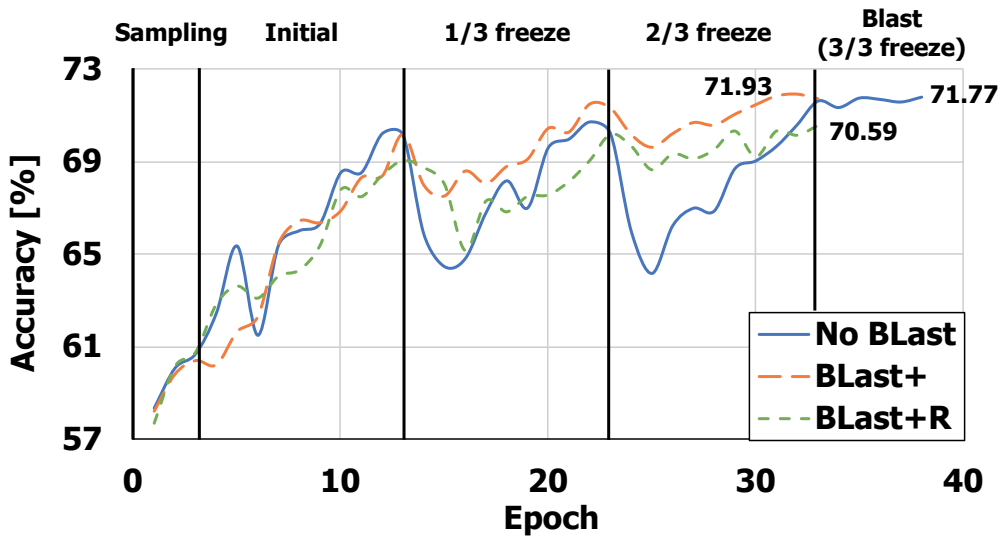


Figure 7.10 Evaluation of BLast(+).

racy in the fine-tuning of 3-bit quantization for MobileNet-v3. As Figure 7.10 shows, we compare three cases. "No BLast" does not freeze weights during fine-tuning. BLast+ freezes weights in the order of AIWQ metric and BLast+R freezes weights in the reverse order of AIWQ metric. The three cases share the same learning rate schedule. As shown in the figure, BLast+ helps to stabilize test accuracy during training. It gives much smaller oscillations at the 3rd (denoted by "1/3 freeze") and 4th ("2/3 freeze") stages where the weights having large AIWQ metric are first frozen minimizing the AIWQ effect. Meanwhile, BLast+R gives the worst accuracy, which reconfirms the effectiveness of freezing the weights with large AIWQ metric earlier than the others.

7.5 Conclusion

In this work, we propose a novel training method called BN last (BLast) and a quantization method called differentiable and unified quantization (DuQ). BLast aims at minimizing the effect of activation instability induced by weight quantization, and DuQ enables the quantization of negative activations found in optimized networks. Based on the proposed methods, we can quantize the state-of-the-art optimized network, MobileNet-v3 into 4 bits with 0.916 % of accuracy loss. In addition, other optimized mobile networks such as MobileNet-v2 and MNasNet are also quantized into 4 bits with negligible accuracy loss. We hope our proposed methods can contribute to advancing towards 4-bit and lower precision computation on embedded devices.

Chapter 8

Conclusion

This dissertation aims to minimize the accuracy loss of quantization to exploit its benefit. Several novel quantization algorithms are proposed, e.g. weight-entropy-based quantization, outlier-aware quantization, and differentiable and unified quantization, DuQ. Besides, additional optimization techniques are proposed to minimize the accuracy drop resulting from quantization, such as a structural concept called precision-highway and a training methodology called BLast(+).

In weight-entropy-based quantization (section 3), the flexible multi-bit quantization method presented enables automated optimization without modification of the original network structure. This work shows the potential of multi-bit quantization that enables sub-6-bit quantization of very deep networks, i.e. AlexNet, GoogLeNet, and ResNet-101 with less than 1 % of top-1 accuracy loss.

In chapters 4 and 5, a novel non-linear quantization algorithm is proposed, called value-aware quantization or outlier-aware quantization. This algorithm is designed considering the characteristics of distributions of weight and acti-

vation, and it shows outstanding results of post-training quantization. Additionally, this helps to reduce memory consumption during training, which enables ResNet-50 training with a 7.5x reduction of memory cost for activation with negligible accuracy loss. Also, the dedicated hardware accelerator, OLAcel, gives a significant amount of energy consumption reduction on ResNet-18 by 62.2 % and 49.5 % compared to the state-of-the-art 16-bit and 8-bit zero-aware accelerators, respectively. This work presents the potential of a hardware accelerator with reduced precision computation.

In chapters 6 and 7, the proposed studies presented a state-of-the-art accuracy of quantized networks based on linear quantization. The precision highway forms a piece of end-to-end high-precision information that minimizes accumulated quantization error across a network. Based on the proposed method, ResNet-18/50 can be quantized into 3-bit without accuracy loss and the 2-bit model also gives a comparable result, whereas LSTM models are also successfully quantized. A novel training method, BN last (BLast) and the differentiable and unified quantization (DuQ) are designed to support reduced precision for the optimized mobile networks, i.e. MobileNet-v2 and MobileNet-v3. BLast is designed to suppress the disturbance of activation instability induced by weight quantization, while the DuQ is applicable for the state-of-the-art optimized structure, e.g. squeeze-excitation module and the swish function. This enables the MobileNet-v3 into 4-bits with less than 1 % accuracy loss. We believe that the proposed algorithms will contribute to the efficient computation of DNNs with reduced precision.

As future work, I plan to extend our work toward quantization-friendly architecture search and quantization algorithms for the BERT model. Automated architecture search is one of the most popular areas of interest in recent [82–84]. Since [82] show the possibility of an automated architecture search, there have been active studies on improving the accuracy of the found model. Furthermore, some studies focused on improving not only accuracy but also other metrics such as performance [84]. To fully utilize the potential of reduced precision, I will extend the idea to an architecture search. In particular, by minimizing the major reason of accuracy degradation of the quantized network, activation instability induced by weight quantization, I hope to find out the quantization-robust network during architecture search.

On the other hand, I will try to optimize the BERT model [85] that shows the outstanding performance for language model and recommendation system. This model is expected to be used frequently, thus it is practically important to optimize the model. Our previous study [17] shows the potential of the quantization for the language model, but I will need to improve our algorithm for the BERT. I will focus on quantization for computation, i.e. linear operator, and embedding table that requires a significant amount of memory.

Bibliography

- [1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [2] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv:1308.3432*, 2013.
- [3] C. Baskin, E. Schwartz, E. Zheltonozhskii, N. Liss, R. Giryes, A. M. Bronstein, and A. Mendelson, “Uniq: Uniform noise injection for non-uniform quantization of neural networks,” *arXiv:1804.10969*, 2018.
- [4] A. Tulloch and Y. Jia, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [5] S. Migacz, “NVIDIA 8-bit inference with TensorRT,” *GPU Technology Conference*, 2017.
- [6] H. Wu, “NVIDIA Low Precision Inference on GPU,” *GPU Technology Conference*, 2019.

- [7] “Snapdragon neural processing engine sdk.” <https://developer.qualcomm.com/docs/snpe/index.html>, 2017. Accessed: 2019-11-16.
- [8] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, “7.1 an 11.5 tops/w 1024-mac butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile soc,” *International Solid-State Circuits Conference (ISSCC)*, 2019.
- [9] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” *European Conference on Computer Vision (ECCV)*, 2016.
- [10] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *arXiv:1606.06160*, 2016.
- [11] J. Choi *et al.*, “Bridging the accuracy gap for 2-bit quantized neural networks,” *arXiv:1807.06964*, 2018.
- [12] S. Jung, C. Son, S. Lee, J. Son, J. Han, Y. Kwak, S. J. Hwang, and C. Choi, “Learning to quantize deep networks by optimizing quantization intervals with task loss,” *Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [13] “Samsung low-power npu solution for ai deep learning.” <https://news.samsung.com/global/samsung-electronics-introduces-a-high-speed-low-power-npu-solution-for-ai-deep-learning>, 2019. Accessed: 2019-11-16.

- [14] E. Park, J. Ahn, and S. Yoo, “Weighted-entropy-based quantization for deep neural networks,” *Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [15] E. Park, S. Yoo, and P. Vajda, “Value-aware quantization for training and inference of neural networks,” *European Conference on Computer Vision (ECCV)*, 2018.
- [16] E. Park, D. Kim, and S. Yoo, “Energy-efficient neural network accelerator based on outlier-aware low-precision computation,” *International Symposium on Computer Architecture (ISCA)*, 2018.
- [17] E. Park, D. Kim, S. Yoo, and P. Vajda, “Precision highway for ultra low-precision quantization,” *arXiv:1812.09818*, 2018.
- [18] E. Park and S. Yoo, “Towards sub-4-bit quantization of optimized mobile networks,” *Computer Vision and Pattern Recognition (CVPR)*, *under review*, 2020.
- [19] Z. Liu *et al.*, “Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm,” *European Conference on Computer Vision (ECCV)*, 2018.
- [20] S. Zhou *et al.*, “Balanced quantization: An effective and efficient approach to quantized neural networks,” *Journal of Computer Science and Technology*, 2017.

- [21] D. Miyashita, E. H. Lee, and B. Murmann, “Convolutional neural networks using logarithmic data representation,” *arXiv:1603.01025*, 2016.
- [22] C. Zhu, S. Han, H. Mao, and W. J. Dally, “Trained ternary quantization,” *International Conference on Learning Representations (ICLR)*, 2017.
- [23] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” *International Conference on Learning Representations (ICLR)*, 2016.
- [24] B. Zhuang *et al.*, “Towards effective low-bitwidth convolutional neural networks,” *Computer Vision and Pattern Recognition(CVPR)*, 2018.
- [25] A. K. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, “WRPN: wide reduced-precision networks,” *arXiv:1709.01134*, 2017.
- [26] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” *International Symposium on Computer Architecture*, 2016.
- [27] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning,” *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.
- [28] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel,

A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, “A configurable cloud-scale DNN processor for real-time AI,” *International Symposium on Computer Architecture (ISCA)*, 2018.

[29] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, “In-datacenter performance analysis of a tensor processing unit,” in *International Symposium on Computer Architecture (ISCA)*, 2017.

[30] J. Albericio, P. Judd, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, and A. Moshovos, “Cnvlutin: Ineffectual-neuron-free deep neural network computing,” *International Symposium on Computer Architecture (ISCA)*, 2016.

- [31] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, “Cambricon-x: An accelerator for sparse neural networks,” *International Symposium on Microarchitecture (MICRO)*, 2016.
- [32] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally, “Scnn: An accelerator for compressed-sparse convolutional neural networks,” *International Symposium on Computer Architecture (ISCA)*, 2017.
- [33] D. Kim, J. Ahn, and S. Yoo, “Zena: Zero-aware neural network accelerator,” *Design & Test*, 2018.
- [34] M. Courbariaux, Y. Bengio, and J. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” in *Neural Information Processing Systems (NIPS)*, 2015.
- [35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Neural Information Processing Systems (NIPS)*, 2012.
- [36] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [37] J. Dai, Y. Li, K. He, and J. Sun, “R-FCN: object detection via region-based fully convolutional networks,” *Neural Information Processing Systems (NIPS)*, 2016.

- [38] W. Zaremba, I. Sutskever, and O. Vinyals, “Recurrent neural network regularization,” *arXiv:1409.2329*, 2014.
- [39] S. Guiaşu, “Weighted entropy,” *Reports on Mathematical Physics*, vol. 2, no. 3, pp. 165–179, 1971.
- [40] S. Guiasu, “Grouping data by using the weighted entropy,” *Journal of Statistical Planning and Inference*, vol. 15, pp. 63–69, 1986.
- [41] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *International Conference on Multimedia (MM)*, 2014.
- [42] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous distributed systems,” *Operating Systems Design and Implementation (OSDI)*, 2016.
- [43] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” *Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [44] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, “Going deeper with embedded FPGA

- platform for convolutional neural network,” *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pp. 26–35, 2016.
- [45] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” *Neural Information Processing Systems (NIPS)*, 2015.
- [46] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, “Building a large annotated corpus of English: The Penn Treebank,” *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.
- [47] B. Ginsburg *et al.*, “NVIDIA Mixed Precision Training on Volta GPUs,” *GPU Technology Conference*, 2017.
- [48] T. Chen, B. Xu, C. Zhang, and C. Guestrin, “Training deep nets with sublinear memory cost,” *arXiv:1604.06174*, 2016.
- [49] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, “The reversible residual network: Backpropagation without storing activations,” *Neural Information Processing Systems (NIPS)*, 2017.
- [50] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, and K. A. Vissers, “Finn: A framework for fast, scalable binarized neural network inference,” *Architecture of Field-Programmable Gate Arrays (FPGA)*, 2017.

- [51] O. Press and L. Wolf, “Using the output embedding to improve language models,” *European Chapter of the Association for Computational Linguistics (EACL)*, 2017.
- [52] H. Inan, K. Khosravi, and R. Socher, “Tying word vectors and word classifiers: A loss framework for language modeling,” *International Conference on Learning Representations (ICLR)*, 2017.
- [53] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Neural Information Processing Systems (NIPS)*, 2019.
- [54] S. Marcel and Y. Rodriguez, “Torchvision the machine-vision package of torch,” *Neural Information Processing Systems (NIPS)*, 2010.
- [55] D. Bakunas-Milanowski *et al.*, “Efficient algorithms for stream compaction on gpu,” *International Journal of Networking and Computing (IJNC)*, 2017.
- [56] J. Konečný *et al.*, “Federated learning: Strategies for improving communication efficiency,” *arXiv:1610.05492*, 2016.
- [57] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “Cacti 6.0: A tool to model large caches,” *HP Laboratories, HPL-2009-85*, 2009.

- [58] “Micron dram calculator.” <http://www.micron.com/support/tools-and-utilities/power-calc>. Accessed: 2019-11-20.
- [59] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Computer Vision and Pattern Recognition*, 2017.
- [60] Y. Ioannou, D. P. Robertson, R. Cipolla, and A. Criminisi, “Deep roots: Improving cnn efficiency with hierarchical filter groups,” *arXiv preprint arXiv:1605.06489*, 2016.
- [61] J. Choi *et al.*, “Pact: Parameterized clipping activation for quantized neural networks,” *arXiv:1805.06085*, 2018.
- [62] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *European Conference on Computer Vision (ECCV)*, 2016.
- [63] F. A. Gers, J. Schmidhuber, and F. A. Cummins, “Learning to forget: Continual prediction with LSTM,” *Neural Computation*, 2000.
- [64] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv:1412.3555*, 2014.
- [65] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” *Computer Vision and Pattern Recognition (CVPR)*, 2017.

- [66] A. K. Mishra and D. Marr, “Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy,” *arXiv:1711.05852*, 2017.
- [67] I. Hubara *et al.*, “Quantized neural networks: Training neural networks with low precision weights and activations,” *arXiv:1609.07061*, 2016.
- [68] Q. He *et al.*, “Effective quantization methods for recurrent neural networks,” *arXiv:1611.10176*, 2016.
- [69] H. Li *et al.*, “Visualizing the loss landscape of neural nets,” *arXiv:1712.09913*, 2017.
- [70] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” *International Conference on Computer Vision (ICCV)*, 2019.
- [71] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv:1704.04861*, 2017.
- [72] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” *Computer Vision and Pattern Recognition (CVPR)*, 2018.

- [73] A. Krizhevsky, V. Nair, and G. Hinton, “The cifar-10 dataset.” <https://www.cs.toronto.edu/~kriz/cifar.html>, 2014. Accessed: 2019-11-16.
- [74] I. Loshchilov and F. Hutter, “SGDR: stochastic gradient descent with warm restarts,” *International Conference on Learning Representations (ICLR)*, 2017.
- [75] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, “Accurate, large minibatch SGD: training imagenet in 1 hour,” *arXiv:1706.02677*, 2017.
- [76] G. E. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv:1503.02531*, 2015.
- [77] P. Izmailov, D. Podoprikin, T. Garipov, D. P. Vetrov, and A. G. Wilson, “Averaging weights leads to wider optima and better generalization,” *Uncertainty in Artificial Intelligence (UAI)*, 2018.
- [78] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *arXiv:1806.08342*, 2018.
- [79] A. Finkelstein, U. Almog, and M. Grobman, “Fighting quantization bias with bias,” *arXiv:1906.03193*, 2019.

- [80] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, “Data-free quantization through weight equalization and bias correction,” *arXiv:1906.04721*, 2019.
- [81] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *International Conference on Machine Learning (ICML)*, 2015.
- [82] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in *International Conference on Learning Representations (ICLR)*, 2017.
- [83] H. Liu, K. Simonyan, and Y. Yang, “DARTS: differentiable architecture search,” *arXiv:1806.09055*, 2018.
- [84] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [85] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” in *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019.

국문초록

딥 뉴럴 네트워크 (DNN)는 활용 범위를 점차 넓혀가며 다양한 분야에 적용되고 있다. 뉴럴 네트워크는 서버 뿐만 아니라 임베디드 기기에서도 널리 활용되고 있으며 이로 인해 뉴럴 네트워크의 효율성을 높이는 것은 점점 더 중요해지는 중이다. 이제 정확도를 유지하면서 속도를 빠르게 하고 에너지 소모를 줄이는 뉴럴 네트워크의 최적화는 필수적 요소로 자리잡았다.

양자화는 가장 효과적인 최적화 기법 중 하나이다. 뉴런의 활성화도 (activation) 및 학습 가중치 (weight)를 저장하는데 필요한 비트 수를 줄임으로써 동일한 양의 데이터 접근과 연산 비용 (칩 면적 및 에너지 소모 등)으로 더 많은 연산이 가능해지며 이로 인해 속도와 에너지 소모를 동시에 최적화할 수 있다. 추후 딥 러닝을 활용하기 위하여 필요할 것으로 예측되는 에너지 효율 및 연산 속도를 만족시키기 위해서 4 비트 혹은 더 적은 정밀도 기반의 양자화 연산이 지대한 공헌을 할 것으로 기대된다.

그러나 양자화의 가장 중요한 단점 중 하나는 데이터의 표현형을 제한하여 자유도가 떨어지게 됨으로서 발생하는 정확도의 손실이다. 이러한 단점을 해결하기 위하여 다양한 연구들이 진행중이다. 최근 일부 연구들은 8 비트의 정밀도에서 뉴럴 네트워크를 활용해 결과를 추론 (inference)하는데 정확도 손실이 거의 없음을 보고하고 있다. 반면 그 외의 다양한 연구들을 통해 4 비트 혹은 더 낮은 정밀도에서 양자화를 적용했을 때 많은 네트워크들의 정확도가 크게 손상되는 현상도 함께 보고되고 있다. 특히 최근 제안된

네트워크들의 경우 성능 향상을 위해 도입한 최적화된 구조가 양자화 하기 어려운 특성을 가져 이러한 현상이 심화된다.

본 논문에서는 양자화된 DNN의 정확도 손실을 최소화하기 위한 다양한 방법들을 제안하였다. 가중 엔트로피 기반 양자화 (Weighted-entropy-based quantization)은 제한된 개수의 양자화 레벨을 최대한 활용하기 위하여 양자화된 데이터의 정보량을 최대화하는 방향으로 양자화를 진행하도록 설계되었다. 이 연구를 통해 아주 깊은 네트워크에서도 뉴런의 활성도와 학습 가중치 모두의 양자화가 적용 가능함을 보였다. 값-의식 양자화 (value-aware quantization), 혹은 예외-의식 양자화 (outlier-aware quantization)는 빈도는 낮지만 큰 값을 가지는 데이터를 큰 정밀도로 저장하는 대신 나머지 데이터에 4 비트 이하의 양자화를 적용하도록 설계된 알고리즘이다. 이는 원본 데이터의 평균과 분산 같은 특성이 양자화된 후에도 유지하도록 도와주어 양자화된 네트워크의 정확도를 유지하는데 기여한다. 이에 더하여 OLAcel이라 명명된 특화 가속기를 제안하였다. 이 가속기는 값-의식 양자화 알고리즘을 통해 양자화된 네트워크를 가속함으로써 정확도 감소는 최소화 하면서 낮은 정밀도의 성능 이득을 최대화한다. 고정밀도-통로 구조 (precision-highway)는 네트워크의 구조를 개선하여 초저정밀도 연산을 수행하면서도 고정밀도 정보 통로를 생성한다. 이는 양자화로 인하여 에러가 누적되는 현상을 완화하여 매우 낮은 정밀도에서 정확도를 개선하는데 기여한다. 학습 기법인 BLast와 미분 가능하고 통합된 양자화 알고리즘 (DuQ)는 MobileNet-v3과 같은 최적화된 모바일형 네트워크를 최적화하기 위하여 제안되었다. 이 방법들을 통해 미미한 정확도 손실만으로 MobileNet-v3의 활성도 및 학습 가중치 모두를 4 비트 정밀도로 양자화하는데 성공하였다.

주요어: 뉴럴네트워크, 최적화, 양자화, 하드웨어 아키텍처, 가속기

학번: 2015-31050

Acknowledgements

어제보다 나은 오늘이 되길 바라면서 하루하루 지내다 보니 벌써 6년의 시간이 지났습니다. 주변 분들의 이해와 격려 덕분에 대학원 생활을 잘 마무리하고 한 명의 연구자로서 출발선에 설 수 있었습니다. 학위 논문을 마무리 지으면서 주어진 이 기회를 빌려 늘 품고 다니던 감사의 마음을 전합니다.

가장 먼저 그동안 저를 지도해주신 유승주 교수님, 정말 감사합니다. 저에게 언제나 본보기자 지향점이 되어주셨고, 앞으로도 그럴 것입니다. 연구자로서, 지도자로서 존경합니다. 석사 과정을 마무리 지을 때 도움을 주셨던 김재준 교수님, 해외 경험의 기회를 주셨던 김재연 박사님, 그리고 박사 심사에 참여하여 귀중한 조언을 해주신 이재욱 교수님, 김장우 교수님, 유민수 교수님, 윤성로 교수님께도 감사의 마음을 전합니다.

연구실 생활 동안 함께 해준 동료들께도 감사의 마음 전합니다. 정 많고 재주 많은 동영이, 박학다식하고 쾌활한 승환 형, 성실하고 든든한 수범이. 좋은 동료들과 함께 연구실 생활을 할 수 있어서 즐겁고 행복했습니다. 꼼꼼하게 신경 써주고 잘 이끌어준 현선 선배와 거리감 없이 유쾌하게 대해주었던 문규 선배, 선배임을 내세우지 않고 함께해준 대현 선배, 다정하고 짹짹하게 잘 따라준 정현이와 스마트하고 자기 일 잘 해내는 택민이와 정현수, 성실하고 꾸준한 한별이 모두 에게도 감사하다는 말 전하고 싶습니다. 그리고 회사에서 오셔서 좋은 모습을 보여주신 형님들, 현승 형님과 지석 형님, 해랑

형님과 원경 형님 덕분에 사회인으로서 갖춰야 할 덕목을 엿볼 수 있었습니다. 이제 막 연구실 생활을 시작한 현영이와 성민이, 남우와 고현수도 지금 꿈꾸는 것들을 이루어 낼 수 있기를 바랍니다. 마지막으로 오랜 시간 함께 했던 태민이. 고뇌를 이겨내고 스스로 만족할만한 성과를 보여줄 수 있기를 바랍니다.

마지막으로 제가 세상에서 가장 사랑하고 존경하는 아버지와 어머니, 역시 가장 사랑하고 자랑스러운 동생. 언제나 믿어주셨고, 안식처가 되어주셨고, 꾸준히 격려해주셨기에 흔들리지 않을 수 있었습니다. 저의 가장 큰 원동력이자 구심점이 되어준 우리 가족들, 언제나 사랑하고 감사합니다.

저와 함께 해주신 모든 분께 감사드립니다. 더 큰 사람이 되어 그동안 받았던 은혜를 갚을 수 있기를 바라면서 앞으로 나아가겠습니다.

2020년 1월.

박은혁 올림.