Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

CAS-2022-5269458

M.Sc. Thesis

A New Logarithmic Quantization Technique and Corresponding Processing Element Design for CNN Accelerators

Longxing Jiang

Abstract

Convolutional Neural Networks (CNN) have become a popular solution for computer vision problems. However, due to the high data volumes and intensive computation involved in CNNs, deploying CNNs on low-power hardware systems is still challenging. The power consumption of CNNs can be prohibitive in the most common implementation platforms: CPUs and GPUs. Therefore, hardware accelerators that can exploit CNN parallelism and methods to reduce the computation burden or memory requirements are still hot research topics. Quantization is one of these methods. One suitable quantization strategy for low-power deployments is logarithmic quantization.

Logarithmic quantization for Convolutional Neural Networks (CNN): a) fits well typical weights and activation distributions, and b) allows the replacement of the multiplication operation by a shift operation that can be implemented with fewer hardware resources. In this thesis, a new quantization method named Jumping Log Quantization (JLQ) is proposed. The key idea of JLQ is to extend the quantization range, by adding a coefficient parameter "s" in the power of two exponents (2^{sx+i}).

This quantization strategy skips some values from the standard logarithmic quantization. In addition, a small hardware-friendly optimization called weight de-zeroing is proposed in this work. Zero-valued weights that cannot be performed by a single shift operation are all replaced with logarithmic weights to reduce hardware resources with little accuracy loss.

To implement the Multiply-And-Accumulate (MAC) operation (needed to compute convolutions) when the weights are JLQ-ed and de-zeroed, a new Processing Element (PE) have been developed. This new PE uses a modified barrel shifter that can efficiently avoid the skipped values. Resource utilization, area, and power consumption of the new PE standing alone are reported. Resource utilization and power consumption in a systolic-array-based accelerator are also reported. The results show that JLQ performs better than other state-of-the-art logarithmic quantization methods when the bit width of the operands becomes very small.

A New Logarithmic Quantization Technique and Corresponding Processing Element Design for CNN Accelerators

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

by

Longxing Jiang
born in ChengDu, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Copyright © 2022 Circuits and Systems Group
All rights reserved.

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF
MICROELECTRONICS

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"A New Logarithmic Quantization Technique and Corresponding Processing Element Design for CNN Accelerators"** by **Longxing Jiang** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 29th November, 2022

Chairman: _____
prof.dr.ir. Rene van Leuken

Advisor: _____
dr. David Aledo Ortega

Committee Members: _____
prof.dr.ir. Rene van Leuken

_____
dr. David Aledo Ortega

_____
prof.dr. Stephan Wong

List of Acronyms

ASIC Application-Specific Integrated Circuit

BNN Binary Neural Network

CNN Convolution Neural Network

DRAM Dynamic Random Access Memory

FPGA Field Programmable Gate Arrays

GPU Graphical Processing Unit

INQ Incremental Network Quantization

ILSVRC ImageNet Large Scale Visual Recognition Challenge

JLQ Jumping Logarithmic Quantization

LUT Look Up Table

MAC Multiplier And Accumulator

NLR No Local Reuse

OS Output Stationery

PE Processing Element

RS Row Stationery

SRAM Static Random Access Memory

STE Straight Through Estimator

SA Systolic Array

TWN Ternary Weight Network

WS Weight Stationery

Abstract

Convolutional Neural Networks (CNN) have become a popular solution for computer vision problems. However, due to the high data volumes and intensive computation involved in CNNs, deploying CNNs on low-power hardware systems is still challenging. The power consumption of CNNs can be prohibitive in the most common implementation platforms: CPUs and GPUs. Therefore, hardware accelerators that can exploit CNN parallelism and methods to reduce the computation burden or memory requirements are still hot research topics. Quantization is one of these methods. One suitable quantization strategy for low-power deployments is logarithmic quantization.

Logarithmic quantization for Convolutional Neural Networks (CNN): a) fits well typical weights and activation distributions, and b) allows the replacement of the multiplication operation by a shift operation that can be implemented with fewer hardware resources. In this thesis, a new quantization method named Jumping Log Quantization (JLQ) is proposed. The key idea of JLQ is to extend the quantization range, by adding a coefficient parameter "s" in the power of two exponents (2^{sx+i}).

This quantization strategy skips some values from the standard logarithmic quantization. In addition, a small hardware-friendly optimization called weight de-zeroing is proposed in this work. Zero-valued weights that cannot be performed by a single shift operation are all replaced with logarithmic weights to reduce hardware resources with little accuracy loss.

To implement the Multiply-And-Accumulate (MAC) operation (needed to compute convolutions) when the weights are JLQ-ed and de-zeroed, a new Processing Element (PE) have been developed. This new PE uses a modified barrel shifter that can efficiently avoid the skipped values. Resource utilization, area, and power consumption of the new PE standing alone are reported. Resource utilization and power consumption in a systolic-array-based accelerator are also reported. The results show that JLQ performs better than other state-of-the-art logarithmic quantization methods when the bit width of the operands becomes very small.

Acknowledgments

Words cannot express my gratitude for the invaluable patience and feedback of my professor, Rene van Leuken. This dissertation could not have been completed without your weekly guidance, and you have generously provided me with a lot of advice and expertise. Furthermore, this thesis would not have been accomplished without the generous help of Dr. David, who gave me a lot of helpful feedback on my research. I should also thank Dr. Neeraj and Professor Alexander, who also inspired me a lot during the weekly meetings. I would also like to thank my classmates for their company and spiritual support. Finally, I would be remiss if I didn't mention my family, especially my parents. Their trust in me kept my spirits and motivation high throughout the thesis project.

Longxing Jiang
Delft, The Netherlands
29th November, 2022

Contents

List of Figures

List of Tables

1 Introduction

An overview of this chapter is as follows: First, the relationship between Convolutional Neural Networks (CNN) optimization and processing elements (PEs) will be explained. Then the problem statements and research questions of this thesis will be introduced, followed by a subsequent introduction to the contributions of this thesis. Finally, an outline of this thesis will be concluded.

CNN is one of the commonly-used networks with practical applications in fields such as visual analytics and image recognition. Researchers continuously develop different efficient CNN models. Taking the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) as an example, the champion in 2012 was the AlexNet architecture[14], which not only served as the first application of machine learning to defeat (by a large margin) all other competitors' algorithms but also achieved an accuracy comparable with the human accuracy; the champion in 2014 was the GoogleNet architecture[5]; another worth-mentioned milestone in 2014 was the VGG architecture[15], which demonstrated the efficiency of using mainly 3x3 stacked kernels and then laid the foundation of the nowadays state-of-the-art networks like ResNet[4]; the champion in 2015 was ResNet-152, whose accuracy rate reach a very high percentage, 96.5%[4]. Recently, with the rapid development of the Internet of Things and smart mobile devices, energy-efficient CNN architecture is increasingly popular in resource-constrained hardware environments. Despite its popularity, due to huge data volume, intensive computation, and frequent memory access in CNNs, deploying a CNN on low-power hardware systems is still challenging. Therefore, in addition to the continuous exploration and development of the CNN model, the hardware deployment optimization of the CNN accelerator gains unprecedented significance.

To make deep neural networks generally easier to deploy on hardware devices like FPGA and ASIC, various quantization algorithms[16][10][9] have been devised to reduce memory requirements. Among all quantization methods, logarithmic quantization is very suitable for low-power inference because as logarithmic weights are represented by powers of two. The memory only needs to store the integer power index instead of the floating point weight. Besides, they fit better the common CNN Gaussian-like distributions of weights and activations, than the more uniform integer quantization. And furthermore, they allow the replacement of the multiplication operation (massively involved in CNN computation) by a shift operation, which can be implemented with fewer hardware resources. Because of all of the previous reasons, logarithmic quantization effectively reduces the CNN memory footprint and reduces the area and power consumption of the PEs involved in CNN computation.

However, current log-quantization methods are facing some challenges. For example, INQ[9] suffers from poor scalability due to some of its global variables and parameters that can only be determined through extensive experiments. In addition, according to the pre-trained baselines of PyTorch implementation of INQ[17], the bit parameter in INQ is a layer-exclusive parameter but not a global parameter. That is to say, the bit parameter in INQ can only determine the specific bit that will be used for each layer, but fail to limit the bit

that will be used for the overall architecture. Consequently, INQ is friendly to hardware that is as flexible as GPU but is unsuitable to hardware that does not have the same degree of flexibility. While DeepShift[10] has poor accuracy in the case of quantization with low activation bits and weight bits. Besides, considering hardware implementation, previous logarithmic quantization methods such as INQ require a dedicated allocation of one bit for zero-value weights that cannot be represented by shift operations. This will increase the burden of in-memory data transmission.

In addition to the quantization method, the PE, which is in relation to the MAC operation speed and power consumption of the entire CNN accelerator, is another critical part of CNN hardware implementation and optimization. Although a single MAC operation is very simple, the huge amount of iterations in the MAC operation in different tasks incur bulky logic computation. Therefore, an excellent CNN accelerator must take into account the two design aspects, logic operations, and memory access, at the same time.

1.1 Problem Statements And Research Questions

Problem statements

This thesis focuses on the following problems:

1. Logarithmic quantization is a good way to reduce memory bandwidth, the bottleneck of power consumption in CNN accelerators. Meanwhile, by encoding the weights in a logarithmic way, not only is achieved a wider range for the same amount of bits (although with "bigger" steps) but bit-shifts replace multiplications on CNNs. However, current state-of-art logarithmic quantization methods still have drawbacks in low-bit quantization cases.
2. Processing elements are in high relation to the overall performance of CNN accelerators. Researchers increasingly explore overall CNN accelerator architecture optimization, but there is still little research into the potential optimization of processing elements.

Research Questions

The aforementioned problems can engender the following corresponding research questions:

1. How to optimize current state-of-art logarithmic quantization methods in extreme low-bit quantization cases for low-power hardware implementation?
2. How to optimize shifter-based processing elements based on logarithmic quantization?

1.2 Thesis Contributions

The main contributions of this thesis can be summarized as follows.

1. A new logarithmic quantization technique named Jumping Logarithmic Quantization (JLQ) is presented. This quantization method can achieve higher accuracy than traditional logarithmic quantization at low-bit quantization by extending the quantization

range. Compared with the state of art logarithmic quantization algorithm, JLQ has obvious advantages in the low-bit (2 or 3-bit) quantization cases.

2. A hardware-friendly weight de-zeroing optimization is proposed. Hardware resources are further reduced by replacing zero-valued weights with logarithmic weights that shift operations can perform.
3. A PE based on JLQ and weight de-zeroing optimization is designed, aiming to perform MAC operations of CNN models with fewer resources, lower area, and lower power consumption.
4. The designed PE is implemented on a real systolic-array-based CNN accelerator for performance verification. Resource utilization, area, and power consumption of the new PE stand-alone and resource utilization, and power consumption in a systolic-array-based accelerator are demonstrated.

All of these contributions led to the acceptance of a paper named "Jumping Shift: A Logarithmic Quantization Method For Low-Power CNN Acceleration" at the DATE 2023 conference.

1.3 Thesis Outline

The rest of the thesis is organized as follows.

Chapter 2: Background This chapter introduces the relevant background of CNN, classical CNN architectures, and commonly-seen CNN data reuse techniques from a hardware perspective.

Chapter 3: Related Work In this chapter, related work about quantization methods will be introduced.

Chapter 4: Jumping Logarithmic Quantization Algorithm and Corresponding PE Design In this chapter, a new logarithmic quantization algorithm named jumping logarithmic quantization is introduced.

Chapter 5: Benchmark Results This chapter includes relative benchmark results of the proposed new quantization algorithm and the analysis of corresponding results.

Chapter 6: Hardware Design In this chapter, a PE design of proposed logarithmic quantization is introduced, followed by relative simulations of the proposed PE and the analysis of the results.

Chapter 7: Conclusions This chapter first draws conclusions for this thesis followed by answering all the previous research questions in detail. And then three possible prospects for future work are presented.

2 Background

This chapter begins with an introduction to the basic concepts of Convolutional Neural Networks (CNNs). After that, the chapter further delves into two classic CNN architectures to gain a comprehensive understanding of its operational complexity from an algorithmic perspective.

2.1 Convolutional Neural Networks

In recent years, there has been a great deal of interest in deep learning. Convolutional neural networks (CNNs), as the most mature type of deep learning models, have naturally attracted attention in various fields. CNNs are used in fields such as medical research [18][19], language processing [20], and visual imagery [14][5][4]. Among them, the most dominant and widely used field of CNN is the field of visual images. Several characteristics of CNNs make them very suitable for tasks related to visual imagery. The local connection, weight sharing, and pooling operation of CNNs can effectively reduce the complexity of the network and the number of parameters, which makes it much easier to train and optimize than other counterparts. In the ImageNet Large Scale Visual Recognition Competition (ILSVRC), the architectures that have obtained breakthrough achievements are all based on CNN.

In general, CNN architectures consist of convolutional layers, pooling layers, and fully connected layers. And two critical processes, forward propagation, and backward propagation are entailed in every CNN architecture.

2.1.1 Convolutional Layers

The convolutional layer, which consists of convolution operation and activation function, is an essential component of a CNN architecture. In CNN, the linear part is the convolution operation, and the nonlinear part is the activation function.

Convolution

An example of convolution operation is shown in Figure.2.1.1. Convolution is a linear operation used for feature extraction. During the convolution process, a small array of numbers, which is called ”kernel”, is multiplied and added correspondingly with another array of numbers called ”input tensor”. Afterward, the output value at the corresponding position of the output tensor can be obtained after the convolution operation. In CNN, there are many kernels, and each different kernel will perform such convolution operations.

A key feature of the convolution operation is weight sharing, which means the kernel in CNN is shared among all input locations. Weight sharing reduces the number of parameters to learn compared to fully connected neural networks.

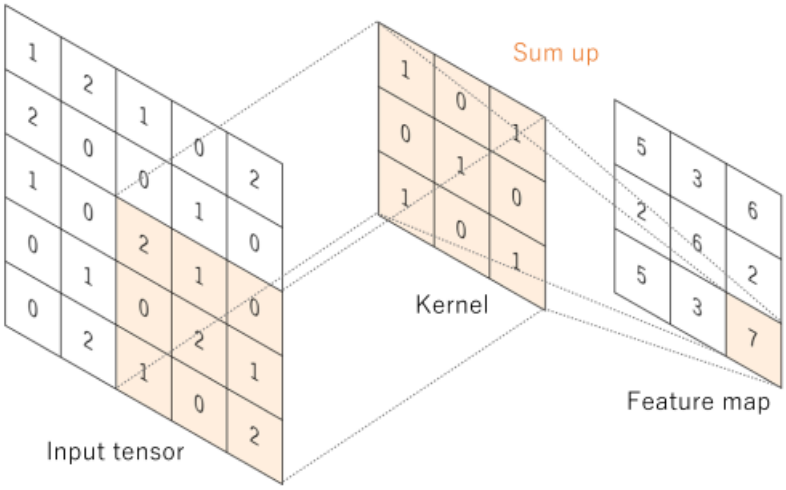


Figure 2.1.1: An example of convolution operation[1].

Nonlinear activation function

The purpose of the nonlinear activation function is to improve the ability to describe the target model, so as to solve the problems that the pure linear model cannot solve. If there is no nonlinear activation function, the output of each layer in CNN is a linear function of the input of the upper layer. In this case, no matter how many layers the CNN has, the final model is still a linear model, which is equivalent to a one-layer model in theory. There are many kinds of nonlinear activation functions. Common types of activation functions are Sigmoid activation function, Tanh activation function, and Relu activation function, all of which are shown in Figure.2.1.2. Since the publication of [14], most CNNs use the Relu activation function. This is because, for the deep network, the sigmoid activation function and the Tanh activation function are prone to the disappearance of the gradient during backpropagation.

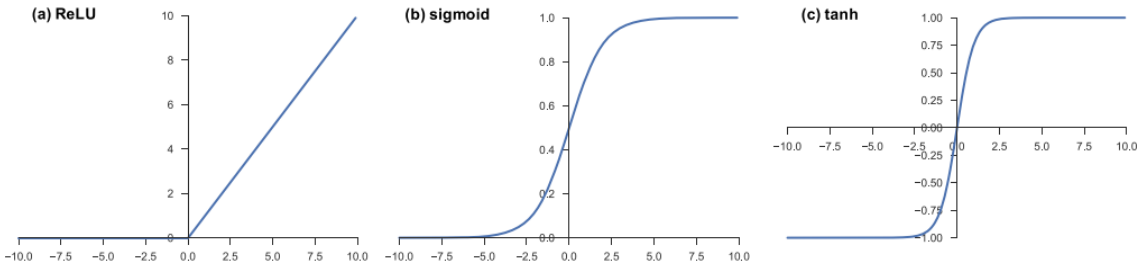


Figure 2.1.2: Examples of activation function[1].

2.1.2 Pooling Layers

Pooling layers provide a down-sampling operation, which is designed to gradually reduce the number of parameters and computational complexity of the model. If the pooling type is max pooling, it will perform an operation that finds the largest number in the kernel as the output; if the pooling type is average pooling, it will perform an operation that finds the average of

all numbers in the kernel as the output. Since the pooling layer will be destructive to the features of the original input data, if the pooling size is too large, the overall performance of the model will be degraded. Therefore, most CNNs choose 2×2 as the pooling size, which shrinks the activation input to 25% of its original size. And an example of 2×2 max pooling is shown in Figure.2.1.3.

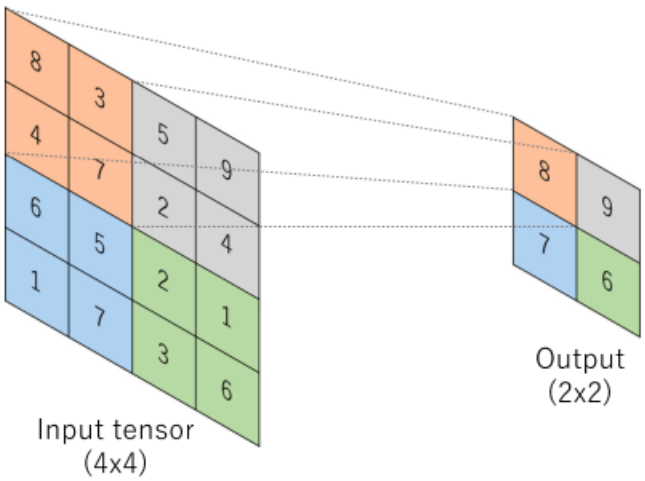


Figure 2.1.3: An example of 2×2 max pooling[1].

2.1.3 Fully Connected Layers

The fully connected layer, also known as the dense layer, can be regarded as a special convolutional layer, which also performs a multiply-add operation. However, unlike the convolutional layer, it does not have a two-dimensional kernel, and the "kernel" of the fully connected layer is completely flattened. Due to the similarity between the fully connected layer and the convolutional layer, in some CNN hardware deployments, convolution layers and fully connected layers often use the same computing module. And an example of the fully connected layer is shown in Figure.2.1.4.

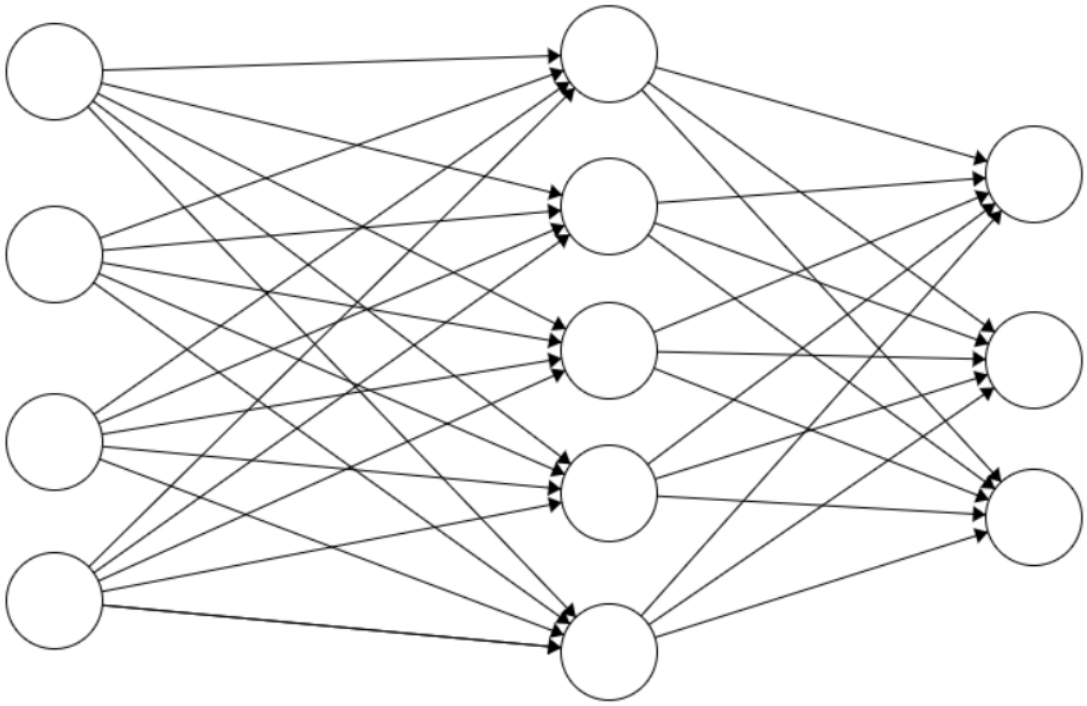


Figure 2.1.4: An example of fully connected layer[2].

2.1.4 Forward Propagation

In CNN, forward propagation refers to the process of convolution calculation and storage of any intermediate variables entailed for gradient computation in memory and outputs for a neural network in order from the input layer to the output layer. A schematic diagram of the forward propagation process is shown in Figure.2.1.5:

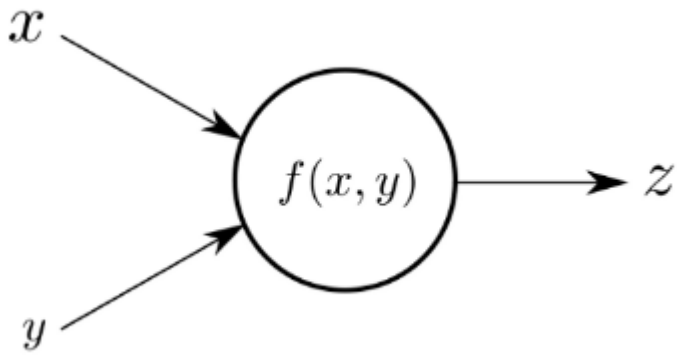


Figure 2.1.5: An example of forward propagation[3].

2.1.5 Backward Propagation

In CNN, backward propagation refers to the process of calculating the gradient of neural network parameters. Unlike forward propagation, this process traverses the network in reverse order, from the output to the input layer, according to the chain rule from calculus. In the backward propagation process, any intermediate variables in relation to partial derivatives will be stored to calculate the final gradient of the loss function with respect to the inputs. A schematic diagram of the backward propagation process is shown in Figure.2.1.6, where L is assumed to be the loss function:

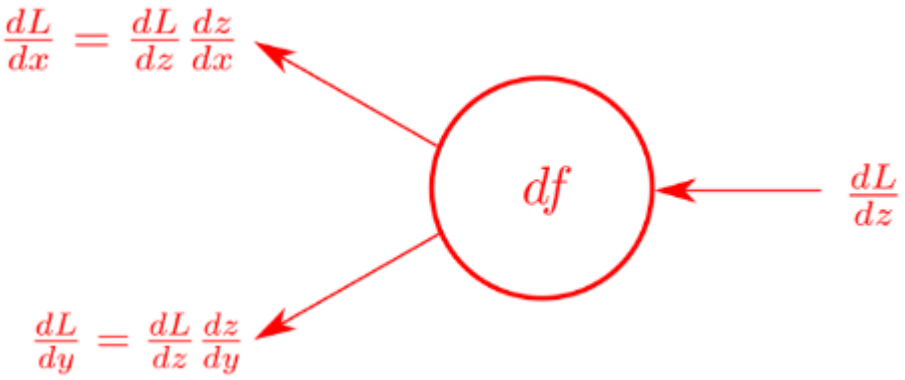


Figure 2.1.6: An example of backward propagation[3].

2.2 Classical CNN Architectures

There are many classic network architectures in CNN. Among them, those that have won the championship in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) over the years attract the most attention from researchers. This subsection will introduce the ResNet architecture and GoogleNet architecture, both of which will be used in subsequent chapters for testing purposes. These two classical CNN architectures are the champions of the ILSVRC in 2014 and 2015.

2.2.1 ResNet Architecture

For CNN, by gradually adding layers to the shallow network, the performance of the model on the training set and test set can often be improved. The model can fit the underlying mappings better owing to the incremental model complexity. However, sometimes, the use of a deeper network will not improve the performance of the model, and even worse, it will cause the performance of the model to decline rapidly. This phenomenon is called ”degeneration”. What ResNet architecture [4] can solve is the ”degeneration” problem of this deep neural network.

ResNet makes the model easier to optimize by adjusting the structure of the model. In ResNet, the stacked layers are named “block”. For each block, the desired underlying mapping is represented as H(x), and the residual function entailed to be fitted is F(x):=H(x)-x. In

this case, the original mapping is recast as F(x)+x. The original authors of ResNet contend that it is much easier to optimize the residual mapping than to optimize the original, unreferenced mapping. Consequently, in ResNet, each stacked layer explicitly fits the layers to the residual mapping. In Figure.2.2.1, an example of the ResNet residual block is displayed.

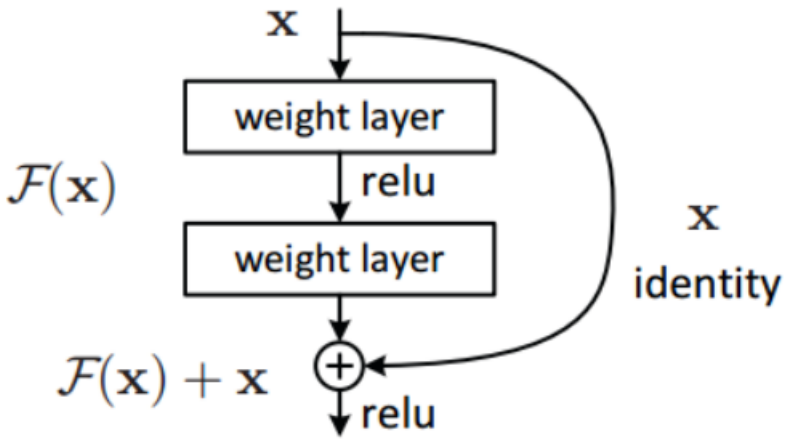


Figure 2.2.1: An example of Residual Block[4].

At the same time, ResNet can also solve the problem of gradient disappearance in convolutional networks. Due to the existence of the residual term x of mapping F(x)+x, in backpropagation, the gradient does not vanish quickly for possible causes of F(x) itself.

2.2.2 GoogleNet Architecture

GoogleNet[5] introduces a novel network structure that included a new type of module called inception. Before the emergence of GoogleNet, other classic network structures, such as AlexNet[14] and VGG[15], achieved good training results by increasing the number of layers of the network. However, the increase in the depth of the network will also bring many negative consequences such as overfitting, gradient disappearance, and gradient explosion. The Inception module of GoogleNet improves the training results from the perspective of increasing the width of the network instead of the depth of the network. The GoogleNet is only 22 layers deep, and its parameters are about 1/12 and 1/3 of its corresponding counterparts, AlexNet and VGG, which means that GoogleNet can extract more features under the same amount of computation resources.

Figure.2.2.2 is the initial inception structure designed by the authors of GoogleNet. Their idea is to replace a small 3x3 kernel with multiple kernels of different types stacked together. The advantage of this design is that the extracted features can be diversified, and there will only be weak correlations between features.

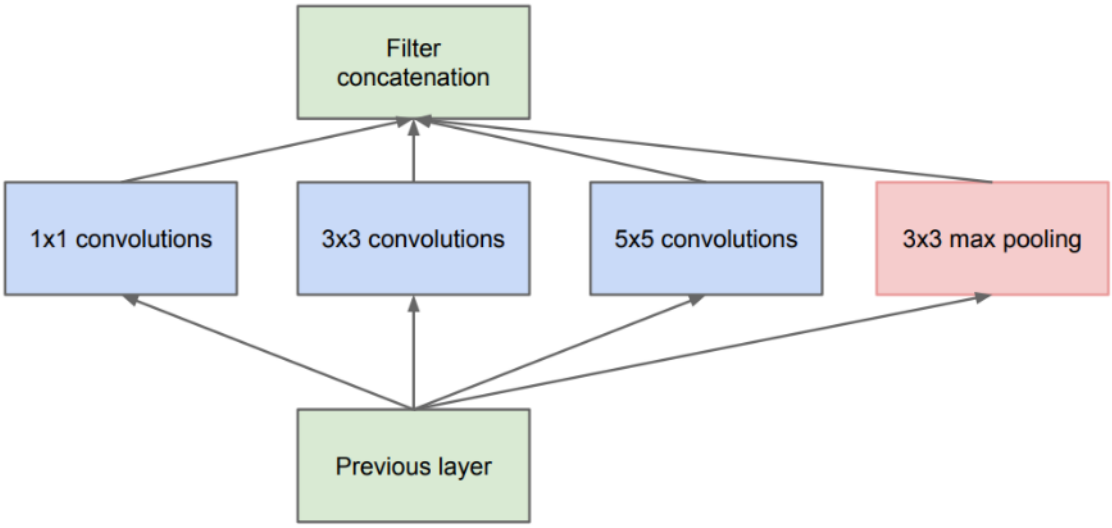


Figure 2.2.2: An example of inception Block[5].

2.3 Expensive Data Movement

Use GoogleNet as an example for further analysis. Although GoogleNet has used some carefully designed optimizations to reduce the overall network parameters as much as possible, the entire network still involves about 1550×10^6 MAC operations and includes about 50MB of weights. This means that for the hardware deployment of high-precision CNN like GoogleNet, both memory bandwidth and computation are the bottlenecks. Owing to these bottlenecks, it is almost impossible to deploy a complete high-precision CNN model on hardware due to the limited resources of most low-consumption hardware. Therefore, during hardware deployment, dynamic random access memory (DRAM) is needed to relieve the pressure of the on-chip memory. However, reading data from DRAM is very energy-consuming and time-consuming. For a simple system like Figure.2.3.1, accessing an element from DRAM costs 200 times more than accessing the same element from the register file (RF)[21].

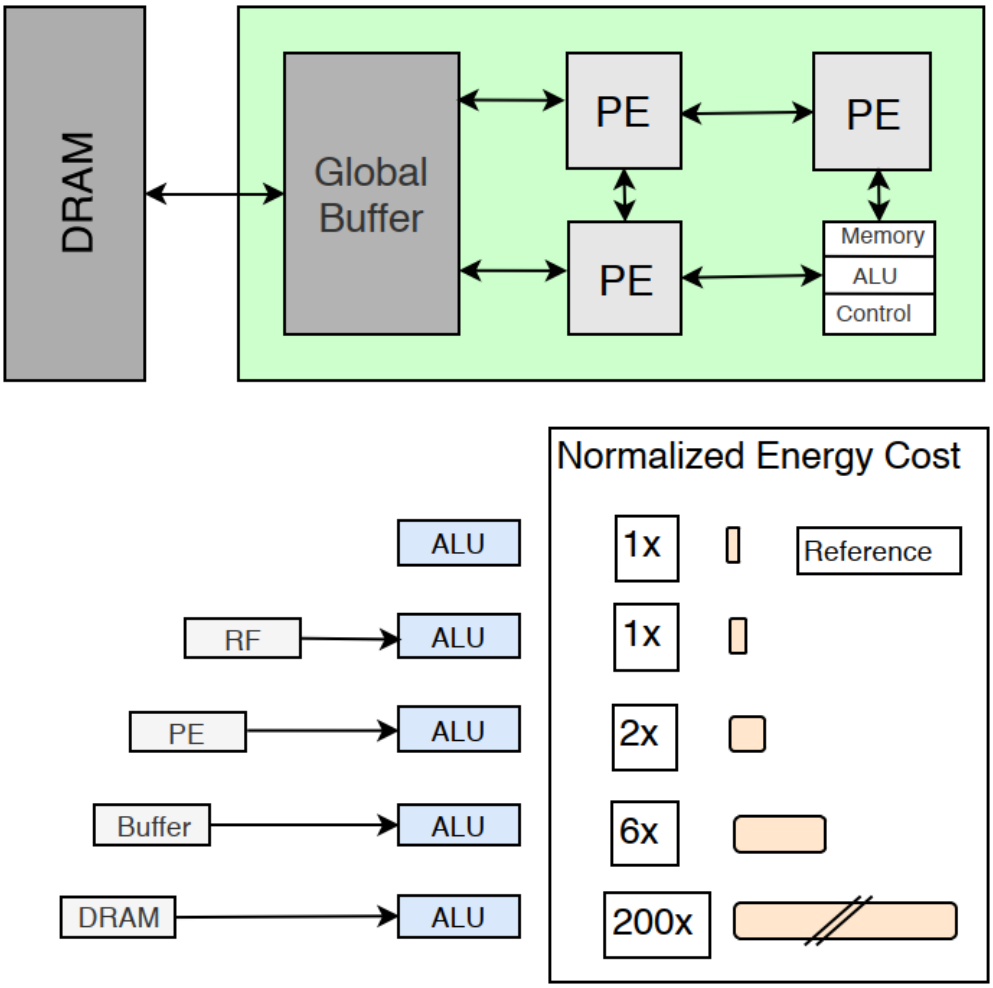


Figure 2.3.1: Hardware module prototype[6].

In theory, without any design strategy, at least 2 memory reads and 1 memory write involved in each MAC operation require access to external memory, which will seriously affect the entire throughput, latency, and energy consumption of hardware systems. Take the GoogleNet introduced earlier for exemplification. Figure.2.3.2 shows the estimated energy consumption between different layers when using the GoogleNet architecture for inference. The energy consumption estimates here are generated using an online tool [7], where the energy unit is measured in terms of the energy required for one MAC operation. Consumption represents the energy required for calculation in this layer. Input feature map, output feature map, and weight respectively correspond to the energy required for the data movement involved in the input features, output features, and weights in this layer. It can be seen that the percentage of energy consumption for data movement in the GoogleNet architecture far exceeds the energy required for computation. Similar conclusions can also be observed in other convolutional network architectures.

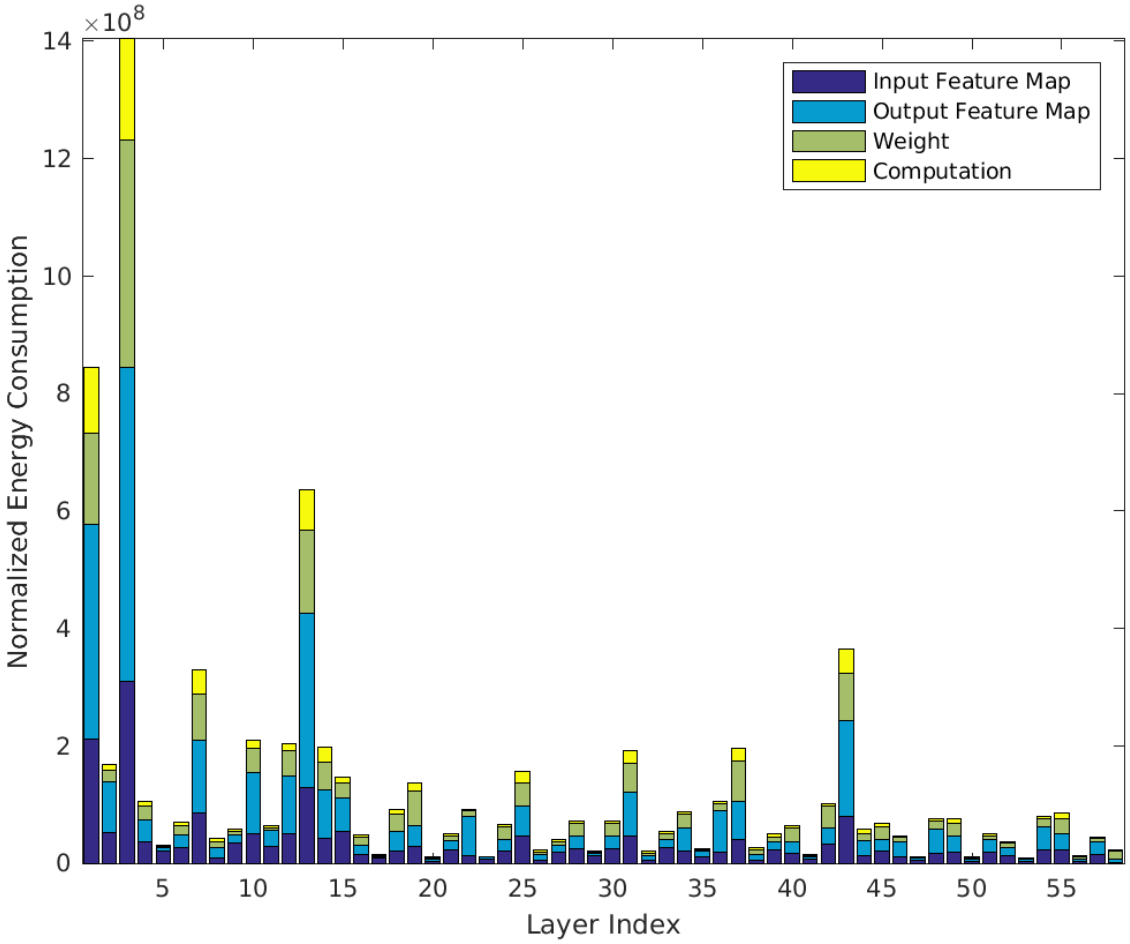


Figure 2.3.2: Energy estimation of GoogleNet[7].

2.4 Data Reuse In CNN Hardware Implementation

In order to reduce the high energy consumption caused by data movement in CNN as much as possible, aspects of data reuse and data flow techniques in CNN need to be considered simultaneously when designing a hardware architecture suitable for CNN deployment:

2.4.1 Data Reuse

The inherent parallelism in CNNs makes data reuse possible. There are three specific data reuse methods: convolution reuse, feature map reuse, and filter reuse.

1. Convolution reuse represents the process where the kernel traverses the input image in a completely parallel way. In this process, the data from one filter is reused in the input features of the same channel. The specific schematic diagram of convolution reuse is shown in Figure.2.4.1:

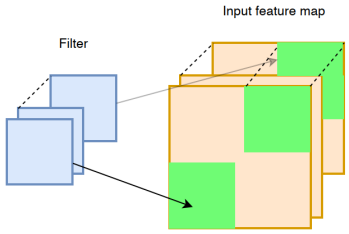


Figure 2.4.1: Convolution Reuse[6].

2. Feature map reuse refers to the situation where the input feature map of one specific channel is used to adapt the filters of multiple channels. The specific schematic diagram of feature map reuse is shown in Figure.2.4.2:

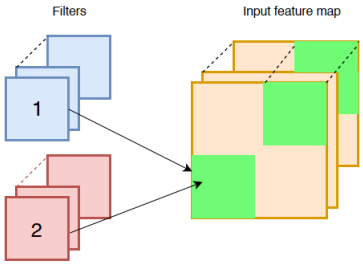


Figure 2.4.2: Feature Map Reuse[6].

3. Filter reuse represents the situation where the filter of the same channel is used to adapt the input features of multiple channels. The specific schematic diagram is shown in Figure.2.4.3:

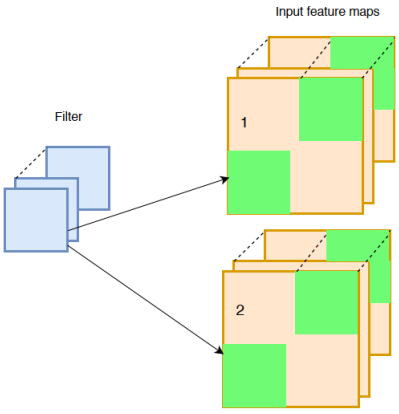


Figure 2.4.3: Filter Reuse[6].

2.4.2 Data Flow Techniques

Representative data flow technologies can be divided into the following four categories. These data flow techniques are Weight Stationery (WS) data flow technique [22], [23], Out-

put Stationery (OS) data flow technique [24], [25], Row Stationery (RS) data flow technique [26], [21], and No Local Reuse (NLR) data flow technique [27],[28] respectively.

1. The WS data flow technique increases the reuse of weighted pixels when performing MAC operations, aiming to minimize the overall energy consumption of reading filter weights. The weights are read one-by-one from DRAM into the registers of each processing element and then remain static. After that, those weights are utilized multiple times to run as many MAC operations as possible until they are no longer needed. A schematic diagram of the WS data flow technique is shown in Figure.2.4.4:

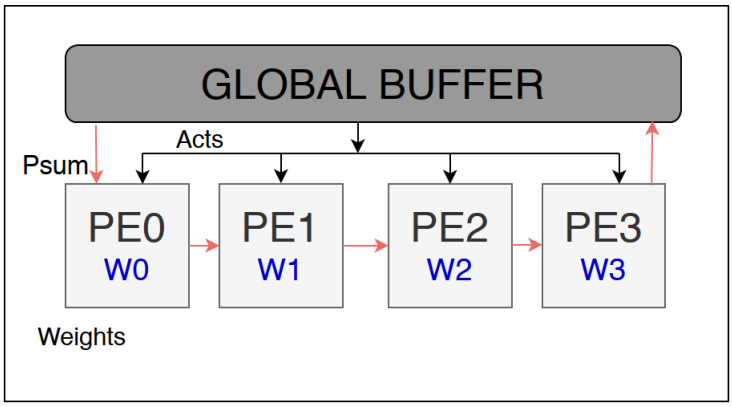


Figure 2.4.4: Weight Stationary Data Flow[6].

2. The OS data flow technique increases the reuse of partial sums when performing MAC operations, aiming to minimize the energy consumption of writing and reading partial products. In the OS data flow, each output feature map pixel is stored in a specific PE and remains stationary, instead of being stored in a specific PE. And the partial sum of one convolution window from one channel is stored in RF and then added to the corresponding partial sum from the next channel. A schematic diagram of the OS data flow technique is shown in Figure.2.4.5:

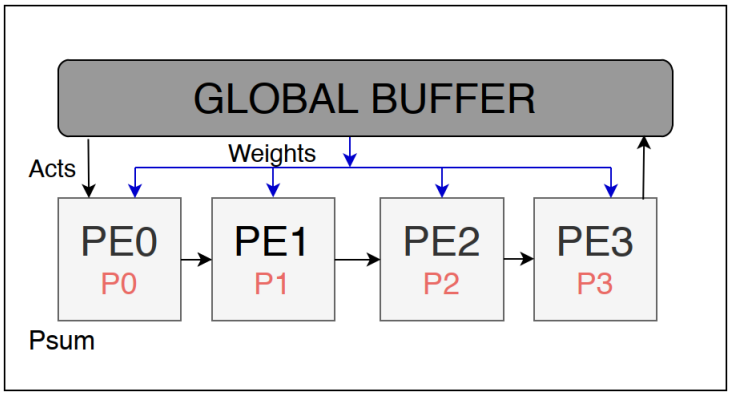


Figure 2.4.5: Output Stationary Data Flow[6].

3. The RS data flow technique simultaneously increases the reuse of weights and the reuse

of partial sums when performing MAC operations. The RS data flow technique stores the input feature map pixels, weight pixels, and output feature map pixels in registers of PE, and reuses these pixels. The input feature map pixels are propagated among the PE arrays in a diagonal way; The weight pixels are propagated between the PE arrays in a lateral way; The output feature map pixels are propagated between the PE arrays in a longitudinal way. A schematic diagram of an RS data flow technique is shown in Figure.2.4.6:

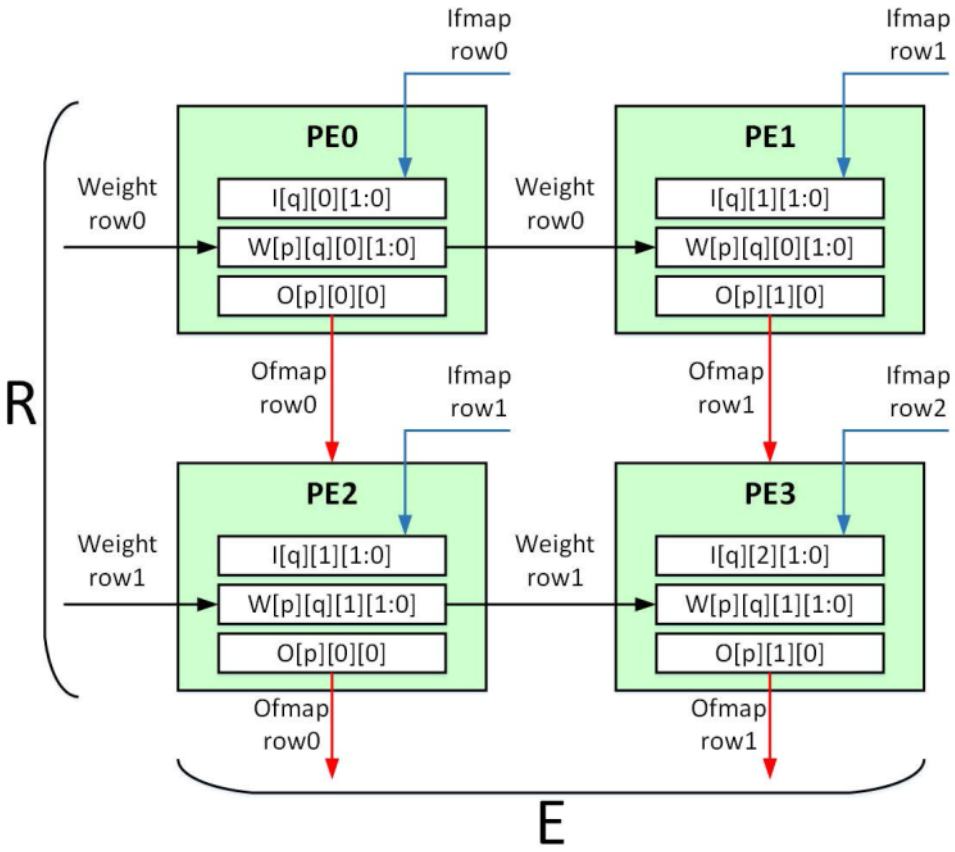


Figure 2.4.6: Row Stationary Data Flow[8].

4. Unlike Weight Stationery, Output Stationery, and Row Stationery data flow techniques, NLR-based PEs do not use registers to store any input features, weights, or output features. Instead, the NLR data flow technique uses a large global buffer (memory) to store input features, weights, and output features. In each cycle, weight pixels are single-cast, while input feature pixels are multi-cast. A schematic diagram of an NLR data flow technique is shown in Figure.2.4.7:

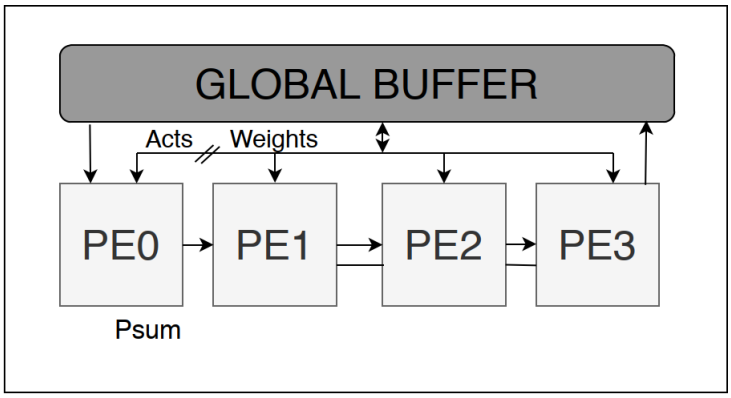


Figure 2.4.7: No Local Reuse Data Flow[6].

3 Related Work

An overview of this chapter is as follows: This chapter begins with a brief introduction to the different log quantization methods. Subsequently, two state-of-art quantization methods, INQ[9] and DeepShift[10] are analyzed in detail. Finally, other related works in low-bit quantization are introduced.

3.1 An Overview Of Different Log Quantization Methods

Logarithmic quantization is able to compress the original CNN, so that the final weights satisfy the form of powers of two, which can be used for power-saving shifting operations for inference.

LogNN [29] was the first to propose logarithmic quantization for CNNs. They propose two methods: in the first method, the weights remain fix-point and the activations are log-quantized. While in the second method, both weights and activations are log-quantized. The problem with the first method is that the outputs need to be quantized before becoming inputs of the next layer, which will cause an overhead. It is more efficient to log-quantize the weights and let activations flow through the network in a fix-point format, which is the method used in this thesis as in most of the other recent works. In the second method, the PE does not perform the shift operation. They propose a PE that performs this custom operation, which is smaller but more complex than a shift operation.

ShiftCNN [30] replaces each multiplication with a set of 2 or 3 shifts, therefore their PE consumes more hardware resources and has higher power consumption than the PE with only a single shifter. Similar to ShiftCNN, in [31] and [32], in order to improve the accuracy, they also replace each multiplication with the sum of two shift operations.

Besides, logarithmic quantization is also applied in approximate computing. In [33], the author proposes an approximate shifter-based PE. And in [34], an approximate logarithmic data representation is proposed for CNN training.

3.2 State-of-art Logarithmic Quantization Methods

Most logarithmic quantization methods start with a pre-trained model, among which the state-of-the-art is INQ [9]. There are also logarithmic quantization methods that support training models from scratch. These quantization methods can be applied for both inference and training. And among them, the state of art quantization method is DeepShift[10].

3.2.1 Incremental Network Quantization (INQ)

The main feature of INQ is implementing incremental retraining from a pre-trained model. The INQ algorithm consists of three steps. The first step is to sort the weights by absolute

value and divide them into two groups according to a certain proportion. The second step is to quantize the group of weights with larger absolute values, and the third step is to re-train the group of weights with relatively smaller absolute values. After these three steps, a certain proportion of the weights have been quantified. And the final step is to increase the quantization scale and repeat the above three steps until the overall quantization is completed. In INQ, the scale of each quantization is a hyperparameter. Unfortunately, there is currently no absolute criterion for the selection of this parameter, which is often obtained through experiments.

Considering the hardware implementation, INQ separates 1 bit for representing zero value and uses the remaining bits for representing the value of the power of two. However, INQ cannot fully utilize the bit representation range. For instance, at 4-bit quantization, INQ has one bit for exclusively representing zero, rendering the remaining 3 bits only able to represent 8 different values rather than represent 16 different values (including or not the zero).

Figure 3.2.1 provides a visualization of the INQ iteration steps.

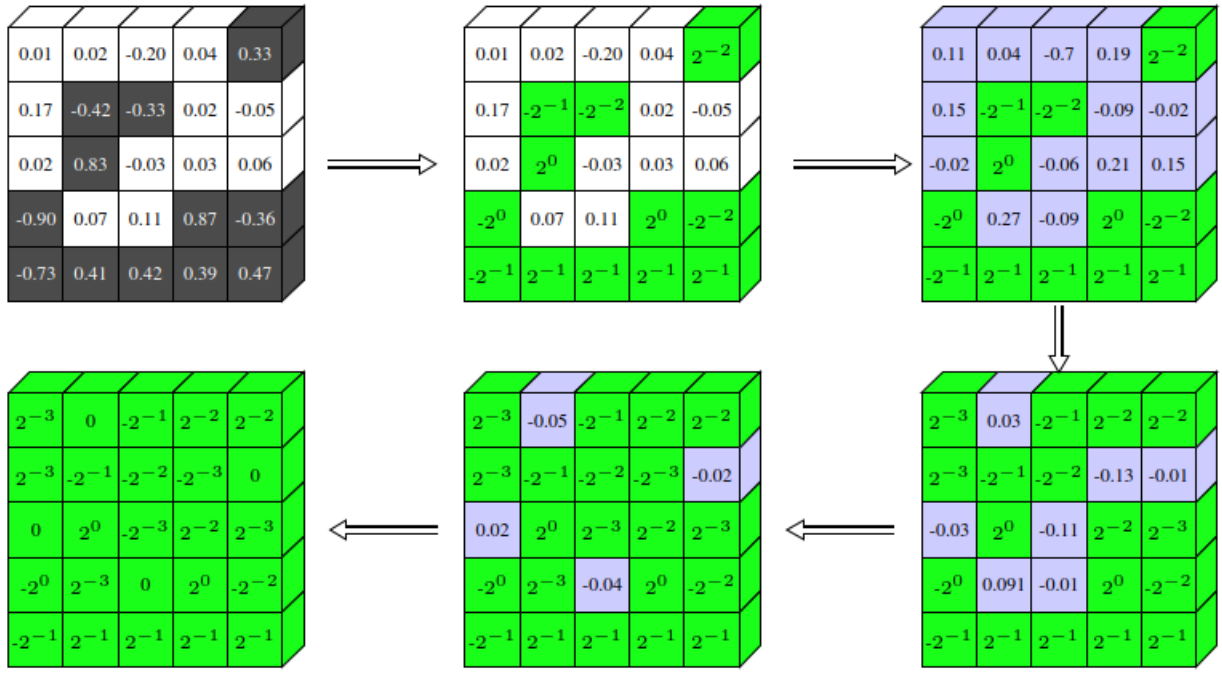


Figure 3.2.1: INQ iteration steps (undergoes from 50%→75%→87.5%→100%)[9].

3.2.2 DeepShift

In [10], the author proposes two methods: DeepShift-Q and DeepShift-PS. DeepShift-Q can be regarded as the normal logarithmic quantization without any optimization approach. In DeepShift-PS, since the power-of-two function is differentiable, the author implements a derivation extension of the backward pass when the weights are logarithmic-quantized. Besides, both DeepShift-Q and DeepShift-PS support training from scratch. A brief schematic diagram of the quantization of DeepShift-Q mode is shown in Figure 3.2.2.

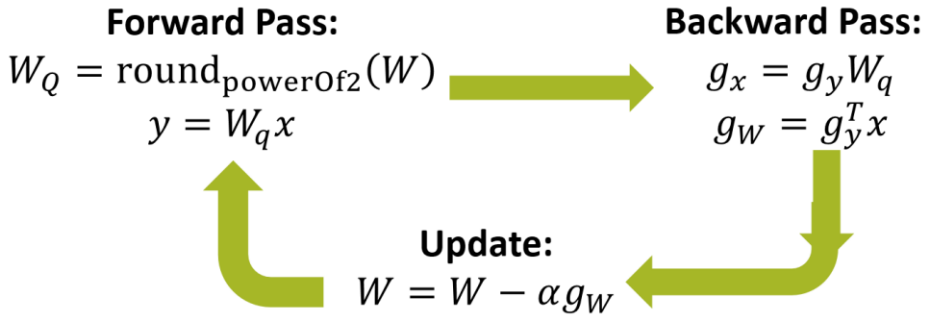


Figure 3.2.2: DeepShift-Q[10].

A brief schematic diagram of the quantization of DeepShift-PS mode is shown in Figure 3.2.3.

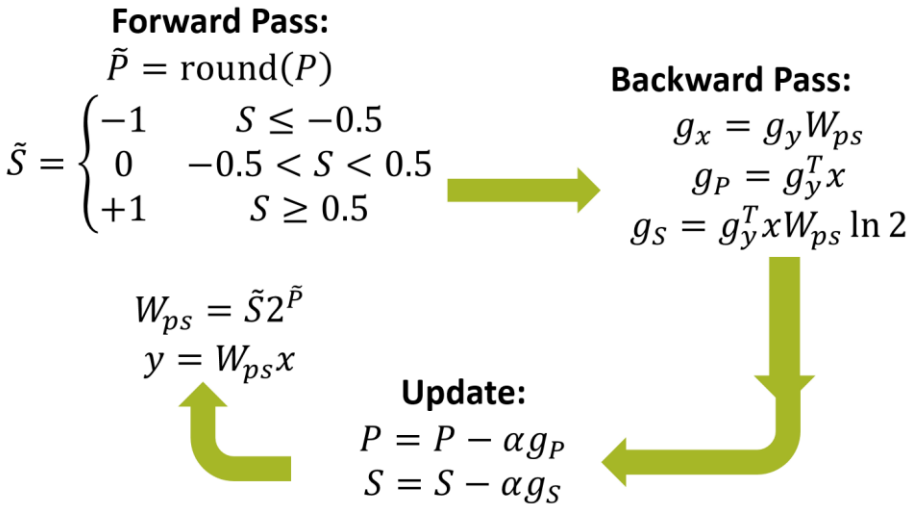


Figure 3.2.3: DeepShift-PS[10].

3.3 Other Worth-mentioned Low-bit Quantization Methods

3.3.1 Binary Net

Binarized Neural Network (BNN)[11] is a neural network that uses only two values of +1 and -1 to represent weights and activations. Compared with the full-precision neural network, BNN can use very simple operations such as XNOR, and popcount to replace original float-32 multiply-accumulate operations to realize the convolution operation, thus saving a lot of memory and computation. The authors of BNN are the first person to propose a way to train neural networks using both binarized weights and activations using stochastic gradient descent. In order to solve the problem of gradient transfer in the binarized weights, the author proposes to use the sign function for float32 weights during the training process to obtain the binarized weights. A brief schematic diagram of the forward pass and backward pass of BNN is shown in Figure3.3.1.

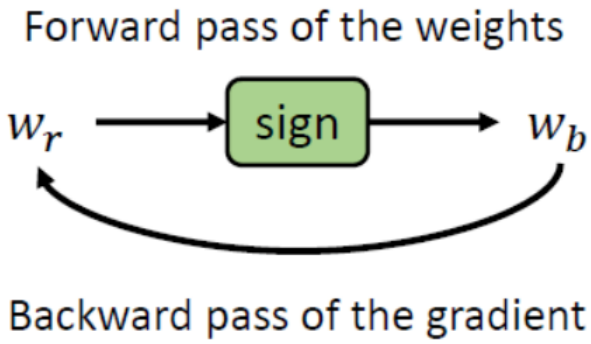


Figure 3.3.1: Forward Pass and Backward Pass In BNN[11].

3.3.2 XNOR-Net

Due to the limited amount of information that can be expressed by binary values, BNN is generally much lower than the full-precision model in terms of model accuracy. However, because BNN can greatly facilitate the deployment of models on resource-constrained devices, researchers have never stopped exploring to improve the accuracy of BNN.

XNOR-Net[12] was proposed shortly after BNN[11]. The authors of XNOR-Net proposed two models, one is BWN (Binary Weight Networks) and the other is XNOR-Net. BWN only uses binarized weights, while activations are still in float32 precision. However, according to the definition of BNN, strictly speaking, BWN cannot be counted in the category of binary networks.

XNOR-Net considers the quantization error on the basis of the original BNN. The author of XNOR-Net proposes a method to extract a scaling factor for each output channel direction of the real-valued weights to restore the information of the binarized weights. At the same time, XNOR-Net also extracts a scaling factor on each pixel in the HW direction for activation, which is used to restore the information of binarized activations. These two scaling factors do not need to be learned and can be obtained by directly calculating the corresponding L1 norm, without affecting the efficient convolution calculation process of binarization. And experimental results show that XNOR-Net outperforms the initial BNN. A brief diagram of convolution in XNOR-Net is shown in Figure3.3.2.

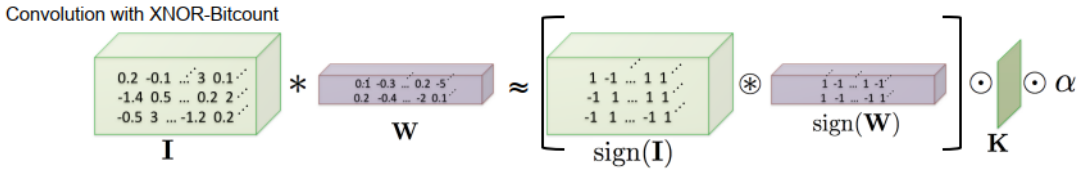


Figure 3.3.2: Convolution In XNOR-Net[12].

3.3.3 ReActNet

In addition to XNOR-Net[12], some researchers have used other optimization ideas to improve BNN[11], of which ReActNet[13] is a representative one. BNN initially only had a Top-1 accuracy of 27% on ImageNet, while ReActNet-C achieved a Top-1 accuracy of

71.4% on ImageNet, which is only about 3% behind the accuracy of the corresponding full-precision model. However, the cost of improving the accuracy is that ReActNet increases a lot of computation, making it difficult to effectively implement in complex tasks such as target detection. The authors of ReActNet have found through extensive experiments that the performance of BNNs is particularly sensitive to changes in the distribution of activations. In other words, the offset and scaling of activations have a significant impact on the performance of the BNN. Therefore, the authors of ReActNet believe that the activations of each layer in the convolutional neural network model have the most suitable offset value and scaling value to make the performance of the entire model optimal. Therefore, ReActNet introduces learnable channel-wise parameter variables to the sign function and PReLU function, allowing the model to automatically learn the best offset and scaling values for each layer. The modified sign function and PReLu function are named ReAct-Sign and ReAct-PReLU respectively, which are core innovation in ReActNet. Images of the ReAct-Sign function and ReAct-PReLU function in ReActNet are shown in Figure3.3.3 and Figure3.3.4 respectively.

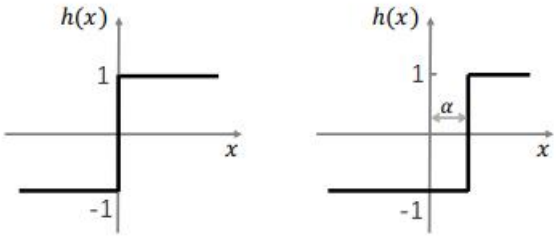


Figure 3.3.3: ReAct-Sign function[13].

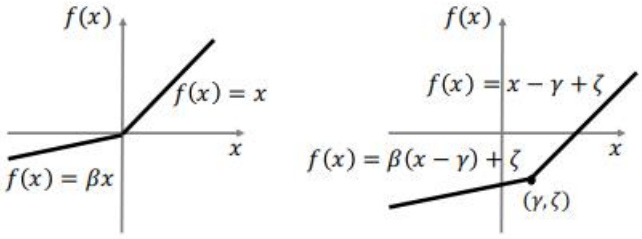


Figure 3.3.4: ReAct-PReLU function[13].

3.3.4 Ternary Weight Networks

After the binary network was proposed, in order to further improve the model accuracy and network expression ability, the ternary weight network(TWN)[35] was proposed. In the ternary weight network, the weights are set to 1, -1, and 0. The author of the ternary weight network believes that ternarization weights have better network expression ability than binarization weights. In state-of-the-art network architectures such as VGG[15], GoogLeNet[5], and Residual Networks[4], the most commonly used convolutional filter size is 3×3. For

binary weights, there are 2^9 (equals 512) different convolution kernels in theory. While for ternary weights, there are 3^9 (equals to 19683) different convolution kernels theoretically. The core part of TWN is the derivation process of threshold *delta* and scale parameter α. First, TWN assumes that the distribution of weights is close to a combination of a normal distribution and a uniform distribution. Then, TWN proposes to use a scale parameter α to minimize the L2 distance between the weights before ternarization and the weights after ternarization. That is to say, a threshold parameter δ is determined theoretically based on prior knowledge of the weight distribution. When the weight is greater than this threshold, this weight becomes 1; When the weight is less than the negative of this threshold, this weight becomes -1; In the residual case, the weight becomes 0. It is also worth mentioning that TWN does not completely eliminate the multiplier operation even though weights are ternarized to 1, -1, and 0. In the actual forward propagation, TWN still needs to multiply each output by the scale parameter.

Jumping Logarithmic Quantization

4

This chapter is outlined as follows: This chapter starts with the assumption of the Jumping Logarithmic Quantization (JLQ), then provides a brief introduction to the error estimation model used. Then this chapter explains the weight de-zero optimization, and subsequently, covers the parameter selection strategy of JLQ.

4.1 Assumption

Many studies indicated that weights in most mainstream non-sparse CNN models generally follow a Gaussian-like distribution [35]. That is to say, the majority of the CNN weights have small values, and only a few outliers have relatively large values. And based on this information, some quantization techniques [16] intentionally use fewer sampling points to quantize weights with large absolute values and use more sampling points to quantize weights with small absolute values. However, some previous studies show that weights with larger absolute values are not inessential. In fact, they are more critical in feature extraction than those with smaller values [36]. Besides, according to the experiment in [37], some quantization methods are highly tolerant of quantization noise, while other quantization methods are not. That is to say, consider to reduce the quantization noise always serves as a well-intended strategy.

Normally, when the bit is sufficient, logarithmic quantization will have a small number of big-absolute-value weights and a large number of small-absolute-value weights, which makes it a perfect solution based on this assumption.

However, in the extreme low-bit situation, for example, 2-bit quantization or 3-bit quantization, traditional logarithmic quantization will fail to strike a balance between big-absolute-value weights and small-absolute-value weights, which might cause great quantization error and generate poor accuracy.

Therefore, considering the feature extraction and quantization noise reduction, we make an assumption that if we want to increase the performance of the logarithmic quantization in the low-bit quantization, both big-value weights and small-value weights need to be taken into consideration.

Based on this assumption, to strike a balance between big-absolute-value weights and small-absolute-value weights in low-bit quantization, a new logarithmic quantization technique called jumping logarithmic quantization (JLQ) is proposed. This quantization technique extends the quantization range by introducing two external parameters instead of increasing the quantization bits so that both big-value weights and small-value weights can be taken into consideration in the extreme low-bit quantization case.

To achieve our goal, we introduce a jumping step parameter ”s” and an initial exponent index parameter ”i” (or pre-shift) in our proposed logarithmic quantization method. In JLQ, the quantization values can be represented as follows:

- **Quantization Weights** = $(\pm 2^{sx+i})$

4.2 Quantization Error Estimation Model

To theoretically determine the hyper-parameters "s" (jumping step) and "i" (initial exponent index) of the JLQ, a quantization error estimation model is proposed. In our model, weights W_i are assumed to be in the range of [-1, 1], and they follow a Gaussian distribution N(μ, δ), where μ is very close to 0. In addition, since the slope of the long tail part of the Gaussian distribution is extremely small, for simplicity, weights located in the [-1, -3δ) and (3δ, 1] intervals are regarded to have a uniform distribution instead of Gaussian distribution in our model. For most typical non-sparse CNN models, the value of δ is between 0.01 and 0.09 [38]. In order to facilitate the comparison of quantization errors engendered by different jumping steps and initial exponent indexes, in this error estimation model, δ is assumed to be the average of 0.01 and 0.09, which is 0.05. And the total number of CNN weights is set to $k \cdot 10^6$. Consider a quantization interval $[W_{down}, W_{up}]$, the logarithmic quantization weight corresponding to this interval is W_{quan}.

Assume that the total number of weights falling in this logarithmic quantization interval is Q, and there are N ($N \to \infty$) different discrete weights in this logarithmic quantization interval. The quantization error formula selected by this estimation model is consistent with the quantization error formula mentioned in [37], which is displayed as (4.2.1):

$$E(x) = \frac{1}{2} \cdot (Quan(x) - x)^2 \tag{4.2.1}$$

When weights obey a uniform distribution, the number of each weight can be consequently regarded as Q/N, and the interval between each weight is (W_{up}-W_{down})/(N-1). Therefore, the total quantized error function F(x) can be represented as (4.2.2):

$$\begin{aligned} F(x) &= \lim_{N\to\infty} \sum_{i=0}^{N-1} E(W_{down} + (W_{up} - W_{down}) \cdot \frac{i}{N-1}) \cdot \frac{Q}{N} \\ &\approx \lim_{N\to\infty} \sum_{i=1}^{N} E(W_{down} + (W_{up} - W_{down}) \cdot \frac{i}{N}) \\ &\cdot \frac{W_{up} - W_{down}}{N} \cdot \frac{Q}{W_{up} - W_{down}} \\ &= \frac{1}{2} \cdot \int_{W_{down}}^{W_{up}} (Quan(x) - x)^2 dx \cdot \frac{Q}{W_{up} - W_{down}} \end{aligned} \tag{4.2.2}$$

When weights obey a Gaussian distribution, the number of each weight can be approximately regarded as equal owing to the fact that the individual probability of each weight tends to be infinitely small as N approaches infinity. Consequently, the total quantization error can still be approximately represented by (4.2.2). The only difference is that the total number of weights within this interval, in this case, can be further determined through the definite integral over the probability density function of the normal distribution, which is shown in (4.2.3):

$$Q = k \cdot 10^6 \cdot \int_{W_{down}}^{W_{up}} \frac{1}{\delta \cdot \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \delta^2}} dx \tag{4.2.3}$$

However, in reality, the number of types of weights is not infinite. Therefore, the authority of the theoretical total quantization error estimated by this model will be slightly degraded.

4.3 Weight De-zero Optimization

In original DeepShift[10], ternary sign operator ($\{-1,0,+1\}$) is used. This sign operator will engender weight with zero value. And in the original INQ[9], one bit is used for zero-value weights representation exclusively. However, since zero value cannot be represented with shift operation, an additional multiplexer is entailed in the real hardware implementation to represent zero value. Consequently, this work proposes a weight de-zero optimization to replace all zero-value weights with logarithmic weights that can be represented by shifters. The following example demonstrates the difference between original weight sets and weight sets after de-zero optimization:

- **Quantization weights with ternary operator (3-bit)**:
  ($\pm 2^0$,$\pm 2^{-1}$,$\pm 2^{-2}$,0)

- **Quantization weights with binary operator (3-bit)**:
  ($\pm 2^0$,$\pm 2^{-1}$,$\pm 2^{-2}$,$\pm 2^{-3}$)

To test the weight de-zero optimization, quantization error with the different sign operator is estimated using (4.2.2).

In general, the estimation process of quantization error involves three steps. The first step is to determine the quantization interval. While there are many ways to determine thresholds for different quantization intervals, for logarithmic quantization, only two are most commonly used.

One way is to use the average of two adjacent logarithmic weights as the threshold of the quantization interval. For example, there are two adjacent logarithmic weights 2^a and 2^b. And corresponding quantization interval threshold is $(2^a + 2^b)/2$.

The threshold of the second method is still in the form of the n-th power of two, and its power exponent is the average of the power exponents of two adjacent logarithmic weights. For instance, for two adjacent logarithmic weights 2^a and 2^b, the corresponding quantization interval threshold is $2^{(a+b)/2}$.

However, after experiments, the results show that different strategies for determining quantization interval thresholds are almost insignificant to accuracy. Consequently, this work does not further explore the optimal threshold in each quantization interval. And this work uses the second method to determine thresholds of different quantization intervals, which is also used in [10].

Take the case of bit width = 2, s = 2, i = -1 (binary operator) as an example: There are a total of 4 quantization intervals in this case:

- 1.(2^{-2},2^0] (corresponding quantization weight:2^{-1})

- 2.(0,2^{-2}] (corresponding quantization weight:2^{-3})

- 3.(-2^{-2},0] (corresponding quantization weight:-2^{-3})

- 4.[-2^{0},-2^{-2}] (corresponding quantization weight:-2^{-1})

The second step is to determine the number of weights in each quantization interval. For instance, using (4.2.3), the number of weights in [0,3δ] can be calculated as 49.87%·k·10^6. The final step is to use (4.2.2) to calculate the estimated quantization error. And in this example, due to symmetry, the total estimated quantization error can be calculated as twice the sum of estimation quantization error in the following three quantization intervals (2^{-2},2^{0}],(3δ,2^{-2}], and (0,3δ], where δ is assumed to be 0.05. And we can repeat the above-mentioned steps to continuously estimate the quantization error under different quantization weight sets. Four different weight sets in the 2-bit quantization case are tested to demonstrate the effect of weight de-zero optimization. Summarizing these results yields the following Table 4.1:

Table 4.1: Theoretical Quantization Error Comparison

weight set	***estimation quantization error***
$\pm2^{0}$**,**$\pm2^{-2}$	16279.95k
$\pm2^{0}$**,0**	3595.98k
$\pm2^{-3}$**,**$\pm2^{-5}$	573.12k
$\pm2^{-3}$**,0**	958.27k

It can be seen that zero-value weights play a significant role in weight sets without small-value weights. However, for weight sets with small-value weights, zero-value weights can be replaced by logarithmic weights with small absolute values. And this substitution theoretically does not accrue more quantization errors, on the contrary, it might reduce quantization errors. Another possible benefit of this replacement is that the weight sets after replacing zero have a larger number of different weights, which can improve the network expression ability.

4.4 Parameter Selection

Since in the proposed quantization error estimation model, both the convolutional network weights and the logarithmic quantization weights are symmetrical at about zero, the quantization error will also share such symmetry. In other words, in this model, the quantization error caused by positive quantization weights and negative quantization weights maintain the same. In order to obtain the theoretical optimal parameters s and i, it is necessary to calculate and compare the estimated quantization errors of different parameter combinations under (4.2.2).

The error estimation process is the same as what is mentioned in the previous section. Summarizing these results yields the following Table 4.2:

Table 4.2: Estimated Quantization Error

bit width, s and i	***estimation quantization error***
bit = 2, s = 2, i = 0	16279.95k
bit = 2, s = 2, i = -1	2158.14k
bit = 2, s = 2, i = -2	920.82k
bit = 2, s = 2, i = -3	**573.12k**
bit = 2, s = 2, i = -4	959.20k
bit = 2, s = 1, i = -3	809.33k
bit = 2, s = 1, i = -4	876.18k
bit = 2, s = 3, i = -2	1244.13k
bit = 2, s = 4, i = -1	1854.92k
bit = 3, s = 2, i = 0	407.23k
bit = 3, s = 2, i = -1	**244.77k**
bit = 3, s = 2, i = -2	542.38k
bit = 3, s = 2, i = -3	876.18k

It can be seen that for 2-bit and 3-bit logarithmic quantization, ”s=2, i=-3” and ”s=2, i=-1” are the theoretically optimal parameter sets respectively. It is worth to mention that some parameter sets such as ”s=1, i=0”, ”s=1, i=-1”, and ”s=1,i=-2” are intentionally neglected since these parameter sets cannot effectively achieve our purpose of extending the quantization range.

5 Benchmark Results

This chapter is outlined as follows: This chapter starts with the statement of the training process, then provides the benchmark results of different networks. Then this chapter analyzes and explains the advantage and disadvantage of JLQ.

We have tested the training results on 3 datasets: CIFAR10[39], CIFAR100[39], and Tinyimagenet[40]. For the CIFAR10 dataset, the original DeepShift-PS and DeepShift-Q are set as the baseline, and two DeepShift modes integrated with JLQ are set as the comparison group to get a preliminary conclusion. And CIFAR100 and Tiny ImageNet datasets are used to verify the scalability of JLQ.

For a fair comparison, we set all the training parameters consistent with those of DeepShift[10]. Different from DeepShift, which only shows the best accuracy results, the accuracy we show is the median after three experiments.

5.1 Benchmark Statement

In order to reduce the burden of memory, researchers often want a quantization method that is able to reduce bit-width for both, weights as well as activation data. In order to demonstrate the advantages of our proposed method in low-bit quantization, most of our experiments are based on 8 activation bits. And since in 4-bit quantization, JLQ might generate smaller weight values than those produced by the traditional logarithm quantization method, the 8-bit activation can be easily shifted to 0 owing to the bit width limit. For a fair comparison, we arranged a small number of experiments with 32 activation bits, which have sufficient bit width to tolerate the weight range expansion brought by JLQ in 4-bit quantization.

Unlike the original DeepShift which use 3 bits to represent the integer part and 5 bits to represent the fraction part for 8-bit activations, all the tested methods used 4 bits to represent the integer part, and 4 bits to represent the fraction part in this case. And since the training of DeepShift-PS at 8 activation bits fluctuates greatly and its loss curve is difficult to converge in this case according to our experiments, we select the highest accuracy during the entire training process as the final accuracy value of a single experiment when doing experiments in relation to DeepShift-PS. Meanwhile, according to the results of DeepShift[10] and Deeplook[37], for logarithmic quantization, the 5-bit quantization only has a slight advantage over the 4-bit quantization. Consequently, we will not consider the 5-bit quantization case in our experiments.

In our test strategy, we will first test the jumping logarithmic quantization with the 2-bit theoretical optimal parameter set ”s=2, i=-3” compared with the DeepShift baseline. At the same time, in order to test the effect of weight de-zero optimization on the accuracy, experiments using the ternary sign operator and binary sign operator will also be performed based on the CIFAR10 dataset. After that, we will test the accuracy of jumping logarithmic quantization with other parameter sets to verify the optimal parameter set prediction provided

by our error estimation model.

In our tables, "W" refers to the number of bits to represent weights, "A" refers to the number of bits to represent activation bits, "T" refers to the ternary sign operator, and "B" refers to the binary sign operator, and "Acc@N" accuracy means that the correct class gets to be in the Top-N probabilities for it to count as "correct". We also highlight the useful data in our table to render the data comparison and conclusion extraction more conveniently.

5.2 CIFAR10 Dataset

The accuracy results shown in table 5.1 are based on the results reproduced by us instead of the results from the original DeepShift paper[10].

Table 5.1: CIFAR10 benchmark from pre-trained, Acc@1

W	A	Sign	Base-PS	JLQ-PS	Base-Q	JLQ-Q
2	8	T	81.75%	90.04%	90.37%	**93.03%**
2	8	B	81.11%	89.89%	89.46%	**93.15%**
3	8	T	90.53%	90.07%	92.67%	**93.27%**
3	8	B	90.37%	89.81%	92.01%	**93.31%**
4	8	T	93.75%	90.18%	**93.95%**	93.29%
4	8	B	93.61%	90.06%	**93.88%**	93.34%
4	32	T	**94.06%**	90.24%	94.05%	93.46%
4	32	B	93.91%	90.17%	**93.95%**	93.37%

We can see that in 2-bit quantization, DeepShift-PS integrated with jumping logarithmic quantization has better performance than the corresponding baseline. Meanwhile, DeepShift-Q integrating with jumping logarithmic quantization achieves higher accuracy than any baseline in 2-bit and 3-bit quantization.

Another interesting phenomenon is that in the 2-bit and 3-bit quantization, the baseline Q using the ternary sign operator has obvious advantages in accuracy compared to the baseline Q using the binary sign operator. This is because, in the case of low-bit quantization, the traditional logarithmic quantization does not have enough bits to generate a logarithmic quantization weight close to 0. And the weight with a value of 0 at this time can be regarded as an expansion of the quantization range, which can avoid a large amount of quantization error caused by weight values concentrated around 0. And this advantage is doomed to become more obvious when the network becomes larger owing to the spur in the number of weights.

This phenomenon is not clearly observed in 4-bit and 5-bit quantization. For the baseline Q combined with jumping logarithmic quantization, the accuracy obtained by using the ternary sign operator is slightly lower than that obtained by using the binary sign operator. This is because when the number of bits becomes sufficient, logarithmic quantization is able to produce logarithmic weights around 0, which weakens the effect of weights with zero value. Similarly, since jumping logarithmic quantization extends the quantization range, generating weight values near 0 in low-bit quantization becomes possible, which makes the weight value of 0 can be replaced by logarithmic weights without loss of accuracy.

We also find some drawbacks of JLQ. For 4-bit or larger bit-width quantization, it brings little to no accuracy improvement than 3-bit quantization. Meanwhile, we can also observe that in 4-bit weight and 32-bit activation cases, even though the weights of JLQ are fully utilized, JLQ still has no advantage over the corresponding baseline. This can be explained by the following two reasons. The first reason is that JLQ adjusts the step size and overdraws the accuracy improvement by expanding the quantization range in advance. Another reason is similar to the reason why 5-bit logarithmic quantization has little improvement over 4-bit logarithmic quantization. For most non-sparse CNN models when the logarithmic weight is less than a certain small value, the mutual substitution between those adjacent small logarithmic weights will become very obvious. For example, adding or removing weight value $\pm 2^{-8}$,$\pm 2^{-9}$ or $\pm 2^{-10}$ by adjusting the quantization range will have almost no impact when there exists 2^{-7} in the original quantization case.

As for the performance of other parameter settings, to fully demonstrate the advantage of extending the quantization range, we intentionally filter parameter sets such as "s=1, i=-1", "s=1, i=-2" and so on, which only achieve small range extension. And the results are shown in table 5.2. JLQ-Q results of "s = 2, i= -3" are mentioned again for the convenience of conclusion extraction.

Table 5.2: CIFAR10 ResNet18 based on other parameter settings of JLQ (Method=JLQ-Q, A=8, Sign=B)

Parameters	W	From pre-trained
s = 1, i = -3	2	89.56%
s = 1, i = -4	2	89.98%
s = 2, i = -1	2	87.81%
s = 2, i = -2	2	90.60%
s = 2, i = -3	2	**93.15%**
s = 1, i = -3	3	93.01%
s = 1, i = -4	3	91.90%
s = 2, i = -1	3	**93.33%**
s = 2, i = -2	3	93.10%
s = 2, i = -3	2	93.31%

We can see that in 2-bit quantization, the parameter set that achieves the best accuracy is "s=2, i=-3", and in 3bit quantization, the best parameter set is s=2, i=-1. Both results are consistent with the prediction of the error estimation model.

To demonstrate our accuracy advantage in low-bit quantization, we also compare our best results with the best results in the original DeepShift paper. The comparison is demonstrated in table 5.3. It can be seen that even though our quantization method has fewer activation bits, it still outperforms its counterpart quantization algorithm.

Table 5.3: CIFAR10 ResNet18 result comparison

Method	W	A	Top1 Accuracy
DeepShift-PS[10]	2	32	92.80%
JLQ-Q(s = 2, i = -3)	2	8	**93.15%**
DeepShift-PS[10]	3	32	92.85%
JLQ-Q(s = 2, i = -1)	3	8	**93.33%**

We can conclude that the DeepShift-Q benefits greatly from JLQ in extremely low-bit quantization cases and can integrate weight de-zero optimization with no accuracy loss.

5.3 CIFAR100 Dataset

As for the CIFAR100 dataset, we simplified our experiments based on the results of CIFAR10. We intentionally removed experiments in relation to Baseline PS and JLQ-PS, considering Baseline Q shows more benefits after integrating with JLQ. We conduct experiments with two network structures, ResNet18 and GoogleNet, to verify the scalability of JLQ. For the convenience of hardware implementation, we keep the weight de-zero optimization and apply it to all experiments of CIFAR100. Besides, since DeepShift baseline[10] also supports training from scratch, we add experiments about training from scratch to further verify the scalability of JLQ. Additionally, we add a control group named "original" that does not implement any quantization to better demonstrate the accuracy comparison. The accuracies based on ResNet18 are shown in table 5.4. The results of JLQ based on ResNet18 under other parameter settings are shown in table 5.5.

Table 5.4: ResNet18 CIFAR100 benchmark (Sign=B)

Method	W	A	From scratch		From pre-trained	
			Acc@1	Acc@5	Acc@1	Acc@5
Original	32	32	74.75%	92.97%	-	-
Base-Q	2	8	28.05%	58.29%	64.97%	86.69%
JLQ-Q	2	8	**71.02%**	**90.97%**	**72.33%**	**91.43%**
Base-Q	3	8	61.79%	87.68%	66.84%	88.45%
JLQ-Q	3	8	**72.73%**	**91.40%**	**73.45%**	**92.17%**
Base-Q	4	8	**74.10%**	**92.46%**	**73.99%**	**92.60%**
JLQ-Q	4	8	72.75%	91.51%	73.33%	92.13%
Base-Q	4	32	**74.33%**	**92.56%**	**74.14%**	**92.71%**
JLQ-Q	4	32	73.15%	91.77%	73.57%	92.36%

Since the CIFAR100 dataset has more types of images than the CIFAR10 dataset, it becomes more difficult to identify the CIFAR100 dataset. In this case, the advantages of JLQ are more obvious. In the 2-bit and 3-bit quantization cases, whether the model is trained from scratch or from a pre-trained model, the accuracy is dramatically improved by implementing JLQ. However, in the 4-bit quantization case, the JLQ shows no advantages compared to the baseline as a trade-off.

As for the parameter sets, according to the results in table 5.5, the optimal parameter set maintains the same as those in CIFAR10.

Table 5.5: Other combinations CIFAR100 ResNet18 (Method=JLQ-Q, A=8, Sign=B)

Parameters	W	From scratch		From pre-trained	
		Acc@1	Acc@5	Acc@1	Acc@5
s = 1, i = -3	2	67.14%	89.09%	69.58%	89.79%
s = 1, i = -4	2	67.71%	89.35%	71.49%	91.09%
s = 2, i = -1	2	60.23%	86.76%	67.00%	87.91%
s = 2, i = -2	2	68.52%	89.75%	69.71%	89.82%
s = 2, i = -3	2	**71.02%**	**90.97%**	**72.33%**	**91.43%**
s = 1, i = -3	3	71.90%	91.31%	73.53%	92.38%
s = 1, i = -4	3	71.38%	91.09%	73.36%	92.21%
s = 2, i = -1	3	**73.02%**	**91.70%**	**73.77%**	**92.45%**
s = 2, i = -2	3	72.81%	91.47%	73.35%	92.14%
s = 2, i = -3	3	72.73%	91.40%	73.45%	92.17%

The accuracies based on GoogleNet are shown in table 5.6. The results of JLQ based on GoogleNet under other parameter settings are shown in table 5.7.

Table 5.6: GoogleNet CIFAR100 benchmark (Sign=B)

Method	W	A	From scratch		From pre-trained	
			Acc@1	Acc@5	Acc@1	Acc@5
Original	32	32	78.17%	94.58%	-	-
Base-Q	2	8	44.97%	76.30%	62.90%	87.07%
JLQ-Q	2	8	**76.51%**	**93.82%**	**76.36%**	**93.65%**
Base-Q	3	8	67.89%	90.44%	67.73%	89.80%
JLQ-Q	3	8	**77.11%**	**94.19%**	**77.27%**	**94.06%**
Base-Q	4	8	**77.52%**	**94.37%**	**78.06%**	**94.55%**
JLQ-Q	4	8	77.15%	94.14%	77.38%	94.19%
Base-Q	4	32	**77.80%**	**94.56%**	**78.09%**	**94.47%**
JLQ-Q	4	32	77.35%	94.27%	77.63%	94.31%

In terms of accuracy, the conclusions in GoogleNet are consistent with those in ResNet. The accuracy after implementing JLQ far exceeds that of Baseline in the case of 2-bit and 3-bit quantization, while this advantage disappears in 4-bit quantization.

Table 5.7: Other combination CIFAR100 GoogleNet (Method=JLQ-Q, A=8, Sign=B)

Base-Q + JLQ	W	A	From scratch		From pre-trained	
			Acc@1	Acc@5	Acc@1	Acc@5
s = 1, i = -3	2	8	72.84%	93.09%	74.45%	93.53%
s = 1, i = -4	2	8	68.46%	90.68%	72.03%	92.08%
s = 2, i = -1	2	8	68.37%	91.16%	68.71%	90.09%
s = 2, i = -2	2	8	72.78%	93.12%	73.92%	93.08%
s = 2, i = -3	2	8	**76.51%**	**93.82%**	**76.36%**	**93.65%**
s = 1, i = -3	3	8	77.19%	94.24%	77.60%	94.21%
s = 1, i = -4	3	8	77.07%	94.17%	77.20%	94.11%
s = 2, i = -1	3	8	77.12%	94.15%	77.31%	94.16%
s = 2, i = -2	3	8	**77.34%**	**94.30%**	**77.63%**	**94.23%**
s = 2, i = -3	3	8	77.11%	94.19%	77.27%	94.06%

As for the parameter sets, the optimal parameter set in 2-bit quantization still maintains the same, while the optimal parameter set in 3-bit quantization is different from the prediction of our error estimation model. The optimal parameter set changes from "s=2,i=-1" to "s=2,i=-2" owing to the variance of weight distribution. However, the accuracy generated in the "s=2,i=-1" parameter set only has slight differences compared with that generated in the "s=2,i=-2" parameter set, which manifests that the prediction of our error estimation model is still very reliable.

5.4 Tiny ImageNet Dataset

In order to test whether JLQ can maintain the accuracy advantage in the case of extremely low-bit quantization (2-bit and 3-bit) on larger datasets, we conducted experiments based on the Tiny ImageNet dataset. The Tiny ImageNet dataset [40] is a modified subset of the original ImageNet dataset [41] with 200 different classes, 100,000 training examples, and 10,000 validation examples. The resolution of the images is only 64x64 pixels, which makes it more challenging to extract information from it than the original ImageNet dataset. The experiment settings are almost the same as those in previous CIFAR100 experiments. The only difference is that we consider two different CNN networks, ResNet18[4], and Inception v3[42] using pre-trained models this time. To get pre-trained models, we first train the networks from scratch in 90 epochs based on Imagenet pre-trained weights. In addition, to increase the baseline of Resnet18, we use the fine-tuning method by removing the max pooling layer to reduce information loss of the image in the early stage of CNN.

The accuracies based on ResNet18, and Inception-v3 are shown in table 5.8, and 5.9 correspondingly.

Table 5.8: ResNet18 Tiny ImageNet benchmark (Sign=B)

Method	W	A	From scratch	
			Acc@1	Acc@5
Original	32	32	60.30%	82.10%
Method	W	A	From pre-trained	
			Acc@1	Acc@5
Base-Q	2	8	37.40%	62.54%
JLQ-Q(s=2,i=-3)	2	8	**58.19%**	**80.82%**
Base-Q	3	8	48.60%	73.51%
JLQ-Q(s=2,i=-1)	3	8	**58.53%**	**81.13%**

Table 5.9: Inception-v3 Tiny ImageNet benchmark (Sign=B)

Method	W	A	From scratch	
			Acc@1	Acc@5
Original	32	32	68.09%	86.89%
Method	W	A	From pre-trained	
			Acc@1	Acc@5
Base-Q	2	8	10.08%	28.81%
JLQ-Q(s=2,i=-3)	2	8	**66.02%**	**85.38%**
Base-Q	3	8	59.83%	81.57%
JLQ-Q(s=2,i=-1)	3	8	**66.37%**	**85.63%**

It can be clearly seen that although the dataset has become larger, compared to the baseline, JLQ still maintains the advantage of accuracy in a very low-bit quantization case.

6 Hardware Design

This chapter is outlined as follows: This chapter starts with a brief introduction of a systolic-array-based accelerator that will be used to implement proposed PEs later, then provides the design of the proposed PE and proposed shifter. Then resource utilization, area, and power consumption of the proposed PE standing alone and in a systolic array prototype are reported.

6.1 Systolic Array System Overview

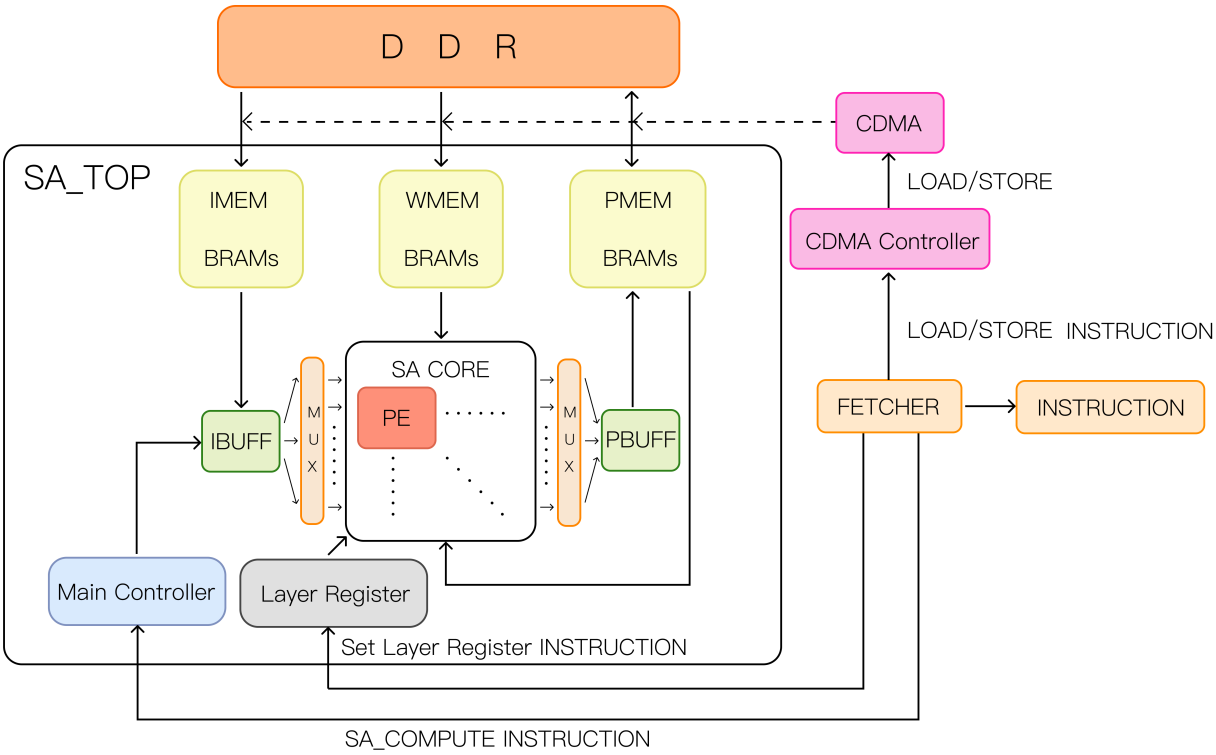


Figure 6.1.1: Top-level hardware architecture of the Neural Network Accelerator.

Since the proposed PE will be implemented in a systolic-array-based accelerator (shown in Figure.6.1.1) to test its performance, this section will provide an overview of this systolic-array-based accelerator. The system flow, systolic array inference flow, and processing element flow are shown in Figure. 6.1.2(1), Figure. 6.1.2(2), Figure. 6.1.2(3) respectively. The computation block is based on systolic arrays and each systolic array is built by 64 Processing Elements (PEs).

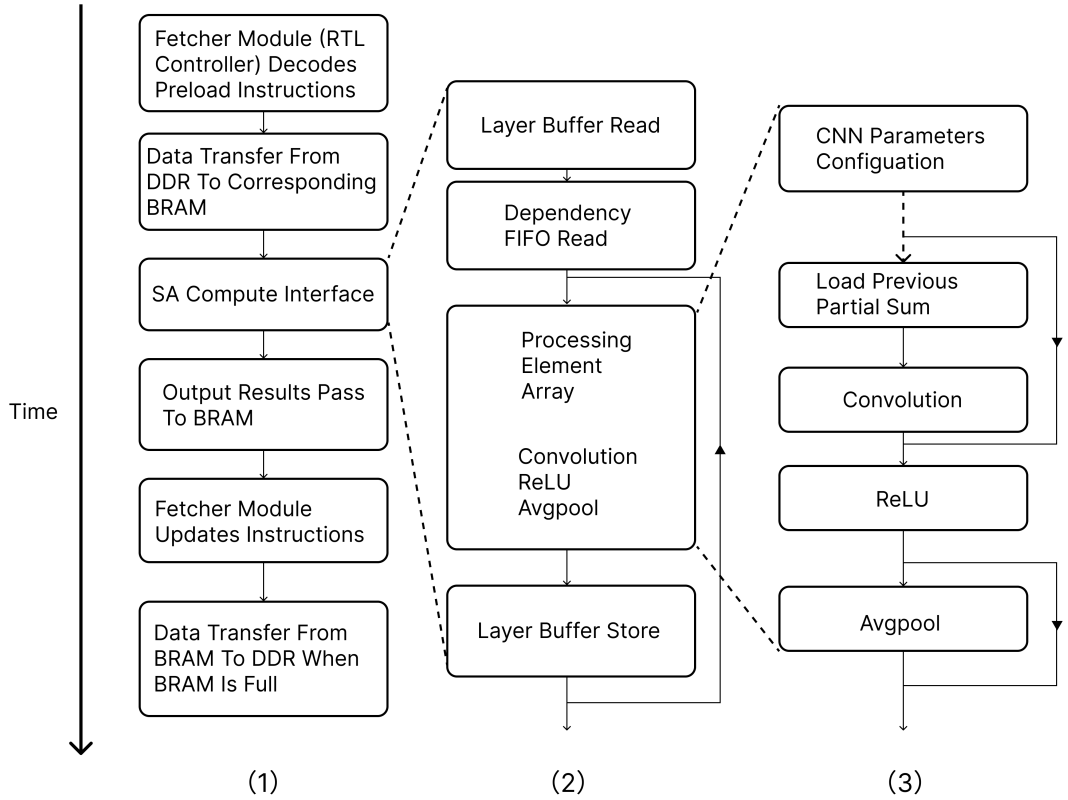


Figure 6.1.2: (1) system timing diagram, (2) inference flow, and (3) Processing unit flow diagram.

The systolic-array-based accelerator is mainly composed of three parts: data movement part, computation part, and control logic part. For the data movement part, since the FPGA's internal available BRAM size is often unable to store the data of a complete network, we load and store data through external DDR3 memory. As for the input data and weight, they are firstly stored in DDR and then passed to corresponding BRAMs (IMEMs and WMEMs). Then, when PMEMs were full after computation, the CDMA will transfer the data in PMEMs to DDR3 to make room for new outputs. The computation part is composed of the systolic array and some other small computation modules, which achieves the convolution operation and Relu activation operation. And the control logic part is composed of the CDMA Controller and the fetcher module. The control logic part is responsible for fetching instructions, controlling CDMA, and coordinating the dependency of different modules and instructions. The control logic part is the side work of this thesis.

6.2 Design Of Processing Element

The processing element serves as the core of neural network acceleration. Therefore, the optimization of a PE will directly affect the efficiency of the overall hardware design. A general weight-stationary shifter-based PE design is illustrated in Figure.6.2.1. The shift

operation of the feature and the weight is first conducted, then followed by a negative or positive value selection and zero value selection.

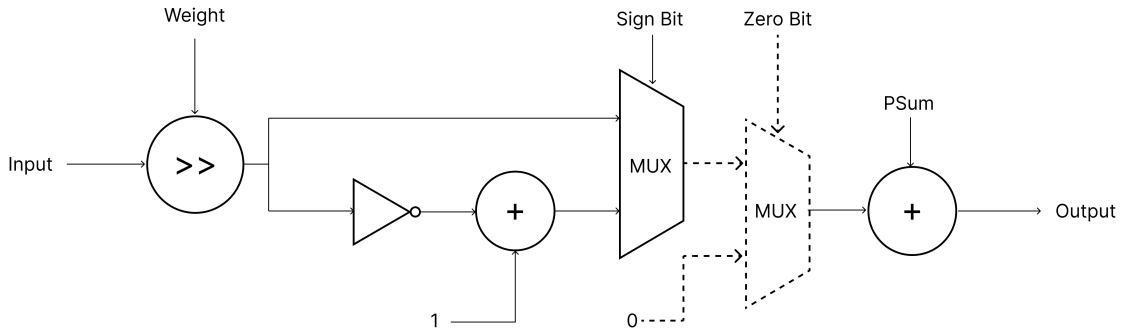


Figure 6.2.1: Shifter-Based PE (Our PE is the solid part, and the dotted part is what we remove from traditional shifter-based PE in our optimization).

Since our quantization method does not have the weight of zero value, our PE removes a multiplexer in relation to zero bit.

6.3 Design Of Shifter

Although JLQ extends the quantization range, it does not mean that it requires a larger shifter for quantization weights. By modifying the barrel shifter, our PE achieves the purpose of range extension without adding any hardware burden. Taking the 3-bit jumping log quantization with step = 2 i = -1 as an example, the logarithmic quantization weights, in this case, are as follows:($\pm 2^{-1}$,$\pm 2^{-3}$,$\pm 2^{-5}$,$\pm 2^{-7}$). These weight values will correspond to the shift operations of $\gg 1$, $\gg 3$, $\gg 5$, and $\gg 7$ respectively. And the operations of $\gg 1$, $\gg 3$, $\gg 5$, $\gg 7$ only entail a 2-bit barrel shifter shown in Figure.6.3.1 instead of a traditional 3-bit barrel shifter to achieve. In this example shifter, the layer of multiplexers that operate right shift one bit is removed. To further save hardware resources, some multiplexers are substituted with AND gates or are neglected directly. The first layer of AND gates and multiplexers represents the $\gg 4$ operation, and the second layer represents the $\gg 2$ operation. Besides, the input is preshifted 1 bit in advance. Consequently, $\gg 1$, $\gg 3$, $\gg 5$, $\gg 7$ operations can be achieved using this shifter.

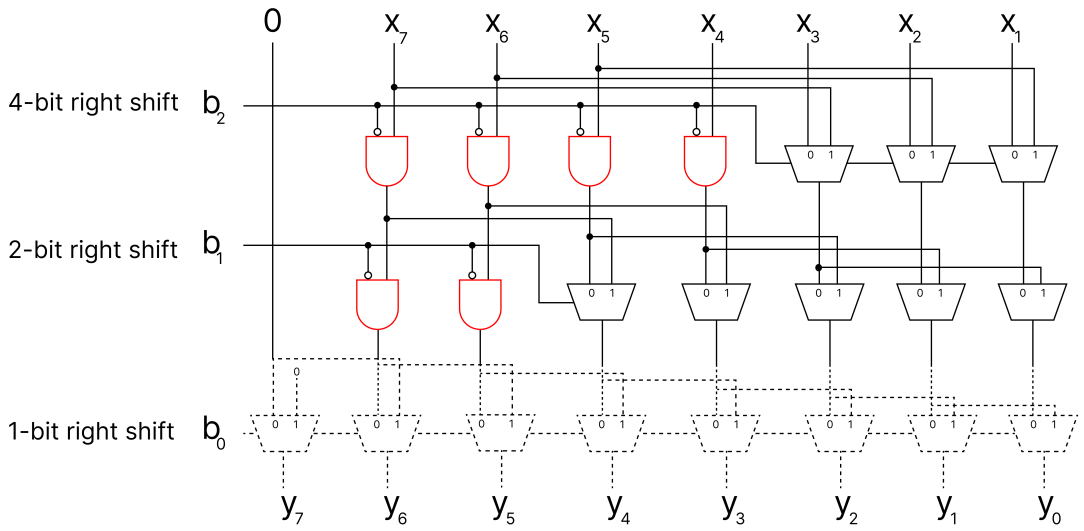


Figure 6.3.1: Shifter Design (Our shifter is the solid part and the red part. The dotted part is what we remove from the traditional 3-bit barrel shifter in our optimization, and the red part indicates the replacement of multiplexers with AND gates. The first column of multiplexers can be directly neglected owing to preshift operation).

6.4 Simulation Results

6.4.1 PE Simulation

In order to calculate area and power consumption, different types of PE were synthesized using the Synopsys Design Compiler(TSMC 28nm technology). And the resource utilization of different types of PE is tested on Xilinx ARTIX7 XC7A100T using Synplify Pro. The basic resulting resource utilization, area consumption, and power consumption for different types of PEs are given in Table6.1. And the normalized results of resource utilization, area, and power consumption are shown in Figure.6.4.1, Figure.6.4.2, and Figure.6.4.3.

Table 6.1: resources consumption in single PE

PE type	W	LUT[a]	A(μm^2)[b]	P(mw)[b]
multiplier-based PE	3	55	221.186	1.3200
shifter-based PE	3	46	209.720	1.1928
our PE	3	33	203.840	1.1879
multiplier-based PE	2	43	200.900	1.2047
shifter-based PE	2	43	202.664	1.1567
our PE	2	28	192.178	1.1516

[a]For Xilinx ARTIX7 XC7A100T FPGA
[b]For TSMC 28nm technology (ASIC)

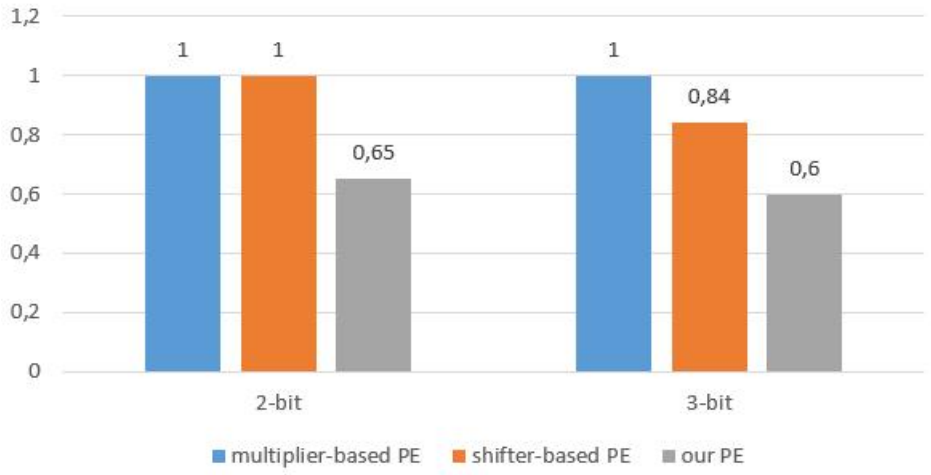


Figure 6.4.1: Normalized LUT Resources.

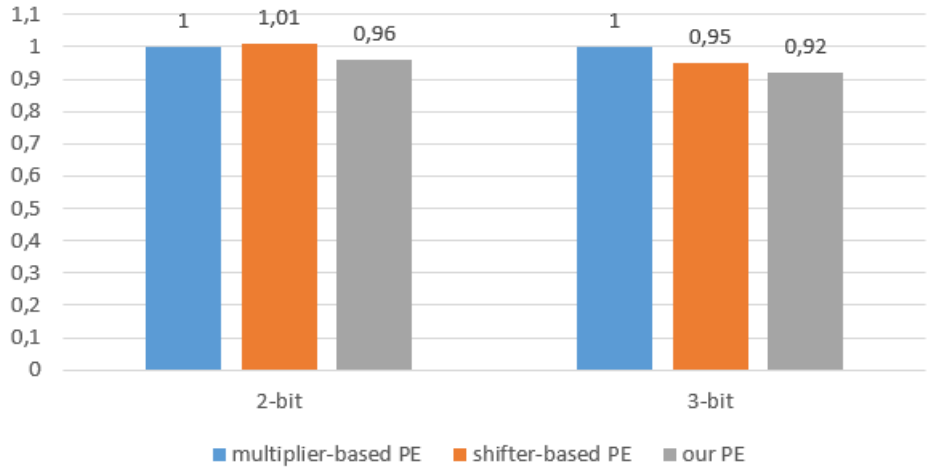


Figure 6.4.2: Normalized Area.

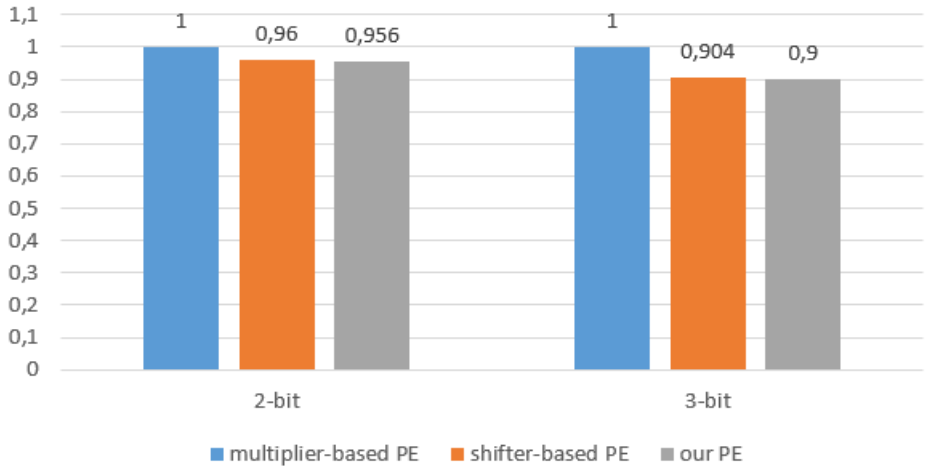


Figure 6.4.3: Normalized Power.

It can be seen that our PE has advantages in LUT utilization, area, and power consumption over both traditional shifter-based PE and multiplier-based PE. Specifically, as shown in Figure.6.4.1, our PE reduces the 35% and 28.6% LUT compared with traditional shifter-based PE in the 2-bit quantization case and 3-bit quantization case respectively. As for area and power consumption, although the differences are not as prominent as those in LUT utilization, our PE still performs better than traditional shifter-based PE.

6.4.2 CNN Accelerator Simulation

An MNIST example is used in the systolic-array-based CNN accelerator to estimate corresponding power consumption. And the network implemented in this accelerator is shown in Table.6.2.

Table 6.2: Simple Convolution Network

number of layers	type	kernel size	stride	input channel	output channel	padding.[a]	activation function
First layer	Convolution	5*5	2	1	16	False	ReLu
Second Layer	Convolution	3*3	2	16	16	False	ReLu
Third Layer	Convolution	5*5	1	16	10	False	ReLu

[a] Padding with 0

Table 6.3 shows the results of comparisons with the same systolic-array-based CNN accelerator implemented with different PE on LUT utilization and power consumption. And the normalized results of LUT utilization and dynamic power consumption are shown in Figure.6.4.4, Figure.6.4.5. The CNN accelerator implemented with the proposed PE achieves less LUT consumption and power consumption compared to that implemented with traditional multiplier-based PE and shifter-based PE under the same bit-width. Specifically, when the weight bit width is 2, the accelerator with our PE has 6% and 7% dynamic power reduction compared to the accelerator with multiplier-based PE and the accelerator with shifter-based PE, respectively. And when the weight bit width is 3, the accelerator with our PE achieves 8% and 5% dynamic power reduction compared to its above-mentioned counterparts respectively.

Table 6.3: Simulation Results Comparison

Accelerator type	W	Quantization Method	Acc.[a]	LUT	DP(w).[b]	SP(w).[c]	TP(w).[d]
With multiplier-based PE	2	Floating Point	96.72%	33088	0.107	0.416	0.523
With shifter-based PE	2	Base-Q(ternary)	96.65%	35066	0.108	0.416	0.524
With proposed PE(for s=1)	2	Base-Q(binary)+preshift.[e]	97.32%	31545	0.102	0.416	0.518
With proposed PE(for s=2)	2	JLQ-Q(S=2,i=-3)	97.57%	31545	0.102	0.416	0.518
With multiplier-based PE	3	Floating Point	97.39%	40686	0.129	0.417	0.546
With shifter-based PE	3	Base-Q(ternary)	97.47%	39920	0.115	0.416	0.531
With proposed PE(for s=1)	3	Base-Q(binary)+preshift.[f]	97.65%	35725	0.110	0.416	0.526
With proposed PE(for s=2)	3	JLQ-Q(S=2,i=-1)	97.76%	35725	0.110	0.416	0.526

[a] based on the full-precision pre-trained model (97.70% accuracy) using the aforementioned implemented network with the same training settings in DeepShift[10]
[b] Dynamic Power
[c] Device Static Power
[d] Total Power
[e] equivalent to JLQ (S=1,i=-3)
[f] equivalent to JLQ (S=1,i=-1)

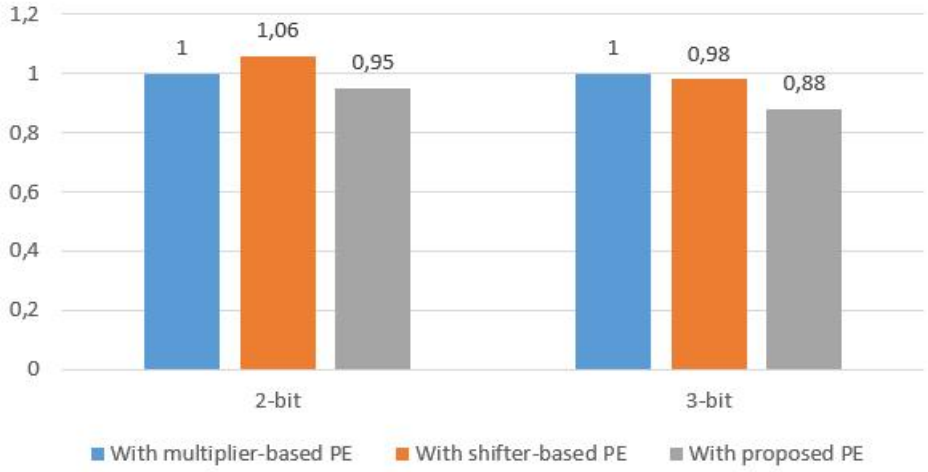


Figure 6.4.4: Normalized LUT Resources for CNN accelerator.

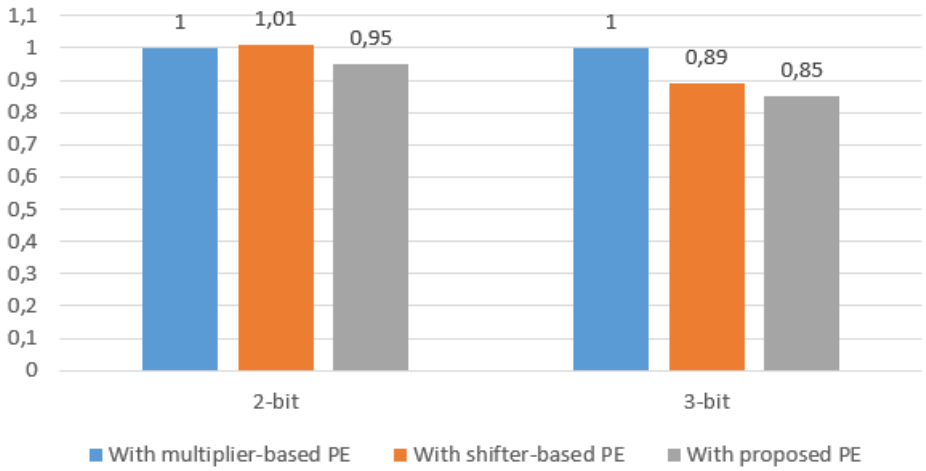


Figure 6.4.5: Normalized Dynamic Power for CNN accelerator.

Since the testing network is small, the number of weights that need to be transferred and the corresponding power consumption are also small. If the dataset and the network become larger, the power consumption reduction and accuracy advantage of JLQ in the low-bit quantization case will become more obvious.

7 Conclusions

This chapter concludes this thesis work. In addition, it addresses the thesis's research questions and then gives some insights about future work based on the thesis results. Afterward, it ends with the paper accepted by the DATE 2023 conference relating to this thesis.

7.1 Thesis Conclusions

In this thesis, firstly, for the software part, we present JLQ, a new network quantization method, to address the problem of how to improve the accuracy of logarithmic quantization in extremely low-bit quantization cases. Unlike existing methods that usually rely on the retraining strategy to alleviate the impact of quantization noise in extremely low-bit quantization cases, JLQ achieves the same purpose by using 2^{sx+i} to extend the quantization range. According to our error estimation model, for JLQ in 2-bit quantization, the optimal parameter settings are "s=2, i=-3". And for JLQ in 3-bit quantization, the optimal parameter settings are "s=2, i=-1". These optimal parameter settings in theory are also verified by our benchmark results. Our benchmark results also show that JLQ has the accuracy advantage in 2-bit and 3-bit quantization cases compared to baseline and counterpart algorithms. And this accuracy advantage will become more obvious for large-scale classification tasks such as TinyImageNet[40]. The drawback of JLQ is that the accuracy advantage disappears in 4-bit (or above) quantization cases. Secondly, for the hardware part, a new shifter that can efficiently JLQ-ed the weights is designed. And considering zero value cannot be represented by the shifter, a weight de-zero optimization is invented, which is already verified in JLQ benchmark results. By incorporating the new shifter and weight de-zero optimization, a new PE is designed. According to our simulation results, our PE has advantages in LUT utilization, area, and power consumption compared to its counterpart PEs. And the advantages still remain when our PE is put in a real systolic-array-based accelerator.

7.2 Addressing Research Questions

The research questions that are listed in Chap. 1.1 were:

1. How to optimize current state-of-art logarithmic quantization methods in extreme low-bit quantization cases for low-power hardware implementation?
2. How to optimize shifter-based processing elements based on logarithmic quantization?

How to optimize current state-of-art logarithmic quantization methods in extreme low-bit quantization cases for low-power hardware implementation?

Compared with other methods that are good at low-bit quantization, such as ternary networks[35], the advantage of logarithmic quantization is that logarithmic quantization

weights are not fixed to 0 and 1. Logarithmic quantization weights can be represented by any n-th power of 2. Therefore, a feasible solution is to test the accuracy using various logarithmic weights sets under low-bit quantization case. And theoretically, a mathematical model can be built to predict the optimal set of logarithmic weights that minimizes the quantization error under low-bit quantization. Previous research has shown that both large and small weights are important. Among them, the importance of the weight with a larger absolute value is reflected in feature extraction, while the importance of the weight with a smaller absolute value is reflected in reducing the quantization error of the entire model. Based on this, this work proposes a logarithmic quantization method called jumping logarithmic quantization (JLQ), which introduces two additional parameters to extend the range of logarithmic weights. By intentionally JLQ-ed some logarithmic weights, the final set of logarithmic weights can contain both large logarithmic weights and small logarithmic weights at the same time. The results show that JLQ is very advantageous in the aspect of accuracy under low-bit quantization such as 2-bit and 3-bit quantization. The only drawback of JLQ is that for quantization cases in 4-bit and above, JLQ loses the advantage of accuracy compared with other logarithmic quantization methods.

How to optimize shifter-based processing elements based on the logarithmic quantization?

There are two approaches to further optimize shifter-based PE. The first approach is to delete the parts that are not just needed in PE as much as possible. Since zero-value weights cannot be processed by the shifter, a multiplexer is entailed in the traditional shifter-based PE to process zero-value weights. This work replaces all zero-value weights with logarithmic weights in the proposed JLQ so that all weights can be processed by shift operations. Therefore, the proposed shifter-based PE does not require an additional multiplexer to deal with zero-value weights. The second approach is to optimize the shifter as much as possible. Since the proposed JLQ in this work intentionally filters some logarithmic weights, this work optimizes the classical barrel shifter so that the proposed PE can efficiently implement JLQ with fewer hardware resources.

7.3 Future Work

We have proven that JLQ (with weight de-zero optimization) performs well on small datasets in extremely low-bit quantization. But the verification based on big datasets such as Imagenet[41] is not yet provided in this work. It is expected to reach similar conclusions since the baseline DeepShift-Q[10] are already proved suitable for Imagenet. In addition, in the aspect of hardware, putting our PE into a real CNN accelerator that can process large networks (for example ResNet[4]) and large datasets (for example Imagenet[41]) to evaluate the overall throughput, resource utilization, and power consumption is not done in our work. We expect a strong reduction in resource utilization and power consumption since JLQ is able to dramatically reduce the bit-width of the memory from the common-used bit down to 3-bit or 2-bit with only little accuracy loss compared to full precision networks. Finally, the integration of non-conflicting optimizations such as incremental re-training (used in INQ[9]), quantization range extension (used in JLQ), padding and activation function optimization (used in ReActNet[13]) and so on is also a very interesting future topic for logarithmic quantization.

7.4 Future Work

We have proven that JLQ (with weight de-zero optimization) performs well on small datasets in extremely low-bit quantization. But the verification based on big datasets such as Imagenet[41] is not yet provided in this work. It is expected to reach similar conclusions since the baseline DeepShift-Q[10] are already proved suitable for Imagenet. In addition, in the aspect of hardware, putting our PE into a real CNN accelerator that can process large networks (for example ResNet[4]) and large datasets (for example Imagenet[41]) to evaluate the overall throughput, resource utilization, and power consumption is not done in our work. We expect a strong reduction in resource utilization and power consumption since JLQ is able to dramatically reduce the bit-width of the memory from the common-used bit down to 3-bit or 2-bit with only little accuracy loss compared to full precision networks. Finally, the integration of non-conflicting optimizations such as incremental re-training (used in INQ[9]), quantization range extension (used in JLQ), padding and activation function optimization (used in ReActNet[13]) and so on is also a very interesting future topic for logarithmic quantization.

7.5 Paper Acceptance

The contributions of this thesis work are also described in the following paper accepted by the DATE 2023 conference.

Jumping Shift: A Logarithmic Quantization Method for Low-Power CNN Acceleration*

***Abstract*—Logarithmic quantization for Convolutional Neural Networks (CNN): a) fits well typical weights and activation distributions, and b) allows the replacement of the multiplication operation by a shift operation that can be implemented with fewer hardware resources. We propose a new quantization method named Jumping Log Quantization (JLQ). The key idea of JLQ is to extend the quantization range, by adding a coefficient parameter "s" in the power of two exponents (2^{sx+i}). This quantization strategy skips some values from the standard logarithmic quantization. In addition, we also develop a small hardware-friendly optimization called weight de-zero. Zero-valued weights that cannot be performed by a single shift operation are all replaced with logarithmic weights that to further reduce hardware resources with almost no accuracy loss. To implement the Multiply-And-Accumulate (MAC) operation (needed to compute convolutions) when the weights are JLQ-ed and de-zeroed, a new Processing Element (PE) have been developed. This new PE uses a modified barrel shifter that can efficiently avoid the skipped values. Resource utilization, area, and power consumption of the new PE standing alone are reported. We have found that JLQ performs better than other state-of-the-art logarithmic quantization methods when the bit width of the operands becomes very small.**

***Index Terms*—Convolutional Neural Network, Low-power hardware acceleration, Logarithmic Quantization, FPGA**

I. Introduction

In recent years, there has been a lot of interest in deep learning. Convolutional Neural Network (CNN), one of the most mature deep learning models, has attracted attention in various fields such as medical research [1] [2], language processing [3], and visual imagery [4] [5] [6]. Despite its popularity, due to its huge data volume, intensive computation, and frequent memory access, deploying a CNN on low-power hardware systems is still challenging. To make deep neural networks generally easier to deploy on hardware devices like FPGA and ASIC, various quantization algorithms [7] [8] [9] have been devised to reduce memory requirements. Among all quantization methods, logarithmic quantization is very suitable for low-power inference because as logarithmic weights are represented by powers of two, the memory only needs to store the integer power index instead of the floating point weight. Besides, they fit better the common CNN Gaussian-like distributions of weights and activations, than the more uniform integer quantization. And furthermore, they allow the replacement of the multiplication operation (massively involved in CNN computation) by a shift operation, which can be implemented with fewer hardware resources.

Because of all of the previous reasons, logarithmic quantization effectively reduces the CNN memory footprint and reduces the area and power consumption of the Processing Elements (PEs) involved in CNN computation.

In this paper, we introduce a logarithmic quantization technique named Jumping Logarithmic Quantization (JLQ) that can achieve higher accuracy than traditional logarithmic quantization at low-bit quantization by extending the quantization range. We performed tests on CIFAR10 [10], CIFAR100 [10], and Tiny ImageNet [11] to evaluate the accuracy based on our quantization technique. Compared with the state of art logarithmic quantization algorithm, our method has obvious advantages in the low-bit (2 or 3-bit) quantization cases. Besides, we develop a hardware-friendly weight de-zero optimization. Hardware resources are further reduced by replacing zero-valued weights with logarithmic weights that can be performed by shift operations. In addition, we design a processing element (PE) architecture based on JLQ and weight de-zero optimization, aiming to perform MAC operations of CNN models with fewer resources, lower area, and power consumption. The processing element (PE) realizes the quantization range expansion of JLQ without introducing additional hardware resources by optimizing the traditional barrel shifter. For ASIC testing, we synthesized our PE design, a multiplier-based PE design, and a traditional shifter-based PE design using TSMC 28nm technology (in the 3-bit and 2-bit quantization case). Compared with other competitors, our PE design has less area and power consumption. In addition, we also conduct experiments regarding the resource consumption of different PEs on Xilinx ARTIX7 XC7A100T FPGA. The results show that our PE design also has advantages in this aspect. Specifically, compared with traditional shifter-based PE, our PE achieves 35% and 28.6% LUT reduction in 2-bit and 3-bit quantization cases respectively.

The rest of this paper is organized as follows. The related work on log quantification is described in Section II. Then, we introduce our proposed quantization method, JLQ, in Section III. Section IV presents the accuracy results of JLQ on various datasets and network structures. In Section V, we present our PE design and the corresponding hardware implementation results. Finally, Section VI is the conclusion.

II. Related Work

LogNN [12] was the first to propose logarithmic quantization for CNNs. They propose two methods: in the first method, the weights remain fix-point and the activations are log-quantized. While in the second method, both weights and activations are log-quantized.

ShiftCNN [13] replaces each multiplication with a set of 2 or 3 shifts, therefore their PE consumes more hardware resources

and has higher power consumption than the PE with only a single shifter. Similar to ShiftCNN, in [14] and [15], to improve the accuracy, they also replace each multiplication with the sum of two shift operations.

In DeepShift [8], the author proposes two methods: DeepShift-Q and DeepShift-PS. DeepShift-Q can be regarded as the standard logarithmic quantization. In DeepShift-PS, since the power-of-two function is differentiable, they implement a further derivation of the backward pass when the weights are logarithmic-quantized, which is the main innovation in this paper. Besides, both DeepShift-Q and DeepShift-PS support training from scratch. We have tested our methods using both DeepShift -Q and -PS equations, however, generally, we got higher accuracy for the DeepShift-Q versions.

The main feature of INQ [9] is implementing incremental retraining from a pre-trained model to increase the quantization accuracy. The INQ algorithm consists of three steps: the first step is to sort the weights by absolute value and divide them into two groups according to a certain proportion; the second step is to quantize the group of weights with larger absolute value; the third step is to retrain the group of weights with relatively smaller absolute value. After iterating these three steps, the overall quantization can be completed. In INQ, the proportion of each quantization is a hyperparameter. Unfortunately, there is currently no absolute criterion for the selection of this parameter, which is often obtained through experiments.

In [16], the author also uses incremental retraining and makes it more adaptive to sparse models through their centralized quantization method.

Besides, logarithmic quantization is also applied in approximate computing. In [17], the author proposes an approximate shifter-based PE. And in [18], an approximate logarithmic data representation is proposed for CNN training. Compared with these methods, in our JLQ method, once the quantization has been performed, the computations are exact. And our JLQ method gets higher accuracy in extreme low-bit quantization cases: 2-bit and 3-bit quantization with 8-bit activations.

III. Jumping Logarithmic Quantization

Many studies indicated that weights in most mainstream non-sparse CNN models generally follow a Gaussian-like distribution [19]. That means, the majority of the CNN weights have small values, and only a few outliers have relatively large values. Based on this information, some quantization techniques [7] intentionally use fewer sampling points to quantize weights with large absolute values and use more sampling points to quantize weights with small absolute values. However, some previous studies show that weights with larger absolute values are not inessential. In fact, they are more critical in feature extraction than those with smaller values [20]. Therefore, based on these two facts, we developed the new logarithmic quantization technique JLQ. This quantization technique extends the quantization range by introducing two external parameters so that both big-value weights and small-value weights can be taken into consideration in the extreme low-bit quantization case. To achieve our goal, we introduce a jumping step parameter "s" and an initial exponent index parameter "i" (or pre-shift) in our proposed logarithmic quantization method. In JLQ, the quantization values can be represented as follows:

- **Quantization Weights** = $(\pm 2^{sx+i})$

A. Quantization Error estimation model

To theoretically determine the hyper-parameters "s" (jumping step) and "i" (initial exponent index) of the JLQ, a quantization error estimation model is proposed. In our model, weights W_i are assumed to be in the range of [-1, 1], and they follow a Gaussian distribution N(μ, δ), where μ is very close to 0. In addition, since the slope of the long tail part of the Gaussian distribution is extremely small, for simplicity, weights located in the [-1, -3δ) and (3δ, 1] intervals are regarded to have a uniform distribution instead of Gaussian distribution in our model. For most typical non-sparse CNN models, the value of δ is between 0.01 and 0.09 [19]. To facilitate the comparison of quantization errors engendered by different jumping steps and initial exponent indexes, in this error estimation model, δ is assumed to be the average of 0.01 and 0.09, which is 0.05. And the total number of CNN weights is set to $k \cdot 10^6$. Consider a quantization interval [W_{down}, W_{up}], the logarithmic quantization weight corresponding to this interval is W_{quan} Assume that the total number of weights falling in this logarithmic quantization interval is Q, and there are N ($N \to \infty$) different discrete weights in this logarithmic quantization interval. The quantization error formula selected by this estimation model is consistent with the quantization error formula mentioned in [21], which is displayed as (1):

$$E(x) = \frac{1}{2} \cdot (Quan(x) - x)^2 \tag{1}$$

When weights obey a uniform distribution, the number of each weight can be consequently regarded as Q/N, and the interval between each weight is (W_{up}-W_{down})/(N-1). Therefore, the total quantization error function F(x) can be represented as (2):

$$F(x) = \frac{1}{2} \cdot \int_{W_{down}}^{W_{up}} (Quan(x) - x)^2 dx \cdot \frac{Q}{W_{up} - W_{down}} \tag{2}$$

When weights obey a Gaussian distribution, the number of each weight can be approximately regarded as equal owing to the fact that the individual probability of each weight tends to be infinitely small as N approaches infinity. Consequently, the total quantization error can still be approximately represented by (2). The only difference is that the total number of weights within this interval, in this case, can be further determined through the definite integral over the probability density function of the normal distribution, which is shown in (3):

$$Q = k \cdot 10^6 \cdot \int_{W_{down}}^{W_{up}} \frac{1}{\delta \cdot \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \delta^2}} dx \tag{3}$$

B. Parameter Selection

Since in the proposed quantization error estimation model, both the convolutional network weights and the logarithmic quantization weights are symmetrical at about zero, the quantization error will also share such symmetry. In other words,

TABLE I
ESTIMATED QUANTIZATION ERROR

bit width, s and i	*estimation quantization error*
bit = 2, s = 2, i = 0	16279.95k
bit = 2, s = 2, i = -1	2158.14k
bit = 2, s = 2, i = -2	920.82k
bit = 2, s = 2, i = -3	**573.12k**
bit = 2, s = 2, i = -4	959.20k
bit = 2, s = 1, i = -3	809.33k
bit = 2, s = 1, i = -4	876.18k
bit = 2, s = 3, i = -2	1244.13k
bit = 2, s = 4, i = -1	1854.92k
bit = 3, s = 2, i = 0	407.23k
bit = 3, s = 2, i = -1	**244.77k**
bit = 3, s = 2, i = -2	542.38k
bit = 3, s = 2, i = -3	876.18k

in this model, the quantization error caused by positive quantization weights and negative quantization weights maintain the same. In order to obtain the theoretical optimal parameters s and i, it is necessary to calculate and compare the estimated quantization errors of different parameter combinations under (2). Summarizing these results yields Table I. It can be seen that for 2-bit logarithmic quantization, s=2 and i=-3 is the theoretically optimal parameter set; while for 3-bit logarithmic quantization, s=2, i=-1 is the theoretically optimal parameter set.

IV. BENCHMARK RESULTS

We have tested the training results on 3 datasets: CIFAR10 [10], CIFAR100 [10], and Tinyimagenet [11]. For the CIFAR10 dataset, the original DeepShift-PS and DeepShift-Q are set as the baseline, and two DeepShift modes integrated with JLQ are set as the comparison group to get a preliminary conclusion. And CIFAR100 and Tiny ImageNet datasets are used to verify the scalability of JLQ. For a fair comparison, we set all the training parameters consistent with those of DeepShift [8]. Different from the display strategy of DeepShift, which only shows the best accuracy results, the accuracy we show is the median after three experiments.

A. Weight de-zero Optimization

In original DeepShift, ternary sign operator ($\{-1, 0, +1\}$) is used. To test the weight de-zero optimization, the ternary sign operator is replaced with the binary sign operator. In the hardware, the advantage of the binary operator is that all the weights produced by the binary operator can be represented using shift operations. An example that can show the difference between implementing the ternary sign operator and the binary sign operator is as follows:

- **Quantization weights with ternary operator (3-bit)**: ($\pm 2^0$,$\pm 2^{-1}$,$\pm 2^{-2}$,0)
- **Quantization weights with binary operator (3-bit)**: ($\pm 2^0$,$\pm 2^{-1}$,$\pm 2^{-2}$,$\pm 2^{-3}$)

B. Benchmark Statement

In order to reduce the burden of memory, researchers often want a quantization method that can minimize bit-width for both, weights and activation data. In order to demonstrate the advantages of our proposed method in low-bit quantization, most of our experiments are based on 8 activation bits. And since the training of DeepShift-PS at 8 activation bits fluctuates greatly and its loss curve is difficult to converge in this case according to our experiments, we select the highest accuracy during the entire training process as the final accuracy value of a single experiment when doing experiments in relation to DeepShift-PS.

In our test strategy, we will first test the jumping logarithmic quantization with the 2-bit theoretical optimal parameter set "s=2, i=-3" compared with the DeepShift baseline. At the same time, in order to test the effect of weight de-zero optimization on the accuracy, experiments using the ternary sign operator and binary sign operator will also be performed based on the CIFAR10 dataset. After that, we will test the accuracy of jumping logarithmic quantization with other parameter sets to verify the optimal parameter set prediction provided by our error estimation model.

In our tables, "W" refers to the number of bits to represent weights, "A" refers to the number of bits to represent activation bits, "T" refers to the ternary sign operator, and "B" refers to the binary sign operator, and "Acc@N" accuracy means that the correct class gets to be in the Top-N probabilities for it to count as "correct". We also highlight the useful data in our table to make the data comparison and conclusion extraction more convenient.

TABLE II
CIFAR10 BENCHMARK FROM PRE-TRAINED, ACC@1

W	A	Sign	Base-PS	JLQ-PS	Base-Q	JLQ-Q
2	8	T	81.75%	90.04%	90.37%	**93.03%**
2	8	B	81.11%	89.89%	89.46%	**93.15%**
3	8	T	90.53%	90.07%	92.67%	**93.27%**
3	8	B	90.37%	89.81%	92.01%	**93.31%**
4	8	T	93.75%	90.18%	**93.95%**	93.29%
4	8	B	93.61%	90.06%	**93.88%**	93.34%
4	32	T	**94.06%**	90.24%	94.05%	93.46%
4	32	B	93.91%	90.17%	**93.95%**	93.37%

C. CIFAR10 Dataset

The accuracy results shown in table II are based on the results reproduced by us instead of the results from the original DeepShift paper.

We can see that in 2-bit quantization, DeepShift-PS integrated with jumping logarithmic quantization has better performance than the corresponding baseline. Meanwhile, DeepShift-Q integrating with jumping logarithmic quantization achieves higher accuracy than any baseline in 2-bit and 3-bit quantization. Another conclusion is that quantization methods using the ternary sign operator only have slight accuracy advantages compared to those using the binary sign operator. That means, using the binary sign operator that is more hardware-friendly is feasible.

We also find some drawbacks of JLQ. For 4-bit or larger bit-width quantization, it brings little to no accuracy improvement than 3-bit quantization. This can be explained by the following

TABLE III
CIFAR10 RESNET18 BASED ON OTHER PARAMETER SETTINGS OF JLQ (METHOD=JLQ-Q, A=8, SIGN=B)

Parameters	W	From pre-trained
s = 1, i = -3	2	89.56%
s = 1, i = -4	2	89.98%
s = 2, i = -1	2	87.81%
s = 2, i = -2	2	90.60%
s = 2, i = -3	2	**93.15%**
s = 1, i = -3	3	93.01%
s = 1, i = -4	3	91.90%
s = 2, i = -1	3	**93.33%**
s = 2, i = -2	3	93.10%
s = 2, i = -3	2	93.31%

two reasons. The first reason is that JLQ adjusts the step size and overdraws the accuracy improvement by expanding the quantization range in advance. Another reason is that for most non-sparse CNN models when the logarithmic weight is less than a certain small value, the mutual substitution between those adjacent small logarithmic weights will become very obvious. For example, adding or removing weight value $\pm 2^{-8}$,$\pm 2^{-9}$ or $\pm 2^{-10}$ by adjusting the quantization range will have almost no impact when there exists 2^{-7} in the original quantization case. As for the performance of other parameter settings, to fully demonstrate the advantage of extending the quantization range, we intentionally filter parameter sets such as "s=1, i=-1", "s=1, i=-2" and so on, which only achieve small range extension. And the results are shown in table III(JLQ-Q results of s = 2, i= -3 are mentioned again for the convenience of conclusion extraction).

We can see that in 2-bit quantization, the parameter set that achieves the best accuracy is "s=2, i=-3", and in 3-bit quantization, the best parameter set is s=2, i=-1. Both results are consistent with the prediction of the error estimation model.

D. CIFAR100 Dataset

As for the CIFAR100 dataset, we simplified our experiments based on the results of CIFAR10. We intentionally removed experiments in relation to Baseline PS and JLQ-PS, considering Baseline Q shows more benefits after integrating with JLQ. We conduct experiments with two network structures, ResNet18 and GoogleNet, to verify the scalability of JLQ. For the convenience of hardware implementation, we keep the weight de-zero optimization and apply it to all experiments of CIFAR100. Additionally, we add a control group named "original" that does not implement any quantization to better demonstrate the accuracy comparison. The accuracy results based on ResNet18 are shown in table IV. The accuracy results based on GoogleNet are shown in table V.

Since the CIFAR100 dataset has more types of images than the CIFAR10 dataset, it becomes more difficult to identify the CIFAR100 dataset. In this case, the advantages of JLQ are more obvious. In terms of accuracy, the conclusions in GoogleNet are consistent with those in ResNet. In the 2-bit and 3-bit quantization cases, whether the model is trained from scratch or from a pre-trained model, the accuracy is dramatically improved by implementing JLQ. However, in the 4-bit quantization case, the JLQ shows no advantages compared to the baseline as a trade-off.

TABLE IV
RESNET18 CIFAR100 BENCHMARK (SIGN=B)

Method	W	A	From scratch		From pre-trained	
			Acc@1	Acc@5	Acc@1	Acc@5
Original	32	32	74.75%	92.97%	-	-
Base-Q	2	8	28.05%	58.29%	64.97%	86.69%
JLQ-Q	2	8	**71.02%**	**90.97%**	**72.33%**	**91.43%**
Base-Q	3	8	61.79%	87.68%	66.84%	88.45%
JLQ-Q	3	8	**72.73%**	**91.40%**	**73.45%**	**92.17%**
Base-Q	4	8	**74.10%**	**92.46%**	**73.99%**	**92.60%**
JLQ-Q	4	8	72.75%	91.51%	73.33%	92.13%
Base-Q	4	32	**74.33%**	**92.56%**	**74.14%**	**92.71%**
JLQ-Q	4	32	73.15%	91.77%	73.57%	92.36%

TABLE V
GOOGLENET CIFAR100 BENCHMARK (SIGN=B)

Method	W	A	From scratch		From pre-trained	
			Acc@1	Acc@5	Acc@1	Acc@5
Original	32	32	78.17%	94.58%	-	-
Base-Q	2	8	44.97%	76.30%	62.90%	87.07%
JLQ-Q	2	8	**76.51%**	**93.82%**	**76.36%**	**93.65%**
Base-Q	3	8	67.89%	90.44%	67.73%	89.80%
JLQ-Q	3	8	**77.11%**	**94.19%**	**77.27%**	**94.06%**
Base-Q	4	8	**77.52%**	**94.37%**	**78.06%**	**94.55%**
JLQ-Q	4	8	77.15%	94.14%	77.38%	94.19%
Base-Q	4	32	**77.80%**	**94.56%**	**78.09%**	**94.47%**
JLQ-Q	4	32	77.35%	94.27%	77.63%	94.31%

E. Tiny ImageNet Dataset

In order to test whether JLQ can maintain the accuracy advantage in the case of extremely low-bit quantization (2-bit and 3-bit) on larger datasets, we conducted experiments based on the Tiny ImageNet dataset. The Tiny ImageNet dataset [11] is a modified subset of the original ImageNet dataset [22] with 200 different classes, 100,000 training examples and 10,000 validation examples. The resolution of the images is only 64x64 pixels, which makes it more challenging to extract information from it than the original ImageNet dataset. To get pre-trained models of Tiny ImageNet Dataset, we train the networks from scratch in 90 epochs based on Imagenet pre-trained weights. After that, we train the networks from those pre-trained models for 15 epochs (Other training parameters are consistent with those in ImageNet experiments mentioned in [8]). In addition, to increase the accuracy of the Resnet18 baseline, we use a fine-tuning method by removing the max pooling layer to reduce information loss of the image in the early stage of CNN. The accuracy results based on ResNet18, and Inception-v3 are shown in table VI, and VII correspondingly.

It can be clearly seen that although the dataset becomes larger, compared to the baseline, JLQ still maintains the advantage of accuracy in a very low-bit quantization case.

TABLE VI
RESNET18 TINY IMAGENET BENCHMARK (SIGN=B)

Method	W	A	From scratch	
			Acc@1	Acc@5
Original	32	32	60.30%	82.10%
Method	W	A	From pre-trained	
			Acc@1	Acc@5
Base-Q	2	8	37.40%	62.54%
JLQ-Q(s=2,i=-3)	2	8	**57.32%**	**80.02%**
Base-Q	3	8	48.60%	73.51%
JLQ-Q(s=2,i=-1)	3	8	**58.36%**	**81.03%**

TABLE VII
INCEPTION-V3 TINY IMAGENET BENCHMARK (SIGN=B)

Method	W	A	From scratch	
			Acc@1	Acc@5
Original	32	32	68.09%	86.89%
Method	W	A	From pre-trained	
			Acc@1	Acc@5
Base-Q	2	8	10.08%	28.81%
JLQ-Q(s=2,i=-3)	2	8	65.98%	85.28%
Base-Q	3	8	59.83%	81.57%
JLQ-Q(s=2,i=-1)	3	8	66.37%	85.43%

V. HARDWARE DESIGN

A. Design Of Processing Element

The processing element serves as the core of neural network acceleration. Therefore, the optimization of a PE will directly affect the efficiency of overall hardware design. A general weight-stationary shifter-based PE design is illustrated in Fig.1. The shift operation of the feature and the weight is first conducted, then followed by a negative or positive value selection and zero value selection.

Since our quantization method does not have the weight of zero value, our PE removes a multiplexer in relation to zero bit and can save one extra bit and one multiplexer compared to traditional PE.

B. Design Of Shifter

Although JLQ extends the quantization range relative to traditional logarithmic quantization, it does not mean that it requires a larger shifter to achieve the corresponding quantization. By modifying the barrel shifter, our PE achieves the purpose of range extension without adding any hardware burden. Taking the 3-bit jumping log quantization with step = 2 i = -1 as an example, the logarithmic quantization weights in this case are as follows:($\pm 2^{-1}$,$\pm 2^{-3}$,$\pm 2^{-5}$,$\pm 2^{-7}$). These weight values will correspond to the shift operations of $\gg 1$, $\gg 3$, $\gg 5$, and $\gg 7$ respectively. And the operations of $\gg 1$, $\gg 3$, $\gg 5$, $\gg 7$ only entail a 2-bit barrel shifter shown in Fig.2 instead of a traditional 3-bit barrel shifter to achieve. In this example shifter, the layer of multiplexers that operate right shift one bit is removed. To further save hardware resources, some multiplexers are substituted with AND gates or are neglected directly. The first layer of AND gates and multiplexers represents the $\gg 4$ operation, and the second layer represents the $\gg 2$ operation. Besides, the input is preshifted 1 bit in advance. Consequently, $\gg 1$, $\gg 3$, $\gg 5$, $\gg 7$ operations can be achieved using this shifter.

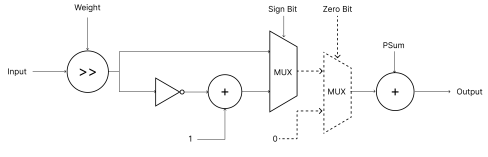


Fig. 1. Shifter-Based PE (Our PE is the solid part, and the dotted part is what we remove from traditional shifter-based PE in our optimization)

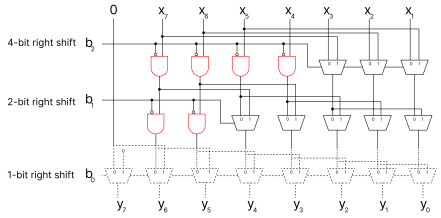


Fig. 2. Shifter Design (Our shifter is the solid part and the red part. The dotted part is what we remove from the traditional 3-bit barrel shifter in our optimization, and the red part indicates the replacement of multiplexers with AND gates. The first column of multiplexers can be directly neglected owing to preshift operation).

TABLE VIII
RESOURCES CONSUMPTION IN SINGLE PE

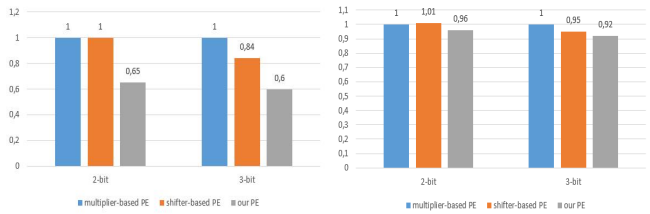
PE type	W	LUT[a]	A(μm^2)[b]	P(mw)[b]
multiplier-based PE	3	55	221.186	1.3200
shifter-based PE	3	46	209.720	1.1928
our PE	3	33	203.840	1.1879
multiplier-based PE	2	43	200.900	1.2047
shifter-based PE	2	43	202.664	1.1567
our PE	2	28	192.178	1.1516

[a]For Xilinx ARTIX7 XC7A100T FPGA
[b]For TSMC 28nm technology (ASIC)

C. Implementation Results

In order to calculate area and power consumption, different types of PE were synthesized using the Synopsys Design Compiler for TSMC 28nm technology. And resource utilization is tested on Xilinx ARTIX7 XC7A100T. The basic resulting resources, area consumption, and power consumption for different PEs are given in Table VIII. And the normalized results of resource utilization and area are shown in Fig.3.

It can be seen that our PE has advantages in hardware utilization, area, and power consumption over both traditional shifter-based PE and multiplier-based PE. Specifically, as shown in Fig.3, our PE reduce the 35% and 28.6% LUT compared with traditional shifter-based PE in 2-bit and 3-bit quantization case respectively. As for area and power consumption, although the differences are not as prominent as those in resource utilization, our PE still performs better than traditional shifter-based PE.



(a) Normalized LUT Resources (b) Normalized Area

Fig. 3. Normalized Results

VI. Conclusion

In this paper, we propose a quantization method called Jumping Logarithmic Quantization (JLQ), a weight de-zero optimization, and a cost-efficient PE design. In the experiments of CIFAR10, CIFAR100, and Tiny ImageNet, after integrating JLQ and weight de-zero optimization, the accuracy of baseline has been greatly improved in both 2-bit quantization and 3-bit quantization. Implementation results show that our PE can maximally reduce the area and power consumption up to 19.7% and 17.2% compared with traditional multiplier-based PE under similar accuracy conditions.

References

[1] C. M. e. a. Gulshan V, Peng L, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," in *JAMA-JOURNAL OF THE AMERICAN MEDICAL ASSOCIATION*, vol. 316, 2016, pp. 2402–2410. [Online]. Available: https://doi.org/10.1001/jama.2016.17216

[2] N. R. e. a. Esteva A, Kuprel B, "Dermatologist-level classification of skin cancer with deep neural networks," in *NATURE*, vol. 542. ICLR 2017, 2017, pp. 115–+. [Online]. Available: https://doi.org/10.1038/nature21056

[3] Y. L. e. a. Chen MC, Ball RL, "Deep Learning to Classify Radiology Free-Text Reports," in *RADIOLOGY*, vol. 286, 2018, pp. 845–852. [Online]. Available: https://doi.org/10.1148/radiol.2017171115

[4] I. S. A. Krizhevsky and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, L. B. F. Pereira, C. J. C. Burges and E. C. A. K. Q. Weinberger, Eds., Curran Associates. Inc., 2012, pp. 1097 – 1105. [Online]. Available: https://doi.org/10.1145/3065386

[5] Y. J. P. S. S. R. D. A. D. E. V. V. A. R. Christian Szegedy1, Wei Liu2, "Going Deeper with Convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, M. Razeghi, G. J. Brown, J. S. Lewis, and G. Leo, Eds. IEEE, 2015, pp. 1 – 9. [Online]. Available: https://doi.org/10.1109/cvpr.2015.7298594

[6] S. R. K. He, X. Zhang and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1512.03385

[7] E. Kalali and R. van Leuken, "A Power-Efficient Parameter Quantization Technique for CNN Accelerators," in *24TH EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN (DSD 2021)*, F. Leporati, S. Vitabile, and A. Skavhaug, Eds. EUROMICRO, 2021, pp. 18–23. [Online]. Available: https://doi.org/10.1109/DSD53832.2021.00012

[8] S. F. e. a. Elhoushi Mostafa, Chen Zihao, "DeepShift: Towards Multiplication-Less Neural Networks," in *CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGITION WORKSHOPS (CVPRW 2021)*. IEEE, 2021, pp. 2359–2368. [Online]. Available: https://doi.org/10.1109/CVPRW53098.2021.00268

[9] Y. G. L. X. Y. C. Aojun Zhou, Anbang Yao, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights," in *Computer Vision and Pattern Recognition*. ICLR 2017, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1702.03044

[10] A. Krizhevsky, "Learning multiple layers of features from tiny images," in *Technical report*. University of Toronto, Department of Computer Science, 2009. [Online]. Available: https://doi.org/10.48550/arXiv.1409.1556

[11] I. L. Patryk Chrabaszcz and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," 2017. [Online]. Available: https://arxiv.org/abs/1707.08819

[12] B. M. Daisuke Miyashita, Edward H. Lee, "Convolutional Neural Networks using Logarithmic Data Representation," 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1603.01025

[13] L. R. Denis A. Gudovskiy, "ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks," 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.02393

[14] Y. W. Chen Yang, Bowen Li, "A Fully Quantitative Scheme With Fine-grained Tuning Method For Lightweight CNN Acceleration," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2017. [Online]. Available: https://doi.org/10.1109/ICECS46596.2019.8964724

[15] L.-R. Z. Z. Z. Jiawei Xu, Yuxiang Huan, "A Low-Power Arithmetic Element for Multi-Base Logarithmic Computation on Deep Neural Networks," in *2018 31st IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/SOCC.2018.8618560

[16] D. B. R. M. C.-Z. X. Yiren Zhao, Xitong Gao, "Focused Quantization for Sparse CNNs," 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1903.03046

[17] C. F. B. Fong, J. Mu, and W. Z. 0012, "A cost-effective cnn accelerator design with configurable pu on fpga," in *2019 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2019, Miami, FL, USA, July 15-17, 2019*. IEEE, 2019, pp. 31–36. [Online]. Available: https://doi.org/10.1109/ISVLSI.2019.00015

[18] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," in *Neural and Evolutionary Computing*, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1603.01025

[19] R. S. Jie Li, "Jizhong dianxing juanji shenjing wangluo de quanzhong fenxi yu yanjiu[Weight Analysis and Research of Several Typical Convolutional Neural Networks]," in *JOURNAL OF QINGDAO UNIVERSITY (Natural Science Edition)*, 2019. [Online]. Available: https://doi.org/10.3969/j.issn.1006-1037.2019.11.13

[20] L. L. Yu Liu, XueJiao Liu, "Optimize FPGA-Based Neural Network Accelerator with Bit-Shift Quantization," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020. [Online]. Available: https://doi.org/10.1109/ISCAS45731.2020.9180919

[21] H. N. Jingyong Cai, Masashi Takemoto, "A Deep Look into Logarithmic Quantization of Model Parameters in Neural Networks," in *Proceedings of the 10th International Conference on Advances in Information Technology*. IEEE, 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1145/3291280.3291800

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

Bibliography

[1] N. M. e. a. Yamashita. R, "Convolutional neural networks: An overview and application in radiology," *Insights into Imaging*, vol. 9, p. 611–629, 2018. [Online]. Available: https://doi.org/10.1007/s13244-018-0639-9

[2] "Fully connected layer." [Online]. Available: https://www.fastaireference.com/tabular-data/fully-connected-layer

[3] "Data science central." [Online]. Available: https://www.datasciencecentral.com/an-elegant-way-to-represent-forward-propagation-and-back/

[4] S. R. K. He, X. Zhang and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1512.03385

[5] Y. J. e. a. Christian Szegedy1, Wei Liu2, "Going Deeper with Convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, M. Razeghi, G. J. Brown, J. S. Lewis, and G. Leo, Eds. IEEE, 2015, pp. 1 – 9. [Online]. Available: https://doi.org/10.1109/cvpr.2015.7298594

[6] S. Marigi Rajanarayana, "Sascnn: A systolic array simulator for cnn," 2019. [Online]. Available: http://resolver.tudelft.nl/uuid:5266a567-9864-4ffd-8e25-0d4d0e5f322a

[7] "Deep neural network energy estimation tool — tool for designing energy-efficient deep neural networks." [Online]. Available: https://energyestimation.mit.edu/

[8] P. C. S. e. a. Park Chan, Park Sungkyung, "Roofline-Model-Based Design Space Exploration for Dataflow Techniques of CNN Accelerators," vol. 8. IEEE, 2020, pp. 172 509–172 523. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3025550

[9] Y. G. e. a. Aojun Zhou, Anbang Yao, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights," in *Computer Vision and Pattern Recognition*. ICLR 2017, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1702.03044

[10] S. F. e. a. Elhoushi Mostafa, Chen Zihao, "DeepShift: Towards Multiplication-Less Neural Networks," in *CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGITION WORKSHOPS (CVPRW 2021)*. IEEE, 2021, pp. 2359–2368. [Online]. Available: https://doi.org/10.1109/CVPRW53098.2021.00268

[11] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1." *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1602.html

[12] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," 2016, cite arxiv:1603.05279v1.pdf. [Online]. Available: http://arxiv.org/abs/1603.05279

[13] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, "Reactnet: Towards precise binary neural network with generalized activation functions." in *ECCV (14)*, ser. Lecture Notes in Computer Science, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., vol. 12359. Springer, 2020, pp. 143–159. [Online]. Available: http://dblp.uni-trier.de/db/conf/eccv/eccv2020-14.html

[14] I. S. A. Krizhevsky and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, L. B. F. Pereira, C. J. C. Burges and E. C. A. K. Q. Weinberger, Eds., Curran Associates. Inc., 2012, pp. 1097 – 1105. [Online]. Available: https://doi.org/10.1145/3065386

[15] A. Z. Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Computer Vision and Pattern Recognition*, 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1409.1556

[16] E. Kalali and R. van Leuken, "A Power-Efficient Parameter Quantization Technique for CNN Accelerators," in *24TH EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN (DSD 2021)*, F. Leporati, S. Vitabile, and A. Skavhaug, Eds. EUROMICRO, 2021, pp. 18–23. [Online]. Available: https://doi.org/10.1109/DSD53832.2021.00012

[17] M. Bonnaerens, "A pytorch implementation of "incre- mental network quantization: Towards lossless cnns with low-precision weights"." [Online]. Available: https://github.com/Mxbonn/INQ-pytorch

[18] C. M. e. a. Gulshan V, Peng L, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," in *JAMA-JOURNAL OF THE AMERICAN MEDICAL ASSOCIATION*, vol. 316, 2016, pp. 2402–2410. [Online]. Available: https://doi.org/10.1001/jama.2016.17216

[19] N. R. e. a. Esteva A, Kuprel B, "Dermatologist-level classification of skin cancer with deep neural networks," in *NATURE*, vol. 542. ICLR 2017, 2017, pp. 115–+. [Online]. Available: https://doi.org/10.1038/nature21056

[20] Y. L. e. a. Chen MC, Ball RL, "Deep Learning to Classify Radiology Free-Text Reports," in *RADIOLOGY*, vol. 286, 2018, pp. 845–852. [Online]. Available: https://doi.org/10.1148/radiol.2017171115

[21] K. e. a. Chen, Yu-Hsin, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *RADIOLOGY*, vol. 52. IEEE, 2016, pp. 127–138. [Online]. Available: https://doi.org/10.1109/JSSC.2016.2616357

[22] B. L. Cavigelli Lukas, "Origami: A 803-GOp/s/W Convolutional Network Accelerator," vol. 27. IEEE, 2017, pp. 2461–2475. [Online]. Available: https://doi.org/10.1109/TCSVT.2016.2592330

[23] D. A. e. a. Gokhale Vinayak, Jin Jonghoon, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 27. IEEE, 2014, pp. 696–+. [Online]. Available: https://doi.org/10.1109/CVPRW.2014.106

[24] C. T. e. a. Du Zidong, Fasthuber Robert, "ShiDianNao: Shifting Vision Processing Closer to the Sensor." IEEE, 2015, pp. 92–104. [Online]. Available: https://doi.org/10.1145/2749469.2750389

[25] L. S. e. a. Liu Daofu, Chen Tianshi, "PuDianNao: A Polyvalent Machine Learning Accelerator," vol. 50. IEEE, 2015, pp. 369–381. [Online]. Available: https://doi.org/10.1145/2694344.2694358

[26] E. J. S. e. a. Chen Yu-Hsin, Krishna Tushar, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," vol. 52. IEEE, 2017, pp. 127–138. [Online]. Available: https://doi.org/10.1109/JSSC.2016.2616357

[27] L. S. e. a. Chen Yunji, Luo Tao, "DaDianNao: A Machine-Learning Supercomputer." IEEE, 2014, pp. 609–622. [Online]. Available: https://doi.org/10.1109/MICRO.2014.58

[28] S. N. e. a. Chen Tianshi, Du Zidong, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," vol. 49. IEEE, 2014, pp. 269–283. [Online]. Available: https://doi.org/10.1145/2541940.2541967

[29] B. M. Daisuke Miyashita, Edward H. Lee, "Convolutional Neural Networks using Logarithmic Data Representation," 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1603.01025

[30] L. R. Denis A. Gudovskiy, "ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks," 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.02393

[31] Y. W. Chen Yang, Bowen Li, "A Fully Quantitative Scheme With Fine-grained Tuning Method For Lightweight CNN Acceleration," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2017. [Online]. Available: https://doi.org/10.1109/ICECS46596.2019.8964724

[32] L.-R. Z. e. a. Jiawei Xu, Yuxiang Huan, "A Low-Power Arithmetic Element for Multi-Base Logarithmic Computation on Deep Neural Networks," in *2018 31st IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2018. [Online]. Available: https://doi.org/10.1109/SOCC.2018.8618560

[33] C. F. B. Fong, J. Mu, and W. Z. 0012, "A cost-effective cnn accelerator design with configurable pu on fpga," in *2019 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2019, Miami, FL, USA, July 15-17, 2019*. IEEE, 2019, pp. 31–36. [Online]. Available: https://doi.org/10.1109/ISVLSI.2019.00015

[34] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," in *Neural and Evolutionary Computing*, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1603.01025

[35] L. Fengfu and L. Bin, "Ternary weight networks." *CoRR*, vol. abs/1605.04711, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1605.html

[36] L. L. Yu Liu, XueJiao Liu, "Optimize FPGA-Based Neural Network Accelerator with Bit-Shift Quantization," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020. [Online]. Available: https://doi.org/10.1109/ISCAS45731.2020.9180919

[37] H. N. Jingyong Cai, Masashi Takemoto, "A Deep Look into Logarithmic Quantization of Model Parameters in Neural Networks," in *Proceedings of the 10th International Conference on Advances in Information Technology*. IEEE, 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1145/3291280.3291800

[38] R. S. Jie Li, "Jizhong dianxing juanji shenjing wangluo de quanzhong fenxi yu yanjiu[Weight Analysis and Research of Several Typical Convolutional Neural Networks]," in *JOURNAL OF QINGDAO UNIVERSITY (Natural Science Edition)*, 2019. [Online]. Available: https://doi.org/10.3969/j.issn.1006-1037.2019.11.13

[39] A. Krizhevsky, "Learning multiple layers of features from tiny images," in *Technical report*. University of Toronto, Department of Computer Science, 2009. [Online]. Available: https://doi.org/10.48550/arXiv.1409.1556

[40] I. L. Patryk Chrabaszcz and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," 2017. [Online]. Available: https://arxiv.org/abs/1707.08819

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015, cite arxiv:1512.00567. [Online]. Available: http://arxiv.org/abs/1512.00567