Research on Efficient CNN Acceleration Through Mixed Precision Quantization: A Comprehensive Methodology

Yizhi He[1], Wenlong Liu[2], Muhammad Tahir[3], Zhao Li[4*], Shaoshuang Zhang[5], Hussain Bux Amur[6]
School of Computer Science and Technology, Shandong University of Technology, Zibo 255049 China[1, 2, 4, 5]
Department of Computer Science, Mohammad Ali Jinnah University, Block 6, P.E.C.H.S, Karachi, 75400, Pakistan[3, 6]

***Abstract*—To overcome challenges associated with deploying Convolutional Neural Networks (CNNs) on edge computing devices with limited memory and computing resources, we propose a mixed-precision CNN calculation method on a Field Programmable Gate Array (FPGA). This approach involves a collaborative design encompassing both software and hardware aspects. Initially, we devised a CNN quantization method tailored for the fixed-point operation characteristics of FPGA, addressing the computational challenges posed by floating-point parameters. We introduce a bit-width strategy search algorithm that assigns bit-widths to each layer based on CNN loss variation induced by quantization. Through retraining, this strategy mitigates the degradation in CNN inference accuracy. For FPGA acceleration design, we employ a flow processing architecture with multiple Processing Elements (PEs) to support mixed-precision CNNs. Our approach incorporates a folding design method to implement shared PEs between layers, significantly reducing FPGA resource usage. Furthermore, we designed a data reading method, incorporating a register set buffer between memory and processing elements to alleviate issues related to mismatched data reading and computing speeds. Our implementation of the mixed-precision ResNet20 model on the Kintex-7 Eco R2 development board achieves an inference accuracy of 91.68% and a computing speed 4.27 times faster than the Central Processing Unit (CPU) on the CIFAR-10 dataset, with an accuracy drop of only 1.21%. Compared to a unified 16-bit FPGA accelerator design method, our proposed approach demonstrates an 89-fold increase in computing speed while maintaining similar accuracy.**

***Keywords*—Convolutional Neural Networks (CNNs); edge computing technologies; Field Programmable Gate Array (FPGA) accelerator; mixed precision quantization; loss variation**

I. INTRODUCTION

Nowadays, deep learning has brought new development opportunities for Internet of Things (IoT). Among them, CNNs are widely used in many areas such as face recognition, autonomous driving, and unmanned air vehicles for their outstanding performance [1]. CNN consists of two stages, which are training and inference. Usually, training is a one-time off-line process, which is based on computing platform with large computing resources and high power consumption. Inference can be deployed on edge computing devices, in order to obtain shorter processing delay and avoid the impact of communication situation [2].

However, with the development of deep learning, the size of CNNs continues to increase to obtain stronger learning ability, resulting in larger network computations, more parameters, and more complex network structures. At the same time, many edge computing platforms have limited storage and computing resources, restriction on power consumption and latency. Therefore, the use of CNN inference on edge computing devices has become an important challenge in the field of Artificial Intelligence (AI) research.

Currently, many FPGA acceleration methods [3] for CNNs have been proposed. With FPGA's parallelism and flexible configuration, it can be deeply customized for the CNN structure through parallel computing methods [7] to provide accelerated services for deep learning, achieving higher performance and power efficiency.

Chen et al. [10] proposed the DianNao CNN accelerator, which adopted a three-level pipeline architecture consisting of multiplication, addition, and sigmoid functions. By reusing the weights stored in the on-chip memory, it reduced the need for accessing off-chip data, thereby lowering memory access power consumption. The CNN accelerator Eyeriss [11] was proposed, which employed a row stationary dataflow to maximize data reuse in the computation array and minimize memory access. The authors of this article reference [12] proposed the energy-efficient and reconfigurable hybrid neural network processor Thinker. Each computing unit in Thinker supports adaptive computation for different data bit-widths required by neural networks. The maximum operating frequency is 200MHz, and the supported data bit-widths are 8-bit and 16-bit.

Additionally, quantization [13] can be used to reduce model sizes and hardware resource consumption, such as replacing original 32-bit floating-point operations with lower precision fixed-point numbers like 8-bit or 16-bit. Jacob et al. [15] proposed an integer quantization method, which uniformly quantizes both weight and activation to 8-bit. Furthermore, there are ultra low-bit quantization methods, such as ternary quantization [16] which quantifies weights into {-*w*, 0, +*w*}, and even binary quantization neural networks [17], which quantize weights and activation values to 1 or -1.

However, using a unified quantization bit-width in ultra low-bit-width situations would significantly affect CNN performance. A highly effective solution to this problem is through mixed precision quantization [20]. It allows each layer

of the CNN model to have different quantization bit-widths. which can greatly preserve the performance. Lin et al. [21] proposed an analytical solution to address the fixed-point quantization problem. It seeks an optimal bit-width allocation strategy across network layers by optimizing the Signal-to-Quantization-Noise Ratio (SQNR). Wang et al. [22] designed a Hardware Aware Quantization (HAQ) algorithm that incorporates inference speed information evaluated by a hardware simulator into the training process, which utilized reinforcement learning to automatically determine quantization strategies. It reduces latency by 1.4-1.95 times and energy consumption by 1.9 times. However, the current methods face challenges in their applicability to FPGA platforms or in terms of high time and space complexity when searching for mixed precision strategies.

In conclusion, if we have an efficient mixed precision search algorithm and can apply the strategies obtained by this algorithm to FPGA platforms; it will be greatly significant for the application of deep learning on AIoT devices. Therefore, we propose a method for implementing a mixed precision CNN model on FPGA, co-designing from software and hardware aspects. The main innovation points are as follow:

1) A quantization method is proposed that is more suitable for FPGA's fixed-point operation characteristics. Combined with a mixed precision strategy search algorithm with the optimization objective of the lowest bit-width for each layer and the constraint of the accuracy of the CNNs achieve mixed precision calculation of CNNs.

2) In terms of FPGA accelerator design, an inter-layer storage and multi-PEs reuse mechanism is proposed based on the streaming processing architecture. And performing folding design between different CNN layers not only retains the flexibility of the streaming processing architecture but also saves hardware resources and improves the computing efficiency of each PE.

3) A new data reading method is designed using a high bit-width data transmission mode to read multiple data in a single cycle, efficiently utilizing bandwidth to transfer data. In addition, a register set is added as a buffer between the Block Random Access Memory (BRAM) and the PE to solve the problem of mismatch between reading and computing speed.

Based on the above, the purpose of this paper is to explore an efficient method for searching mixed precision quantization strategies and implementing mixed precision computation of CNN on the FPGA platform. The goal is to achieve high performance CNN computations within limited computational and storage resources.

This paper is structured as follows: Section II elucidates the mixed precision quantization method and outlines the strategy for bit-width exploration in the context of CNNs. In Section III, the FPGA platform accelerator is detailed, encompassing the overall architecture, parallel processing elements (PE), and an efficient data transfer mechanism. Section IV delves into the experimental results of quantization and assesses the performance of the mixed precision accelerator. Finally, Section V provides a comprehensive conclusion and future work for the paper.

II. Mixed Precision Quantization for CNNs

A. Design of Quantization Method

Although many existing quantization methods can reduce the parameter storage of CNNs through encoding and decoding, the quantized parameters are still computed using floating-point numbers [23]. When implemented to the FPGA platform, the fixed-point number used differs in precision from the original floating-point number, which leads to calculation error and causes drop of inference accuracy. Therefore, a quantization method suitable for FPGA has been designed to convert the parameters in the CNN model into fixed-point numbers, and retrain the quantized model to reduce the degradation of its accuracy. The quantization process consists of two steps: first, the floating-point number is transformed into a fixed-point number, which is left shifting, amplified, rounded, and truncated to a *n*-bit fixed-point integer. Then, the fixed-point decimal value is restored to an approximately original value through right shifting.

The process of converting a 32-bit floating-point number X to a n-bit integer X_{int} is shown as follows:

$$X_{int} = clamp(round(2^{n-l-1} \cdot X), Q_{min}, Q_{max}) \quad (1)$$

Where l is the number of bits in the integer part of X, $Q_{min}=-2^{n-1}$, $Q_{max}=2^{n-1}-1$, $round()$ is the rounding function, and $clamp()$ is defined as follows:

$$clamp(x,a,b) = \begin{cases} a, & x < a \\ x, & a \le x \le b \\ b, & x > b \end{cases} \quad (2)$$

For example, the process of quantizing a decimal with a floating-point number of 1.253 to an 8-bit fixed-point number (assuming 1 sign bit, 3 bits of integer width, and 4 bits of decimal width) can be shown in Fig. 1.

The floating-point number 1.253 is shifted left by 4 bits (multiplied by 2^4) and then amplified to obtain 20.048. Then, it is rounded and clamped to obtain an integer value of 20. Finally, the quantized integer value is shifted right by 4 bits (divided by 2^4) to obtain the fixed-point value of 1.25. Hence, it is possible to replace 32-bit floating-point numbers with 8-bit fixed-point numbers in order to make them more easily deployable in FPGA.

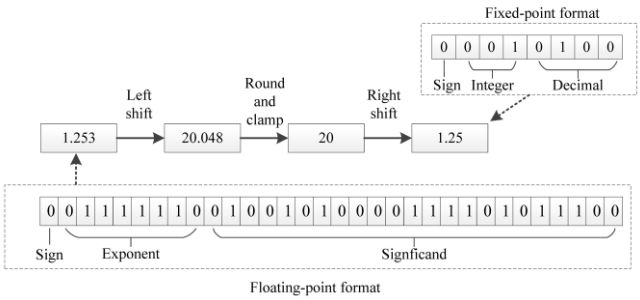

Fig. 1. Quantization Process Diagram.

Suppose the activation value A and weight parameter W in the CNNs are quantized to A_{int} and W_{int} respectively, then the convolutional process can be transformed into,

$$Y = A*W + b = A_{int} * W_{int} \cdot 2^{l_A + l_W + 1 - 2n} + b \tag{3}$$

where b represents bia, l_A and l_W represent the integer bit width of activation value A and weight parameter W, respectively.

Through (3), the floating-point number in the convolution process can be quantized into n-bit integer. After completing the convolution calculation, right shifting can restore the approximate value to the original result. Using shifting and integer operations instead of floating-point arithmetic can reduce the computational resource requirements for floating-point operations, which is easy to implement in FPGA. Moreover, the quantized fixed-point CNN can be retrained, greatly reducing the accuracy drop caused by directly implementing the CNN model to the FPGA platform.

B. Quantization Bit-Width Strategy Search Algorithm

Allocating appropriate quantization bit-widths for each layer in a CNN model is an important challenge in implementing mixed precision operations. Assuming there are 20 convolutional layers in a CNN model, each convolutional layer can be assigned a bit-width value ranging from 1 to 32. There are 32^{20} quantization schemes, and several schemes produce different inference accuracy and occupy different hardware resources. Therefore, selecting a suitable quantization scheme that has high accuracy and is also easy to deploy on FPGA requires an efficient strategy search algorithm. A novel CNN mixed precision quantization method is proposed by using quantization loss variation [25] to adjust the bit-widths of each layer, which can avoid this exponential search space.

1) Calculation of quantization loss variation: The quantization loss variation is an important indicator of bit-width allocation. We found that the loss variation is related to both the first and second derivatives (Hessian matrix) information of each quantization layer and the quantization error. It can be expressed by as shown as follows using Taylor expansion:

$$\Delta L = L(W_Q) - L(W) \approx g(W)^T \Delta W + \frac{1}{2} \Delta W^T H(W) \Delta W \tag{4}$$

$g()$ represents the first derivative of the weight parameter W in the full-precision CNNs, $H()$ represents the second derivative, $L()$ represents the cross-entropy loss function commonly used in CNNs, and W_Q represents the weight parameter of the quantized CNNs. When studying the loss variation brought by second-order information, it is very difficult to directly calculate the relevant values of the Hessian matrix due to the large amount of weight parameters in CNNs. Therefore, the power iteration method can be used to approximate the maximum eigenvalue of the Hessian matrix. For each quantization layer, perturbations can be added in the direction of the corresponding eigenvector of the Hessian matrix as the quantization error [25], and then calculates the corresponding loss variation. Thus, we add perturbations in both gradient direction and Hessian matrix eigenvector direction separately in each quantization layer as the quantization error ΔW, as shown in (5).

$$\Delta W = \begin{cases} \lambda \times g(W) \\ \lambda \times H_i(W) \cdot V_i \end{cases} \tag{5}$$

The λ can be used to adjust the size of perturbation. $H_i(W) \cdot V_i$ represents the eigenvector corresponding to the maximum eigenvalue of the Hessian matrix for the *i-th* quantization layer, which can be calculated by (6) and the power iteration method.

$$\frac{\partial (g_i^T V_i)}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} V_i + g_i^T \frac{\partial V_i}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} V_i = H_i V_i \tag{6}$$

V_i is a random vector with the same dimension as the *i-th* quantization layer. The quantization error ΔW in the gradient direction is applied to the selected quantization layer and calculate the perturbed CNN loss variation ΔL_1. Then, the quantization error in the eigenvector direction of the Hessian matrix is applied to this quantization layer and the new loss variation is recalculated as ΔL_2. Finally, the maximum value between ΔL_1 and ΔL_2 is taken as the loss variation caused by this quantization layer.

2) Search method design: To reduce the search space, adjacent quantization layers in the CNNs with the same structural characteristics (such as kernel size, number of channels, and padding method) are merged into quantization blocks. The method described in last section is used to calculate the loss variation of different quantization blocks. High bit-widths are assigned to the quantization blocks that cause large loss variations, while low bit-widths are used for those that cause small loss variations. Then, the model is retrained according to the bit-width allocation strategy. The feedback results from the training are fed back to the policy search module, and the output policy is adjusted based on the feedback until the best bit-width allocation method is found. The specific process is shown in below Algorithm 1.

Algorithm 1: Quantization Bit-Width Policy Search

```
Input：Pre_Model, Dataset, Loss, Blocks, Acc_Set
Output：Bit-Width Policy
1：Blocks = Sort (Blocks, Loss)
2：Bit_Width = [16, 15, 14, … 4, 3, 2]
3：Index = 0
4：While (Index < length (Bit_width)) do
5：    New_policy = upgrade_policy (Bit_width [Index], Blocks)
6：    Model = Quantize (Pre_Model, New_policy)
7：    for i in range(epoch) do
8：        Acc = Train (Model, Dataset)
9：    if (Acc ≥ Acc_Set) then:
10：       Index = Index + 1
11：   else:
12：       Restore (Blocks)
13：       Choice_next_block (Blocks)
14: return New_policy
```

In this algorithm, a full precision pretrained model is used as the Pre_Model, and the corresponding Loss variation is used as input. The layer in pretrained model is merged into Blocks. Dataset is used as model training. Then, based on step 1, the quantization blocks are sorted in descending order according to the Loss values they produce. For the quantization block with a larger Loss value, it is quantized first, and the assigned bit-width is not less than that of the quantization block with a smaller Loss value. The quantization bit-width is set from 2 to 16 bits according to step 2, and the Index set by step 3 is used to select the bit-width value. Initially, every quantization block selects 16 bits, as shown in step 5. Next, a new CNN model is quantized based on the strategy generated by step 5, and the model is iteratively trained for epoch times. The highest training result is recorded as Acc, as shown in steps 7 and 8. Then, the highest training result is compared with the required accuracy Acc_Set in step 9 to make a decision. If the requirement is met, the Index value is increased according to step 10, and an attempt is made to reduce the bit-width of the quantization block. If the requirement is not met, the current qutization is restored the last policy, then the next quantization block is selected to adjust the bit-width value according to steps 12, 13. By following the above steps, the appropriate bit-width is selected for each quantization block in sequence.

III. Mixed Precision Accelerator Design

The mixed precision quantization of CNN achieves a balance between accuracy and compression rate by selecting appropriate bit-widths for the weight and activation parameters in each layer [26]. Therefore, in the accelerator, a design method is adopted that uses multiple types of Conv modules to support different bit-width precision operations. This can avoid inefficiency of high-bit-width arithmetic units used by low-bit-width operations. The structure is shown in Fig. 2.

In advance, the calculation method of each layer in CNN model is written into the finite state machine (FSM) in the form of instruction according to the sequence. The input image can be stored in an off-chip memory, and during computation, burst transmission via the AXI4 bus can be used to read the input image from the off-chip memory to the Conv module [27]. The entire computing process only needs to read the input image from off-chip memory once to reduce the high latency and energy consumption caused by off-chip reading and writing operations.

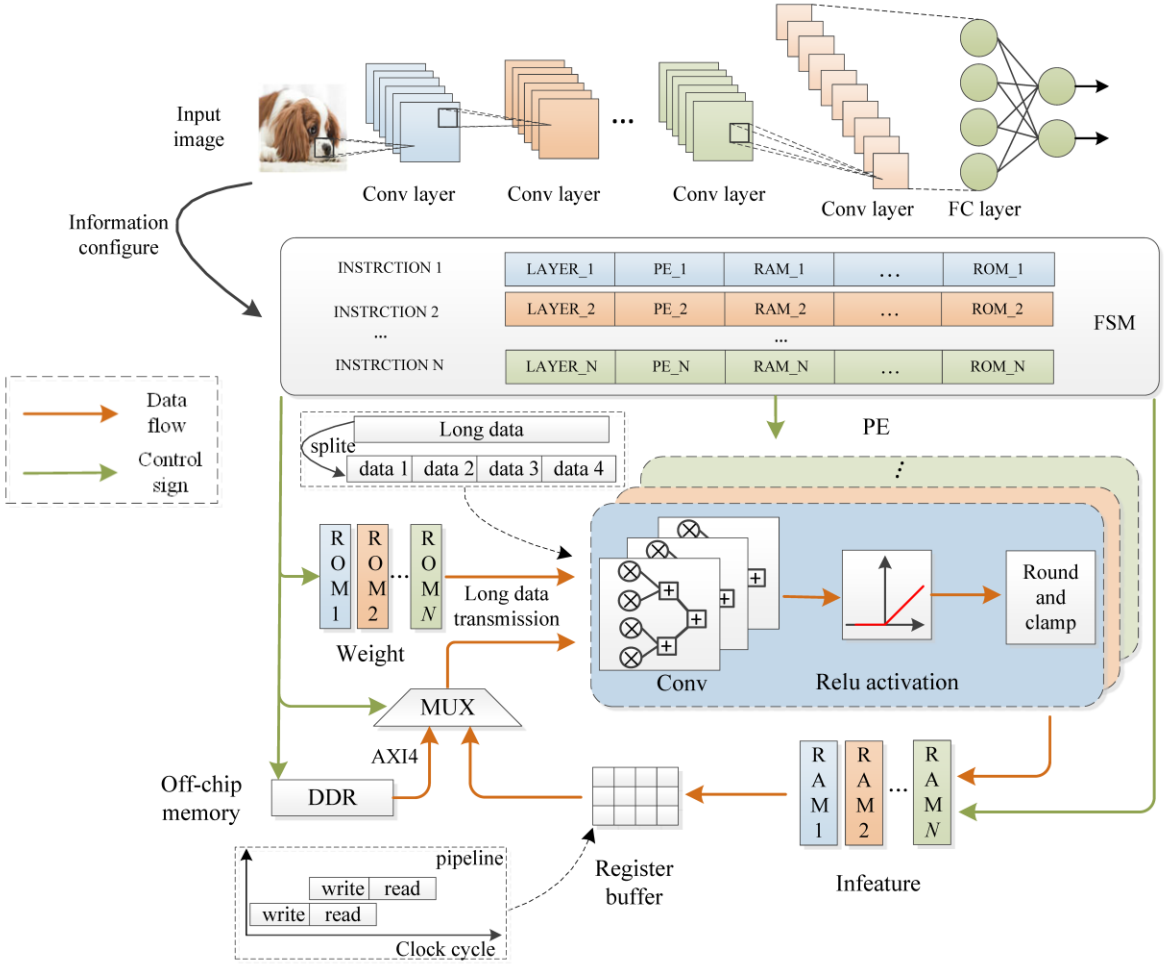

Fig. 2. The mixed precision accelerator design structure.

Then, the FSM sends instruction to control the reading of weight parameters from the Weight Read-Only Memory (ROM) and sends them to the designated PE for convolution, activation, quantization (Round and Clamp). Then the calculation results are cached in the RAM specified by the instruction, and execute subsequent processing according to the next instruction provided by FSM.

For the intermediate buffer data reading and writing operations, their throughput can easily become a bottleneck for the entire accelerator performance due to the limitations of the BRAM reading and writing interfaces. This can cause a problem of mismatch between data calculation speed and reading speed. Therefore, a register set is introduced as a data buffer. The true dual-port RAM and pipeline are simultaneously used, which greatly increase the speed of reading data into PE.

In term of weight parameter storage, the quantized weight parameters of each layer in the CNN are stored in the ROM array (Weight ROM) built of on-chip RAM resources according to the address order. During circuit initialization, the parameters are stored in the on-chip memory in a ROM initialization method. The read width is set to the sum of all weight parameter bit-widths in a single convolution kernel. Data slicing allows for extracting specific bits from a signal by using indices. So the entire long bit-width data can be split and sent sequentially to the Conv module after reading the data. In ROM IP core provided by Xilinx, the maximum read width can reach 4608 bits, which is sufficient to read the weight parameters in a convolutional kernel within one cycle.

The overall processing adopts a streaming processing architecture, and the CNN is folded according to the mixed precision quantization results in order to share PEs and memory resources among different layers. This not only alleviates the problem of large resource consumption of the streaming processing architecture, but also allows the saved resources to be used in each Conv module to improve computation parallelism. Meanwhile, the entire network can still be designed with pipeline processing. Each computing module can serve as a stage pipeline, forming a large-scale pipeline design to reduce computation latency under the condition of multiple input data.

A. PE Design

The PE structure is shown in Fig. 3. It is composed of the Conv module, Relu activation, Round and Clamp module. In the same PE, all multipliers are the same type. For example, the multipliers in Fig. 3 are all $M\times N$ type, where M corresponds to the weight bit-width and N corresponds to the feature data bit-width. The number of multipliers in each group is set according to the convolution kernel size. Assuming the convolution kernel size is $K\times K$, then K^2 multipliers are set as one group. The number of groups S is set to the least common multiple of input channels of all quantization layers using this PE. This can fully utilize all multipliers and improve computation efficiency when calling this PE.

In each convolution module, there are multiple layers of adder trees and a multiplexer. The number of adder operations can be controlled based on the input channel number of the current layer to adapt to different computation modes with different input channel numbers. For example, if the convolution channel S is set to 64 in the PE and the current network layer has 32 input channels, then two output pixels can be obtained through a binary adder tree of depth 5. When computing a network layer with 64 input channels, one output pixel needs to be obtained through a binary adder tree of depth 6. Therefore, we can reuse the PE for different layers by setting a multiplexer to output results at different layers of the adder tree.

In CNN models, to accelerate the convergence speed, prevent problems such as gradient explosion, gradient vanishing, and overfitting, many networks have Batch Normalization (BN) layers [28]. The process of calculating the BN layer with convolutional result Y through (7), can be represented as:

$$Y_{bn}=\gamma\cdot\left(\frac{Y-\mu}{\sqrt{\sigma^2+\varepsilon}}\right)+\beta=\gamma\cdot\left(\frac{W*A+b-\mu}{\sqrt{\sigma^2+\varepsilon}}\right)+\beta \tag{7}$$

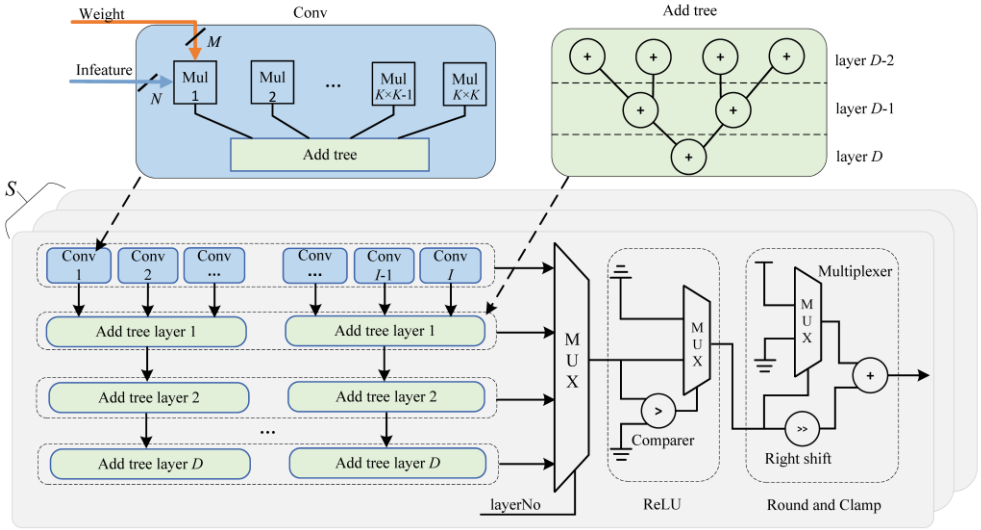

Fig. 3. Structure diagram of the PE unit.

Where μ and σ represent the mean and standard deviation within a batch, γ represents the scaling parameter, β represents the offset parameter, and ε is a very small constant set to 0.001. However, the additional computation brought by this layer makes it more difficult to implement the CNNs on FPGA platforms. Thus, the convolution, batch normalization and quantization operation are integrated together to solve this problem. Firstly, the process of integrating convolution and BN can be represented as follows:

$$Y = W' * A + b' \tag{8}$$

$$W' = \frac{\gamma W}{\sqrt{\sigma^2 + \varepsilon}} \quad b' = \frac{\gamma(b - \mu)}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

ωηερε ,

Furthermore, the quantization operation (3) can be merged into (8) as follows:

$$Y = W'_{int} * A_{int} \cdot 2^{l_A + l_W - 2n+1} + b' \tag{9}$$

The low bit-width fixed-point weight W_{int} can be obtained directly by using the optimal parameters for the weight W' and quantization bit-width n and scale factor l_A. All of them can be obtained from algorithm 1. By integrating the three operations, many complex calculations can be completed during quantization training, the trained parameters W'_{int} can be stored directly in the on-chip ROM of FPGA during circuit initialization. So, this significantly reduces lots of calculations on the FPGA.

In addition, an activation layer is often used after the convolution operation to increase the non-linear ability of the CNN. To simplify the design, we also integer the activation layer into PE. The logic structure of the commonly used ReLU activation function is shown in the Fig. 3, which uses a comparer and multiplexer. The comparer compares the input value with zero, and the result controls the multiplexer to output either the input value or 0 as the activation value.

The convolution operation with M bits weights and N bits input feature values will result in $M+N$ bits of convolution results. However, in the next layer of the network, the bit-width for the computation has already been specified by the quantization strategy, and the convolution result needs to be quantized to the specified bit-width for the next layer. Therefore, we perform Round and Clamp operations on the activated convolution result to truncate the length of the data to the specified bit-width as the input feature value for the next layer. In this way, the intermediate calculation results can also be stored in low bit-width BRAM, which saves the storage resources. The Round and Clamp operations consist of a multiplexer and an adder, which determine whether the first bit after the reserved bits of the data is equal to 1. If it is 1, the rounding operation will add 1 to the reserved data result. Otherwise, it will keep the original value.

With the above design, calculating one pixel in the output feature map requires multiplication unit, adder tree, activation, and quantization process. Considering the delay caused by these processes, it is difficult to output one result value in each cycle. Therefore, we perform pipeline to improve computing efficiency, and the specific process is shown in Fig. 4.

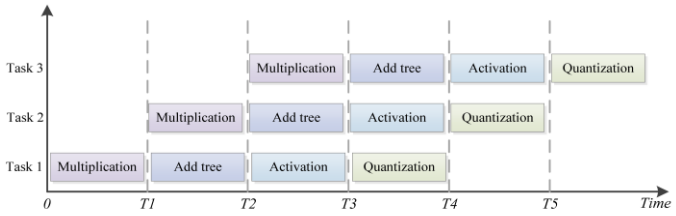

Fig. 4. The PE pipeline.

We assume that each process requires one clock cycle Δt, and calculating one output pixel requires m processes. According to the above settings, if no pipeline is applied, the number of cycles required to calculate all the pixels in the output feature map can be expressed as:

$$T_i = mH_{out}W_{out}\Delta t \tag{10}$$

W_{out}, H_{out} represent the width and height of the output feature map, which can be calculated based on the convolution process.

$$\begin{cases} H_{out} = \dfrac{H_i - K + 1}{S} \\ W_{out} = \dfrac{W_i - K + 1}{S} \end{cases} \tag{11}$$

H_i, and W_i are the height and width of the input feature map, K is the size of the convolution kernel, and S represents the convolution stride. With the pipeline, every clock cycle can generate one output pixel value except for the first output pixel point calculation process. The total number of calculation cycles can be reduced to the result expressed in (12). By comparing (10) and (12) , the computing efficiency has got improved significantly.

$$T_i = (H_{out} \cdot W_{out} - 1)\Delta t + m\Delta t \tag{12}$$

B. Folding Design

In one layer of the CNNs, there are many identical structures of channels, and even in many CNNs, there are many layers with the same structure and convolution method. When there are many channels in each layer, full parallel design according to the layer order in FPGA will inevitably use a large amount of calculation and storage resources, which may even exceed the existing resources of the FPGA [29]. Moreover, in CNNs, data needs to be passed in layer order, and it is difficult to parallelize between layers. Therefore, based on the above two calculation characteristics, we fold the original data flow to reduce the number of PE, and map it to hardware using the method of PE reuse. So different network layers can share PE while still keep the original computing efficiency. The process is shown in Fig. 5.

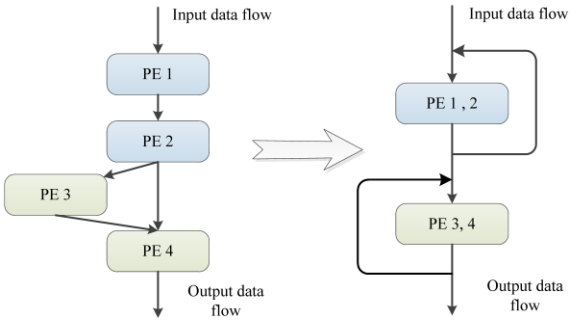

Fig. 5. Streaming folding diagram.

When different layers share the same PE, a control module needs to be introduced because there are differences in weight parameters, convolution strides, and other information between each layer of the CNN. The control module can dynamically switch the computing mode according to the requirements of different layers, such as adjusting the address for loading pre-trained weights, the source of input feature, and the address for storing intermediate calculation results. So, we need to pre-analyze the structural parameters of the selected CNN, compile the layer number, the storage address of the pre-trained weight, the convolution stride, the BRAM position where the intermediate cached result is stored, and other information of each layer into one instruction data and write it into FSM. After FSM outputs instructions, they are divided into segments. The calculation and data flow in FPGA can be uniformly allocated through various instruction segments. The length L of each segment can be macro-defined according according to (13).

$$L_i = \lceil log_2 N_i \rceil \tag{13}$$

N_i represents the number of species included in the *i-th* segment. If it is assumed to perform calculations on the first channel (64 channels in total) of the first layer (20 layers in total) of the network, the format of this instruction provided by FSM is shown in Fig. 6.

C. Optimization of Data Reading Methods

In the design of mainstream accelerators [31], to avoid high latency caused by accessing off-chip memory, weight parameters and intermediate calculation results are usually stored into on-chip BRAM memory in sequential order directly. However, when reading data, only one or two data can be read per clock according to the corresponding address. After quantization to low-bit-width fixed-point numbers, if the data is still transmitted according to the previous method, it will result in underutilization of bandwidth resources and a mismatch between data reading speed and computation speed. Therefore, a new data reading method is proposed to reorganize low-bit-width data by concatenating multiple low-bit-width data into one long word. At the same time, true dual-port RAM and register set are used in combination with data reuse to improve reading speed.

This method for reading data through 3×3 convolution is explained as follows: When reading the data, it is necessary to first set up a two-dimensional register set as a buffer between RAM and Conv module, as shown in Fig. 7.

LAYER_ID	CHANNEL_ID	PE_ID	WEIGHT_ID	RESULT_ID	STRIDE
5'b00001	6'b000001	2'b01	6'b000001	6'b000001	1'b0

Fig. 6. The instruction format.

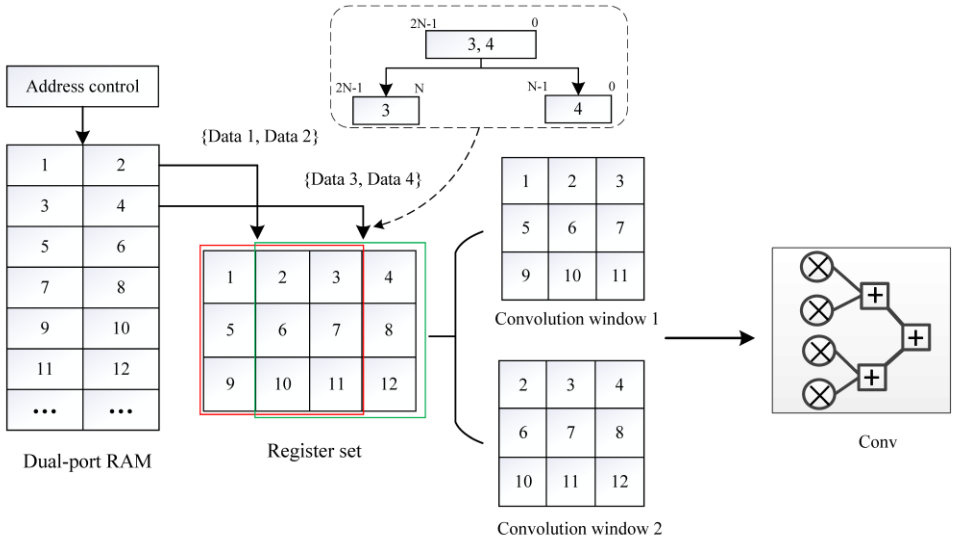

Fig. 7. Buffer structure diagram

Double-port RAM is a type of memory component that has two independent data ports. The write data width of a double-port RAM is set to twice the read data width, *2N* bits. When reading the data, two addresses are set as adjacent numbers, so that four adjacent data can be read and stored into the first row of the two-dimensional register set within one clock cycle. Then, in the next clock cycle, four adjacent data are read according to the address and stored into the second row of the register set. After the register set is full, the data from columns 1 to 3 can be directly read to achieve 3×3 full parallel convolutional calculation. Then, the data from columns 2 to 4 can be read to complete the second convolution, while reusing the second and third columns of data, thus reducing the overall data access. Additionally, when FPGA resources are sufficient, a larger register set and higher reading data width can be set, which not only can realize a higher data reuse ratio, but also can achieve a faster pipeline operation.

IV. Experiments and Result

To verify the effectiveness of the proposed method in this paper, the Resnet20 model was selected for validation on the CIFAR-10 dataset. The deep learning framework PyTorch was used to complete the quantization training experiment and mixed precision strategy search process on a server equipped with an NVIDIA Tesla T4 GPU. The quantized Resnet20 model was designed using the Verilog language on a Kintex 7 Eco R2 development board, and data reports were generated through synthesis and implementation with Vivado 2019.2 to analyze resource utilization and power consumption.

A. Resnet20 Quantization Experiment

The full precision Resnet20 model is iteratively trained on the CIFAR-10 dataset for 300 times. The accuracy of the training result is 92.89%, which was used as the pretrained model for quantization. We set the inference accuracy of the quantized model to not be less than 91.5%. The main network structure of Resnet20 is shown in Fig. 8, which has three types of residual blocks, with the convolution kernel size, channel number, and padding mode being the same in each type. According to this structural feature, we divided the network into five major structural blocks. The first layer is the first structural block, layers 2-7 are the second structural block, layers 8-14 form the third structural blocks, layers 15-21 form the fourth structural block, and the last fully connected layer is the fifth structural block.

Based on the above settings, we search for the quantization strategy in Algorithm 1, the final quantization bit-width results of the Resnet20 are shown in Fig. 9.

The activation and weight bit-widths of the first and fifth structural blocks are both quantized to 8 bits, the weight bit-widths of the second and third structural blocks (layers 2-14) are quantized to 7 bits, and the weight bit-widths in the fourth structural block (layers 15-21) are quantized to 8 bits, with the activation bit-width being 7 bits.

After mixed precision quantization and retraining of Resnet20, the inference accuracy is 91.68%, with only a 1.21% loss compared to the full precision Resnet20.

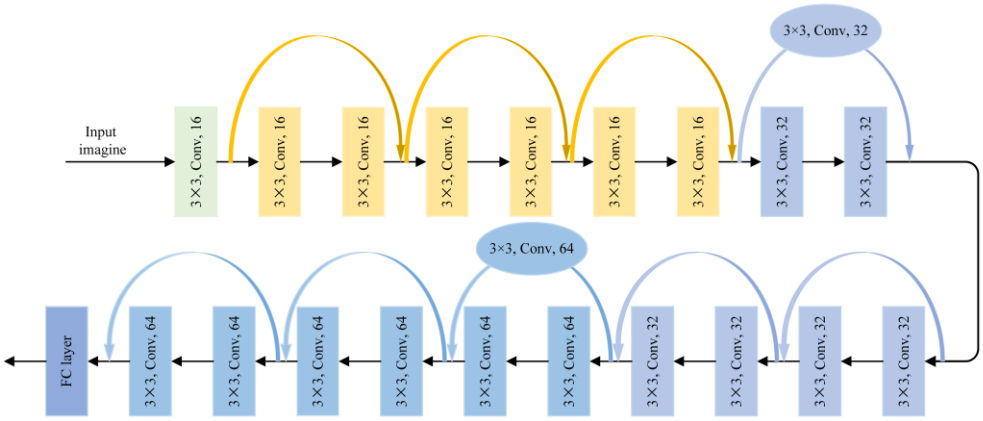

Fig. 8. Partial structure diagram of Resnet20.

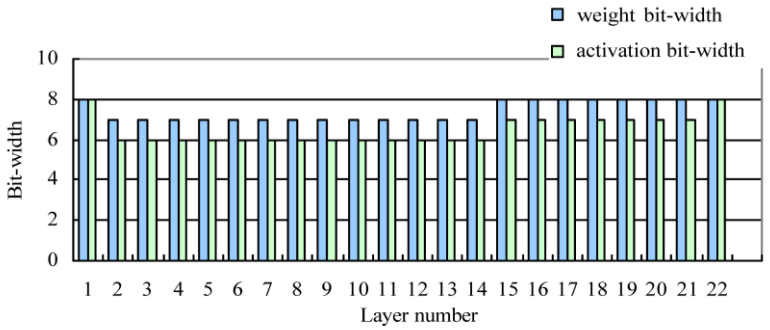

Fig. 9. Quantization bit width strategy for Resnet20.

Table I shows the comparative results of related work. In [33], the method of uniformly quantizing to 8 bits was used, and the accuracy was 90.7%. But in our paper, by reasonably allocating different bit-widths for each quantization layer, the accuracy is still 0.98% higher than their method even with lower bit-widths. In [34], after quantizing the full precision model to 8 bits and directly transplanting it to the FPGA platform, the calculation error caused by replacing the original floating-point numbers with fixed-point numbers resulted in an accuracy loss of 6.92%. By comparing this method, it can be seen that our paper greatly reduces the loss of CNN accuracy by performing quantization training and selecting a more suitable quantization method for the FPGA, making the quantized CNN closer to the full precision one.

B. Accelerator Design Experiment

1) Analysis of FPGA resource usage: Firstly, we tested the resource requirements for different bit-width multipliers in this experiment by using the synthesis tool in Vivado 2019.2, as shown in Table II. It can be seen that it takes 2 Digital Signal Processors (DSPs), 128 Look UpTables (LUTs) and 299 Flip Flops (FFs) to complete 32-bit floating-point multiplication. If DSP is not used, it would require 606 LUTs and 805 FFs to construct a 32-bit floating-point multiplier. By quantizing the multiplication operation to no higher than 8 bits, the resource usage can be reduced significantly. For example, the 7bits×bits multiplier only needs 45 LUTs and 13 FFs. The main chip xc7k325tffg676-2 in the development board used in this experiment has only 840 DSPs, but it has 203800 LUTs. Despite the limited number of DSPs available, additional multipliers are still required to achieve high parallel computations. So both DSPs and LUTs are needed to be used in combination to perform convolution operations in this paper.

Table III displays the Multiply-Accumulate operations (MACs) and resource usage and for each PE. According to the quantization results of Section IV.A in the first block, calculations are performed using 8bits8bits multipliers, therefore, PE 1 is equipped with 432 DSPs of the 8bits×8bits type, allowing for concurrent processing of 48 sets of 3×3 convolutions. The 7bits×6bits multipliers can be used for the calculations in both the 2nd and 3rd structure blocks according to the quantization results. So, these two structure blocks (the 2nd to 14th quantization layers) are designed to be folded and share a single PE (PE 2) with 1152 multipliers, which can compute up to 128 parallel convolutions of 3×3. The PE 3 used in the fourth structure block can use 8bits×7bits multipliers, so the 15th to 21th quantization layers are folded and also designed with 1152 multipliers. In PE 2 and PE 3, all the multipliers are implemented using LUTs to compensate for the limited on-chip DSP resources. In the 22nd layer, which is a fully connected layer, the computation process is carried out in PE 4. With its structure consisting of 64 inputs and 10 outputs, it is designed with 10 DSPs capable of simultaneously performing 10 parallel Multiply-Accumulate operations.

TABLE I. QUANTIZATION EXPERIMENT RESULTS OF RESNET20.

Quantization Method	Weight Bit-Widths	Weight Integer Bit-Widths	Activation Bit-Widths	Activation Integer Bit-Widths	Accuracy (%)
Baseline	32	/	32	/	92.89
[33]	8	/	8	/	90.7
[34]	8	/	8	/	84.81 (91.73)
Ours	7/8	2	6/7/8	4	91.68

TABLE II. COMPARISON OF RESOURCE USAGE FOR DIFFERENT MULTIPLIERS

Multiplier Type (*m*-bit × *n*-bit)	LUTs	FFs	DSPs	Power (W)
32×32 (float)	128	299	2	0.184
32×32 (float)	606	805	0	0.191
16×16 (fixed)	280	32	0	0.192
8×8 (fixed)	71	16	0	0.173
8×8 (fixed)	0	0	1	0.173
8×7 (fixed)	63	15	0	0.172
7×6 (fixed)	45	13	0	0.17

TABLE III. RESOURCE USAGE FOR PE UNITS

Quantized layer number	Convolution kernel size	MACs	PE number	Multiplier type (*m* bits×*n* bits)	LUTs	DSPs	FFs	BRAMs
1	16×3×3×3	884736	1	8×8	7040	432	11008	8
2-7	16×16×3×3	28311552	2	7×6	67768	0	40392	48
8-14	32×16×3×3/32×32×3×3	28311552	2	7×6	67768	0	40392	48
15-21	32×64×3×3/64×64×3×3	28311552	3	8×7	89528	0	49144	96
22	64×10(fully connected layer)	1280	4	8×8	325	10	166	0

TABLE IV. RESOURCE CONSUMPTION FOR RESNET20

Resnet20	Resource usage	Available	Utilization (%)
LUT	186257	203800	91.39
FF	148385	407600	36.4%
DSP	442	445	52.6
BRAM(36Kb)	152	840	34.2

The resource requirements for the entire Resnet20 are shown in Table IV, where the DSP resource utilization rate is 52.6% and LUT resource utilization rate is 91.39%. DSP resources are used for the first quantized layer and the last fully connected layer, while both shared accelerators are constructed entirely using LUTs with low bit-width multipliers to minimize resource usage. Storage areas can be recycled, thereby saving a large amount of BRAM resources.

2) Analysis of the buffer reading and writing: In the Resnet20, all convolution kernel sizes are 3×3, so only one data reading and writing method needs to be designed. This paper sets the register set size to 6×8, and each set can hold 48 data to provide the data required for 24 convolutions. Using the method described in Section 3.3 to read and write data, four data can be written into the register set per cycle, so it will take 12 cycles to fill the entire register set.

To achieve efficient computation, pipeline is added to the writing and reading operations. The specific process is shown in Fig. 10. When reading data from rows 4-6, new data is written into rows 1-3, and the data in rows 4-6 can be used to complete reading and calculation in 6 cycles. During these 6 cycles, the data in rows 1-3 can be updated by writing new data into them. Similarly, when reading data from rows 1-3, new data is written into rows 4-6. This can achieve the ability to read 9 data required for a 3×3 convolution every cycle.

3) FPGA accelerator performance analysis: In order to validate the advantage of FPGA accelerator, we evaluated the computational efficiency of three different platforms: CPU (Intel Core i5-12400), GPU (NVIDIA RTX 2070 SUPER), and FPGA. Table V shows the results of Resnet20 model inference time and energy consumption on CIFAR-10 dataset using three different platforms. Since the CPU and GPU platforms perform better with large batch sizes, we set the batch size to multiple values to obtain their highest performance. The inference time per image was obtained by dividing the total time by the batch size.

The experimental results show that, the computing speed is 4.27 times faster than that of the CPU after quantization, layer fusion, and parallel computation using FPGA, and the required power only needs 5% of the CPU's power consumption. Moreover, because we design a large number of parallel PEs while also deeply customizing the CNN structure, our work achieves similar computational speed under a power consumption of only 6.2% of the GPU platform. Since our design retains the characteristics of the streaming processing architecture, it can still adopt pipeline design when applied to the computation of multiple sets of input images, and each PE can serve as a stage pipeline, which can further improve the overall computational performance.

TABLE V. COMPARISON WITH OTHER EXPERIMENTAL PLATFORMS

Platform	Frequency (MHz)	Latency (ms)	Power consumption (W)	Speedup ratio
CPU	2500	2.01	65	1
GPU	1605	0.54	215	3.72
FPGA	100	0.47	13.31	4.27

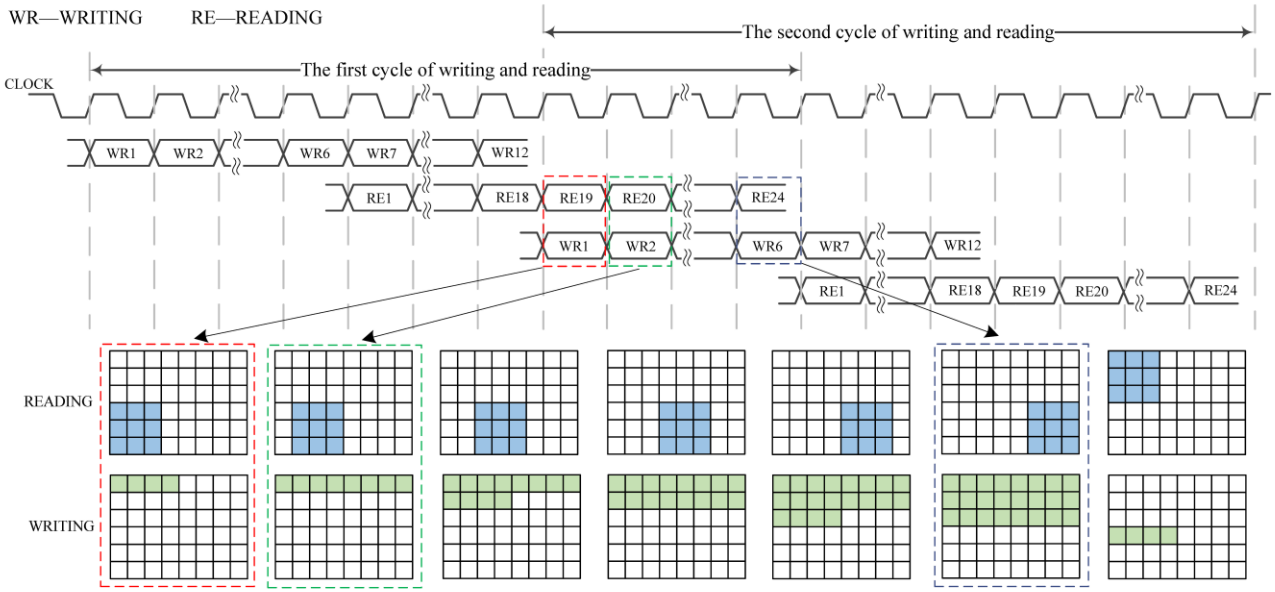

Fig. 10. Reading and writing data pipeline diagram.

Table VI shows the comparative results of other related work. In [35], the FFT method is used to reduce a certain amount of computational complexity, but still uses 16 bits long-data for computation. Due to the high computational complexity brought by long-bit-width data operations, in the case of limited DSP resources, it limits the computational parallelism. But multiple types of PEs are used in our paper which is suitable for different types of low-bit-width data operations to avoid low computational efficiency of a single long-bit-width PE structure. Therefore, compared with this method, our paper shortens the time by 89.3 times and reduces power consumption by 21.8 times. In addition, our paper reasonably allocates bit-widths for each layer of the CNN to improve its accuracy by 0.43%. In [36], they deploy a binary Resnet20 using an Application Specific Integrated Circuit (ASIC), and all network parameters are quantized to 1 bit. This method reduces computational complexity to the minimum via ultra low bit-widths, achieves higher frequency of operations and lower power consumption as shown in Table VI. However, this ultra low bit-width quantization method also produces a large precision error, which greatly affects the performance of the CNN. But the mixed precision quantization method and retraining the quantized model in our paper achieve a higher accuracy rate of 9.88% than that. In terms of computational speed, by efficiently utilizing the development board resources and increasing the parallelism of convolution calculation through multiplication circuits built with LUTs, it has achieved lower computational latency. In [37], the Winograd algorithm is used to reduce the computational load, but 16-bit data operation limits the overall performance, the mix precision low bit-width CNN in our work is more effective.

TABLE VI. PERFORMANCE COMPARISON

Work	Experiment Platform	Frequency (MHz)	Bit-Width (bits)	Latency (ms)	Throughput (GOP/S)	Energy (mJ)	Accuracy
[35]	Zynq 7020	154	16	42	/	137	91.25
[36]	ASIC 65nm	1000	1	0.98	/	3.8	81.8
[37]	ZynqZ7035	150	16	/	43.5	/	/
our	Kintex 7	100	6-8MP	0.47	179	6.26	91.68

V. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a high-performance CNN design method tailored for edge computing. Employing quantization methods and a strategy search algorithm in the software algorithm mitigated the significant accuracy loss associated with quantizing CNN models. In the FPGA accelerator design, we implemented a reuse structure based on a streaming processing architecture. This involved designing different Processing Elements (PEs) according to the characteristics of the CNN model structure and quantization bit-width. Notably, different network layers could share the same PE, optimizing resource utilization. For efficient data transmission, we adopted a strategy of packing quantized low-bit-width data into long words. This approach fully leveraged high-bandwidth data transfer, utilizing a register set as a buffer and employing a data reuse method to achieve synchronous data reading and computing. The validation of our method using Resnet20 on the CIFAR-10 dataset demonstrated its effectiveness. Comparative analysis with other computational platforms and related works revealed that our CNN accelerator outperformed a unified 16-bit FPGA accelerator design, achieving an 89-fold increase in computing speed with lower power consumption. Specifically, our CNN accelerator exhibited a computing speed 3.72 times faster than the GPU (RTX 2070 SUPER), while consuming only 6.2% of its power. In conclusion, our research presents a novel approach to high-performance CNN design for edge computing, showcasing substantial improvements in computing speed and power efficiency compared to existing methods. As part of future work, we plan to explore further optimizations and scalability of our approach, addressing potential challenges and extending its applicability to broader CNN architectures and datasets.

ACKNOWLEDGMENT

This research was funded by the National key R&D Program of China (2022YFE0107300).

REFERENCES

[1] Q. Jian, P.Y.Zhang and X. J. Wu. “FPGA implementation method for a configurable CNN Co-accelerator,” Journal of Electronics, vol. 47, no. 7, pp. 1525-1531, 2019.

[2] X. Peng. , J. Yu, B. Yao. L. Liu, Y. Peng. “A Review of FPGA-Based Custom Computing Architecture for Convolutional Neural Network Inference,” Chinese Journal of Electronics, vol. 30, no. 1, pp. 1-17, 2021.

[3] K. Guo. “Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 1, pp. 35-47, 2018.

[4] Y. Yu, C. Wu, T. Zhao, K Wang, and L. He, “OPU: An FPGA-based overlay processor for convolutional neural networks,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 1, pp. 35–47, 2020.

[5] Y. Wu, L. Kai, Y. Liu, et al. “Progress and trend of deep learning FPGA accelerator,” Chinese Journal of Computers, vol. 42, no. 11, pp. 2461-2480, 2019.

[6] A. Shawahna, Sait. S. M, El-Maleh. A. “FPGA-based accelerators of deep learning networks for learning and classification: a review,” IEEE Access, vol. 7, pp. 7823-7859, 2018.

[7] Z. J. Lin, X. W. Gao, X. P Chen, et al. “Design of high parallel CNN accelerator based on FPGA for AIoT,” The Journal of China Universities of Posts and Telecommunications, vol. 29, no. 05, pp. 1-9, 2022.

[8] Q. Dou, Y. Deng, R. Deng, et al. “Laius: an energy-efficient FPGA CNN accelerator with the support of a fixed-point training framework,” International Journal of Computational Science and Engineering, vol. 21, no. 3, pp. 418-428, 2020.

[9] Y. Ma, Y. Cao, S. Vrudhula, J.S. Seo, “Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks,” In Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 45–54, 2017.

[10] T. Chen, Z. Du, N. Sun, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," SIGARCH Computer Architecture News, vol. 42, no. 1, pp. 269-284, 2014.

[11] Y. Chen, T. Krishna, J. S. Emer, V. Sze. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, 2017.

[12] S. Yin, P. Ouyang, S. Tang. "A High Energy Efficient Reconfigurable Hybrid Neural Network Processor for Deep Learning Applications," IEEE Journal of Solid-State Circuits, vol. 53, no. 4, pp. 968-982, 2018.

[13] X. Ruan, W. Hu, Y. Liu. "Dynamic sparsity and model feature learning enhanced training for convolutional neural network-pruning," SCIENTIA SINICA Technologica, vol. 52, no. 5, pp. 667-681, 2022.

[14] J.Wang. "Lightweight and real-time object detection model on edge devices with model quantization," Journal of Physics: Conference Series, vol. 1748, no. 3, pp.1-10, 2021.

[15] B. Jacob, S. Kligys, B. Chen, et al. "Quantization and Training of Neural Networks for Efficient Integer-arithmetic-only Inference" Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704-2713, 2018.

[16] J. Achterhold, J. M. Koehler, A. Schmeink, et al. "Variational Network Quantization," International Conference on Learning Representations,, pp. 1-18, 2018.

[17] M. Rastegari, V. Ordonez, J. Redmon, et al. "Xnor-net: Imagenet Classification Using Binary Convolutional Neural Networks," European Conference on Computer Vision, Springer, Cham, pp. 525-542, 2016.

[18] Z. Liu, B. Wu, W. Luo, et al. "Bi-real Net: Enhancing the Performance of 1-bit Cnns with Improved Represent-ational Capability and Advanced Training Algorithm," Proceedings of the European Conference on Computer Vision(ECCV), pp. 722-737,2018.

[19] Z. Liu, Z. Shen, M. Savvides, et al. "Reactnet:Towards precise binary neural network with generalized activation functions," European conference on computer vision, Springer, Cham, pp. 143-159, 2020.

[20] E. Soufleri and K. Roy, "Network Compression via Mixed Precision Quantization Using a Multi-Layer Perceptron for the Bit-Width Allocation," IEEE Access, vol. 9, pp. 135059-135068, 2021.

[21] D. Lin, S. Talathi, S. Annapureddy, "Fixed point quantization of deep convolutional networks," International conference on machine learning, pp. 2849-2858, 2016.

[22] K. Wang, Z. Liu Z, Y. Lin, et al. "Haq: Hardware-aware Automated Quantization with Mixed Precision," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8612-8620, 2019.

[23] R. Q. Wang, et al. "Deep Neural Network Compression for Plant Disease Recognition," Symmetry, vol. 13, no. 10, pp.1-17, 2021.

[24] P. J. He, Z. Wu, S. Zhang, et al. "Deep network quantization via error compensation," IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 9, pp. 4960-4970, 2022.

[25] Z. Dong, Z. Yao, A. Gholami, et al. "Hawq: Hessian Aware Quantization of Neural Networks with Mixed-precision," Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 293-302 ,2019.

[26] Z. Dong, Z. Yao, D. Arfeen, et al. "Hawq-v2: Hessian Aware Trace-weighted Quantization of Neural Networks," Advances in Neural Information Processing Systems, vol. 33, pp. 18518-18529, 2020.

[27] Y. Yu, C. Wu, T. Zhao, K. Wang and L. He, "OPU: An FPGA-Based Overlay Processor for Convolutional Neural Networks," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 1, pp. 35-47, 2022.

[28] J. Wang, S. Li, Z. An,et al. "Batch-normalized deep neural networks for achieving fast intelligent fault diagnosis of machines," *Neurocomputing*, 2018, vol. 329, pp. 53-65.

[29] G. Li, "Block Convolution: Toward Memory-Efficient Inference of Large-Scale CNNs on FPGA," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 5, no. 41, pp.1436-1447, 2022.

[30] M. Cho, Y. Kim. "FPGA-Based Convolutional Neural Network Accelerator with Resource-Optimized Approximate Multiply-Accumulate Unit," Electronics, vol. 10, no. 22, pp. 1-16, 2021.

[31] M. Sait. "Optimization of FPGA-based CNN accelerators using metaheuristics," The Journal of Supercomputing, vol. 79, no. 4, pp. 4493-4533, 2023.

[32] P. Tommaso, R. Emilio, D. Gianmarco, et al. "A Multi-Cache System for On-Chip Memory Optimization in FPGA-Based CNN Accelerators," Electronics, vol. 10, no. 20, pp. 1-18, 2021.

[33] Gao Z, Zhang H, Yao Y, et al. "Soft error tolerant convolutional neural networks on fpgas with ensemble learning," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 3, pp. 291-302, 2022.

[34] J Hu, Ying. G, Q. Tian, et al. "Hardware implementation of neural network accelerator based on RISC-V," Electronics & Packaging, vol. 23, no. 2, pp. 1-6, 2023.

[35] Abtahi T , Shea C , Kulkarni A , et al. "Accelerating Convolutional Neural Network With FFT on Embedded Hardware," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1-24, 2018.

[36] Hosseini M, Mohsenin T, et, al. "Binary Precision Neural Network Manycore Accelerator," ACM Journal on Emerging Technologies in Computing Systems(JETC), pp . 1-27, 2021.

[37] Y. Yu, P. Zhang, H. Gong, et al. "Lightweight Network Hardware Acceleration Design for edge computing," Computer Science, vol. 50, no. S2, pp. 832-838, 2023.