Hindawi
International Journal of Distributed Sensor Networks
Volume 2024, Article ID 8018810, 11 pages
https://doi.org/10.1155/2024/8018810

Research Article

Custom Network Quantization Method for Lightweight CNN Acceleration on FPGAs

Lingjie Yi,[1] **Xianzhong Xie**,[1] **Yi Wan**,[1] **Bo Jiang**,[1] **and Junfan Chen**[2]

[1]*School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, China*
[2]*Chongqing Haiyun Jiexun Technology, Chongqing, China*

Correspondence should be addressed to Xianzhong Xie; xiexzh@cqupt.edu.cn

Received 6 September 2023; Revised 19 February 2024; Accepted 20 March 2024; Published 2 April 2024

Academic Editor: Kavita Pandey

The low-bit quantization can effectively reduce the deep neural network storage as well as the computation costs. Existing quantization methods have yielded unsatisfactory results when being applied to lightweight networks. Additionally, following network quantization, the differences in data types between the operators can cause issues when deploying networks on Field Programmable Gate Arrays (FPGAs). Moreover, some operators cannot be accelerated heterogeneously on FPGAs, resulting in frequent switching between the Advanced RISC Machine (ARM) and FPGA environments for computation tasks. To address these problems, this paper proposes a custom network quantization approach. Firstly, an improved PArameterized Clipping Activation (PACT) method is employed during the quantization aware training to restrict the value range of neural network parameters and reduce the loss of precision arising from quantization. Secondly, the Consecutive Execution Of Convolution Operators (CEOCO) strategy is utilized to mitigate the resource consumption caused by the frequent environment switching. The proposed approach is validated on Xilinx Zynq Ultrascale+MPSoC 3EG and Virtex UltraScale+XCVU13P platforms. The MobileNetv1, MobileNetv3, PPLCNet, and PPLCNetv2 networks were utilized as testbeds for the validation. Moreover, experimental results are on the miniImageNet, CIFAR-10, and OxFord 102 Flowers public datasets. In comparison to the original model, the proposed optimization methods result in an average decrease of 1.2% in accuracy. Compared to conventional quantization method, the accuracy remains almost unchanged, while the frames per second (FPS) on FPGAs improves by an average of 2.1 times.

1. Introduction

Convolutional neural networks (CNNs) are among the most commonly used networks in artificial intelligence (AI) applications. Moreover, they are widely applied in computer vision [1], natural language processing [2], and embedded systems [3, 4]. However, the complex network structures and continuously stacked convolution layers in such networks create a heavy burden regarding the storage and computational resources of resource-limited hardware devices. Under these circumstances, numerous researchers have put for compression methods, such as approximation, quantization, and pruning, to reduce the size of the networks [5–7]. These compression techniques play a crucial role in decreasing memory bandwidth usage by leveraging redundancy and irrelevance.

As an essential technique for network compression, quantization converts the parameters from 32-bit floating-point precision to 8-bit or lower, therefore, reducing the network's computational intensity, parameter sizes, and memory consumption. However, network quantization often leads to accuracy loss. Therefore, quantizing networks without significantly sacrificing precision has become a hot research topic within this field. To compensate the quantization loss, Jiao et al. [8] utilized DoReFa-Net's bit-wise convolution kernels for network quantization. Choi et al. [9] introduced the PACT method, which enhances the quantization of activations during the training process without

inducing significant performance degradation. While post-training quantization (PTQ) [5, 10] compresses the trained networks by quantizing parameters to lower bitwidth through partial validation data, the errors introduced by these approximations accumulate during forward propagation computations, leading to a significant performance dropout [11]. Jacob et al. [12] proposed a quantization aware training (QAT) method that injects quantizers into the network graph, computes network parameters during the training process, and applies straight-through estimator to approximate gradients. However, most of these methods [13, 14] only inject quantizers before the convolution operators, while many other operators, in more complex network structures, are left unquantized.

During the training process of CNNs, the support of the graphics processing unit (GPU) is typically required. However, using a GPU dedicated for network inference after the network is trained may increase the system's overall performance and resource consumption. To efficiently deploy and execute network simulation, Kotlar et al. [15] explored several operation locations and suitable underlying hardware architectures, including multicore processors, many-core processors, FPGAs, and application-specific integrated circuits (ASICs). As a result and due to their lower power consumption and flexible configurable hardware resources, FPGAs are the most suitable for deployment and inference as they have gradually become a new research hotspot [16–18]. However, implementing CNNs on FPGAs also faces several challenges. For instance, the conventional network VGG16 has 138 million parameters, which requires 255 MB of storage when applying a 32-bit format [19]. Furthermore, transferring these values to off-chip memory also incurs performance and energy overheads. In addition, due to the unique structure and limited resources of the FPGAs, numerical adjustments are required for inference deployment of networks.

To address the series of issues when using FPGAs regarding the acceleration of the CNN inference, this paper primarily implemented the following measures:

(i) A quantization aware training method that combines the improvements related to the PACT method is proposed, to quantize the networks. Specifically, during the quantization training process, an improved PACT method is applied for numerical preprocessing to compressed the values and facilitate quantization mapping

(ii) A quantization strategy, which is called Consecutive Execution Of Convolution Operators (CEOCO), is proposed, where the quantizers are added after the interruption of consecutive convolution operations. This is followed by retraining to ensure the consistency of input and output numerical types of the convolution operators, enabling them to be executed consecutively on FPGAs

(iii) The effectiveness of the proposed method is verified by conducting experiments on the Xilinx Zynq Ultrascale+MPSoC 3EG and the Virtex UltraScale +XCVU13P platforms. Moreover, MobileNetv1, MobileNetv3, PPLCNet, and PPLCNetv2 networks are employed as the experimental testbeds

To sum up, the remainder of the paper is organized as follows: Section 2 provides the background, followed by Section 3 that describes the proposed method. Furthermore, Section 4 presents the experimental process and its results, and finally, Section 5 concludes the paper and suggests future ideas to enhance this work.

2. Background

In this section, the background of the research conducted in this paper will be presented. Therefore, this section will introduce the following three aspects: firstly, the network architecture of the lightweight networks series, which are MobileNets and PPLCNets, will be presented. Moreover, the different network parameters as well as the computational requirements with the conventional networks will be introduced. Secondly, the quantization methods used in network compression, specifically the fixed-point scalar quantization and the PACT methods that reduce the number of bits, will be provided. Thirdly, the graph optimization, performed by deep learning (DL) frameworks when deploying networks to target hardware, such as FPGAs, will be explained.

2.1. Lightweight CNNs. Conventional classification networks, such as VGG and ResNet, have large network sizes and a high number of parameters. Without lightweight optimization, they are not suitable for deployment on edge devices [20, 21]. To address this issue, researchers proposed a lightweight CNN called MobileNetv1. The fundamental component of MobileNetv1 comprises depthwise separable convolution followed by pointwise convolution. This architecture not only maintains high classification accuracy, but it has also smaller parameters and computational complexity. Additionally, MobileNetv2 introduces an inverted residual structure featuring a linear bottleneck, which serves to further enhance the network's performance. In contrast, MnasNet introduces a lightweight attention model, based on the squeeze-and-excitation structure, leveraging strengths from different networks. Furthermore, MobileNetv3 combines the structures of the aforementioned three networks to further enhance the network depth and improve the learning capability. Furthermore, PPLCNet and PPLCNetv2 are two lightweight networks characterized by their simple structures and high efficiency. They perform exceptionally well in computer vision tasks although having smaller model sizes and computational requirements.

Compared to the conventional networks, the selected network parameters, multiadd, and size are represented in Table 1. It can be clear that the lightweight networks exhibit significant reductions at several levels. However, when deploying these lightweight networks on resource-limited devices, there are still challenges that need to be solved.

A comparative analysis of the runtime of various layers is also conducted in the experimental networks. To reach

Table 1: Details of different networks.

Network	Parameters (M)	Multiadd (GFLOPs)	Size (M)
VGG16	138.55	15.47	489
ResNet34	21.81	3.68	83.86
ResNet50	25.61	4.11	98.55
MobileNetv1	4.25	0.58	12.7
MobileNetv3	5.5	0.23	17.1
PPLCNet	2.96	0.16	7.62
PPLCNetv2	6.6	0.6	23.4

Table 2: The time distribution of each network structure.

Network	Conv (%)	Depthwise Conv (%)	BN (%)	Other (%)
MobileNetv1	49.3	9.7	24.9	16.1
MobileNetv3	62.3	3.6	15.6	18.5
PPLCNet	54.16	7.4	19.4	19.04
PPLCNetv2	60.89	4.4	11.5	23.21

this objective, four networks are deployed where the images are resized to $3 \times 224 \times 224$ pixels. They are then inserted into networks for inference. The runtime simulation is shown in Table 2. It is important to mention that the main time consumption is concentrated on the calculations of the convolution layers and batch normalization layers.

2.2. Quantization Methods. The fixed-point scalar quantization is a common technique used for quantization CNNs as it converts floating-point representations into fixed-point representations through the reduction of the number of bits for weights and activations. Moreover, fixed-point representations can significantly reduce the storage space required for the network and improve the inference speed. This is mainly due to fixed-point arithmetic that can be more efficiently executed on hardware as the use of fewer bits can save memory bandwidth and accelerate computation. Furthermore, in convolution calculations, the dot product operation is performed on matrices, and the matrix $W \in R^{m \times n}$ is assumed to be split into $m \times n$ blocks t_{mn} as represented here below:

$$W = \begin{pmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{m1} & \cdots & t_{mn} \end{pmatrix}. \tag{1}$$

Moreover, fixed-point scalar quantization is applied on W, transforming this matrix from a floating-point domain to a fixed-point domain using scale and bias. In addition, the fixed-point domain can be also converted back to the floating-point domain. The conversion formulas are as follows:

$$\begin{aligned} W_{\text{int}} &= \text{clamp}\left(-2^b, 2^b, \text{round}\left(\frac{W}{S} + Z\right)\right), \\ \widehat{W} &= S \times (W_{\text{int}} - Z), \end{aligned} \tag{2}$$

where W represents the original floating-point matrix, W_{int} represents the quantized fixed-point matrix, $\widehat{W}$ represents the dequantized floating-point matrix from W_{int}, round represents the round-to-nearest function, and clamp represents the truncation function. We compute the scale and bias as $S = (\max W - \min W)/(2^b - 1)$ and $Z = \text{round}(\max W_{\text{int}} - (\max W/S))$, and max and min represent maximum and minimum functions. Note that the round function is used in the conversion process to approximate the quantized fixed-point numbers to the nearest integer, in order to minimize the error during the conversion between the floating-point and fixed-point values.

Furthermore, the PACT quantization stands out as the most effective fixed-point quantization technique, achieving simultaneous quantization of both weights and activations. This method utilizes a trainable parameter, denoted as α as a truncation threshold. The computation formula for PACT is represented as follows:

$$y = \text{PACT}(x) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}, \tag{3}$$

where x represents the input data and y is the truncated result. The core principle of the PACT technique is to dynamically control the precision of quantization and truncation through the parameter α. The value of this parameter is adaptively adjusted based on the statistical characteristics of the data, aiming to achieve better quantization results. This approach enables a reduction in computational and storage requirements while maintaining network performance, thereby enhancing the efficiency and precision of quantized networks.

2.3. Computational Graph Optimization. A computational graph is used to depict the computation logic and the state of a deep neural network during the training and inference processes. It comprises fundamental unit data structures, known as tensors, along with operation units referred to as operators. In a computational graph, operators are denoted as nodes, and the directed edge-connected nodes represent the propagation of tensor states [22, 23]. Moreover, DL frameworks accurately dispatch operators to available computational resources, such as GPUs, NPUs, or FPGAs, to be ready for execution. During the network inference, each operator in the network reads/writes data from/to the registers, which can be time-consuming and computationally demanding. Therefore, the concept of operator fusion has been introduced to address this issue [24].

In fact, the operator fusion involves merging multiple consecutive operators into a more complex one to reduce data reads and writes in registers. For instance, within the

batch normalization folding, batch normalization is defined as a map of the output x:

$$\text{BatchNorm}(x) = \gamma\left(\frac{x-\mu_B}{\sqrt{\delta_B^2+\varepsilon}}\right)+\beta, \tag{4}$$

where γ is a coefficient for BatchNorm, β represents the bias, B is the minimum batch data, μ_B and δ_B^2 indicate the mean and variance of the B in the current layer, and ε is a small positive number greater than zero. During network inference, the parameters of BN are fixed. Thus, we can rewrite the terms such that BN operation is fused with the linear layer $y = Wx$:

$$y = \frac{\gamma}{\sqrt{\delta_B^2+\varepsilon}}(Wx-\mu_B)+\beta. \tag{5}$$

Using γ' to represent $\gamma/\sqrt{\delta_B^2+\varepsilon}$, Eq. (5) can be simplified as follows:

$$y = \gamma' Wx - \gamma'\mu_B + \beta. \tag{6}$$

By integrating the BN operator into the linear operator, the overhead of data transfer and register access can be minimized, resulting in enhancing the network inference efficiency.

The PaddleLite deployment tool performs regular operator fusion operations [25], such as convolution, BN, and *ReLU* operators. Except for the *ReLU* activation function, other activation functions, such as the *Swish* and *HardSwish* functions, are determined based on the operators' hardware support. In the computation graph, a subset of operators that possess data dependencies and are compatible with the target hardware is denoted as a subgraph. Moreover, PaddleLite not only fuses operators but also performs subgraph fusion by merging adjacent subgraphs into one to accelerate inference on heterogeneous chips. This merging process reduces the resource overhead of the platform switching.

3. The Proposed Method

In this section, the custom network quantization method is presented in detail. Firstly, the floating-point (FP) network is quantized using the QAT along with the improved PACT method. Secondly, the CEOCO method is applied to retrain the networks, and then, the subgraph fusion technique is utilized. The pipeline of the overview framework is illustrated in Figure 1.

In more detail, the blue boxes represent the steps, the orange circles indicate the convolution operators, and the green circles show other nonparameterized operators. As for the gray/dark gray circles, they represent the quantizers, commonly referred to as fake nodes in engineering. Finally, $Q(\cdot)$ represents the quantization and dequantization function.

3.1. QAT with Improved PACT Method. The quantization method used in the QAT is called fixed-point scalar quantization. It converts the floating-point representation into a fixed-point representation. The QAT is a simulation-based quantization technique, it updates parameters by simulating the errors introduced by quantization. Moreover, all variables used for forward and backward propagation are represented as floating-point. Therefore, the weight quantization and activation quantization nodes are injected into the computation graph to simulate the effects of quantization on the variables.

To facilitate hardware implementation, the symmetric quantization is adopted for both weights and activations. This approach uniformly maps values, within the positive and negative range, to a fixed range of the fixed-point integers, where the zero-point is a constant equal to 0. Moreover, weights undergo channel-wise symmetric quantization, allowing more precise quantization accuracy at the channel level. In addition, activations are subjected to layer-wise symmetric quantization. We denote certain convolution layer weights in the pretrained model by $W = \{W^1, W^2, \cdots W^n\}$; there are a total of n convolution kernels. Finally, the weights quantization and dequantization are performed according to the following equation:

$$\widehat{W}^i = Q\left(W^i, S_{W^i}\right) = \text{clamp}\left(-127,127,\text{round}\left(\frac{W^i}{S_{W^i}}\right)\right)\times S_{W^i}, \tag{7}$$

where the scale is

$$S_{W^i} = \frac{\max\left(\text{abs}\left(W^i\right)\right)}{127}. \tag{8}$$

In this case, the round represents the round-to-nearest function, clamp represents the truncation function, and max and abs represent absolute value and average value functions, respectively. The channel-wise quantization is not determined by the number of channels in the convolutional kernel, but through the total number of output channels in the convolutional layer. This is done to reduce computational complexity and speed up the inference process. In this approach, quantization is performed based on the number of output channels, and each output channel is assigned a separate scaling factor S_{W^i} for data type mapping. To introduce the quantization error during the forward propagation, $\widehat{W}^i$ is deployed for calculations. This facilitates gradient updates for weight adjustment during backward propagation. In situations involving round-to-nearest operation, the gradient is either zero or undefined everywhere. To address this problem, the straight-through estimator is employed.

The PACT method involves preprocesses weights and activations through quantization. Although weights typically lie in the range of -1 to 1, the activations can span an unlimited range. Thus, in this study, preprocessing is exclusively applied to activations. The original PACT method truncates positive activations during quantization; however, it ignores negative activations, leading to an error in calculating the

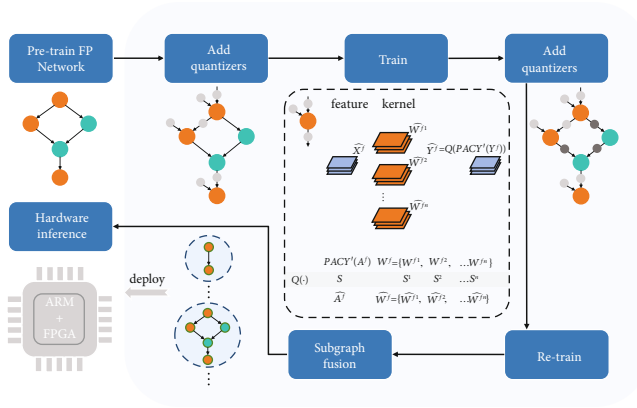


Figure 1: The overview framework with custom network quantization method.

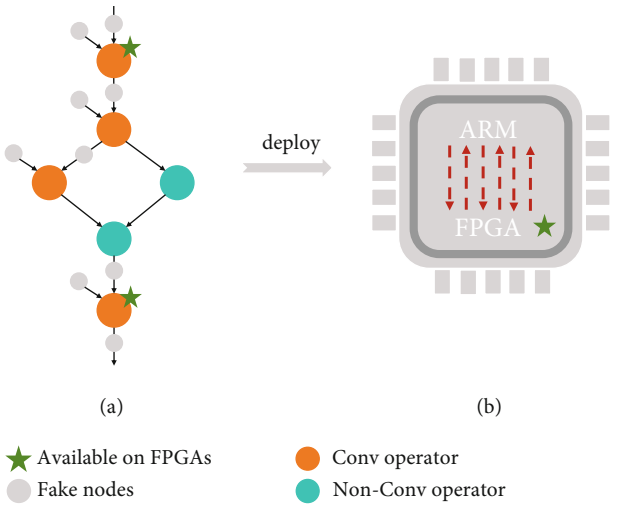


Figure 2: Network heterogeneous scheduling execution diagram.

scaling factor for quantization. Based on the activation properties of *ReLU* and *HardSwish* functions used in the experimental network, the minimum activations are either 0 or -3 whereas the maximum value can tend toward infinity. The improved PACT method in this paper is defined as follows:

$$y = \mathrm{PACT}'(x) = \begin{cases} -\beta, & x \in (-\infty, -\beta) \\ x, & x \in [-\beta, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}, \tag{9}$$

where the PACT′ presents the improved PACT method. It introduces truncation limits not only for values greater than zero but also uses a trainable parameter β as a truncation to constrain values within the range of [-β,0]. This approach enhances the quantization and simplifies the calculation of scaling factors. In this paper, the initial threshold value α is set as 20, and value β is set as 3 arising from the *HardSwish* function. During the training process, both thresholds are adjusted through gradient propagation and eventually converge to suitable truncation thresholds, respectively.

Next, activations are quantized and dequantized: $\widehat{X} = Q(\mathrm{PACT}'(X), S_X)$, where S_X is calculated using the strategy of sampling the moving average absolute maximum value. This strategy can reduce the network's sensitivity to noise and redundant information, thereby improving the network's generalization ability.

3.2. CEOCO in Retraining and Subgraph Fusion. The DL framework accurately allocates operators to the target hardware for execution and carries out heterogeneous scheduling of network operations, as shown in Figure 2.

Figure 2(a) represents the network structure composed of operators, and the red arrow in Figure 2(b) illustrates the diagram of frequent platform switching during heterogeneous scheduling. Considering the limited resources of FPGAs, the input and output data types of the operators, executed on the target FPGA hardware, are restricted to the int8 fixed-point integer. Hence, some convolution operators can be executed on FPGAs whereas the remaining should be executed on ARM. To achieve consecutive execution of the convolution operators on FPGAs and reduce the interaction cost between ARM and FPGAs, several convolution operators must be added to the subgraph, making the convolution operators conform to the hardware execution conditions as much as possible.

In the conventional quantization method, only the convolution operator is quantized whereas the other operators are not quantized. For example, in Figure 2, the convolution operators that are not marked as available will result in activation outputs with floating-point values, which can lead to several problems. Therefore, the network structure must be identified by adding fake nodes before these unquantized operators, as shown in Figure 3.

Figures 3(a)–3(c) represent network architectures, while Figures 3(d)–3(f) show the subgraph fused network architectures of the corresponding DL frameworks. In more detail, Figure 3(a) undergoes a transformation by adding fake nodes to become Figure 3(b). If the target hardware

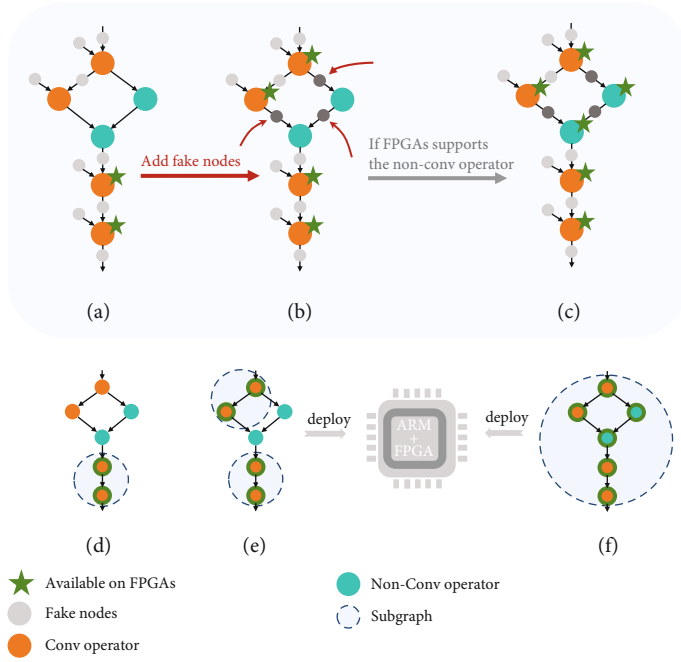


Figure 3: The CEOCO method diagram.

Input: Data, quantizers, pre-trained FP network with N convolutional layers
Output: The quantized network inference model
1: Add quantizers before convolution operators;
2: **for** $i = 1 ; i \leq N ; i \longleftarrow i + 1$ **do**
3: Forward propagation by $Q(\cdot)$ to weights of the network $\widehat{W}$ and by $Q(PACT'(\cdot))$ to activations of the network $\widehat{X}$;
4: Backward propagation by STE to update network parameters;
5: **end for**
6: Add quantizers before non-convolution operators;
7: Re-train the network and subgraph fusion;
8:**return** quantized network inference model;

Algorithm 1: Framework of custom network quantization.

supports the execution of other nonconvolution operators, Figure 3(b) can be further transformed into Figure 3(c).

During the retraining process, the fake nodes will record the data flowing through them and calculate the scaling factor. Subsequently, the activations at this stage will be quantized. When performing quantization training, the neural network not only obtains weights but also determines scaling factors for each layer's weights and activations, denoted as S_Y, S_X, and S_{W^i}, respectively. During network inference in the convolution layer, all variables are represented as 8-bit integer. From this, the formula for quantized inference can be expressed as follows:

$$S_Y Y_{\text{int}} = \sum_{i=1}^{N} S_{W^i} W^i_{\text{int}} \times S_X X_{\text{int}}, \tag{10}$$

organized as

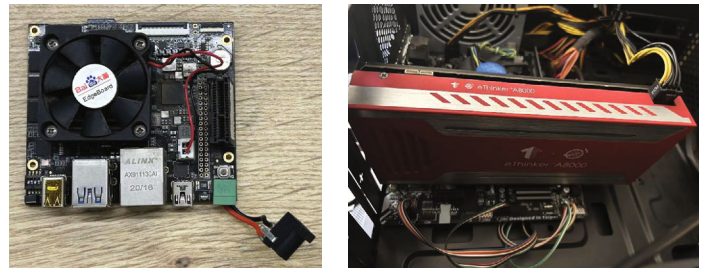
$$Y_{\text{int}} = \frac{S_{W^i} S_X}{S_Y} \sum_{i=1}^{N} W^i_{\text{int}} X_{\text{int}}, \tag{11}$$

where N presents the number of convolution kernels. Moreover, we assume that $M = S_{W^i} S_X / S_Y$, $M = 2^{-n} M_o$, where M represents the floating-point and M_o is a fixed-point. In the quantization inference, M can be replaced by M_o using the bit-shifting technique, so that all the data types in Eq. (11) become fixed-points.

The fake nodes are added to the successors of the convolution operators, and the deployment tool will incorporate the preceding convolution operators into the subgraph according to specific rules. Then, all convolution operators will be implemented on FPGAs using 8-bit integer arithmetic. Thus, this will enable the consecutive execution of the convolution operations. Therefore, the above procedure is summarized as Algorithm 1.

Table 3: Configuration of experimental environment.

Program		Detail
Hardware environment	CPU	Intel(R) Xeon(R) Gold 6148 CPU @ 2.40 GHz
	GPU	Tesla V100 (graphics memory 16 GB)
	Memory	100 GB
Software environment	Operating system	Linux Ubuntu 16.04
	Deep learning framework	PaddlePaddle 2.3.0
	Acceleration library	CUDA 12.0
	Python	Python 3.7.8
	Datasets	miniImageNet, CIFAR-10, OxFord 102 Flowers
Inference platform		Xilinx Zynq Ultrascale+MPSoC 3EG (XCZU3EG) Virtex UltraScale+XCVU13P (XCVU13P)



(a) (b)

Figure 4: Inference platform: (a) XCZU3EG and (b) XCVU13P.

Table 4: The comparison between the FP networks and the optimized networks.

Network	Size (M)	Top-1 accuracy (%)	Accuracy loss (%)	Compression ratio (%)
MobileNetv1-F	12.7	90.72	—	—
MobileNetv3-F	17.1	94.75	—	—
PPLCNet-F	7.62	91.51	—	—
PPLCNetv2-F	23.4	94.42	—	—
MobileNetv1-O	3.5	89.77	0.95	72.44
MobileNetv3-O	4.7	93.84	0.91	72.51
PPLCNet-O	2	89.68	1.83	73.75
PPLCNetv2-O	5.7	93.46	0.96	75.64

4. Experimental Evaluation

This section outlines the comprehensive experimental design employed in this work. Table 3 presents the relevant configuration of software, hardware, and inference hardware platform used in this study, as well as the exhibited experimental datasets.

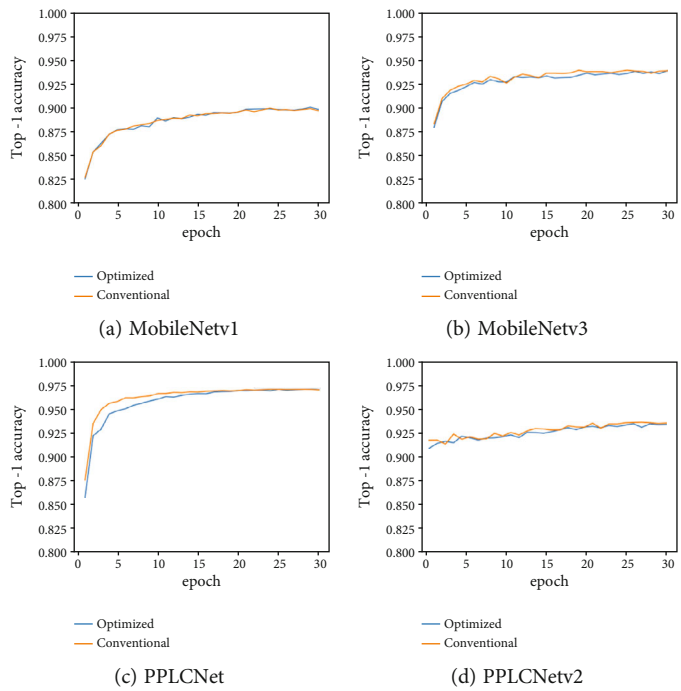
The program was deployed on specialized hardware platforms called Xilinx Zynq Ultrascale+MPSoC 3EG and Virtex UltraScale+XCVU13P, as presented in Figure 4.

4.1. Experimental Setting. This experiment uses the publicly available *miniImageNet* dataset [26]. It consists of 100 categories with a total of 60,000 color images. Each category has 600 sample images. Moreover, the dataset is divided into a training set and a validation set using an 8 : 2 ratio. The proposed method was deployed for training, and the following models have been deployed: MobileNetv1, MobileNetv3, PPLCNet, and PPLCNetv2. Furthermore, the Momentum optimizer was used with momentum set to 0.9 and fixing the batch size to 64. As for the learning rate, it was adjusted according to a cosine schedule during the 30 epoch training period. Moreover, the initial learning rate was set to 0.00375. In the training phase, each image was randomly cropped to 224×224 and randomly flipped horizontally. In the evaluation phase, the image was firstly resized to 256 along the short edge; then, a center crop of size 224×224 was applied.

4.2. Experimental Recognition Results and Analysis. The experimental comparison of network size, compression

Table 5: The comparison between the conventional quantized networks and the optimized networks.

Network	Top-1 accuracy (%)	XCZU3EG		XCVU13P		Size (M)	Compression ratio (%)
		Inference time (ms)	Time reduction (%)	Inference time (ms)	Time reduction (%)		
MobileNetv1-C	89.98	29.44	—	8.04	—	3.7	70.86
MobileNetv3-C	93.90	53.97	—	14.92	—	5.1	70.17
PPLCNet-C	89.56	16.35	—	4.30	—	2.2	71.12
PPLCNetv2-C	93.61	47.81	—	13.79	—	6.0	74.35
MobileNetv1-O	89.77	16.44	44.16	4.32	46.27	3.5	72.44
MobileNetv3-O	93.84	18.89	64.99	5.01	66.42	4.7	72.51
PPLCNet-O	89.68	9.88	39.57	2.69	37.44	2.0	73.75
PPLCNetv2-O	93.46	23.01	51.87	6.10	55.76	5.7	75.64



(a) MobileNetv1 (b) MobileNetv3

(c) PPLCNet (d) PPLCNetv2

Figure 5: The comparison diagram of the validation Top-1 accuracy.

effect, and recognition accuracy of the four FP networks and the optimized networks in this paper was conducted on *miniImageNet*. The results are shown in Table 4.

∗-F is the FP network, and ∗-O is the optimized network in this paper. It is evident that the sizes of all four networks before quantization are 12.7 M, 17.1 M, 7.62 M, and 23.4 M, respectively. However, after quantization, their sizes are reduced to 3.5 M, 4.7 M, 2 M, and 5.7 M achieving compression rates of 72.44%, 72.51%, 73.75%, and 75.64%, respectively. A major contributing factor to this compression is the quantization of weights in the convolution layers. By converting them from float32 to int8, the network size is reduced by approximately 73.58% in average. This remarkable compression effect is particularly noticeable in the convolution layers, which contain a significant part of the computed parameters. Furthermore, the optimized networks demonstrate a marginal decrease in accuracy on the validation set. Specifically, the Top-1 accuracy drops by 0.95%, 0.91%, 1.83%, and 0.96%, respectively. However, they are still meeting the precision requirements for image recognition tasks.

Furthermore, the deployment and inference of both the conventional quantized network and the optimized network are performed using XCZU3EG and XCVU13P chips. Therefore, the effects on the Top-1 accuracy and inference speed on the validation set are displayed in Table 5.

∗-C represents the conventional quantized network, which used QAT with the PACT method for training. The FPS on XCZU3EG increased to 1.79x, 2.86x, 1.65x, and 2.08x, respectively. On XCVU13P, the FPS increased to 1.86x, 2.98x, 1.60x, and 2.26x, respectively. The optimization methods achieve an improvement in the compression ratio of 1.58%, 2.34%, 2.63%, and 1.29%, respectively, compared to conventional quantization.

Table 6: The comparison among the FP network, conventional quantized network, and the optimized network.

Network	CIFAR-10/Flowers Top-1 accuracy(%)	XCZU3EG FPS
MobileNetv3-F	87.11/90.79	—
MobileNetv3-C	85.97/89.84	18.53
MobileNetv3-O	86.02/89.71	52.94

Table 7: Object detection network YOLOv3 model size, mAP@0.5, and FPS.

Network	Size (M)	mAP@0.5 (%)	FPS
YOLOv3-F	89.12	79.64	—
YOLOv3-C	24.45	78.77	3.79
YOLOv3-O	22.01	78.69	8.58

The variation in acceleration effects across different networks arose from the differences in inference performance lies in the situation where convolutional operators are executed consecutive in the original network. However, in general, the use of the method described in this paper has led to enhance the speed while maintaining accuracy. Therefore, Figure 5 presents the convergence of the validation set during the training of the four networks.

The diagrams of Figure 5 illustrate the comparison of accuracy between the validation sets during the training process. Moreover, Figures 5(a)–5(d) show, respectively, the comparison of the validation performance among MobileNetv1, MobileNetv3, PPLCNet, and PPLCNetv2 using the conventional quantization and the custom network quantization. Moreover, it can be observed that the accuracy does not show any noticeable loss, while there is a significant enhancement in inference speed.

To validate the universality of the optimization method proposed in this paper, we conducted experiments on *CIFAR-10* [27] and *OxFord 102 Flowers* [28]. The *CIFAR-10* dataset contains ten categories, each consisting of 6,000 images. It is composed of 50,000 training images and 10,000 validation images. The *Flowers* dataset encompasses a diverse collection of 102 different floral species. It consists of a total of 8,189 images, with 2,040 images used for training and the remaining images used for validation.

We adopted the same training settings as mentioned above for the *miniImageNet* dataset, with the only difference being that for the *CIFAR-10* dataset; the input image size is 32×32 pixels. Table 6 presents the accuracy of the original MobileNetv3, as well as the accuracy of the network after conventional quantization methods and the proposed optimization methods in this paper.

According to the observation results in Table 6, it can be concluded that both methods used to process the quantized model resulted in accuracy that was relatively close to the original accuracy. The accuracy only decreased by less than 1.1%. This finding serves as evidence for the effectiveness of the optimization method proposed in this paper on different datasets.

The optimization methods mentioned in this paper are also applicable to accelerating the inference of object detection networks in the field of object detection. For the one-stage object detection model YOLOv3, we replace its original backbone network with the lightweight network MobileNetv3. The experiments were conducted using the *PASCAL VOC* [29] dataset and validated the inference on the XCZU3EG platform. The input image size is 608×608 pixels. Table 7 presents the comparison of model size, average precision (mAP), and FPS.

We have observed from Table 7 that the original mAP@0.5 achieved by YOLOv3 is 79.64%. By employing both conventional quantization techniques and the optimized quantization method proposed in this paper, we have managed to reduce the model size from 89 M to 24.45 M and 22.01 M, respectively, while maintaining the accuracy drop within 1%. Furthermore, our proposed optimization method has demonstrated 2.26 times faster inference acceleration than the conventional quantization method on hardware platforms.

4.3. Ablation. The optimization methods, employed in this study, were subjected to ablation experiments, using the MobileNetv3-C network as the baseline. Moreover, the experimental results with the inclusion of the improved PACT and CEOCO methods on top of the baseline are shown in Table 8.

It can be observed that the improved PACT method, proposed in this paper, has achieved a 0.59% improvement in the original accuracy. Additionally, the adoption of the CEOCO strategy resulted in a significant boost at the level of the inference speed. These experimental results validate the effectiveness of all the optimization strategies described in the paper.

The CEOCO strategy focuses on adding fake nodes based on the model structure, enabling consecutive scheduled execution of convolutional operators on FPGAs. We conducted ablation experiments by inserting fake nodes before different nonconvolution operator structures in MobileNetv3 and performing inference on XCZU3EG, as shown in Table 9. We used "baseline+PACT"" from Table 8 as the baseline for this ablation experiment. Due to the negligible impact of individual operators on accuracy and FPS, we uniformly add fake nodes based on operator types.

According to Table 9, it can be observed that in the MobileNetv3 network structure, the number of preceding convolution operators for pool, elementwise-add, elementwise-mul, and dropout operators is 9, 16, 20, and 1, respectively. Fake nodes were uniformly added based on operator types. It was found that as the number of preceding convolution operators increases, the precision degradation becomes more significant, but the improvement in FPS becomes more pronounced. Additionally, the preceding nodes of pooling operators in the MobileNetv3 network are all in the form of residual structures. Therefore, if fake nodes are inserted solely in front of pooling operators, their preceding nodes will also not be included in the subgraph.

Table 8: The ablation experiment 1.

Network	PACT′	CEOCO	Top-1 accuracy (%)	FPS
Baseline			93.90	18.53
Baseline + PACT′	✓		94.49	18.50
Baseline + PACT′ + CEOCO	✓	✓	93.84	52.94

Table 9: The ablation experiment 2.

Network	Number of preceding convolution op	Top-1 accuracy (%)	FPS
Baseline	0	94.49	18.50
Baseline + pool	9	94.36	17.06
Baseline + elementwise_mul	16	94.21	31.51
Baseline + elementwise_add	20	94.10	36.29
Baseline + dropout	1	94.49	19.28
Baseline + all	46	93.84	52.94

5. Conclusion

This paper analyzes the structure of CNNs and addresses the challenges of deploying on ARM+FPGA heterogeneous chips. Therefore, a method, called custom network quantization, is proposed in this work. It involves the improved PACT method and the CEOCO strategy. The experiment involves training MobileNetv1, MobileNetv3, PPLCNet, and PPLCNetv2 networks using the *miniImageNet*, *CIFAR-10*, and *OxFord 102 Flowers* datasets and deploying them on XCZU3EG and XCVU13P chips to test the performance of the different methods. The results demonstrate that the Top-1 accuracy of these optimized networks decreases by 1.2% in average, while compressing four networks to approximately fourth of their original sizes. Moreover, inference on the chip exhibits a significant improvement at the speed level.

As for the future works, we will proceed with exploring other methods for model compression, aiming to further reduce the size of the model while ensuring that the loss in network recognition accuracy maintains an acceptable range.

Data Availability

The datasets used or analyzed during the current study are available from the corresponding author on reasonable request. The code is available from the corresponding author on reasonable request.

Conflicts of Interest

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Acknowledgments

This work is partially supported by the Special Key Project of Technological Innovation and Application Development of Chongqing (CSTB2022TIAD-KPX0057).

References

[1] X. Yao, Q. Wu, P. Zhang, and F. Bao, "Weighted adaptive image super-resolution scheme based on local fractal feature and image roughness," *IEEE Transactions on Multimedia*, vol. 23, pp. 1426–1441, 2021.

[2] B. Alshemali and J. Kalita, "Improving the reliability of deep neural networks in NLP: a review," *Knowledge-Based Systems*, vol. 191, article 105210, 2020.

[3] J. C. Vieira, A. Sartori, S. F. Stefenon, F. L. Perez, G. S. De Jesus, and V. R. Q. Leithardt, "Low-cost CNN for automatic violence recognition on embedded system," *IEEE Access*, vol. 10, pp. 25190–25202, 2022.

[4] Z. Chen, J. Chen, G. Ding, and H. He, "A lightweight CNN-based algorithm and implementation on embedded system for real-time face recognition," *Multimedia Systems*, vol. 29, no. 1, pp. 129–138, 2023.

[5] J.-y. Li, Y.-k. Zhao, Z.-e. Xue, C. A. I. Zheng, and L. I. Qing, "A survey of model compression for deep neural networks," *Chinese Journal of Engineering*, vol. 41, no. 10, pp. 1229–1239, 2019.

[6] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5113–5155, 2020.

[7] A. Kumar, A. M. Shaikh, Y. Li, H. Bilal, and B. Yin, "Pruning filters with L1-norm and capped L1-norm for CNN compression," *Applied Intelligence*, vol. 51, pp. 1152–1160, 2021.

[8] L. Jiao, L. Cheng, W. Cao, X. Zhou, and L. Wang, "Accelerating low bit-width convolutional neural networks with embedded FPGA," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Ghent, Belgium, 2017IEEE.

[9] J. Choi, Z. Wang, S. Venkataramani, P. I. J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: parameterized clipping activation for quantized neural networks," 2018, http://arxiv.org/abs/1805.06085.

[10] T. Dinh, A. Melnikov, V. Daskalopoulos, and S. Chai, "Subtensor quantization for Mobilenets," in *In Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pp. 126–130, Springer, 2020.

[11] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. H. Hassoun, "Post-training piecewise linear quantization for deep neural networks," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, pages 69–86*, Springer, 2020.

[12] B. Jacob, S. Kligys, B. Chen et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, Salt Lake City, 2018.

[13] K. Uday, S. M. Meena, P. Joshua et al., "Performance improvements in quantization aware training and appreciation of low precision computation in deep learning," in *Advances in Signal Processing and Intelligent Recognition Systems: 6th International Symposium, SIRS 2020, Chennai, India, October 14–17, 2020, Revised Selected Papers 6*, pp. 90–107, Springer, 2021.

[14] C. Liu and Z. Dong, "A selective quantization approach for optimizing quantized inference engine," in *2023 11th International Conference on Information Systems and Computing Technology (ISCTech)*, pp. 92–99, IEEE, Qingdao, China, 2023.

[15] M. Kotlar, D. Bojić, M. Punt, and V. Milutinović, "Survey of deployment locations and underlying hardware architectures for contemporary deep neural networks," *International Journal of Distributed Sensor Networks*, vol. 15, no. 8, Article ID 155014771986866, 2019.

[16] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Computing and Applications*, vol. 32, no. 4, pp. 1109–1139, 2020.

[17] S. Li, Y. Luo, K. Sun, N. Yadav, and K. K. Choi, "A novel FPGA accelerator design for realtime and ultra-low power deep convolutional neural networks compared with Titan X GPU," *IEEE Access*, vol. 8, pp. 105455–105471, 2020.

[18] M. A. Zhichao Zhang, P. Mahmud, and A. Z. Kouzani, "FitNN: a low-resource FPGA-based CNN accelerator for drones," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21357–21369, 2022.

[19] S. Sordillo, A. Cheikh, A. Mastrandrea, F. Menichelli, and M. Olivieri, "Customizable vector acceleration in extreme-edge computing: a RISC-V software/hardware architecture study on VGG-16 implementation," *Electronics*, vol. 10, no. 4, p. 518, 2021.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, http://arxiv.org/abs/1510.00149.

[21] A. Gholami, K. Kwon, B. Wu et al., "SqueezeNext: hardware-aware neural network design," in *In Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 1638–1647, Salt Lake City, 2018.

[22] B. Pang, E. Nijkamp, and W. Ying Nian, "Deep learning with tensorflow: a review," *Journal of Educational and Behavioral Statistics*, vol. 45, no. 2, pp. 227–248, 2020.

[23] A. Paszke, S. Gross, F. Massa et al., "Pytorch: an imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] T. Chen, T. Moreau, Z. Jiang et al., "{TVM}: An automated {end-to-end} optimizing compiler for deep learning," in *In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, USENIX, 2018.

[25] Y. Ma, D. Yu, T. Wu, and H. Wang, "Paddlepaddle: an open-source deep learning platform from industrial practice," *Frontiers of Data and Domputing*, vol. 1, no. 1, pp. 105–115, 2019.

[26] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra, "Matching networks for one shot learning," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[27] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: KNN-based ensemble of classifiers," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1192–1195, IEEE, Las Vegas, NV, USA, 2016.

[28] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729, Bhubaneswar, India, 2008.

[29] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.