LSFQ: A Low Precision Full Integer Quantization for High-Performance FPGA-based CNN Acceleration

Zhenshan Bao, Kang Zhan, Wenbo Zhang, Junnan Guo
Faculty of Information Technology
Beijing University of Technology
Beijing, China
baozhenshan@bjut.edu.cn, kangkang@emails.bjut.edu.cn, zhangwenbo@bjut.edu.cn, guojn@emails.bjut.edu.

***Abstract*—Neural network quantization has become an important research area. Deep networks run with low precision operations at inference time offer power and space advantages over high precision alternatives, and can maintain high accuracy. However, few quantization can demonstrate this advantage on hardware platform, because the design of quantization algorithm lacks the consideration of actual hardware implementation. In this paper, we propose an efficient quantization method for hardware implementation, a learnable parameter soft clipping fully integer quantization (LSFQ), which includes weight quantization and activation quantization with learnable clipping parameter method. The quantization parameters are optimized automatically by back propagation to minimize the loss, then the BatchNorm layer and convolutional layer are fused, and the bias and quantization step size are further quantized. In this way, LSFQ accomplishes integer-only-arithmetic. We evaluate the quantization algorithm on a variety of models including VGG7, mobile-net v2 in CIFAR10 and CIFAR100. The results show that when the quantization reaches 3-bit or 4-bit, the accuracy loss of our method is less than 1% compared with the full-precision network. In addition, we design an accelerator for the quantization algorithm and deploy it to the FPGA platform to verify the hardware-awareness of our method.**

Index Term—Low Precision Quantization， convolutional neural network (CNN) accelerator， field-programmable gate array (FPGA)

I. INTRODUCTION

As a powerful tool in machine learning, deep neural networks have achieved remarkable success in various computer vision and image processing tasks. Moreover, it exceeds the human ability in many specialties. However, the deep neural network is difficult to be applied on the resource-constrained devices due to its large amount of parameters and computation. At present, the common solution is to deploy it in the data center and use the high performance and expensive devices for computing. However, there are also problems such as the high electricity cost caused by excessive power consumption, or the long transmission delay caused by limited network conditions. What's more, neural networks have a lot of redundancy and should be optimized in some ways. Therefore, it is critical to design a more efficient and lower inference cost network architecture. More and more researchers begin to pay attention to the compression of deep neural networks.

Among many compression methods, quantization is the key step to deploy the neural network to the exclusive hardware platform. Quantization of high precision floating point networks is represented by the reduced bit width. It can also greatly reduce static parameters and intermediate parameters in network inference, thus reducing memory footprint and computational intensity. Quantization is also closely related to the realization of specialized hardware that maps network inference processes to low-energy and low-precision integer or fixed point arithmetic circuits. The advantages of using quantization algorithm to create networks for low-precision hardware have been demonstrated in several deployed systems [1-4].

There have been many outstanding low-precision quantitative studies. Recently, the most worthy researches lie on the hardware-ware design, like the hardware-aware automated quantization with mixed precision (HAQ) [5] and the SqueezeNext [6]. Based on the previous researches, we get the following design rules.

1) **Aim to the full integer quantization**. In previous studies, weight quantization and activation quantization were the core of the research, and many works were carried out on these local computing units, because these were the main computational content of neural networks. There are other common computing units in neural networks, such as the BatchNorm layer. From hardware awareness, these units also bring computation and are not easily mapped to low-precision integer or fixed-point arithmetic circuits. In addition, in order to ensure the accuracy of many researches, the first and last layers of the neural network are often not quantized. Such quantization algorithm is difficult to retain the accuracy in the hardware-friendly neural network. So we think our research direction should be full integer quantization. When designing the quantization algorithm, all the computing units in the neural network should be considered. According to the experimental analysis, the weight, bias and activation function of any layer in the CNN neural network should be quantified, means the whole integer quantization, which is consistent with Zhao X's point of view [7].

2) **Aim to the linear symmetric quantization**. Generally speaking, the quantization algorithm is divided into non-linear quantization and linear quantization. Nonlinear quantization refers to the representation of the quantized data with nonlinear encoding, which requires additional transformation to obtain the correct algorithm results. For example, in some classical deep compression scheme [8, 9], a large amount of lookup tables are consumed and its efficiency of execution would be compromised. In contrast, linear quantization can be more suitable for the low precision operation of accelerators. Linear quantization can be divided into symmetric quantization and asymmetric quantization. Asymmetric quantization takes one more parameter than symmetric quantization and requires additional subtraction or linear operations before multiplication. Therefore, symmetric quantization is more suitable for common accelerator chip design, and there is no need to re-design the data path in these hardware. Due to technical versatility and hardware awareness, the linear symmetry quantization is taken as our research direction.

To sum up, a new quantization method would be proposed, which includes the following contributions:

2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS) | 978-1-6654-1503-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/COOLCHIPS52128.2021.9410327

- LSFQ, as a new weight quantization method, is proposed to find the optimal quantization truncation parameter and the distribution parameter during training. Then, two parameters are introduced into the quantizer to indicate the clip position and the degree of sparse distribution, and are learned through back propagation.
- Different from other quantization methods, the constraints from hardware resources are considered adequately in ours. The whole network is quantified, which includes the bias and the step size, from the first layer to the last one.
- Based on the LSFQ, we designed a low precision CNN accelerator on the target platform.

II. Related Work

In this work, we focus on a hardware-oriented approach to quantization. In general, there are two main categories of hardware-aware quantization methods. The first is a quantitative approach without training. In 2011, Vincent Vanhoucke[10] proposed the linear fixed-point 8-bit quantization technology and greatly reduced the computing cost under X86 CPU, which is the first paper in the area. At present, almost all the well-known quantization methods are 8-bit, such as NVIDIA tensorRT and Google quantization [11].

While, the second one aims to obtain a lower level wide less than 4-bit network by quantitative training method, and achieve a precision similar to that of the full-precision network at the same time. With this approach, the weights, the activations, and even the gradient parameters can be quantified to very low-bit widths, so as to reach a rapid inference on the specific target hardware. The core trouble lies in the parameter dispersion, which is mainly caused by the bad defined to the back propagation. Dorefa-net [12] used the Straight-Through-Estimator [13] to estimate the neuron gradient. With the k-bit quantization to the weight and activation parameters, the training and inference process can be accelerated while ensuring the accuracy, which can be effectively applied to CPU, FPGA, ASIC and GPU and other hardware. In recent years, the quantization approaches have been improved from many aspects. In 2018, Choi J added adaptive parameters to clip the output of the activation, which is named PACT [14]. In 2020, Esser S K proposed LSQ [15], which adaptive training on the quantization step size, and Zhao X carried out LLSQ [7], which adjusts the quantization scales adaptively.

III. Learnable Parameter Soft Clipping Full Integer Quantization

In this section, we firstly give the overview of our proposed scheme of quantization, LSFQ. Then the scheme is elaborated, including the low precision representation, the BatchNorm layers fusion, and the deployment of quantization model on our specialized integer-only CNN accelerator for fast inference.

A. Overview of LSFQ

Compared with these previous works, our proposed quantization scheme focuses on the constraints imposed by the real hardware. The low-bit integer data for inference is utilized to compute the entire neural network which includes the convolutional (Conv) layer, the fully connection (FC) layer, the BatchNorm (BN) layer. The weight, bias, activation, and step size factors are all quantized to low-bit integers. The overview of the quantization scheme is shown in Figure 1.

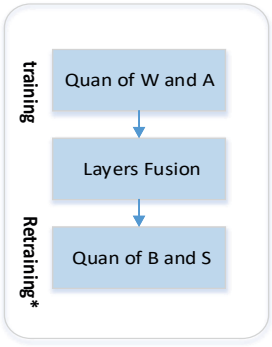

Figure 1: Overview of LSFQ

The quantization scheme can be divided into three steps. 1) Training of the network with quantized weights and activations to low-bit (2-4bit); 2) Conv-BN or FC-BN fusion for efficient inference; 3) Quantization of bias and step size.

B. Soft Clipping Quantization

It is shown that the weight distribution of deep neural network is subject to normal distribution with a mean value of 0, so the linear symmetric quantization in weight quantization is adopted. Its general form is:

$$q_w = round\left(\frac{clip(w,-c,c)}{s}\right) \tag{1}$$

$$s = \frac{c}{2^{k-1}-1} \tag{2}$$

$$\widehat{w} = sq_w \tag{3}$$

Where, $w \in \mathbb{R}$ is one kernel of one layer weights, the variable $c \in \mathbb{R}^+$ is the quantization parameter, known as the clipping value, $s \in \mathbb{R}^+$ can calculated by c, known as the step size. While $q_w \in \{-2^{k-1}+1, \dots, -1, 0, 1, \dots, 2^{k-1}-1\}$ is the integer values flowing in the integer accelerator and $\widehat{w} \in \{-(2^{k-1}-1)s, \dots, -s, 0, s, \dots, (2^{k-1}-1)s\}$ is the quantized weights. $clip(v,l,h)$ returns v with values below l set to l and values above h set to h, $round(v)$ rounds v to the nearest integer.

$Clip$ can intercept the dense part of weight distribution for quantization and reduce the overall quantization error. Gradient $\frac{\partial \widehat{w}}{\partial w}$ can be computed. Thus,

$$\frac{\partial \widehat{w}}{\partial w} = \begin{cases} 0\ , & w \in (-\infty, -c) \\ 1\ , & w \in (-c, c) \\ 0\ , & w \in (c, +\infty) \end{cases} \tag{4}$$

From eq(4), when $w < -c$ or $w > c$, Gradient $\frac{\partial \widehat{w}}{\partial w}$ is set to be 0, which leads to the precision loss of the quantization because the w parameter cannot be well learned. To solve this problem, we use the $soft\_clip$ instead of $clip$, which is defined as follows:

$$soft\_clip(x,a,b) = b \cdot tanh(ax) \tag{5}$$

Where, $a \in \mathbb{R}^+$ and $b \in \mathbb{R}^+$ are the parameters of $soft\_clip(x,a,b)$, a and b are used to adjust $soft\_clip(x,a,b)$ to approximate $clip(v,l,h)$, as shown in the figure 2.

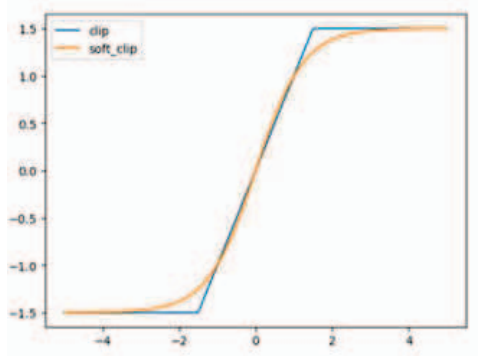

Figure 2: The comparison between $clip(x, -1.5, 1.5)$ and $soft\_clip(x, 0.8, 1.5)$

The weight quantization of $soft\_clip$ is as follows:

$$q_w = round\left(\frac{soft\_clip(w, \alpha, \beta)}{s}\right) \tag{6}$$

Here, q_w, s are same as eq(1), the variable $\alpha, \beta \in \mathbb{R}^+$ are the quantization parameter. They are dynamically adjusted via gradient descent-based training with the objective of minimizing the accuracy degradation arising from quantization.

As the defined above, we use α and β as our quantization parameters. We provides a way to learn them from the training loss by the following gradient through the quantizer to the soft clipping parameter:

$$\frac{\partial \widehat{w}}{\partial \alpha} = \beta(1 - tanh^2(\alpha w)) \tag{7}$$

$$\frac{\partial \widehat{w}}{\partial \beta} = tanh(\alpha w) \tag{8}$$

The gradient $\frac{\partial \widehat{w}}{\partial w}$ can be computed as:

$$\frac{\partial \widehat{w}}{\partial w} = a \cdot b \cdot (1 - \tanh^2(ax)) \tag{9}$$

C. Learned Clipping Parameter Activation

The above method of soft clipping quantization is also applicable to the activation quantization. However, since the activation quantization requires real-time processing of feature map, and the $tanh(x)$ will bring unnecessary computation, PACT[14] as the activation quantization is chosen, which can be defined as:

$$q_x = round\left(\frac{clip(x, 0, c)}{s}\right) \tag{10}$$

$$s = \frac{c}{2^k - 1} \tag{11}$$

$$\hat{x} = sq_x \tag{12}$$

Where, $x \in \mathbb{R}$ is one kernel of one layer activations, the variable c, s are the same ones as eq(1). While $q_x \in \{0,1, \dots, 2^k - 1\}$ is the integer flowing in the integer accelerator and $\hat{x} \in \{0, s, \dots, (2^k - 1)s\}$ is the quantized activations. c is taken as the quantization parameter. Thus, the gradient can calculated by:

$$\frac{\partial \hat{x}}{\partial c} = \begin{cases} 0 & , x \in (-\infty, c) \\ 1 & , x \in (c, +\infty) \end{cases} \tag{13}$$

In the same way, the gradient $\frac{\partial \hat{x}}{\partial x}$ can be computed using the Straight-Through-Estimator (STE) to estimate it as 1.

D. Conv-BN Fusion and FC-BN Fusion

In the quantized neural networks, the convolutional layer or the fully connection layer is usually followed by batch norm layer. As a result, the Conv-BN and FC-BN fusion would be designed to reduce the latency of network inference by removing additional computation overhead. The operator of quantized Conv and FC layers can be expressed as:

$$o_c = s_w q_w s_x q_x + b \tag{14}$$

Here, s, q are the same as eq(1), $s_w q_w$, $s_x q_x$ are the quantized weights and activations matrixes, the b is the bias and o_c is the output feature matrix. Note that s_w, s_x and b are full precision values.

The BN layer can be define as:

$$o_b = \frac{o_c - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \varphi \tag{15}$$

Where μ and σ^2 are EMA statistics, ϵ is a small value of constant. γ and φ are learned parameters in BN layers.

Obviously, BN layers can be merged. And the merging is defined as:

$$o_b = s_b q_w s_x q_x + b_b \tag{16}$$

$$s_b = \rho s_w;\ b_b = (b - \mu)\rho + \varphi;\ \rho = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \tag{17}$$

Where s_b, b_b are merged step size and bias parameters. They are of the full precision.

E. Bias and Step Size Quantization

To get integer-only-arithmetic, both biases and step size should be quantified. Firstly, $s_b$and s_x need be multiplied to get the step scale s_f. Therefore, the eq(16) is converted into the following form:

$$o_b = s_f q_w q_x + b_b \tag{18}$$

$$s_f = s_b s_x \tag{19}$$

Then, s_f and b_b are quantified. Notice that s_f is usually lower than 1, and the round operation will incur huge error, so it should be amplified to a certain extent and be integer-processed with round operation. The specific method is as follows:

$$o_b \approx \widehat{o_b} = \frac{q_s q_w q_x + q_b}{2^{l\_shift}} \tag{20}$$

$$q_s = round\left(2^{l\_shift} s_f\right) \tag{21}$$

$$q_b = round(2^{l\_shift} b_b) \tag{22}$$

Where, $\widehat{o_b}$ is quantitative o_b, q_s, q_b are the amplified integers s_f and b_b. l_shift is a parameter, which we call the left shift factor, which we initialize to 12. When the left shift factor is set 12, the gap with $\widehat{o_b}$ and o_b is negligible, almost no precision loss. This is also demonstrated in subsequent section. When the left shift factor is set small, such as 4, the accuracy needs to be restored by retraining. The quantization method of Step Size and bias of BN fusion layer is the above, but it is not difficult to find that the method is also applicable to a single convolutional layer or a fully connection layer.

IV. Accelerator Architecture

In this section, we discuss in detail the architecture of the processor, which efficiently supports the inference process of ours quantized networks for CNNs.

A. Overview of Accelerator

Figure 3 shows the overall structural design of the accelerator. Our accelerator (highlighted in light yellow) can be called by the CPU and used to compute the entire neural network at once. The CPU provides scheduling and control for the accelerator. The whole accelerator is a streaming structure, and all the network parameters are placed on the chip, and the data flow is organized in a similar way to FINN[16]. Input flows into the accelerator from off-chip memory, and the results flow back into memory after convolution computation at each layer. Due to the all weights on-chip design, all the layers can work in parallel, and the intermediate results do not have to be frequently swapped with off-chip memory, which saves the time of data transport. Among them, MVAU module realizes convolution or full connection calculation, batch regularization operation, activation calculation and other functions, which are important modules of hardware implementation and will be described in detail later. Pool is the pooling module, and the Controller module controls the coordination of the whole accelerator.

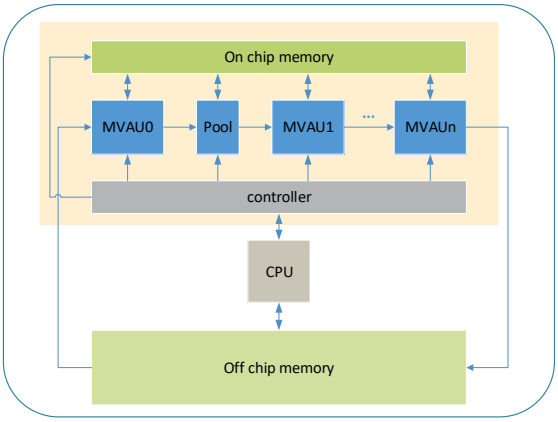

Figure 3: Accelerator Architecture

B. The Matrix–Vector–Activation Unit

In order to deploy the quantized network to the specialized integer-only CNN accelerator or other integer hardware, the linear symmetric quantization is usually used and all the parameters and characteristic data are quantized to the integer ones. LSFQ can be formulated as follow:

$$q_a = round\left(clip\left(\frac{q_s q_w q_x + q_b}{2^{l\_shift}}, 0, 2^k - 1\right)\right) \qquad (23)$$

And then the operation of eq(23) should be implemented on the target platform. So we designed the following hardware structure, the LSFQ implement unit is designed, as shown in Figure 4. Activation and weights are quantized to the 4-bit network of the accelerator PE cell schematic, $l_{shift} = 12$. MUL and Adder tree are designed for the vector computing unit of low-bit, and accumulator gets the calculation result of $q_w q_x$. S memory stores q_s(16-bit) and B memory stores $q_b + round(\frac{q_s}{2})$(24-bit). Clip module truncates data between [0, $2^{16} - 1$]. The ">>" module moves data 12-bit to the right, which ensure that the output is 4-bit. The MUL in the figure is a SIMD multiplier that is capable of calculating PN multiplications per cycle and PN can be set manually. Therefore, the PE needs to receive PN data every time.

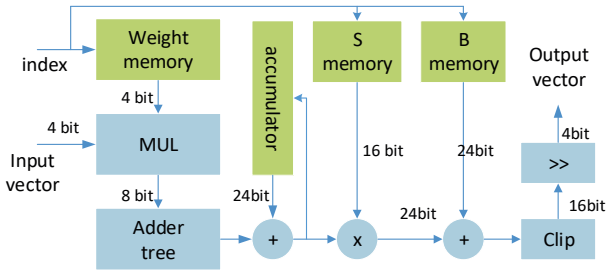

Figure 4: computing element

As shown in Figure 5, the above PE organization is combined with an input and output buffer to form the core calculation module Matrix-Vector -Activation Unit designed by us. The corresponding eq(23) is responsible for Matrix Vector multiplication calculation, and contains batch regularization and Activation calculation formulas. Therefore, MVAU can be configured to achieve CONV-BN-RELU, FC-BN-RELU, CONV, FC, etc. The number of PE in MVAU is configurable, and we record the number of PE in each MVAU as PM. Since each MVAU module corresponds to a layer in the convolutional network, PN and PM corresponding to MVAU can be adjusted with more computing capacity of that layer. So as to realize the balance calculation between the layers.

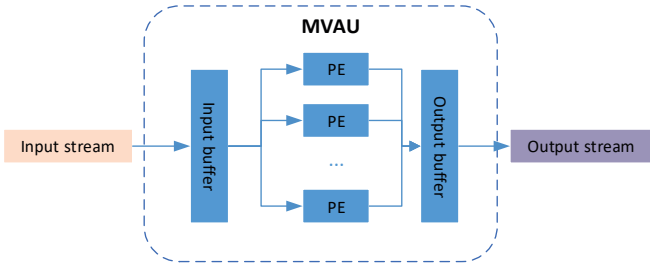

Figure 5: Matrix–Vector–Activation Unit

V. Results

In this section, we test the LSFQ method on CIFAR 10 and CIFAR 100, and compare the accuracy with some advanced quantitative methods. Then we implement the quantized network on FPGA, and compare the speed, accuracy, resource consumption and other aspects with some previous acceleration schemes. Finally, we applied the whole scheme on the DAC System Design Contest 2020 (DAC-SDC2020) [24] dataset and verified the usability of the scheme.

A. VGG7 on CIFAR 10

Firstly, the LSFQ is validated on CIFAR10 VGG7. The VGG7 network structure is similar to that used in [17], including 6 convolutional layers, 3 max-pooling layers, and a full connection layer. The numbers of convolutional layer filters are 128,128,256,265,512,512. We use Adam optimizer and cosine learning rate scheduler to train LSFQ-VGG7 model. Specifically, we trained 200 epochs with the initial learning rate of 1E-3 for the full precision and 4-bit quantitative model without pre-training. For the 3-bit quantized model, 50 epochs were trained with the parameters of the 4-bit model as the pre-training parameters and the starting learning rate of 2E-4; for the 2-bit quantized model, 50 epochs were trained with the parameters of the 3-bit model as the pre-training parameters.

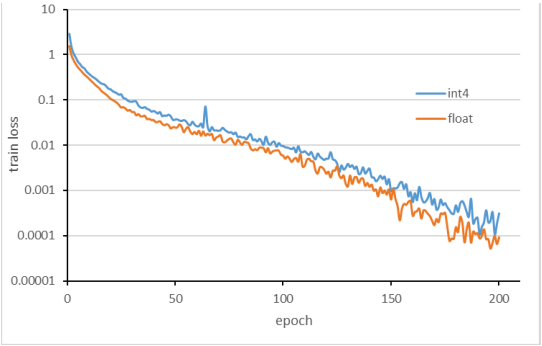

Figure 6: Training loss across epochs with 4-bit quantitative model and full-precision model on cifar10.

Table 1: Comparison with previous quantization methods on CIFAR-10. The bit width for weights (w), activations (a), bias (b) and step size(s) are given.

Method	#Bitsw/a/b/s	Acc(%)	Degradation
SR+DR[19]	Reference	93.05	-
	8/8	92.94	0.11
XNOR-net[20]	Reference	-	
	1/1	88.83	-
WAGE[21]	Reference	-	
	2/8	93.22	-
LR Net [22]	Reference	93.4	
	1/32	93.18	0.22
	2/32	93.26	0.14
RQ[17]	Reference	93.05	
	8/8	93.30	-0.25
	4/4	91.57	1.48
	2/2	90.92	2.07
LSFQ	Reference	92.53	
	4/4	92.86	-0.33
	3/3	92.41	0.12
	2/2	92.12	0.41
LSFQ	4/4/24/16	92.86	-0.33
	3/3/24/16	92.42	0.11
	2/2/24/16	92.12	0.42

We present the training loss of 4-bit quantitative model and full-precision model on cifar10, as shown in Figure 6. It can be seen that there is a small difference between the 4-bit model and full-precision model in terms of convergence speed, and the final loss of both can be less than 0.001. Therefore, compared with the full-precision model, the LSFQ quantitative model does not need more epochs to obtain similar results.

LSFQ-VGG7 network on CIFAR10 is compared with previous research work as shown in the table 1.

Because we quantify the first and the last layer, so the first layer of input data needs to quantify, so the input image data directly by line divided by 255 (equivalent to quantify to 8-bit), rather than in the same way as with conventional did normalized processing, so our training result is slightly lower than previous work results.

From table 1, we can see that our method is close to full accuracy results in 3-bit and 4-bit quantization, and the accuracy loss in 2-bit quantization is within 0.5%, which is relatively competitive with previous work.

To verify the performance of LSFQ on a network with fewer parameters, we manually reduced the number of filters for VGG7. The numbers of convolution layer filters of vgg7-tiny1 below are 64, 64,128,128,256,256. The numbers of convolution layer filters of vgg7-tiny2 is 32,32,64,64,128,128. In addition, the results show in table2.

Table 2: reduced network vgg7-tiny1 and vgg7-tiny2 on CIFAR10.

Method	#Bits w/a/b/s	Acc(%)	Degradation
VGG7-tiny1	Reference	91.43	
	4/4/24/16	91.59	-0.16
	3/3/24/16	91.75	-0.32
	2/2/24/16	89.72	1.72
VGG7-tiny2	Reference	89.23	
	4/4/24/16	89.42	-0.19
	3/3/24/16	89.03	0.02
	2/2/24/16	86.65	2.58

It can be seen that LSFQ can also have a good quantization accuracy on small size neural network, and the accuracy after 3-bit and 4-bit quantization is still comparable to the full accuracy.

B. Mobile-net v2 on CIFAR 100

Further, to verify that our approach is equally effective on an efficient network, we quantified the Mobile-net v2 [23] network and tested it on the CIFAR 100 dataset. The training method is the same as that in CIFAR 10 above. The results are shown in the table 3.

It can be seen that our method can still maintain a relatively high precision on efficient model, which is difficult to compress in other works. The accuracy loss of 4-bit quantization is controlled within 1%.

Table 3: LSFQ- Mobile-net v2 network on CIFAR100

Method	# Bits w/a	Top1	Top5
Mobile-net v2	Reference	71.01	91.18
	4/4	70.34	90.69
	3/3	69.84	90.55
	2/2	67.74	89.52

C. VGG7-tiny2 on FPGA

We deployed the 4-bit quantified VGG7-tiny2 network to the ultra96 V2 FPGA board. Moreover, in the speed, precision, resource consumption and other aspects of evaluation with other works, the results are shown in the table 4.

Due to the high accuracy of our hardware-aware quantization (LSFQ), our accelerator achieves a good balance between speed and accuracy. Although more DSPS are used in our design, DSP is the inherent resource of FPGA, and we think it is not a disadvantage to make full use of FPGA resources.

D. LSFQ on DAC-SDC

With the full quantification and hardware Design scheme, our team participated in the DAC-SDC2020 [24] and won the champion. The task of DAC-SDC 2020 was to implement the object detection algorithm with deep neural networks on the specified device. We needed to design the object detection algorithm and evaluate it on the ULTRA 96 V2 FPGA board.

Table 4: Comparison of our low-precision integer neural network accelerator with other works

Accelerator	platform	kLUTs	DSP	BRAM	Power	FPS	BIT-width	Accuracy
[18]	Zynq XC7Z020	46.9	3	94	4.7	168.4	1-2b	88.54
FINN-R	Zynq XC7Z045	46.2	-	186	11.7	21.9k	1b	80.1
Ours	Zynq Zu3EG	43.9	360	150.5	5.6	**6.4k**	4b	**89.42**

We designed network structure similar to VGG as the backbone and YOLO V2 object detection method as the back-end. During the contest, we used a combination scheme, including Dorefa-net weight quantization and PACT activation quantization. After the contest, we further studied and proposed the LSFQ. The final implementation comparison of the two schemes is shown in table 5.

Table 5: Comparison the whole scheme on 2020 DAC-SDC

Method	# Bits w/a	IoU	FPS	Power(W)
Reference	32/32	0.701	-	-
Dorefa+PACT	4/4	0.656	212.7	6.65
LSFQ	4/4	**0.679**	215.4	6.65

It can be seen that the accuracy (IoU) has been significantly improved by using the LSFQ, which further verify the hardware awareness and practicability of our quantization method.

VI. CONCLUSION

In this paper, we propose a new method, which is a learnable parameter soft clipping full integer quantization, and give a scheme to quantified the entire network to integer. In addition, the BN layer is fused to reduce unnecessary computing overhead, and an efficient computing structure is designed for the accelerated calculation of our quantized neural network algorithm. From the experimental results, our quantization algorithm has good performance on various models, which test on CIFAR10, CIFAR100. For 3-bit and 4-bit quantization, its accuracy loss is less than 1%, which is competitive with some previous works. In addition, the hardware acceleration has achieved good precision and speed. At present, our method loses some accuracy in efficient network with 2-bit quantization, and the quantization bit width used for bias and quantization step size is large. Therefore, in the future, we will deeply explore the low-bit quantization and low-bit quantization of bias and quantization step size of efficient network.

REFERENCES

[1] N. P. Jouppi, C. Young, N. Patil et al., "In-datacenter performance analysis of a tensor processing unit," in Proceedings of the 44th Annual International Symposium on Computer Architecture, 2017, pp. 1–12.

[2] S. K. Esser, P. A. Merolla, J. V. Arthur et al., "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing," in Proc Natl Acad Sci U S A, 2016.

[3] J. Qiu, J. Wang, S. Yao et al, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in Field Programmable Gate Arrays, 2016, pp. 26-35.

[4] B. Li, L. Liu, Y. Jin et al., "Designing Efficient Shortcut Architecture for Improving the Accuracy of Fully Quantized Neural Networks Accelerator," in 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC). 2020.

[5] K. Wang, Z. Liu, Y. Lin et al., "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," in Computer Vision and Pattern Recognition, 2019: 8612-8620.

[6] A. Gholami, K. Kwon, B. Wu et al., "SqueezeNext: Hardware-Aware Neural Network Design," in Computer Vision and Pattern Recognition, 2018, pp. 1638-1647.

[7] X. Zhao, Y. Wang, X. Cai et al., "Linear Symmetric Quantization of Neural Networks for Low-precision Integer Hardware," in International Conference on Learning Representations, 2020.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in ICLR, 2016.

[9] E. Park, J. Ahn, S. Yoo et al., "Weighted-Entropy-Based Quantization for Deep Neural Networks," in computer vision and pattern recognition, 2017, pp. 7197-7205.

[10] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in Deep Learning and Unsupervised Feature Learning Workshop, NIPS, 2011.

[11] B. Jacob, S. Kligys, B. Chen, et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in Computer Vision and Pattern Recognition, 2018, pp. 2704-2713.

[12] S. Zhou, Y. Wu, Z. Ni et al., "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," arXiv: Neural and Evolutionary Computing, 2016.

[13] Y. Bengio, N. Leonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," arXiv: Learning, 2013.

[14] J. Choi, "PACT: Parameterized clipping activation for quantized neural networks," arXiv: Computer Vision and Pattern Recognition, 2018.

[15] S. K. Esser, J. L. Mckinstry, D. Bablani et al., "LEARNED STEP SIZE QUANTIZATION," in International Conference on Learning Representations, 2020..

[16] M. Blott, T. B. Preuser, N. J. Fraser et al., "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," ACM Transactions on Reconfigurable Technology and Systems, 2018.

[17] C. Louizos, M. Reisser, T. Blankevoort et al., "Relaxed Quantization for Discretized Neural Networks," in International Conference on Learning Representations, 2019.

[18] L. Jiao, C. Luo, W. Cao et al., "Accelerating low bit-width convolutional neural networks with embedded FPGA," in 2017 27th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2017.

[19] P. Gysel, J. Pimentel, M. Motamedi et al., "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, 2018.

[20] M. Rastegari, V. Ordonez, J. Redmon et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in European Conference on Computer Vision, 2016, pp. 525-542.

[21] S. Wu, G. Li, F. Chen et al., "Training and Inference with Integers in Deep Neural Networks," 2018.

[22] O. Shayer, D. Levi, E. Fetaya et al., "Learning Discrete Weights Using the Local Reparameterization Trick," in International Conference on Learning Representations, 2018.

[23] M. Sandler, A. Howard, M. Zhu et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Computer Vision and Pattern Recognition, 2018.

[24] DAC SDC 2020.https://dac.com/content/2020-system-design-contest (2020-07-26)