IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received December 2, 2021, accepted December 21, 2021, date of publication December 27, 2021, date of current version December 31, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3138756*

# DoubleQExt: Hardware and Memory Efficient CNN Through Two Levels of Quantization

**JIN-CHUAN SEE[1], HUI-FUANG NG[1], HUNG-KHOON TAN[1], (Member, IEEE), JING-JING CHANG[1], WAI-KONG LEE[2], (Member, IEEE), AND SEONG OUN HWANG[2], (Senior Member, IEEE)**
[1]Faculty of Information and Communication Technology (FICT), Universiti Tunku Abdul Rahman, Kampar, Petaling Jaya 31900, Malaysia
[2]Department of Computer Engineering, Gachon University, Seongnam 13120, South Korea

Corresponding authors: Hui-Fuang Ng (nghf@utar.edu.my) and Seong Oun Hwang (sohwang@gachon.ac.kr)

This work was supported in part by the Ministry of Higher Education (MOHE), Malaysia, through the Fundamental Research Grant Scheme, under Grant FRGS/1/2021/ICT02/UTAR/02/3; and in part by the Universiti Tunku Abdul Rahman Research Fund (UTARRF) under Project IPSR/RMC/UTARRF/2019-C1/L06. The work of Wai-Kong Lee was supported by the Brain Pool Program through the National Research Foundation of Korea (NRF) by the Korean Government through the MSIT under Grant 2019H1D3A1A01102607. The work of Seong Oun Hwang was supported by the Ministry of Science and ICT (MSIT), South Korea, under the High-Potential Individuals Global Training Program supervised by the Institute of Information and Communications Technology Planning and Evaluation (IITP), under Grant 2021-0-01532.

**ABSTRACT** To fulfil the tight area and memory constraints in IoT applications, the design of efficient Convolutional Neural Network (CNN) hardware becomes crucial. Quantization of CNN is one of the promising approach that allows the compression of large CNN into a much smaller one, which is very suitable for IoT applications. Among various proposed quantization schemes, Power-of-two (PoT) quantization enables efficient hardware implementation and small memory consumption for CNN accelerators, but requires retraining of CNN to retain its accuracy. This paper proposes a two-level post-training static quantization technique (DoubleQ) that combines the 8-bit and PoT weight quantization. The CNN weight is first quantized to 8-bit (level one), then further quantized to PoT (level two). This allows multiplication to be carried out using shifters, by expressing the weights in their PoT exponent form. DoubleQ also reduces the memory storage requirement for CNN, as only the exponent of the weights is needed for storage. However, DoubleQ trades the accuracy of the network for reduced memory storage. To recover the accuracy, a selection process (DoubleQExt) was proposed to strategically select some of the less informative layers in the network to be quantized with PoT at the second level. On ResNet-20, the proposed DoubleQ can reduce the memory consumption by 37.50% with 7.28% accuracy degradation compared to 8-bit quantization. By applying DoubleQExt, the accuracy is only degraded by 1.19% compared to 8-bit version while achieving a memory reduction of 23.05%. This result is also 1% more accurate than the state-of-the-art work (SegLog). The proposed DoubleQExt also allows flexible configuration to trade off the memory consumption with better accuracy, which is not found in the other state-of-the-art works. With the proposed two-level weight quantization, one can achieve a more efficient hardware architecture for CNN with minimal impact to the accuracy, which is crucial for IoT applications.

**INDEX TERMS** Convolutional neural network, quantization, Internet of Things, deep learning, field programmable gate array.

## I. INTRODUCTION

Recently, the use of convolutional neural network (CNN) in IoT applications is becoming popular, which were demonstrated through a series of recent works ( [1]–[5]). IoT sensors are battery operated, placed at remote area and are expected to operate over a long duration without replacing the battery. This implies that low energy consumption is a critical design factor. Conventional CNN computation using Graphical Processing Unit (GPU) is not ideal in such scenario, as it leads to high power consumption. As such, efforts have been made to implement dedicated CNN accelerators using Field Programmable Gate Array (FPGA) to achieve a more energy efficient solution. However, CNN

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaolong Li.

computation involves floating-point operations that are slow and complex when implemented on FPGA. Due to this reason, techniques to quantize the CNN network into integer values are being researched recently. This allows the CNN computation to be performed in integer domain, which is faster and more area efficient for FPGA implementation.

In general, quantization of CNN can be performed through retraining (*e.g.,* [8]–[11]) or post-training static quantization (*e.g.,* [12], [13]). The retraining methods are known to recover most of the accuracy caused by quantization, but requires retraining of CNN. However, retraining activities involve additional cost (infrastructures with GPU) and time. Moreover, due to privacy issues, we may not always able to access the original training dataset to perform retraining. This is especially true for training data of medical related applications [6], [7] and user data for recommendation system. Hence, the retraining method only suitable for scenario where the training dataset is available.

Post-training static quantization directly quantize a pre-trained 32-bit floating-point CNN and does not require retraining; our work follows this direction. Post-training quantization is a more practical and generic solution as it does not involve the training process with original dataset. This technique is able to reduce the model size, retain the accuracy to an acceptable level ( [12], [13]). For instance, Wu *et al.* [13] proposed to quantize ResNet-50 to 7-bit integer, which reduces the classification accuracy only by 0.16%. These proposed techniques achieve low degradation in accuracy, but they require integer multipliers, which is usually mapped to Digital Signal Processing (DSP) units that are limited in number in FPGA.

In recent years, several works ( [14], [15]) were shown to perform $log_2$ (a.k.a power-of-two (PoT)) weight quantization, whereby the CNN model memory size can be greatly reduced. At the same time, the PoT weight quantization technique allows integer multipliers to be replaced by shifters to lower the complexity for CNN hardware implementation. This is because when the multiplicand is in PoT form, the multiplication can be performed in simple shift process without using a costly multiplier. However, their technique involves retraining in order to retain a suitable accuracy level.

This work proposes a two-level weight quantization scheme which achieves several advantages compared to state-of-the-art works: no-retraining, low accuracy degradation, low memory consumption and low hardware complexity. Experiments on quantized network accuracy of ResNet-20 and ResNet-56 were performed for CIFAR10 dataset classification. To cater for real-world application, ImageNet dataset classification accuracy were assessed for the quantized ResNet-18 and ResNet-34 using our proposed quantization scheme. Hardware resource utilization were also assessed by implementing the hardware design for our proposed scheme on Artix-7 and ZCU102 Xilinx FPGA. The contributions of this paper are summarized as follows:

1) DoubleQ, a two-level post-training static weight quantization technique, is proposed. DoubleQ first quantizes the CNN weights into 8-bit, and then further quantizes it into the PoT form. This allows the processing element (PE) to be implemented using lower complexity shifters rather than the costly DSP units.
2) Since DoubleQ quantizes the CNN weights into PoT, allowing them to be multiplied using shifter, only their PoT exponents are needed for the hardware. This allows a more compact storage for the PoT weights as the exponents can be encoded using lesser bitwidth (Refer to Section II-B).
3) To recover the accuracy, DoubleQExt, a mixed 8-bit and PoT weight quantization scheme, is proposed to perform PoT quantization only on less informative layers. In return, it requires a heterogenous core for hardware implementation to support 8-bit multiplication and shifting operation. This approach slightly trades-off the hardware area for a better accuracy compared to DoubleQ, but still consumes lesser memory for storage as compare to 8-bit quantization. It can be controlled to quantize lesser or more layers in PoT quantization; flexible trade-off between lower memory consumption and better accuracy of the target CNN. This is a feature which is not found in existing PoT weight quantization schemes.
4) For ResNet-20, DoubleQ saves 37.50% of memory compared to a single level of 8-bit weight quantization, with 7.28% accuracy degradation. With DoubleQExt, the accuracy degradation is reduced to only 1.19% and 23.05% memory reduction achieved compared to a single level 8-bit weight quantization.
5) Experimental result shows that our proposed scheme achieves 1% less degradation than the state-of-the-art PoT scheme [16] with comparable memory compression rate.

Table 1 summarizes the brief description on some of the frequently used quantization schemes in this paper.

Our paper is organized as follows: Section II presents the background on quantization and the motivation of this work. Section III presents our methodology on DoubleQ. Section IV discusses our experimental result and comparison with related work. Lastly, we conclude our work in Section V.

## II. BACKGROUND AND MOTIVATION OF DoubleQ

### A. QUANTIZATION OF CONVOLUTIONAL NEURAL NETWORKS (CNN)

Quantization refers to the transformation process that converts the 32-bit floating-point into low precision integer values. In general, it scales down the storage size for the input feature maps and kernel maps required to perform convolution, reduces overall memory transaction required between on-chip and off-chip memory storage, and further resolves the exponential growth in size of the computation data for CNN as the network goes deeper. At least 4× of memory reduction is expected when 32-bit values are quantized into 8-bit. At the same time, inference speed can also be improved. For instance, an 8-bit integer multiplier

**TABLE 1.** Important quantization scheme and terminologies in this paper.

| Quantization Scheme | Description |
|---|---|
| Quant8 | Perform 8-bit post training quantization on pretrained CNN. |
| DoubleQ | 1) Perform 8-bit post training quantization on pretrained CNN. 2) Then, quantize all layers of the 8-bit CNN again to 8-bit representable Power-of-Two (PoT) representable values. |
| DoubleQExt | 1) Perform 8-bit post training quantization on pretrained CNN. 2) Then, selectively quantize less informative layers of the 8-bit CNN to 8-bit representable Power-of-Two (PoT) representable values. (Refer to Section III-B) 3) Layers that are not selected remain in 8-bit. |

is much faster than its 32-bit floating-point counterpart. Some existing quantization techniques require retraining of the network (LQ-Nets [9], [17]), which is not flexible in many occasions. Hence, this work only focuses on post-training static quantization that does not require retraining. The quantization of weights is described below [18], [19]:

$$q_{int} = round(float \div scale + offset) \quad (1)$$

$$scale = (float_{max} - float_{min}) \div (q_{max} - q_{min}) \quad (2)$$

$$offset = q_{min} - (float_{min} \div scale) \quad (3)$$

In Eq. 1, floating-point (*float*) values are quantized into integers ($q_{int}$) by dividing it with a *scale*, and adding the *offset* in the respective layer. Eq. 2 shows how *scale* is calculated using the maximum and minimum values from the original floating-point parameter distribution in the respective layer. $q_{max}$ and $q_{min}$ is the maximum and minimum value of the desired precision in integer (e.g., $q_{max} = 127$ and $q_{min} = -128$ for signed 8-bit quantization). Eq. 3 shows the equation to calculate *offset*, an integer value that represents the value zero of floating-point before quantization with respect to the targeted quantized range. By employing these three equations, 32-bit floating-point can be quantized into 8-bit integer with little to no accuracy degradation. This enables floating-point multipliers to be replaced with integer multipliers, which are much faster and smaller in chip area.

### B. MOTIVATION: WHY ANOTHER LEVEL OF QUANTIZATION?

Power-of-Two (PoT) quantization has been explored in several previous weight quantization works ([14]–[16]) and the quantized results on floating point weights are widely reported. Central to this idea is the observation that PoT multiplication can be easily achieved through bit-shifting whereby the costly multipliers are replaced by simple shifting hardware. However, recent work on PoT quantization is performed directly on floating-point weights ([14], [15]) and typically requires retraining of the network to recover the accuracy. We aim to achieve retrain-less PoT weight quantization, without degrading the accuracy severely. As such, this work aims to explore PoT weight quantization from integer domain, instead of floating-point domain.

Trained network weights are real numbers that typically lie within the range [−1,1] and exhibit normal distribution [15]. Performing PoT weight quantization on these floating-point numbers would result in power-of-two number with negative exponents (see Table 2). In base-2, multiplication between activation and PoT weight with negative exponent due to recent year PoT weight quantization scheme [14]–[16] is essentially a division operation, which is performed through right shift. Designing a hardware module to handle this truncation incurs more hardware consumption, which defeats the original purpose of replacing multipliers with shifters.

**TABLE 2.** Comparison of PoT quantization on floating-point weights vs 8-bit quantized weights.

| $q_{pot} = sign(float) * 2^{round(log_2 \lvert float \rvert)}$ | | | |
|---|---|---|---|
| *float* | -0.5 | 0.1356 | 0.0378 |
| $q_{pot}$ | $-2^{-1}$ | $2^{-3}$ | $2^{-5}$ |
| $q_{pot} = sign(int) * 2^{round(log_2 \lvert int \rvert)}$ | | | |
| *int* | -8 | 30 | 59 |
| $q_{pot}$ | $-2^{3}$ | $2^{5}$ | $2^{6}$ |

By leveraging the retrain-less benefit of post-training static weight quantization, this paper proposes DoubleQ, a two-level weight quantization scheme. It first performs post-training static 8-bit quantization on the network weight, followed by another level of PoT weight quantization. After two levels of quantization, the resulting PoT values are in positive exponents, enabling multiplication in base-2 using left-shifting, which is very efficient for hardware implementation. This also greatly improves the speed performance and reduces the memory consumption (network weights size). To implement this in hardware using shifters, the PoT weights can be converted to their exponent form to further reduce the storage space. Since weight is available offline, the PoT weight quantization and its conversion to encoded 5-bit format can be performed before executing the model. Hence, no weight quantization overhead will be incurred during hardware execution. Figure 1 shows a sample PoT quantization from 8-bit integer and its storage in 5-bit encoded format. However, since 5-bit is not a conventional datatype in existing computer system, we propose to store the encoded 5-bit weight using datatype *unsigned short int*. At each 16-bit location, 5-bit encoded weight will be stored in a pack of three. This allows a more compact storage compared to storing 8-bit weight. The comparison of 8-bit and 5-bit weight storage is illustrated in Fig 2.

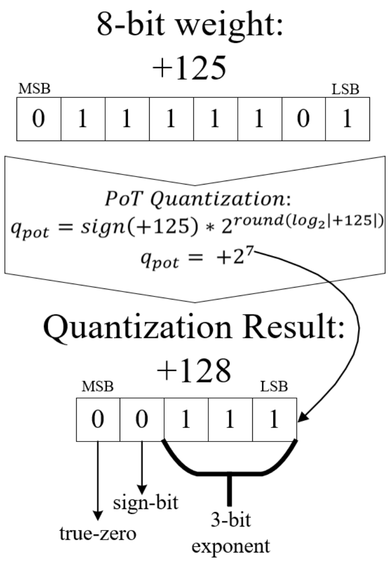


**FIGURE 1.** Example of PoT quantization on 8-bit integer to 5-bit PoT weight. The PoT weight is converted to encoded format for compact storage.

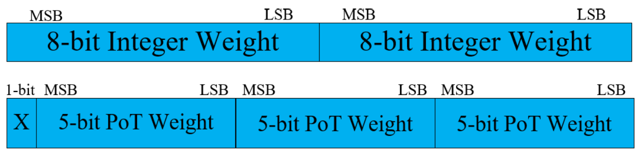


**FIGURE 2.** Comparison between 8-bit integer weight and 5-bit PoT weight storage using *unsigned short int* datatype. "X" is a 1-bit don't-care, and will be discarded during computation.

### C. RELATED WORKS

Efficient CNN hardware implementation has been actively research, to exploit useful applications of CNN on embedded devices. However, CNN in general involves heavy computation of floating-point multiplication and accumulation (MAC), which requires the use of GPU. Due to this reason, Li *et al.* [20] introduced a high throughput and low power hardware architecture on FPGA, which is more energy efficient compared to GPU. In terms of FPGA implementation, floating-point number multiplication is usually mapped to Digital Signal Processing (DSP) units, which are limited on FPGA. Hence, the previous works that proposed CNN architecture implementations on FPGA, usually focuses on how to reduce hardware resource consumption for the floating-point MAC operations.

In the work by Wang *et al.* [21], the authors proposed to compute CNN convolution using parallel Fast Finite Impulse Response Algorithm (FFA), an algorithm for digital signal processing. Their approach involves expressing the floating-point input feature map and weight parameter as time domain term, and computed efficiently using the proposed FFA algorithm. As reported in [21], the FFA algorithm was able to reduce 33% multiplications from total computation in CNN layers with $3 \times 3$ kernels. Another approach in work by Zhang *et al.* [22] however, instead of reducing number of multiplications, they proposed to overlap multiplication operations using lesser DSP units. Zhang *et al.* [22] proposed customized 9-bit floating-point format, as opposed to conventional 32-bit floating-point format. The reduced floating-point format enables them to overlap four floating-point multiplications using only 2.25 DSP slices, as opposed to a single conventional 32-bit floating-point multiplier that requires at least 18 DSP slices. While these work ( [21], [22]) has shown that it is possible to efficiently implement floating-point operations for CNN in FPGA, it is still relatively less efficient to be implemented as opposed to integer hardware.

It is worth noticing, however, in recent years, various quantization solution have been proposed, specifically, post-training static quantization [23]–[25] that does not involve retraining of the neural network. In the work by Fang *et al.* [23], the authors proposed to split the weight distribution into multiple regions. For each region, the weights are then quantized with their respective scaling factors to convert to their respective integer ranges. However, owing to the requirement of the proposed scheme, each region of the weight distribution represents a separate computation path, due to the differences in their scaling factors. This requirement, as commented by the authors [23], specified that at least three or more accumulators are required based on the number of regions the weight distribution is split into. The number of accumulators (at least three) tied to each multiply and accumulate (MAC) processing element (PE) might require more hardware resource for implementation, not to mention that existing CNN accelerators usually implements large number of PE in parallel to achieve high performance computation. In the work by Kim *et al.* [24] and Cai *et al.* [25], they proposed efficient search algorithm to find the optimal bitwidth for each of the layer weights. This scheme results in mixed precision for the network weights, where each layer might be represented using a different bitwidth. While the mixed-precision scheme is good to reduce overall network memory size for storage, the mixed data bitwidth could cause overhead during hardware access to decode from its compacted storage format. Furthermore, due to multiple bitwidth requirements between layers, a universal PE would be needed to support efficient computation of various bitwidth. This might not reduce the hardware resource implementation even though the data bitwidth has been reduced.

As such, supported by breakthrough of integer quantization for CNN, integer implementation for CNN in FPGA can also be seen in recent years work. While integer quantization reduces hardware implementation complexity for CNN, it does not reduce the computation load as compared to FP domain. To maintain performance with limited number of DSP, Lee *et al.* [26] performs algorithmic improvement on CNN convolution computation to optimize DSP utilization. In the work by Lee *et al.* [26], the Double MAC architecture was proposed and implemented on the off-the shelf DSP unit in FPGA. A typical DSP unit on FPGA performs a single multiplication between 2 input operands at every cycle.

The Double MAC technique proposed trades-off extra logic to be designed on top of the DSP, but enables 2 multiplications to take place between 3 operands. Their approach enables 2x throughput improvement for the convolutional layer.

All works ( [21], [22], [26]) mentioned above proposes efficient techniques that emphasizes on reducing resource requirement for MAC operations, but they are still highly dependent on DSP units. On FPGA, not only the DSP are limited, they also consume large area, which can be a limitation to some applications that need small hardware area.

While multiplication operation is inevitable in CNN, it is worth noticing that, multiplier-less solution can be seen in several work, by replacing multiplications using shifters. This was made possible due to quantization to base-2 domain [16] or algorithmic modification [28]. In the work by Xu *et al.* [16], their proposed scheme quantizes weights into a mix of base-2 and base-$\sqrt{2}$. However, their proposed scheme quantizes weight directly from the floating-point domain. As discussed in Section II.B, the resultant weight will be in PoT with negative exponents, which can be achieved as through right shifting in base-2 domain. While efficient to be implemented as hardware, there is no guarantee that the division in PoT would not yield any remainder and could cause value truncation. This effect might add another level of quantization error, on-top of the already induced quantization error due to constraining full precision 32-bit to narrower precision floating-point value, so that it is representable in base-2 domain.

Besides quantization, there are some other recent works that focus on developing efficient hardware architecture through hardware/software co-design approach. One of the representative works along this direction is from You and Wu [27], wherein a framework was proposed to optimize the sparse CNN and implemented on FPGAs.

Another interesting work was presented by Parmar and Sridharan [28], their proposed technique modifies the underlying CNN MAC operation between input feature map and weight parameters. The weight parameters are substituted with trigonometric functions, which enables the CNN MAC operation to be performed using CORDIC algorithm. When translated into hardware, the CORDIC algorithm requires only adders, shifters and multiplexers, as opposed to multipliers in conventional MAC operation. However, the proposed CORDIC algorithm when executing in hardware, requires a minimum of 3 clock cycle to produce one product term, as opposed to conventional shifting or multiplication operation that can be achieved within 1 clock cycle. The proposed scheme [28] trades off computation speed for efficient CNN FPGA implementation.

## III. THE PROPOSED DoubleQ SCHEME

This section describes the proposed post-training static quantization that does not require any form of retraining. We first introduce the proposed DoubleQ and discuss its limitations. Then, we present a technique to improve the accuracy of DoubleQ, which is named as DoubleQExt.

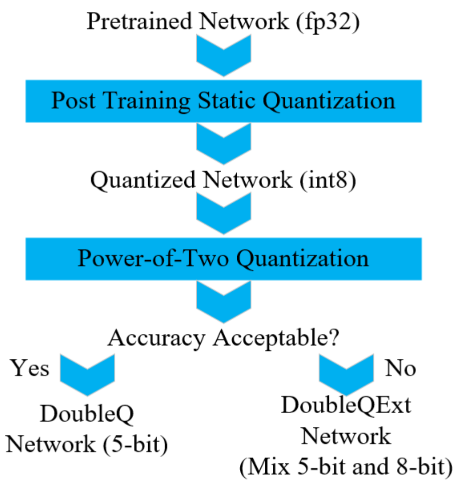

Pretrained Network (fp32)
Post Training Static Quantization
Quantized Network (int8)
Power-of-Two Quantization
Accuracy Acceptable?
Yes
No
DoubleQ Network (5-bit)
DoubleQExt Network (Mix 5-bit and 8-bit)

**FIGURE 3.** DoubleQ and DoubleQExt quantization process.

### A. OVERVIEW OF DoubleQ: TWO LEVELS OF QUANTIZATION

The proposed DoubleQ scheme (Refer to Figure 3) goes through two levels of quantization. A pretrained 32-bit floating-point CNN is first quantized into an 8-bit integer CNN. The 8-bit quantization is applied to both weight and activation. However, unlike weight that can be quantized offline, quantization parameter for the activation is obtained by inferencing on a small set of calibration data (extracted from the test dataset). Then, activation is quantized to 8-bit on the fly using these precomputed quantization parameters. For the weight, another level of quantization using the $log_2$ function is applied to the 8-bit integer weights to obtain PoT quantization. The resulting weights from these two levels of quantization are then used in the inference process.

To evaluate the effect of DoubleQ, a preliminary experiment was conducted on MNIST and CIFAR10 dataset for classification tasks. For MNIST dataset, we used a pretrained network adopted from [29]; for CIFAR10, ResNet-20 and ResNet-56 networks from [30] were used. All networks are pretrained in 32-bit floating-point (Float). These networks are first quantized into 8-bit (Quant8), followed by PoT quantization on the convolutional layers. The results are shown in Table 3.

**TABLE 3.** Top-1 accuracy and memory requirement comparison between each scheme.

| Network | Scheme | Top-1 Accuracy (Degradation)† | Memory (Reduction) |
|---|---|---|---|
| MNIST | Float | 98.24% | 21.00kB |
| | Quant8 | 98.23% (-0.01%) | 5.25kB (75.0%) |
| | DoubleQ | 98.33% (+0.09%) | 3.28kB (84.4%) |
| CIFAR10 (ResNet-20) | Float | 92.96% | 1.08MB |
| | Quant8 | 92.28% (-0.68%) | 270.26kB (75.0%) |
| | DoubleQ | 85.00% (-7.96%) | 168.91kB (84.4%) |
| CIFAR10 (ResNet-56) | Float | 94.44% | 3.4MB |
| | Quant8 | 93.92% (-0.52%) | 850.86kB (75.0%) |
| | DoubleQ | 90.77% (-3.67%) | 531.79kB (84.4%) |

† *"Degradation"* throughout this paper refers to accuracy difference between the quantized model and original model. (*e.g.*, *DoubleQ - Float*, *DoubleQ - Quant8*)

From Table 3, we can see that DoubleQ does not degrade the accuracy of MNIST dataset classification. However, its memory consumption is reduced by 84.4% when compared to Float. This memory reduction is achieved by storing DoubleQ weights in exponent format, which is representable within 5-bit range. For CIFAR10, classification using ResNet-20 with DoubleQ suffers 7.28% accuracy degradation compared to Quant8. However, the larger ResNet-56 network accuracy degradation is only at 3.67%. Similar to MNIST dataset classification, 84.4% memory reduction is achieved as well for both ResNet-20 and ResNet-56 when compared to Quant8. These results show that DoubleQ is a promising solution to reduce the memory consumption at the expense of accuracy degradation. This technique does not require any retraining compared to existing works [14], [15]. DoubleQ simplifies the implementation of CNN accelerator, as it only requires shifter, which is much resource efficient for hardware implementation compared to 32-bit floating-point and 8-bit multiplier. However, this level of accuracy degradation might be undesirable for some critical applications that could only tolerate minimal classification error. To cater for such requirement, we propose another strategy to improve the accuracy of DoubleQ, which is presented subsequently.

### *B. DoubleQExt: RECOVERING THE ACCURACY*

To recover accuracy degradation introduced in DoubleQ, DoubleQExt (Refer to Figure 3) is proposed. DoubleQExt only performs two-level quantization on selected convolutional layers, in contrast to DoubleQ that affects all layers. The selection criteria in DoubleQExt are based on a *Minimum Recovery Threshold* (M.R.T), which is the targeted recovery percentage from the accuracy degradation experienced by applying DoubleQ.

**Algorithm 1** Pseudocode of DoubleQExt

```
1: Set all layers to double quantization (DoubleQConv)
2: Evaluate initial accuracy
3: Initialize critical layer to 0
4: do
5:     for j in layer list do
6:         Convert layer[j] to Quant8Conv
7:         Evaluate latest accuracy
8:         Compare latest accuracy with initial accurracy
9:         if recovered accuracy is highest then
10:            Revert previous critical layer to DoubleQConv
11:            Record j as critical layer
12:        end if
13:    end for
14:    Remove critical layer from layer list
15: while accuracy recovery < M.R.T
```

Algorithm 1 shows detailed steps to select the best combination of convolutional layers to apply double quantization (*DoubleQConv*), which eventually incurs the least accuracy degradation. Hence, its main purpose is to find the critical layers that should be remained in 8-bit precision.

Firstly, *DoubleQConv* is applied on all convolutional layers to find the initial accuracy (*line 1*∼*2*) before starting the selection process (*line 4*∼*15*). Applying *DoubleQConv* on all layer's results in a drop in accuracy. The following steps recover some of the accuracy loss by identifying the critical layers to be reverted to a less aggressive single quantization (*Quant8Conv*). During the selection process, all existing convolutional layer (*line 5*) will be assessed, wherein *Quant8Conv* (*line 6*) is applied. Following this, its classification accuracy is evaluated (*line 7*) and compared with the initial accuracy (*line 8*), to calculate the recovered accuracy. If the currently recovered accuracy is the highest (*line 9*), the current layer will be selected as the new critical layer (*line 11*) and the previously selected layer will be reverted back to *DoubleQConv* (*line 10*). After all existing layers have been assessed, the selected critical layer is removed from the existing layer list (*line 14*) to prevent it from being assessed again in subsequent iteration. The process stops when the algorithm achieves the desired M.R.T (*line 15*).

**TABLE 4.** Performance recovered for ResNet-20 using DoubleQExt.

| M.R.T | Critical Layers | Accuracy Recovered (Degradation) | Total parameter size (Memory reduced) |
|---|---|---|---|
| 0% | N/A | 85.00% (-7.28%) | 168.91 kB (37.50%) |
| 50% | 0, 5, 8 | 89.09% (-3.19%) | 173.39 kB (35.84%) |
| 60% | 0, 5, 8, 19 | 90.20% (-2.08%) | 187.22 kB (30.73%) |
| 70% | 0, 5, 8, 17, 19 | 90.68% (-1.60%) | 201.04 kB (25.61%) |
| 80% | 0, 5, 8, 14, 17, 19 | 91.09% (-1.19%) | 207.95 kB (23.05%) |
| 90% | 0, 5, 8, 10, 14 to 20 | 91.56% (-0.72%) | 253.65 kB (6.15%) |

## IV. RESULT AND DISCUSSION

### *A. NETWORK PERFORMANCE ASSESSMENT*

Table 4 shows the performance recovered by applying Algorithm 1 (DoubleQExt) to ResNet-20 and evaluated on CIFAR10 for different values of M.R.T. "*Critical Layers*" are layers that are sensitive to DoubleQ and should be remained as 8-bit, to achieve better classification accuracy. "*0% M.R.T*" refers to DoubleQ ResNet-20 without the improvement from DoubleQExt algorithm. By only removing three layers (0, 5 and 8), at least 50% of accuracy was recovered. When DoubleQExt is applied with higher M.R.T, most of the accuracy can be recovered at the expense of lesser memory reduction. In other words, by mixing 8-bit and PoT quantization at different layers, the accuracy degradation is much lesser compared to applying DoubleQ at all layers.

Note that M.R.T can be used as a metric to select the desired accuracy and memory trade-off. Referring to ResNet-20, applications that requires high accuracy may choose minimum accuracy degradation (i.e., $\leq 2\%$) by setting M.R.T to 70% $\sim$ 90% with a higher memory consumption. On the other hand, one can sacrifice accuracy to achieve low memory consumption by choosing an M.R.T value of 50% $\sim$ 60%.

**TABLE 5. Performance on ImageNet DAtaset using DoubleQExt.**

| Network | Scheme | Top-1 | Top-5 | Memory |
|---|---|---|---|---|
| ResNet-18 | Float | 69.76% | 89.08% | 44.7MB |
| | DoubleQExt | 67.97% | 88.04% | 8.06MB |
| ResNet-34 | Float | 73.30% | 91.42% | 85.1MB |
| | DoubleQExt | 71.41% | 90.30% | 14.3MB |

This flexible trade-off feature has been shown to be very useful, which is not found in other state-of-the-art works.

To further assess the effectiveness of DoubleQExt, we performed experiment on two representative networks (ResNet-18 and 34 from [31]) for ImageNet classification task. From Table 4, it can be seen that, the DoubleQExt reduces the top-5 accuracy of ResNet-18 by 1.04% only, while achieving 82.0% of memory reduction. Similarly, in ResNet-34, the accuracy is degraded only by 1.12%, while achieving 83.2% of memory reduction. The memory size of ResNet-18 and 34 after DoubleQExt quantization is relatively small, which is suitable for FPGA implementations.

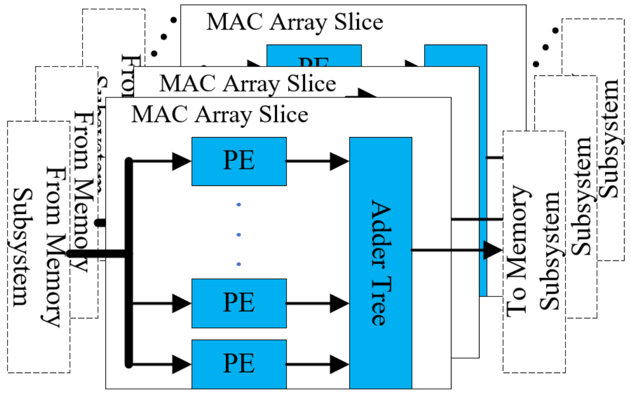


**FIGURE 4. Top-level view of proposed hardware architecture.**

## *B. HARDWARE IMPLEMENTATION ASSESSMENT*

Figure 4 illustrates the top-level view of the proposed hardware architecture. Multiple multiply-and-accumulate (MAC) array slices are implemented in parallel to increase computation throughput. In each slice, processing elements (PE) performs multiplication and the resultant product is accumulated using adder tree. Each PE is implemented using only LUT and FF, or only DSP. Three hardware schemes are implemented: conventional 8-bit quantization (Quant8); double quantize all layer (DoubleQ), mixed 8-bit and double quantization (DoubleQExt) with 80% M.R.T (Refer to Section IV-A). The PE internals differ between three hardware schemes: conventional 8-bit quantization (Quant8); double quantize all layer (DoubleQ), mixed 8-bit and double quantization (DoubleQExt) with 80% M.R.T (Refer to Section IV-A). Their respective PE implementation are shown in Figure 5.

For Quant8 scheme, the multipliers are implemented using DSP units. Whereas for DoubleQ and DoubleQExt scheme, their implementation mainly uses Look-Up Tables (LUT) and Flip-Flops (FF) only. The DoubleQ consists of a shifter that

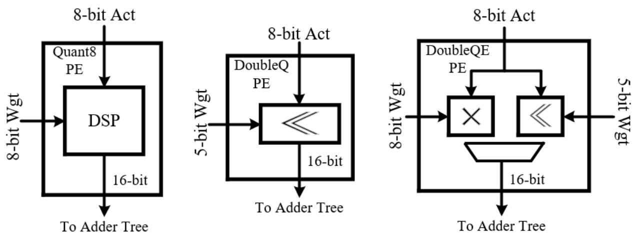


**FIGURE 5. Comparison of PE implementation based for different quantization scheme.**

takes in 8-bit inputs and 5-bit shift amount (PoT weight) and computes a 16-bit output. For DoubleQExt PE, it consists of 8-bit multiplier, a shifter and a multiplexer to select outputs between the shifter and multiplier. The multiplexer selects output based on configuration obtained from DoubleQExt algorithm as discussed in Section IV-A. While the number of components is more as compared to Quant8 and DoubleQ, DoubleQExt needs lesser hardware resources compared to Quant8, and achieved better accuracy compared to DoubleQ. These comparisons are further shown in Table 6.

**TABLE 6. Implementation result for ResNet-20 on Artix-7 (xc7a100tcsg324).**

| Scheme | Quant8 | DoubleQExt | DoubleQ |
|---|---|---|---|
| PE | 240 | 512 | |
| DSP (Equivalent LUT and FF)† | 240 (131280 LUT; 83280 FF) | N/A | |
| Non-DSP resource | 1530 LUT; 480 FF | 56768 LUT; 14336 FF | 27584 LUT; 14848 FF |
| Total LUT and FF | 132810 LUT; 83760 FF | 56768 LUT; 14336 FF | 27584 LUT; 14848 FF |
| Top-1 Accuracy (Degradation): | 92.28% | 91.09% (-1.19%) | 85.00% (-7.28%) |
| Total param. size (Mem. reduced): | 270kB | 208kB (23.05%) | 169kB (37.50%) |

† Equivalent resource if DSP is implemented using LUT and FF. Data obtained using Vivado IP generator.

Table 6 shows the comparison between each scheme when implemented for ResNet-20 on Artix-7 (xc7a100tcsg324), synthesized using Vivado 2017.2. As mentioned in Section I, the 8-bit integer multiplier in Quant8 was mapped into DSP by the synthesizer. This limits the design to only able to implement 240 PE, which is the maximum number of DSP available on Artix-7. This shows that, DSP limitation on an FPGA is a crucial design consideration for conventional 8-bit quantized CNN hardware implementation. To further compare Quant8 with DoubleQ and DoubleQExt, we calculated the equivalent Look-Up Tables (LUT) and Flip-Flops (FF) count used by the total number of DSP by Quant8. The equivalent LUT and FF count are obtained from synthesis report of Vivado IP generator by generating an equivalent multiplier size that follows the setting of a single DSP unit on Xilinx FPGA, which is a 25 x 18 multiplier.

For DoubleQ and DoubleQExt, we constrained the synthesizer to map into purely LUT and FF. This allows us to implement more PE as compared to Quant8 design.

However, DoubleQExt requires both 8-bit multiplier and shifter PE for mixed layer computation. Hence, DoubleQExt consumes more LUT compared to DoubleQ that only uses shifter-based PE. Similarly, DoubleQExt also consumes more memory compared to DoubleQ. However, DoubleQExt requires 57.2% LUT and 82.9% FF lesser than Quant8; it can also achieve a higher accuracy (91.09% vs 85.00%) compared to DoubleQ and lower memory consumption compared to Quant8.

**TABLE 7. Comparison with existing mixed precision scheme [25] and hardware design requirement.**

| | Cai et al. [25] | Our work (DoubleQExt) |
|---|---|---|
| Activation Bit-Width | 4, 6 and 8-bit | 8-bit |
| Weight Bit-Width | 2, 4 and 8-bit | 5 and 8-bit |
| Operation Requirement | 2, 4, 6 and 8-bit multiplications | 8-bit multiplication and shifting |

## C. COMPARISON WITH RELATED WORKS

Table 7 compares the existing mixed precision scheme with DoubleQExt. The work by Cai *et al.* [25] proposed a mixed precision of 2, 4, 6 and 8-bit solution. Their scheme requires a PE design that supports 4 different bit-widths computations. This introduces a more complicated design requirement, as the data path and control circuitry need to be able to perform scalable multiplication operation, on the fly. Such requirement would introduce more hardware resources during FPGA implementation. In contrast, our DoubleQExt scheme only needs to switch between two modes, which is multiplication or shifting. In comparison, DoubleQExt has a simpler requirement which is much more hardware efficient to be implemented in FPGA. It should be noted however, DoubleQExt is much more suitable for error-tolerant applications.

**TABLE 8. Comparison with floating point [22] and 8-bit integer [28] system.**

| | Zhang et. al. [22] | Parmar et. al. [28] | Our work (DoubleQExt) |
|---|---|---|---|
| FPGA | Kintex-7 | Virtex-5 | Artix-7 |
| Number of PE | 9 | 9 | 9 |
| Act/Wgt | 9-bit FP/ 9-bit FP | 8-bit INT/ 8-bit INT | 8-bit INT/ 8 and 5-bit INT |
| DSP (Equivalent LUT and FF)$^\dagger$ | 2.25 (1231 LUT; 781 FF) | N/A | N/A |
| Non-DSP resource | 1125 LUT; 576 FF | 1465 LUT; 519 FF | 1020 LUT; 120 FF |
| Total LUT and FF | 2356 LUT; 1357 FF | 1456 LUT; 519 FF | **1020 LUT; 120 FF** |

$^\dagger$ Equivalent resource if DSP is implemented using LUT and FF. Data obtained using Vivado IP generator.

In Table 8, DoubleQExt is compared with the floating-point system [22] and 8-bit integer system [28]. To ensure fair comparison, we synthesize our proposed DoubleQExt with the layout of nine PE, which is identical to [22] and [28]. Zhang *et al.* [22] proposed a 9-bit floating point system, which has lower hardware area than conventional 32-bit floating point system. Their implementation not only requires DSP unit, but also requires a lot of LUT and FF. The proposed DoubleQExt consumes 56.7% and 91.2% lesser LUT and FF respectively compared to [22]. Since DoubleQExt does not consume any DSP, it is much smaller in terms of hardware. Parmar and Sridharan [28] proposed a CORDIC based algorithm to replace the multiply and accumulate (MAC) required in CNN computation. DoubleQExt consumes 30.0% and 76.9% lesser LUT and FF respectively compared to [28]. Although they do not require DSP, unlike DoubleQExt, their technique needs retraining. In terms of memory consumption, [22] stores the network weights in 9-bit, while [28] stores them in 8-bit. DoubleQExt requires lesser memory consumption as the network weights are stored in a mixture of 8-bit and 5-bit size. This shows that DoubleQExt is more hardware and memory efficient compared to state-of-the-art works.

**TABLE 9. Comparison with Xu *et al.* [16].**

| | Xu et. al. [16] | Our Work | |
|---|---|---|---|
| FPGA | ZCU104 | ZCU102 | |
| Max Frequency | 200MHz | 250MHz | |
| PE | 1512 | 1024 | |
| GOPs | 604.8 | 512.0 | |
| Scheme | SegLog | DoubleQ | DoubleQExt |
| LUT | 113k | 69152 | 113184 |
| FF | Not reported | 28672 | 27648 |
| Network | ResNet-34 | | |
| Top-5 Float (Mem. size) | Not reported | 91.42% (85.1MB) | |
| Top-5 (Mem. size) | Not reported | 81.14% (13.3MB) | 90.30% (14.3MB) |
| Memory compressed | x6.4 | x6.4 | x6.0 |
| Degradation | -2.11% | -10.28% | **-1.12%** |

Table 9 compares the proposed solutions with SegLog, a state-of-the-art PoT scheme by Xu *et al.* [16]. SegLog performs a mix of $log_2$ and $log_{\sqrt{2}}$ quantization on the CNN weights. To ensure a fair comparison, we performed the DoubleQ and DoubleQExt experiment following the same experiment settings as performed by Xu *et al.* [16]. We use the same network, ResNet-34, to perform the same task, image classification, on the same dataset, which is ImageNet to ensure the validity of our comparison. While not mentioned, we believed their [16] quantized accuracy is measured against the benchmarking result of original ResNet-34 (fp32) on ImageNet classification. We performed our DoubleQ and DoubleQExt experiment and compare with the original fp32 ResNet-34 to calculate the accuracy degradation. We then compare the accuracy degradation from our experiment against the result of Xu *et al.* [16]. With the same experimental setting and benchmarking target, the relative difference between Xu *et al.* [16] and our accuracy degradation serves to prove the significance of our proposed scheme. DoubleQExt can achieve x6.0 memory reduction with 1.12% accuracy degradation, compared with the 32-bit floating-point ResNet-34. Compared to SegLog [16], DoubleQExt achieved 1% lesser degradation, but consumes 6.25% more memory. In terms of hardware implementation, the LUT consumption of DoubleQExt is similar to SegLog

(∼113k). However, if high accuracy is not mandatory, the proposed DoubleQ can save 38.8% LUT compared to SegLog. Furthermore, the memory size and accuracy can be trade-off flexibly in DoubleQExt; this is a useful feature not found in SegLog.

The use of FPGA based CNN accelerator is becoming a trend recently. This is also in line with the emerging trend in using FPGA based processor [32] for IoT applications. The proposed DoubleQExt technique is useful for IoT applications, in which the memory size and accuracy can be trade-off according to the requirements (i.e., high accuracy or low memory consumption).

## V. CONCLUSION AND FUTURE WORK

### A. CONCLUSION

This paper proposes DoubleQ, a post-training static quantization scheme for reduced CNN hardware implementation. DoubleQ performs two levels of quantization (8-bit and PoT quantization) on the CNN network to significantly reduce the memory consumption (37.50% further reduction from 8-bit quantization), in the expense of reduced accuracy. This enables the hardware implementation to replace multipliers with low complexity shifters. DoubleQExt, a strategic layer selection algorithm is also proposed to improve the accuracy of DoubleQ. As a result, the original DoubleQ ResNet-20 with 7.28% accuracy degradation (compare to 8-bit network) is at 1.19% only, while achieving 23.05% memory reduction after applying DoubleQExt. Comparison with the state-of-the-art CNN accelerator [16] further shows that DoubleQExt achieve 1% lesser degradation (with x6.0 times memory compression compared to 32-bit floating-point network) on ImageNet classification. DoubleQExt also allows flexible trade-off in memory size and accuracy, which is a distinctive feature that is not commonly found in existing works. This can be a promising solution for IoT applications, wherein sensor nodes implemented on FPGA may have different memory constraints and accuracy requirements.

### B. FUTURE WORK

The use of deep learning in IoT systems is limited due to the large memory consumption. A typical sensor node implemented using microcontroller, may only have a few megabytes (MB) of RAM storage. For instance, STM32F743 [33] is a high-end microcontroller used to build IoT sensor nodes, it only has 2MB of flash memory and 1MB of RAM storage to store all the firmware and data. Due to this reason, popular deep learning networks like ResNet-20 (1.08MB) and ResNet-56 (3.40MB) are almost impossible to be stored in such constrained devices. However, by applying the proposed DoubleQ (see Table 2), the networks are compressed down to less than 600 kilobytes (KB) only. This allows the use of such deep learning networks in IoT applications, in the expense of some accuracy lost, which is within the tolerance. Considering an FPGA-based sensor node, DoubleQExt does not utilize any DSP, hence it can be a more lightweight solution compared to the previous works (see Table 7). A practical implementation and evaluation of such system under actual IoT application scenario would be an interesting future direction.

## REFERENCES

[1] B. Y. Cai, R. Alvarez, M. Sit, F. Duarte, and C. Ratti, "Deep learning-based video system for accurate and real-time parking measurement," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7693–7701, Oct. 2019, doi: 10.1109/JIOT.2019.2902887.

[2] S. Khan, K. Muhammad, S. Mumtaz, S. W. Baik, and V. H. C. de Albuquerque, "Energy-efficient deep CNN for smoke detection in foggy IoT environment," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9237–9245, Dec. 2019, doi: 10.1109/JIOT.2019.2896120.

[3] G.-J. Horng, M.-X. Liu, and C.-C. Chen, "The smart image recognition mechanism for crop harvesting system in intelligent agriculture," *IEEE Sensors J.*, vol. 20, no. 5, pp. 2766–2781, Mar. 2020, doi: 10.1109/JSEN.2019.2954287.

[4] W. Liu, "Beach sports image detection based on heterogeneous multiprocessor and convolutional neural network," *Microprocessors Microsyst.*, vol. 82, Apr. 2021, Art. no. 103910, doi: 10.1016/j.micpro.2021.103910.

[5] H. Jiang, X. Li, and F. Safara, "IoT-based agriculture: Deep learning in detecting apple fruit diseases," *Microprocessors Microsyst.*, Aug. 2021, Art. no. 104321, doi: 10.1016/j.micpro.2021.104321.

[6] H. Bi, J. Liu, and N. Kato, "Deep learning-based privacy preservation and data analytics for IoT enabled healthcare," *IEEE Trans. Ind. Informat.*, early access, Oct. 8, 2021, doi: 10.1109/TII.2021.3117285.

[7] Y. Liu, Z. Ma, X. Liu, S. Ma, and K. Ren, "Privacy-preserving object detection for medical images with faster R-CNN," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 69–84, 2022, doi: 10.1109/TIFS.2019.2946476.

[8] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards less CNNs with low-precision weights," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–14.

[9] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *Computer Vision—ECCV*. Munich, Germany: Springer, 2018, pp. 373–390, doi: 10.1007/978-3-030-01237-3_23.

[10] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," Jul. 2018, *arXiv:1805.06085*.

[11] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Mach. Intell.*, vol. 3, pp. 675–686, Jun. 2021, doi: 10.1038/s42256-021-00356-5.

[12] T. Dinh, A. Melnikov, V. Daskalopoulos, and S. Chai, "Subtensor quantization for mobilenets," in *Proc. Embedded Vision Workshop, 16th Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 126–130.

[13] D. Wu, Q. Tang, Y. Zhao, M. Zhang, Y. Fu, and D. Zhang, "EasyQuant: Post-training quantization via scale optimization," Jun. 2020, *arXiv:2006.16669*.

[14] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," Mar. 2016, *arXiv:1603.01025*.

[15] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Mar. 2020, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=BkgXT24tDS.

[16] J. Xu, Y. Huan, B. Huang, H. Chu, Y. Jin, L.-R. Zheng, and Z. Zou, "A memory-efficient CNN accelerator using segmented logarithmic quantization and multi-cluster architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 6, pp. 2142–2146, Jun. 2021, doi: 10.1109/TCSII.2020.3038897.

[17] C. Y. Lo, F. C. M. Lau, and C.-W. Sham, "Fixed-point implementation of convolutional neural networks for image classification," in *Proc. Int. Conf. Adv. Technol. Commun. (ATC)*, Ho Chi Minh City, Vietnam, Oct. 2018, pp. 105–109, doi: 10.1109/ATC.2018.8587580.

[18] B. Jacob. (2017). *Gemmlowp: A Small Self-Contained Low-Precision GEMM Library*. [Online]. Available: https://github.com/google/gemmlowp

[19] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," Apr. 2020, *arXiv:2004.09602*.

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

[21] J. Wang, J. Lin, and Z. Wang, "Efficient hardware architectures for deep convolutional neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1941–1953, Nov. 2018, doi: 10.1109/TCSI.2017.2767204.

[22] Z. Zhang, D. Zhou, S. Wang, and S. Kimura, "Quad-multiplier packing based on customized floating point for convolutional neural networks on FPGA," in *Proc. 23rd Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2018, pp. 184–189, doi: 10.1109/ASPDAC.2018.8297303.

[23] 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*. Cham, Switzerland: Springer, 2020, pp. 69–86, doi: 10.1007/978-3-030-58536-5_5.

[24] Y. Kim, J. Lee, Y. Kim, and J. Seo, "Robust quantization of deep neural networks," in *Proc. 29th Int. Conf. Compiler Construct.*, San Diego, CA, USA, Feb. 2020, pp. 74–84, doi: 10.1145/3377555.3377900.

[25] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "ZeroQ: A novel zero shot quantization framework," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 13166–13175, doi: 10.1109/CVPR42600.2020.01318.

[26] S. Lee, D. Kim, D. Nguyen, and J. Lee, "Double MAC on a DSP: Boosting the performance of convolutional neural networks on FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 888–897, May 2019, doi: 10.1109/TCAD.2018.2824280.

[27] W. You and C. Wu, "RSNN: A software/hardware co-optimized framework for sparse convolutional neural networks on FPGAs," *IEEE Access*, vol. 9, pp. 949–960, 2021.

[28] Y. Parmar and K. Sridharan, "A resource-efficient multiplierless systolic array architecture for convolutions in deep networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 2, pp. 370–374, Feb. 2020, doi: 10.1109/TCSII.2019.2907974.

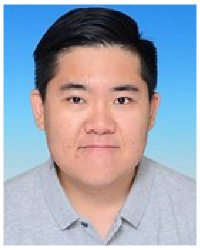
[29] G. Koehler. (Feb. 17, 2020). *MNIST Handwritten Digit Recognition in PyTorch*. [Online]. Available: https://nextjournal.com/gkoehler/pytorch-mnist

[30] Y. Li. (2020). APoT quantization. GitHub Repository. [Online]. Available: https://github.com/yhhhli/APoT_Quantization

[31] *Torchvision.models—Torchvision 0.11.0 Documentation*. Accessed: Jun. 15, 2021. [Online]. Available: https://pytorch.org/vision/stable/models.html

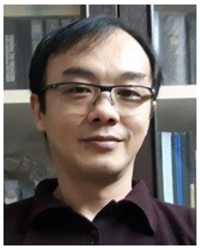
[32] W.-P. Kiat, K.-M. Mok, W.-K. Lee, H.-G. Goh, and R. Achar, "An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications," *Microprocess. Microsyst.*, vol. 73, Mar. 2020, Art. no. 102966, doi: 10.1016/j.micpro.2019.102966.

[33] *STM32H743/753 Microcontroller*. Accessed: Oct. 15, 2021. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32h743-753.html



**JIN-CHUAN SEE** received the master's degree in computer science from Universiti Tunku Abdul Rahman (UTAR), where he is currently pursuing the Ph.D. degree with the Faculty on Information and Communication Technology. His current research interest includes deep learning algorithm implementation on field programmable gate-array.

**HUI-FUANG NG** received the Ph.D. degree in biosystems and agricultural engineering from the University of Minnesota, USA, in 1996. He was a Software Engineer at PPT Vision Inc., MN, USA, from 1996 to 2003. He joined the Department of Computer and Information Science, Asia University, Wufong, Taiwan, from 2003 to 2013. He is currently with the Department of Computer Science, University Tunku Abdul Rahman (UTAR), Malaysia. He is also the Chairperson of the Centre for IoT and Big Data, UTAR. His research interests include image processing, computer vision, and machine learning.



**HUNG-KHOON TAN** (Member, IEEE) received the B.Eng. degree in computer engineering from Universiti Teknologi Malaysia, Malaysia, and the M.Phil. and Ph.D. degrees in computer science from the City University of Hong Kong, Hong Kong. He worked as a Test Development Engineer and a Senior Design Engineer with the Research and Development Center, Altera, Penang, Malaysia, from 1999 to 2004. He has been with the Department of Computer Science, Faculty of Information and Communication Technology (FICT), Universiti Tunku Abdul Rahman (UTAR), since 2011. His current research interests include action recognition in videos, image and video content analysis, and anomaly detection.

**JING-JING CHANG** received the bachelor's degree in chemical engineering and the Ph.D. degree in control and automation engineering from Universiti Putra Malaysia, in 2011 and 2016, respectively. She is currently an Assistant Professor with the Department of Computer and Communication Technology, Universiti Tunku Abdul Rahman.

**WAI-KONG LEE** (Member, IEEE) received the B.Eng. degree in electronics and the M.Eng.Sc. degree from Multimedia University, in 2006 and 2009, respectively, and the Ph.D. degree in engineering from University Tunku Abdul Rahman (UTAR), Malaysia, in 2018. From 2009 to 2012, he served as a Research and Development Engineer for several multinational companies, including Agilent Technologies (now known as Keysight), Malaysia. He served as an Assistant Professor and the Deputy Dean (research and development) for the Faculty of Information and Communication Technology, UTAR. He was a Visiting Scholar at Carleton University, Canada, in 2017, Feng Chia University, Taiwan, in 2016 and 2018, and OTH Regensburg, Germany, in 2015, 2018, and 2019. He is currently a Postdoctoral Researcher at Gachon University, South Korea. His research interests include cryptography, GPU computing, numerical algorithms, the Internet of Things (IoT), and energy harvesting. He served as a Reviewer for several international journals, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, in 2016 and 2017, IEEE SENSORS JOURNAL, IEEE INTERNET OF THINGS JOURNAL, from 2018 to 2021, and IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, from 2018 to 2021.

**SEONG OUN HWANG** (Senior Member, IEEE) received the B.S. degree in mathematics from Seoul National University, in 1993, the M.S. degree in information and communications engineering from the Pohang University of Science and Technology, in 1998, and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology, South Korea. He worked as a Software Engineer with LG-CNS Systems, Inc., from 1994 to 1996. He also worked as a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), from 1998 to 2007. He worked as a Professor with the Department of Software and Communications Engineering, Hongik University, from 2008 to 2019. He is currently a Professor with the Department of Computer Engineering, Gachon University. His research interests include cryptography, cybersecurity, and artificial intelligence. He is an Editor of *ETRI Journal*.

• • •