IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 1, 2020, accepted January 23, 2021, date of publication January 27, 2021, date of current version February 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054879

Zero-Centered Fixed-Point Quantization With Iterative Retraining for Deep Convolutional Neural Network-Based Object Detectors

SUNGRAE KIM AND HYUN KIM, (Member, IEEE)
Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea
Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Hyun Kim (hyunkim@seoultech.ac.kr)

This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korean Government (MSIT) (Development of AI Deep-Learning Processor and Module for 2,000 TFLOPS Server) under Grant 2020-0-01305, and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2019R1A6A1A03032119.

ABSTRACT In the field of object detection, deep learning has greatly improved accuracy compared to previous algorithms and has been used widely in recent years. However, object detection using deep learning requires many hardware (HW) resources due to the huge computations for high performance, making it very difficult to run real-time on embedded platforms. Therefore, various compression methods have been studied to solve this problem. In particular, quantization methods greatly reduce the computational burden of deep learning by reducing the number of bits used for weights and activation functions in deep learning. However, most of these existing studies targeted only object classification and cannot be applied to object detection. Furthermore, most of the existing quantization studies are based on floating-point operations, which requires additional effort when implementing HW accelerators. This paper proposes an HW-friendly fixed-point-based quantization method that can also be applied to object detection. In the proposed method, the center of the weight distribution is adjusted to zero by subtracting the mean of weight parameters before quantization, and the retraining process is iteratively applied to minimize the accuracy drop caused by quantization. Furthermore, while applying the proposed method to object detection, performance degradation is minimized by considering the minimum and maximum values of weight parameters of deep learning networks. When applying the proposed quantization method to representative one-stage object detectors, You Only Look Once v3 and v4 (YOLOv3 and YOLOv4), detection accuracy similar to the original networks (*i.e.*, YOLOv3 and YOLOv4) with a single-precision floating-point format (32-bit) is maintained despite expressing weights with only about 20% of the bits compared to a single-precision floating-point format in COCO dataset.

INDEX TERMS Convolutional neural network, deep neural network, fixed-point quantization, network compression, object detector, YOLOv3, YOLOv4.

I. INTRODUCTION

In recent years, with the development of the Graphics Processing Unit (GPU), there has been tremendous development in the field of deep neural networks (DNNs). Because convolutional neural networks (CNNs) are used in the field of computer vision, the accuracy of object detection and classification increases dramatically [1]–[4]. However, because a DNN uses many layers, a large number of parameters are required, which significantly increases computational complexity [5]–[7]. In particular, for DNN-based object detection, classification, and localization are performed simultaneously, which requires vast computation [8]–[11]. Recently, the use of embedded platforms, such as autonomous vehicles and mobile devices has also increased dramatically. Because embedded platforms operate on battery-generated power and consist of limited hardware (HW) resources, embedded platform-based object detectors require low-power, light-weight, and fast network architectures while maintaining object detection accuracy.

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

To satisfy these requirements, various network compression studies, such as approximate computing, data quantization, and pruning, have been actively conducted [12]–[31]. Pruning methods reduce the number of parameters by removing unnecessary weights, filters, and layers, resulting in a less-complex network. In contrast, data quantization methods benefit from memory access time, inference time, and power consumption in the accumulation and multiplication of parameters by reducing the number of bits for parameters in the networks. Quantization methods have been actively researched because they have a lower decrease in accuracy than other compression methods and do not change the network structure, only reducing the bits that are allocated to the weight and activation parameters. For the weight quantization techniques that have been most actively studied, the value of the weight is expressed in a single-precision floating-point format (*i.e.*, 32-bit), and the number of weight parameter types used in the network is reduced through quantization [21]–[23].

There are two major problems with these conventional weight quantization methods. First, performance declines significantly when existing techniques are applied to object detection because they only focus on object classification. Second, existing quantization methods still include floating-point operations, so there is no significant advantage in terms of processing speed or power consumption in HW implementation. In HW architectures, floating-point arithmetic requires much more energy consumption for addition and multiplication operations than fixed-point arithmetic [24]. Moreover, in recent years, many studies have been conducted to accelerate CNNs through system-on-chip (SoC) design or field programmable gate array (FPGA) implementation for mobile devices. For such implementations, it is necessary to change the weights to an HW-friendly fixed-point format rather than a floating-point format [24].

This paper proposes a fixed-point-based quantization method appropriate for HW implementation (e.g., SoC design and FPGA implementation) that can be applied to object detection to overcome the shortcomings of existing studies. The proposed scheme quantizes the weights of each layer with optimally quantized bits using the grid search technique while increasing the quantization ratio gradually during the iterative retraining process. In particular, in the process of quantization, by subtracting the mean of weight parameters from all weight parameters in each layer, the weights are distributed around zero, thereby compensating for the performance degradation caused by the fixed-point representation. The proposed quantization technique enables computation suitable for the HW structure and, thus, supporting fast operation while significantly reducing computational complexity and power consumption in embedded platforms. Moreover, to maintain a light-weight effect while minimizing performance degradation even in DNN-based object detectors, we propose an adaptive quantization method considering the minimum and maximum weights of each layer. Consequently, the proposed method outperforms existing methods in both object classification and detection in terms of the trade-off between accuracy and computational complexity. Experimental results show that the proposed method with high compatibility achieves excellent performance not only in image classification networks but also in various object detectors. Especially, when applying it to the representative one-stage object detector, You Only Look Once v3 and v4 (YOLOv3 and YOLOv4) [10], [11], while maintaining the advantage of fixed-point-based quantization suitable for HW implementation, the proposed quantization method achieves approximately 80% weight parameter reduction with a negligible decrease in the mean average precision (mAP). The main contributions of this paper are summarized as follows:

- 1) HW-friendly fixed-point quantization: Fixed-point quantization accompanied by the zero-centered weight distribution and the iterative retraining scheme not only minimizes performance degradation caused by quantization, but also maximizes the light-weight effect of quantization in hardware implementation.
- 2) Mixed precision design for object detectors: The adaptive quantization technique based on the bit allocation method that considers the weight distribution within each layer minimizes performance degradation even in object detectors.
- 3) Quantization with high compatibility: The proposed technique can be easily applied to various object detectors including YOLOv3 [10] and YOLOv4 [11].

The rest of the paper is organized as follows. Section II provides background on the quantization and object detection. Section III describes the proposed quantization method. In Section IV, the proposed method is verified through the experimental results and compared with previous studies. Finally, Section V concludes the paper.

II. BACKGROUND

This section describes the related studies, the basic principle of the quantization method, and the YOLO algorithms that will be the target application of the proposed quantization method.

A. RELATED WORKS

Several studies applying quantization to CNNs have been conducted because of their advantages of robustness against performance degradation, ease of applying the concept of approximate computing, and lack of needing to change the network structure [21]–[26].

Most recent quantization studies [21]–[23] have focused on INT quantization with integer parameters and floating-point scale factors to prevent an accuracy decrease in networks that perform image classification. Especially, Jung *et al.* [21] found that a reduction in accuracy does not occur when using only four bits for both weight and activation parameters, even for the ImageNet dataset [32]. However, as mentioned previously, these studies on INT quantization [21]–[23] accompany floating-point operations, and consequently still have single-precision floating-point format

weights and activations. Therefore, the actual HW resource saving and speed-up in these studies are less than fixed-point quantization with the fixed-point scale factor. Nevertheless, in the field of computer vision, research in this direction has been actively conducted because it is most important to minimize performance degradation. However, in an embedded platform or HW accelerator environment, it is much more efficient to change the weight or activation parameters only to a fixed-point format, and therefore, quantization with a fixed-point format is still an important research topic concerning practicality and HW efficiency. Furthermore, as deep learning becomes more actively used in mobile application, the importance of practicality is expected to increase.

Accordingly, fixed-point format-based quantization studies have been conducted. Enderich *et al.* [25] change the distribution of weights multimodally on a fixed-point format but did not use a method to remove bias of the weight distribution, making it difficult to use in networks where the distribution of weights is not located on a zero basis. Lin *et al.* [26] use the signal-to-quantization-noise ratio (SQNR) to ensure that the error resulting from the fixed-point quantization is minimized and mathematically assign the bits to each layer for the fixed-point quantization. However, mathematically minimizing errors does not necessarily lead to an accuracy reduction. In particular, these fixed-point-based quantization studies have the advantage of easy HW implementation, but it is difficult to minimize performance degradation, so these studies have been conducted with the goal of applying to relatively simple image classification. Consequently, when they are applied to object detection, there is a problem that performance degradation is extremely significant.

B. PARAMETER QUANTIZATION

Quantization is a compression method to reduce large sets of parameters, such as input data, weights, and activation maps, generally represented by 32-bit floating-point, to a smaller discrete set of it with specific number of bits. By reducing the bits for the parameters in CNNs, which are composed mostly of convolution layers, it is possible to significantly reduce the network capacity and the amount of computation and power in HW design [6]. However, excessive quantization leads to an accuracy degradation in object classification/detection due to distortion of input data and deformation of learning parameters. Therefore, the number of bits to be reduced should be determined by considering the trade-off between the reduction in computational complexity and distortion of accuracy.

Quantization methods can be divided in two main ways, depending on the number representation. The first is to quantize an existing 32-bit floating-point number to a low-bit floating-point number (*e.g.*, half-precision floating-point number), and the second is to quantize an existing 32-bit floating-point number to a low-bit fixed-point (*e.g.*, INT8). Fig. 1 shows the 8-bit representation of floating-point and fixed-point quantization. The floating-point number has both exponent bits and mantissa bits that represent the degree of scale and resolution, respectively. Therefore, the floating-point quantization can represent a relatively wide range, and the quantization step-size does not need to be uniform. However, in the low-bit representation of 8bit or less, the number representation is more disadvantageous than the fixed-point number. On the other hand, the fixed-point quantization has a uniform quantization step-size and a relatively limited range of representations because several parameters have a shared exponent (*i.e.*, scale factor). However, since there are more bits allocated to mantissa, fixed-point quantization may have better solution than floating-point quantization in the low-bit representation. In addition, it should be noted that floating-point arithmetic costs approximately 9.1 times more energy for addition and 1.2 times more energy for multiplication than fixed-point arithmetic in HW architectures [24].

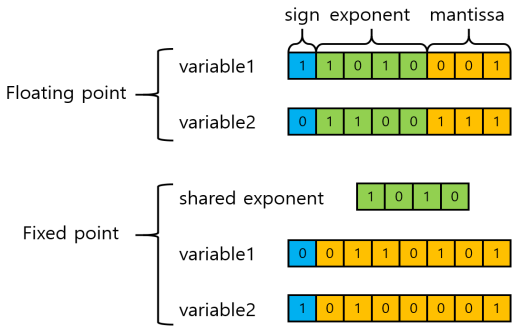


FIGURE 1. Representation of floating-point and fixed-point numbers.

Fixed-point quantization can also be classified into two types according to a scale factor. In case of integer (INT) quantization with fixed-point parameters and floating-point scale factors, the number of quantized bits with integer representation determines the number of candidates for weight or activation parameters, but the actual weight or activation values are represented as single-precision floating-point number because each candidate is multiplied by 32-bit floating-point scale factors. Therefore, there is no significant advantage in speed-up and power saving on HW design. On the other hand, when quantizing both parameters and scale factor to a fixed-point number as shown in Fig. 1, the benefits of quantization can be maximized although a slight performance drop may occur. Moreover, in this method, the multiplication of 2^n (i.e., shared exponent) can be replaced by shifting operation, which results in a lot of benefits in HW design. Specifically, when the shared exponent is 2^n and quantization is carried out with m bit, the fixed-point quantization points may be expressed as $k \times 2^n$, where the value k is as follows:

$$-\frac{2^m-2}{2} \leq k \leq \frac{2^m-2}{2} \quad (k \in integer). \tag{1}$$

Fig. 2 illustrates the distribution of the number of weights. Fig. 2(a) representing the distribution of the number of pre-trained weights is changed as Fig. 2(b) and Fig. 2(c) when

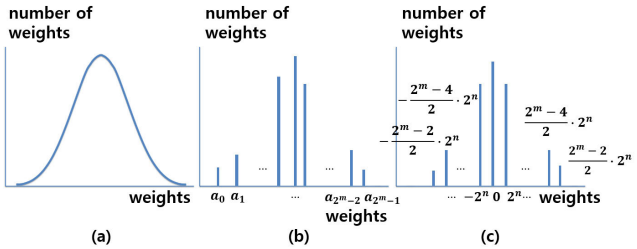


FIGURE 2. Distribution of the number of weights. (a) Normal (*i.e.*, Pre-trained) weights. (b) *m*-bit quantized weights based on floating-point representation. (c) *m* -bit quantized weights based on fixed-point representation.

quantized to *m*-bit floating-point format and *m*-bit fixed-point format, respectively. Fig. 2(c) shows that the fixed-point quantization has 2^m-1 quantization points distributed around zero at uniform quantization steps according to (1). This results in the advantage of reducing HW complexity by making the distribution of quantization points symmetric [27], but this also makes it difficult to avoid performance degradation.

C. YOLO NETWORKS

The YOLO algorithm [33], the most representative one-stage detection algorithm, enables fast object detection by dividing the image into multiple grids and analyzing information about each grid, in contrast to the region proposal method of two-stage detectors. In particular, the feature map of the detection layer is designed to output bbox coordinates, objectness scores, and class scores to enable detection of multiple objects with a single inference. Because of this network structure (i.e., the processing with grid units), YOLO had problems with low detection accuracy, although the detection processing speed is very fast compared to conventional methods. The YOLOv2 algorithm [34] proposed to solve these problems by improving detection accuracy by adding batch normalization to all convolution layers, an anchor box, multi-scale training, and fine-grained features while maintaining operating speed by changing the filter sizes of the existing YOLO to 1 × 1 and 3 × 3 filters. However, there is still the problem of low detection performance for small and dense objects.

To address the shortcomings of YOLOv2, as depicted in Fig. 3(a), the most recently proposed YOLOv3 [10] adds a residual skip connection to manage vanishing gradient problems in deep networks. Furthermore, for detecting small objects, up-sampling and concatenation techniques that preserve fine-grained features are applied, and detection layers at three different scales are added according to the feature map size using a similar way approach as the feature pyramid network [35]. In other words, YOLOv3 is a fully-convolutional network composed of only 1 × 1 and 3 × 3 size convolution filters and illustrates outstanding accuracy for objects of various sizes while maintaining a high processing speed similar to YOLO [33] and YOLOv2 [34].

In order to develop a more efficient and powerful object detection model, as depicted in Fig. 3(b), YOLOv4 [11]

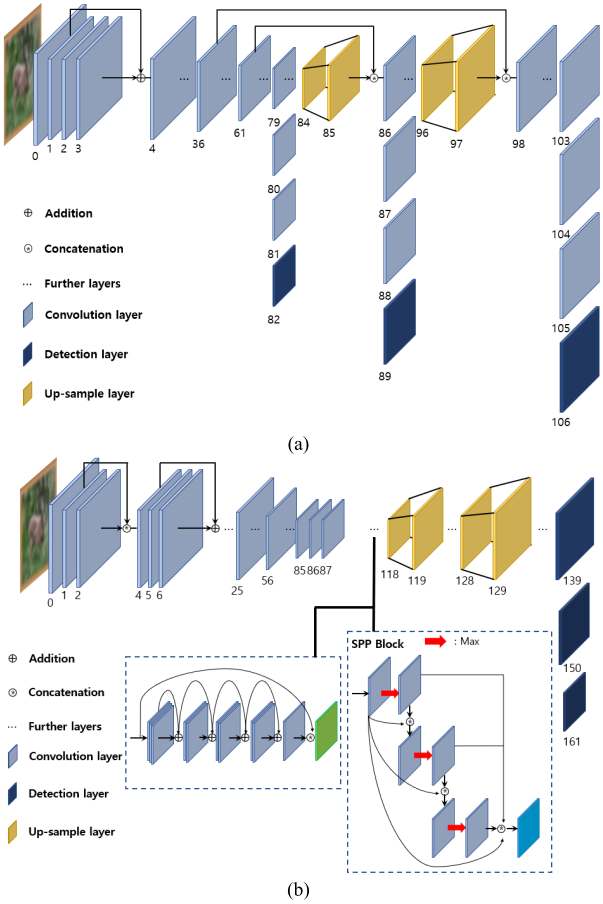


FIGURE 3. The network architecture. (a) YOLOv3. (b) YOLOv4.

applies the existing algorithms in two ways. One is the "Bag-of-Freebies", which includes training strategies and pre-processing methods, and the other is the "Bag-of-Specials", which includes architecture-related plugin modules and post-processing methods. Moreover, by selecting CSPDarknet53 as the backbone and utilizing the SPP block, it is possible to significantly improve the detection accuracy for relatively small objects at a fast-operating speed.

Given these advantages, YOLOv3 [10] and YOLOv4 [11] are widely used in domains where both accuracy and speed are important. In particular, it is widely used in embedded platforms, where many recent studies have been conducted [13]. However, even this one-stage object detector is constructed by stacking many layers with high complexity, so the network size is still large, and significant computation is required. Consequently, it is still challenging to support such a network for real-time operations in an embedded platform environment, so various compression techniques are required.

III. PROPOSED METHODS

A. OVERVIEW OF THE PROPOSED FIXED-POINT QUANTIZATION

The entire flowchart and algorithm of the proposed quantization method are depicted in Fig. 4 and Algorithm 1,

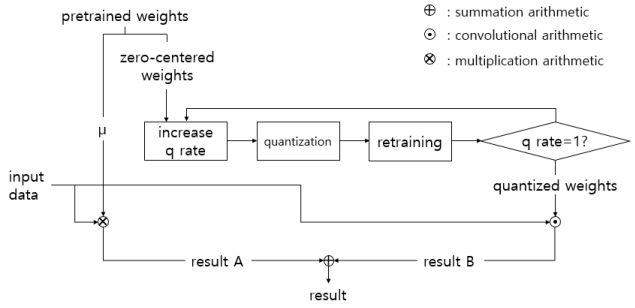


FIGURE 4. Flowchart of the proposed quantization.

Algorithm 1 Quantized Convolution Algorithm

Input: pretrained weights(W), input data(x)
Parameter: μ, num_weights, q_rate, m, n, k, max, min
Output: result

1: Let $\mu = 0$, q_rate =0, result_A =0, result_B =0
2: **while** q_rate !=1**do**
3: increase q_rate
4: **while** num_weights **do**
5: if (W > max) max = W
6: if (W < min) min = W
7: $\mu = \mu + W$
8: **end while**
9: n $\fallingdotseq \log_2$ ((max-min)/2^m-2)
10: $\mu = \mu$/num_weights
11: **while** num_weights **do**
12: $W_b = W - \mu$
13: if (k-q_rate/2)*$2^n \leq$ Wb $\leq$ (k + q_rate/2)*2^n
14: Wbq = Quan(Wb)
15: **end while**
16: *retraining*
17: **end while**
18: result_A = $\mu^* \sum$ x
19: **while** num_weights **do**
20: result_B = result_B + Wbq · x
21: **end while**
22: result = result_A + result_B
23: **return** result

respectively. As shown in Fig. 4, the proposed method processes the pretrained weights by dividing them into two parts: zero-centered weights and mean of weights (μ). Zero-centered weights are calculated by subtracting μ from the pretrained weights. μ does not need to apply the quantization process, and the convolutional operation of μ and input data can be replaced by simple multiplication because it has only one value in a certain group of weights. On the other hand, zero-centered weights are quantized and retrained iteratively, gradually increasing the quantization rate until the *q_rate* becomes 1. The well-quantized weights through this iterative process undergo a convolution operation with input data. After that, those two parts are added to get a better approximate result of the original calculation.

For a more detailed explanation, Algorithm 1 illustrates an algorithm for the fixed-point quantization of weight parameters to m bits in a filter unit. Variables m, n, k in the algorithm are described in Section II-B, and *num_weights* is the total number of weight parameters in the filter. The μ is the mean of the weight parameters in the filter, and *min* and *max* are the minimum and maximum weight values in the filter, respectively. In this paper, the existing convolution operation is divided into two operations, and finally, these two results from each operation are accumulated to calculate the final result. The final result is denoted as *result*, and the results of each of the two operations are denoted as *result_A* and *result_B*, respectively. *q_rate* is the ratio of weights which are being quantized in this step and has a value between 0 and 1. If the *q_rate* is 1, all parameters are quantized, and if it is 0, all parameters are not quantized. $*$ is a multiplication operation, and $\cdot$ is a convolutional operation. *Quan*() is a quantization function that performs deterministic quantization based on the parameters, *min*, *max*, m, n, and k, mentioned above.

The proposed method iteratively performs the quantization and retraining process by increasing the quantization rate gradually (line 3 of Algorithm 1) with sufficiently pre-trained weight parameters, until the *q_rate* becomes 1 (*i.e.*, repeat the **while** statement from line 2 to line 17 of Algorithm 1). It should be noted that the degree of the q_rate increase is determined empirically depending on networks, and always has a value between 0 and 1. For the quantization process, by limiting the range of the weight distribution based on the *max* and *min* values within the filter (lines 5 and 6 of Algorithm 1), the step-size of the fixed-point quantization (*i.e.*, 2^n) can be as sophisticated as possible (line 9 of Algorithm 1). Furthermore, by calculating the average of the weight parameters in the filter and subtracting this average from the weight parameters, it is possible to maximize the utilization of the characteristics of fixed-point-based quantization that performs quantization around zero (lines 7, 10, and 12 of Algorithm 1). Through this zero-centered process, detailed in Section III-B, weight parameters that are not distributed around zero can also be quantized with a small loss to suit the characteristics of the fixed-point-based method. After the zero-centered scheme, a quantization and retraining process is required. As depicted in Fig. 2(c), in fixed-point quantization, the quantized values are fixed to $k \times 2^n$ (k and n are an integer), but not all weight parameters before quantization exist around 2^n. In other words, they can be located in ambiguous points between adjacent quantized points. If these ambiguous values are quantized to inappropriate values, they cause a large reduction in accuracy and have a dominant impact on the performance of DNNs. Therefore, in this paper, only the weight parameters which are definitely close to quantized points (*i.e.*, $k \times 2^n$) are quantized first to $k \times 2^n$ and then retrained (lines 13 and 14 in Algorithm 1). After that, by gradually increasing the *q_rate* to expand the quantization range, this process is repeated until all weight parameters are quantized with the appropriate values. (*i.e.*, *q_rate* == 1). Section III-C describes the iterative

retraining in detail. After the quantization and retraining process, the proposed fast convolutional operation is performed with those quantized weight parameters (from line 18 to line 22 of Algorithm 1).

B. FAST CONVOLUTIONAL ARITHMETIC WITH THE ZERO-CENTERED WEIGHT DISTRIBUTION

As explained in Section II-B, when fixed-point quantization is performed, all weight parameters are converted to an integer k multiple of 2^n (*i.e.*, $k \times 2^n$), which means the quantization step-size is uniformly determined as 2^n, and the center of these quantized values is located at zero. Furthermore, the step-size varies significantly according to networks or filters and, depending on this value, overflow may occur and the resolution of data may decrease. In other words, the determination of the step-size strongly influences performance, so it is difficult to apply a quantization process suitable for the characteristics of each network or filter without considering this step-size. In this paper, to optimize the step-size of fixed-point quantization, the maximum and minimum values of each filter are considered, and within this range (*i.e.*, from the minimum value to the maximum value), the step-size is determined by dividing the range uniformly according to the specific target bit. As a result, the range is limited to ensure that the step-size is as sophisticated as possible. With an optimal step-size, it is possible to perform the quantization process more accurately, which minimizes the loss of the weight parameters.

Fig. 5 illustrates the medians of maximum and minimum weight parameters for each layer when training the VGG-7 network with Cifar-10 dataset [35] and the YOLOv3/v4 networks with COCO dataset [36]. As depicted in Fig. 5, the weight parameters of each layer are not always distributed around zero, and this trend is more apparent in the object detection (i.e., YOLOv3 and YOLOv4) than the image classification (i.e., VGG-7). In YOLOv3, as the layer deepens, the degree that the median deviates from zero tends to become more severe. In the near-end layer, the median of the minimum and maximum values is nearly 0.3 in YOLOv3, which is 15 times larger than 0.021 in VGG-7. Moreover, some layers (i.e., beginning and ending layers) in YOLOv4 have a higher median absolute value compared to YOLOv3. Nevertheless, if a fixed point-based technique that always sets the center to zero and quantizes parameters is applied to networks with such a non-zero centered weight distribution, the error due to the quantization becomes magnified. Especially, for the YOLOv3 and YOLOv4 networks, where most of the weight parameters are not distributed around zero, it is challenging to apply the existing fixed-point quantization, which leads to serious performance degradation. To address this issue, this paper uses a zero-centered effect that moves the center of the weight distribution to zero by finding the mean value of the weight parameters and subtracting the mean value from each weight parameter. Consequently, it is possible to minimize the quantization error that occurs when fixed-point quantization is applied.

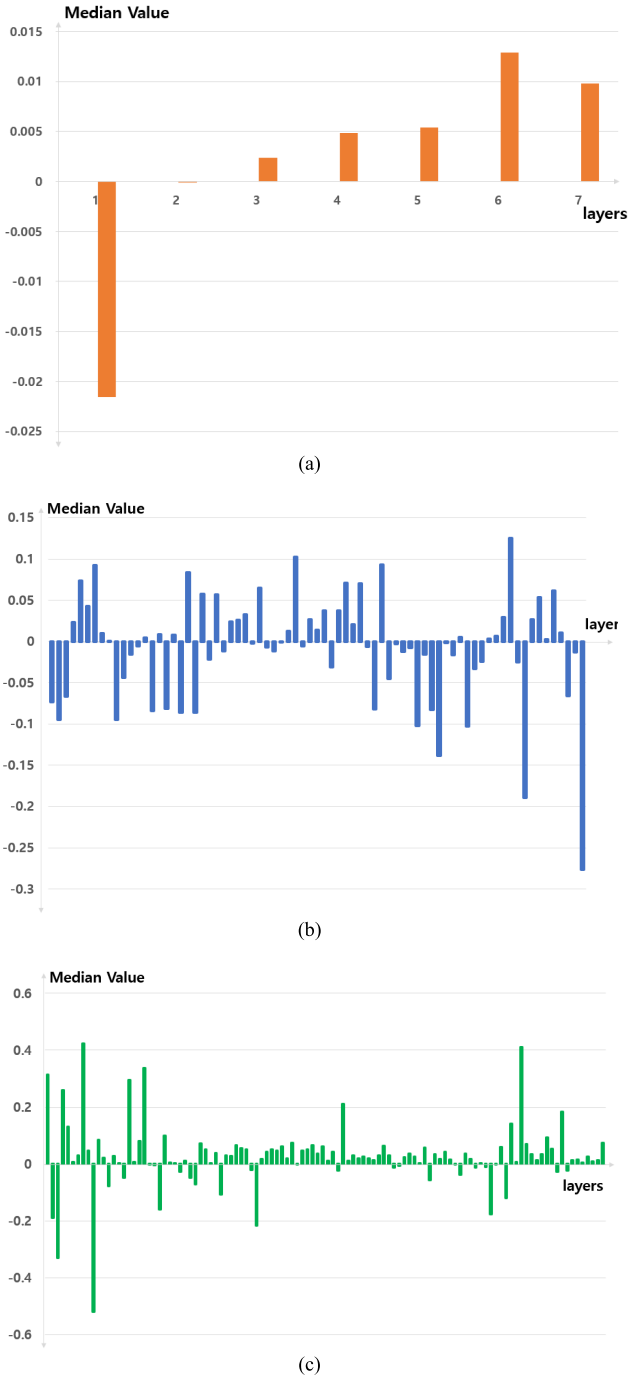


FIGURE 5. The median of maximum weights and minimum weights. (a) VGG-7 in Cifar10 dataset. (b) YOLOv3 in COCO dataset. (c) YOLOv4 in COCO dataset.

However, the zero-centered technique of the weight parameters accompanies changes in the convolutional arithmetic process. As depicted in Fig. 4, even if the zero-centered weight parameters are quantized, when performing the convolution operations with inputs, the subtracted mean value should be included in the convolution operation. This is because the average (μ) is artificially subtracted, and a process of adding it again appropriately before the convolution

operations is required to obtain the final correct result. This process can be expressed as follows:

$$\sum W \cdot x = \sum ((W_b + \mu) \cdot x) \quad (2)$$

$$= \sum (W_b) \cdot x + \mu \sum x \quad (3)$$

$$\approx \sum Quan(W_b) \cdot x + \mu \sum x \quad (4)$$

The parameters, Wb, *Quan*(), and μ, also can be seen in Algorithm 1. The first term in (2) illustrates the general convolution operations. W and x are the weights and input data, respectively, and the convolution operation can be expressed as W·x. If Wb is the value obtained by subtracting the average of the weight parameters in the filter (μ) from W, (2) is established because it simply divides the W into Wb and μ. Because μ is not affected by sigma ($\sum$), (3) is also established. Then, as depicted in (4), only Wb in the first term of (3) can be quantized by the proposed method with specific bits and the convolution operation of the quantized Wb and the input is performed. Next, by adding the multiplication result of the sum of input data x and the μ (*i.e.*, $\mu \sum x$) to this approximated value (*i.e.*, $\sum Quan$(Wb)), the same final convolution result as when W is directly quantized can be obtained. This process is illustrated in Fig. 6.

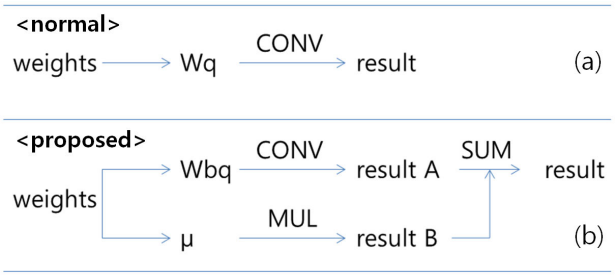


FIGURE 6. Flow of CONV arithmetic operation. (a) Original method. (b) Proposed method.

Because the *Quan*(Wb) in (4) is expressed with the quantized bits rather than original 32 bits, the convolution operation with this approximated value is much faster and operates at much lower power than the conventional convolution operation with a 32-bit floating-point format. This advantage is more prominent in HW accelerator designs. It should be noted that the second term of (4) consists of simple summations and a single multiplication rather than a complex convolution operation, so it is much simpler to calculate the result. Accordingly, this paper removes bias of the weight distribution to compensate for the shortcomings of fixed-point quantization, which must proceed around zero, and recovers them with only a small amount of computation (*i.e.*, the combination of simple summations and one multiplication), such that its usage has been extended to include weight parameters of all networks where the weight distribution is not zero-centered.

C. ITERATIVE RETRAINING SCHEME INCREASING QUANTIZATION RATE

As depicted in Fig. 2 (c), the quantization points of the fixed-point quantization are $k \times 2^n$ (k and n are an integer). Therefore, if the weights near $k \times 2^n$ are quantized to $k \times 2^n$, no significant loss occurs. However, ambiguous values between adjacent quantized points present a challenge in determining which quantization point to quantize, and the effect of this decision on the final performance on DNNs is severe. Furthermore, during the retraining process, the values of weight parameters are fine-tuned. Based on these two observations, in this paper, the values near the quantized points are first quantized to the corresponding close quantized point and the retraining is performed because values near the quantized points are more likely to have negligible accuracy loss after quantization and less likely to be changed due to retraining. Because it is impossible to apply this concept after only one instance of retraining, the quantization and retraining process is iteratively performed by increasing the quantization rate.

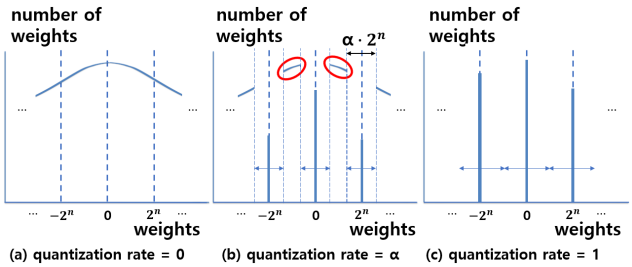


FIGURE 7. Distribution of weight parameters according to each quantization rate.

Fig. 7 illustrates the iterative retraining process when the quantization rate is 0, α, and 1, respectively, in the case where the weights exist around zero. As depicted in Fig. 7(a), when the distribution of the weights is given, the value of n is determined through the maximum and minimum values of the weight parameters in a filter (line 9 of Algorithm 1). The values near multiples of 2^n, such as -2^n, 0, and 2^n, do not cause a large loss even when quantized with these values. Therefore, as depicted in Fig. 7(b), the quantization process is applied only to the weight parameters within the range that is fixed according to the quantization rate (*i.e.*, close to -2^n, 0, 2^n), and the rest of the weight parameters marked with red circles in Fig. 7(b) are retained (line 13 of Algorithm 1). Then, all of these weight parameters are retrained. During this retraining process, the ambiguous weight parameters (*i.e.*, the red circled parameters in Fig. 7(b)) can be fine-tuned and closer to one of the values to be quantized. By fine-tuning these ambiguous values and gradually expanding the range of quantization according to the increase in the quantization rate, iterative quantization and retraining techniques can compute the final result, as depicted in Fig. 7(c).

TABLE 1. The portion of the ambiguous weight parameters in VGG-7, YOLOv3, and YOLOv4 networks.

	Method	Non-retrained network	Retrained network
Portion (%)	VGG-7 (Cifar-10)	23.33	8.73
	YOLOv3 (COCO)	22.95	18.01
	YOLOv4 (COCO)	20.06	14.65

Table 1 illustrates how the portion of ambiguous weight parameters between the adjacent quantized points decreases when the retraining process is performed with a quantization rate of 0.8 in the VGG-7 network, YOLOv3 network, and YOLOv4 network. If the quantization rate is 0.8, weight parameters corresponding to $(k + 0.4) \cdot 2^n \leq \mathrm{W} \leq (k + 0.6) \cdot 2^n$ are excluded from quantization, and the values of these weight parameters are changed during the retraining process. In Table 1, the second column illustrates the portion of the weight parameters that are excluded from quantization before the retraining, and the third column presents the portion of the weight parameters that are excluded from quantization after the retraining. The results illustrate that the portion of the ambiguous weight parameters is significantly reduced during retraining from 23.33%, 22.95%, and 20.06% to 8.73%, 18.01%, and 14.65% in VGG-7, YOLOv3, and YOLOv4, respectively, implying that iterative retraining can more divide the ambiguous values more distinctly, thereby minimizing the estimation loss that occurs during quantization.

D. APPLICATION TO THE OBJECT DETECTOR WITH THE BIT ALLOCATION METHOD

In applying to the object detectors, YOLOv3 and YOLOv4, the proposed quantization method is much more effective because the weight parameters of YOLOv3 and YOLOv4 are much more prone to not being distributed around zero, as mentioned in Section III-B, although the weight distribution varies depending on datasets. Moreover, YOLOv3 and YOLOv4 have many layers, as depicted in Fig. 3, so there are many weight parameters that have ambiguous values between adjacent quantized points, and inadequate quantization of these parameters directly leads to a degradation in performance. Consequently, in object detectors, it is essential to shift the center of the weight distribution to zero by subtracting the mean of weight parameters and reduce the proportion of weight parameters with ambiguous values through iterative retraining.

Furthermore, DNN models that perform object detection, including YOLO, have a significant depth of layers, and each layer has a different role, so the minimum number of bits required to express the weight parameters is different for each layer. In specific layers, even if the number of bits used to express the weight parameters is significantly reduced, the mAP value is not significantly changed. In addition, even in a specific layer, when the number of bits to be allocated for weight parameters is reduced, the mAP increases similar to the effect of preventing overfitting. Based on this observation, the optimized bits allocated to each layer in YOLOv3 and YOLOv4 are adaptively determined using the grid search method. For grid search, quantization is performed on a given bit for the weight parameters in the entire network model and mAP is measured by lowering the number of bits assigned to the weight parameters in each layer. Accordingly, the network is configured bit-adaptively by determining the number of bits that produces the maximum mAP in each layer.

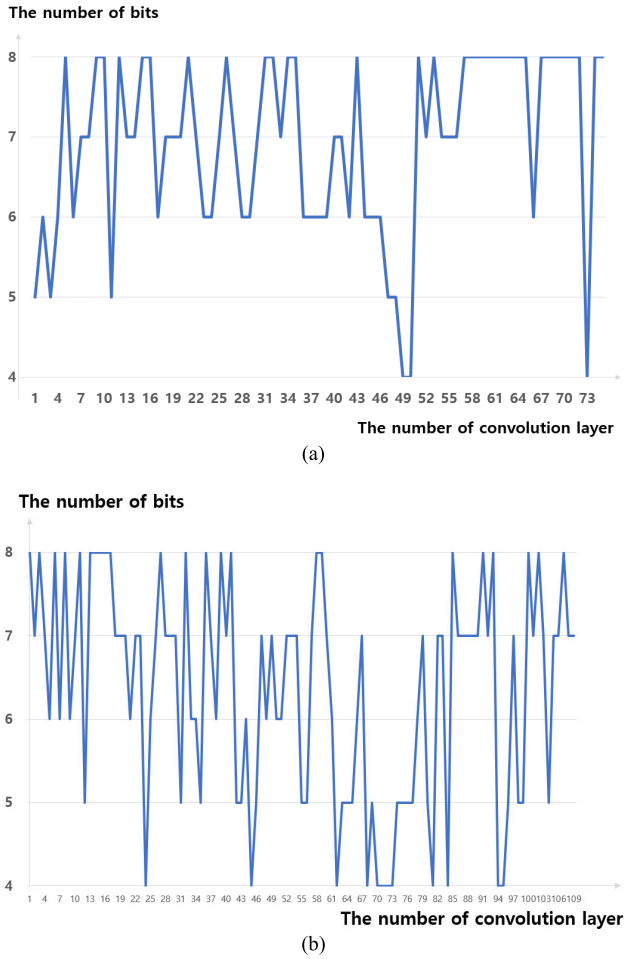


FIGURE 8. Number of bits used in each convolutional layer. (a) YOLOv3. (b) YOLOv4.

Fig. 8 presents the number of bits achieving the highest mAP for each layer in YOLOv3 and YOLOv4. In Fig. 8(a), to acquire these results, all weight parameters in YOLOv3 are quantized to 8 bits that do not affect performance degradation, and then the mAP is measured by changing the allocated weight bits of the specific layer while maintaining the weight bits of other layers at a fixed 8 bits. The results show that several layers can be quantized up to 4 bits. Finally, the bits determined according to the previously described method are allocated to each layer in YOLOv3. In Fig. 8(b), the optimal quantized bit for each layer in YOLOv4 is also measured in a similar way. The figure shows that YOLOv4 tends to have a high proportion of lower bit (i.e., 4 bits) layers that

lead to optimal mAPs in the middle of the network, and the front and end layers tend to require relatively higher bits (i.e., 8 bits) to produce optimal mAPs. In both YOLOv3 and YOLOv4 networks, after optimizing the bits for each layer, an optimally-quantized object detector with minimal performance degradation can be configured using the techniques presented in the previous subsections.

IV. EXPERIMENTAL RESULTS

A. EXPERIMENTAL ENVIRONMENT

To measure the accuracy and the weight size of each model, the experiments are performed on the server platform with Linux OS and GeForce RTX 2080Ti (10.1 CUDA). To verify the effect of the proposed quantization method on image classification, the experiments are conducted using the Cifar-10 dataset in VGG networks. To obtain an additional complexity reduction effect, the activation function parameters are also expressed as 4 bits through a Relu function by applying the technique in Section III-B. Usually, in HW implementation, the Relu function is used more widely than leaky Relu to express more numbers by removing the sign bit [37]. On the other hand, to verify the effect of the proposed techniques on object detection, experiments are conducted using the COCO dataset in the YOLOv3 and YOLOv4 networks. Considering the characteristics of object detection, the activation parameters maintain the full precision of the original network (*i.e.*, 32-bit) without applying quantization to minimize performance degradation. The mAP is measured at .5 IOU (Intersection Over Union).

TABLE 2. Accuracy change by applying the quantization method in CIFAR-10 dataset with VGG networks.

Method	Accuracy (%)	Accuracy Drop (%)	W/A bits
Original network	92.07	-	32/32
Basic Quantization + Basic Retraining	91.71	-0.36	4/4
Zero-Centered Quantization + Basic Retraining	91.74	-0.33	4/4
Basic Quantization +Iterative Retraining	91.79	-0.28	4/4
Zero-Centered Quantization + Iterative Retraining	**91.97**	**-0.1**	**4/4**

B. PERFORMANCE EVALUATION ON IMAGE CLASSIFICATION

Table 2 illustrates the accuracy results of the existing VGG-7 network (*i.e.*, 32-bit full decision) [2] and the quantized VGG-7 networks with the various options in the second column. In the fourth column, the number of bits used for weight and activation parameters in each network is presented. If the fixed-point quantization and retraining process are applied without both the zero-centered quantization technique in Section III-B and the iterative retaining technique in Section III-C, respectively, a significant accuracy degradation of 0.36% occurs. In contrast, if each of the zero-centered quantization and iterative retraining techniques is applied, the performance degradations are reduced to 0.33% and 0.28%, respectively. When both techniques are applied, the performance degradation is 0.1%, which is negligible despite the reduction of both weight and activation parameters from 32 to 4 bits. It should be noted that these results have been achieved only with pure fixed-points, with no floating-points being used for weight or activation parameters.

TABLE 3. Comparison results of accuracy drop in CIFAR-10 dataset with VGG networks.

Method	Model	Accuracy drop (%)	W/A bits
Proposed method	VGG-7	**-0.1**	**4/4**
Enderich *et al.* [25]	VGG-7	-0.15	2/32
SYMOG [38]	VGG-7	-0.19	2/32
Lin *et al.* [26]	VGG-8	-0.37	4/4
Miyashita *et al.* [39]	VGG-8	-0.31	4/5

Table 3 presents the results of comparing the performance with the existing fixed-point quantization schemes on image classification targets. Compared to Enderich *et al.* [25] and SYMOG [38], where only the weights are quantized with a fixed-point format, although the proposed method used two more bits for the weight parameters, the bits used for activation parameters are significantly reduced to 1/8, and the accuracy degradation is also less than both of them. Compared with the proposed method and Lin *et al.* [26], both studies allocate the same number of bits to weight and activation parameters, and the accuracy of the proposed method is 0.27% higher than that of Lin *et al.* [26]. Miyashita *et al.* [39] used log-scale quantization to replace multiplication operations with add operations in HW design. However, despite using 1 bit more for the activation parameter than the proposed method, the accuracy drop of Miyashita *et al.* [39] is greater by 0.21% than that of the proposed method.

TABLE 4. mAP comparison results on YOLOv3 using each method in COCO dataset.

Method	mAP	Average W bits
Original YOLOv3[10]	54.64	32
Quantized YOLOv3 (Basic/Proposed)	52.60 / 54.52	8
	50.59 / 53.64	7
	42.61 / 50.05	6
	14.52 / 46.95	5
	0 / 42.02	4
Proposed YOLOv3 with bit allocation	**54.52**	**6.58**

C. PERFORMANCE EVALUATION ON OBJECT DETECTION

Table 4 presents the mAP results of the original YOLOv3 network [10] and the quantized YOLOv3 network with the various options. In the third column, the average number of bits used for weight parameters in each network is presented. If the proposed quantization methods in Sections III-B and C

are not applied (*i.e.*, Quantized YOLOv3–Basic), performance degradation is extremely severe even if it is quantized to 8 bits. In contrast, when applying the zero-centered quantization technique in Section III-B and the iterative retraining technique in Section III-C, the performance degradation is insignificant at 0.12% until the number of bits allocated to weight parameters in YOLOv3 is quantized to 8 bits. However, if the number of bits allocated to weight parameters is reduced from 8 to 4 bits, a more significant mAP decrease occurs, as depicted in the table. Even if the proposed quantization technique is applied, the quantization to 7 bits also causes a significantly large mAP decrease, and if the quantized bits are even fewer, the reduction in accuracy is so severe that it is difficult to use in a real application. However, by applying the proposed bit allocation method in Section III-D, the average number of bits required for weight parameters can be decreased to an average of 6.58 bits while maintaining the same mAP with 8-bits quantization. Reducing the number of bits leads to a decrease in memory access time, inference time, and power consumption when designing HW accelerators. In conclusion, when both the proposed quantization methods and bit-allocation method are applied, it is possible to maintain performance similar to that of the original YOLOv3 with only approximately 20% bits for expressing the weight parameters in YOLOv3. Similarly, it should be noted that these results are achieved by representing the weight parameters in fixed-point format only, without any floating-point format.

TABLE 5. mAP comparison results on YOLOv4 using each method in COCO dataset.

Method	mAP	Average W bits
Original YOLOv4[11]	64.17	32
Quantized YOLOv4 (Basic/Proposed)	62.39 / 62.9	8
	58.68 / 62.42	7
	30.05 / 61.32	6
	0 / 58.26	5
	0 / 34.15	4
Proposed YOLOv4 with bit allocation	**63.54**	**5.99**

Table 5 presents the mAP results of the original YOLOv4 network [10] and the quantized YOLOv4 network with the various options. It should be noted that the structure of this table is exactly the same as Table 4, but the experimental results obtained by replacing YOLOv3 with YOLOv4 are presented. In YOLOv4, performance degradation due to the basic quantization is more severe than YOLOv3 in all the number of quantized bits. However, the proposed method can considerably mitigate the performance degradation even if bit allocation is not applied. It should be noted that the performance difference between the basic quantized model and the proposed quantized model is getting more significant as the allocated quantized bit becomes lower. Finally, by using the proposed quantization model with adaptive bit allocation, we can achieve a 0.64% higher mAP than the proposed 8-bit quantization model, even though the average number of bits is 2.01 bits less.

TABLE 6. mAP Comparison results on each network in COCO dataset.

Method	mAP (%)	W bits	Weight size (MB)
Original YOLOv3 [10]	54.64	32	237
YOLOv2 [34]	48.1	32	194
Zhang *et al.* [40]	48.1	16	97
YOLOv3-tiny [10]	33.1	32	33.79
Gaussian YOLOv3-tiny [13]	39.3	32	33.82
Nayak *et al.* [41]	28.09	8	30.48
Proposed quantized YOLOv3	**54.52**	**6.58**	**48.96**

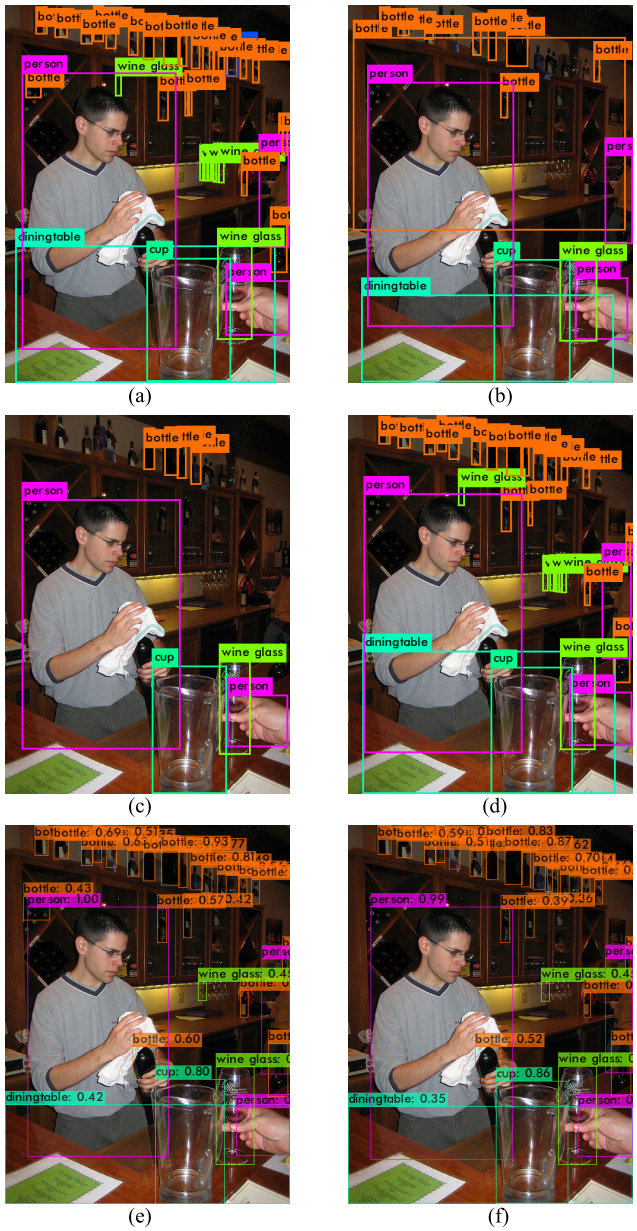


FIGURE 9. Detection results of the various object detectors. (a) YOLOv3. (b) YOLOv2. (c) Quantized YOLOv3 (Basic-6bits). (d) Proposed quantized YOLOv3. (e) YOLOv4. (f) Proposed Quantized YOLOv4.

Table 6 compares the proposed network with other networks in terms of accuracy and weight size. Compared to YOLOv2 network [34] and its quantized model [40],

the proposed network achieves 6.42 higher mAP even though the weight size of the proposed network is approximately one-quarter and one-half smaller, respectively. Compared to YOLOv3-tiny [10], Gaussian YOLOv3-tiny [13], and Nayak *et al.* [41], the weight size of the proposed network is approximately one-third larger, but the mAP of the proposed network is much higher than that of YOLOv3-tiny [10], Gaussian YOLOv3-tiny [13], and Nayak *et al.* [41] by 21.42, 15.22, and 26.43, respectively. In particular, through the comparison result with Gaussian YOLOv3-tiny [13], which enhances the accuracy of a small-scale network, it can be seen that effectively reducing the size of a large-scale network is more efficient than improving the performance of a small-scale network. Consequently, the proposed network is superior in terms of the trade-off between accuracy and network size.

For visual evaluation, Fig. 9 illustrates the detection results of the various object detectors. In the experiment, the detection threshold is set to 0.5, which is the default test threshold of the original YOLO network. In Figs. 9 (a)-(d), the results of the original YOLOv3 and the proposed network illustrate similar performance except that some small objects are missing. The proposed network is superior to the quantized YOLOv3 with the 6-bit basic scheme and YOLOv2. In particular, in the quantized YOLOv3 with the 6-bit basic scheme and YOLOv2, almost all small objects are missing, whereas the proposed quantized YOLOv3 identifies almost all of the same small objects despite reducing the network size to 1/5 of the original YOLOv3 network. In Figs. 9(e) and 9(f), the detection results of the original YOLOv4 and the proposed YOLOv4 model are presented. The original YOLOv4 and the proposed quantized YOLOv4 network show almost similar detection results except for one or two small objects. Although the average number of bits required for each weight parameter in the proposed YOLOv4 has been reduced to an average of 5.99 bits, the proposed YOLOv4 detects significantly more objects than the original YOLOv3.

V. CONCLUSION

In this paper, we propose a fixed-point-based quantization method specialized for embedded platforms to compensate for the problems of floating-point-based quantization methods. Furthermore, an adaptive application method that can expand to the object detection is proposed. When applying the proposed techniques to YOLOv3 and YOLOv4, the mAP similar to that of the original network using a single-precision floating-point format can be achieved using only approximately 20% of the number of bits for weight parameters, while maintaining the advantage of fixed-point-based quantization suitable for HW implementation. Consequently, the proposed algorithm can contribute significantly to the commercialization of deep learning algorithms by advancing the implementation of object classification/detection HW accelerators for embedded platforms.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 1, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[4] J. Choi, D. Chun, H. Kim, and H.-J. Lee, "Gaussian YOLOv3: An accurate and fast object detector using localization uncertainty for autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 502–511.

[5] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 3873–3882.

[6] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

[7] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in FPGA design for CNN-based object detectors," *IEEE Trans. Circuits Syst. Video Technol.*, early access, Aug. 31, 2020, doi: 10.1109/TCSVT.2020.3020569.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2014, pp. 580–587.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Oct. 2016, pp. 21–37.

[10] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*. [Online]. Available: http://arxiv.org/abs/1804.02767

[11] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*. [Online]. Available: http://arxiv.org/abs/2004.10934

[12] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[13] J. Choi, D. Chun, H.-J. Lee, and H. Kim, "Uncertainty-based object detector for autonomous driving embedded platforms," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 16–20.

[14] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A survey of FPGA-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1–26, 2019.

[15] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave Gaussian quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5918–5926.

[16] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1389–1397.

[17] D. T. Nguyen, N. H. Hung, H. Kim, and H.-J. Lee, "An approximate memory architecture for energy saving in deep learning applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 5, pp. 1588–1601, May 2020.

[18] D. T. Nguyen, H. Kim, H. J. Lee, and I. J. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

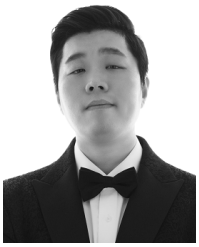
[19] Z. Zhou, W. Zhou, X. Lv, X. Huang, X. Wang, and H. Li, "Progressive learning of low-precision networks for image classification," *IEEE Trans. Multimedia*, early access, Apr. 23, 2020, doi: 10.1109/TMM.2020.2990087.

[20] Y. Xu, W. Dai, Y. Qi, J. Zou, and H. Xiong, "Iterative deep neural network quantization with lipschitz constraint," *IEEE Trans. Multimedia*, vol. 22, no. 7, pp. 1874–1888, Jul. 2020.

[21] S. Jung, C. Son, S. Lee, J. Son, J. J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4350–4359.

[22] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2019, pp. 7948–7956.
[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149
[24] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
[25] L. Enderich, F. Timm, L. Rosenbaum, and W. Burgard, "Learning multimodal fixed-point weights using gradient descent," in *Proc. 27th Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, 2019, pp. 1–6.
[26] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jun. 2016, pp. 2849–2858.
[27] J. Faraone, N. Fraser, M. Blott, and P. H. W. Leong, "SYQ: Learning symmetric quantization for efficient deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4300–4309.
[28] J.-I. Guo, C.-C. Tsai, J.-L. Zeng, S.-W. Peng, and E.-C. Chang, "Hybrid fixed-point/binary deep neural network design methodology for low-power object detection," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 3, pp. 388–400, Sep. 2020.
[29] M. Wess, S. M. P. Dinakarrao, and A. Jantsch, "Weighted quantization-regularization in DNNs for weight memory minimization toward HW implementation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2929–2939, Nov. 2018.
[30] M. Kiyama, Y. Nakahara, M. Amagasaki, and M. Iida, "A quantized neural network library for proper implementation of hardware emulation," in *Proc. 7th Int. Symp. Comput. Netw. Workshops (CANDARW)*, Nov. 2019, pp. 136–140.
[31] R. Z. Tan, X. Chew, and K. W. Khaw, "Quantized deep residual convolutional neural network for image-based dietary assessment," *IEEE Access*, vol. 8, pp. 111875–111888, 2020.
[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
[33] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
[34] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7263–7271.
[35] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. 4, 2009.
[36] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2014, pp. 740–755.
[37] L. Lai, N. Suda, and V. Chandra, "Deep convolutional neural network inference with floating-point weights and fixed-point activations," 2017, *arXiv:1703.03073*. [Online]. Available: http://arxiv.org/abs/1703.03073
[38] L. Enderich, F. Timm, and W. Burgard, "SYMOG: Learning symmetric mixture of Gaussian modes for improved fixed-point quantization," *Neurocomputing*, vol. 416, pp. 310–315, Nov. 2020.
[39] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," 2016, *arXiv:1603.01025*. [Online]. Available: https://arxiv.org/abs/1603.01025
[40] S. Zhang, J. Cao, Q. Zhang, Q. Zhang, Y. Zhang, and Y. Wang, "An FPGA-based reconfigurable CNN accelerator for YOLO," in *Proc. IEEE 3rd Int. Conf. Electron. Technol. (ICET)*, May 2020, pp. 74–78.
[41] P. Nayak, D. Zhang, and S. Chai, "Bit efficient quantization for deep neural networks," 2019, *arXiv:1910.04877*. [Online]. Available: http://arxiv.org/abs/1910.04877

SUNGRAE KIM received the B.S. degree in electrical and information engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2019. His research interests include SoC design and compression schemes for deep neural networks.



HYUN KIM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2009, 2011, and 2015, respectively. From 2015 to 2018, he was with the BK21 Creative Research Engineer Development for IT, Seoul National University, as a BK Assistant Professor. In 2018, he joined the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, where he is currently working as an Assistant Professor. His research interests include algorithm, computer architecture, memory, and SoC design for low-complexity multimedia applications and deep neural networks.