IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

DOI:10.1587/transfun.2023EAP1163

Publicized:2024/08/05

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Accelerating CNN Inference with an Adaptive Quantization Method Using Computational Complexity-Aware Regularization

Kengo NAKATA[†], **Daisuke MIYASHITA**[†], **Jun DEGUCHI**[†], ***Nonmembers,***
and **Ryuichi FUJIMOTO**[†], ***Senior Member***

SUMMARY Quantization is commonly used to reduce the inference time of convolutional neural networks (CNNs). To reduce the inference time without drastically reducing accuracy, optimal bit widths need to be allocated for each layer or filter of the CNN. In conventional methods, the optimal bit allocation is obtained by using the gradient descent algorithm while minimizing the model size. However, the model size has little to no correlation with the inference time. In this paper, we present a computational-complexity metric called MAC×bit that is strongly correlated with the inference time of quantized CNNs. We propose a gradient descent–based regularization method that uses this metric for optimal bit allocation of a quantized CNN to improve the recognition accuracy and reduce the inference time. In experiments, the proposed method reduced the inference time of a quantized ResNet-18 model by 21.0% compared with the conventional regularization method based on model size while maintaining comparable recognition accuracy.
key words: *deep learning, convolutional neural networks, inference, quantization, mixed-precision computing*

1. Introduction

Convolutional neural networks (CNNs) have achieved high recognition accuracy in image classification and object detection tasks [1]–[3]. However, inference of CNNs requires millions to billions of multiply–accumulate (MAC) operations, resulting in huge amounts of latency and energy consumption. Quantization is an effective technique for reducing computational costs and accelerating CNN inference by lowering the bit precision [4]–[6]. For example, lowering the bit precision from 8 bits to 4 bits allows MAC operations to be executed twice as fast on general-purpose computing devices such as NVIDIA Tesla T4 [7] and A100 [8] graphical processing units (GPUs), as well as on dedicated accelerators [9], [10]. Recent studies [11]–[17] have proposed layer-wise or filter-wise quantization methods that reduce computational costs without drastically reducing accuracy by allocating the optimal bit width depending on the layer or filter.

A promising technique for determining the optimal bit allocation is to employ the gradient descent algorithm [18]–[20]. The quantization step size, which determines the bit width of weights, is set as a learnable parameter along with the weights and is updated by the stochastic gradient descent (SGD) algorithm to minimize the classification error. After iterative updates, the optimal bit allocation can be derived from the optimized quantization step size and weight parameters.

Uhlich et al. [20] proposed a conventional gradient descent–based regularization method based on the model size and memory footprint of the quantized CNN. They were able to optimize the quantization step size and weight parameters to minimize the classification error under the constraints imposed by the model size and memory footprint. Consequently, the optimal bit allocation is obtained for achieving a high recognition accuracy at less than the target model size and memory footprint. However, their method focuses only on minimizing the model size and memory footprint and does not directly consider the processing time for CNN inference (i.e., inference time) during optimization. In other words, their optimal bit allocation is not always ideal for reducing the inference time.

In this paper, we present a computational-complexity metric that does not focus on the model size or memory footprint but instead considers the number of MAC operations and the bit widths of a quantized CNN, which we show to be strongly correlated with the inference time. We use this metric to propose a regularization method and optimization flow [21] for obtaining an optimal bit allocation to achieve a high recognition accuracy at less than a target computational complexity or inference time. We empirically demonstrate that models optimized by our proposed method achieve a better recognition accuracy with a shorter inference time than models optimized by Uhlich et al.'s conventional method.

The remainder of this paper is organized as follows. Section 2 describes related works and their issues. Section 3 introduces the computational-complexity metric. Section 4 presents the proposed regularization method and optimization flow using the computational-complexity metric. Section 5 presents the experimental results, and Section 6 gives our conclusions.

[†]The author is with Kioxia Corporation, Yokohama, 247-8585 Japan.

2. Related work and issues

2.1 Methods for optimal bit allocation

CNNs comprise tens or hundreds of layers with thousands of filters [1], [2]. Finding the optimal bit allocation for each layer or filter by an exhaustive search is computationally costly*. Many methods have been proposed for effectively determining the optimal bit allocation of quantized CNNs, for example, the analysis-based [11], [13]–[15], [22], reinforcement learning–based [16], [17], [23], and gradient descent–based [18]–[20].

The analysis-based approach determines the bit allocation by observing the distribution of weight values at each layer or filter [11], [22] or by measuring the sensitivity of the recognition accuracy to quantizing the weight and activation values [13]–[15]. Observing the distribution and evaluating the sensitivity do not require iterative forward and backward operations to update the weight parameters. Therefore, the analysis-based approach has a low computational cost and is useful for determining the bit allocation immediately. However, it is insufficient for improving the recognition accuracy at smaller bit widths or shorter inference times compared with the other approaches, which iteratively update the weight parameters to determine the optimal bit allocation.

The reinforcement learning–based approach [16], [17], [23] repeats a series of processes that include setting the bit allocation, fine-tuning weight parameters, and evaluating the accuracy and inference time for various bit allocations. Reinforcement learning is used to search for the optimal bit allocation based on the evaluation results of the accuracy and inference time. This approach is effective when the characteristics of the optimization target (e.g., the hardware used for inference) are not entirely revealed or when the objective function is non-differentiable (i.e., loss gradients cannot be calculated). However, the weight parameters need to be fine-tuned over tens to hundreds of epochs each time the bit allocation is changed, so the search for the optimal bit allocation is time-consuming.

Recent studies [24], [25] have empirically shown that dedicated hardware for mixed-precision computing [26] can reduce the processing time in proportion to the bit precision. In other words, when such hardware is used for inference, the characteristics of the optimization target are already revealed. Therefore, optimization does not necessarily require the reinforcement learning–based approach but instead can be effectively achieved by the gradient descent–based approach.

In this paper, we focus on a computational-complexity metric that can be approximately differentiable, and we apply this metric to the gradient descent–based approach.

The gradient descent–based approach [18], [19] uses a gradient descent algorithm such as SGD to determine the bit allocation. This approach sets the quantization step size, which determines the bit width of weights, as a learnable parameter along with the weights. It iteratively updates the weight and step size parameters to minimize the value of the loss function. After the updates, the bit allocation is determined based on the optimized weight and step size parameters.

To obtain a more compact quantized CNN, Uhlich et al. [20] proposed a regularization method that uses the model size and memory footprint of the quantized CNN. When the model size is used as a regularization term, it is added to the loss $\mathcal{L}$ to give

$$\mathcal{L} = \mathcal{L} + \lambda M, \tag{1}$$

where λ is a hyperparameter that adjusts the regularization effect. M is the model size of the quantized CNN, and it is calculated as

$$M = \sum_l (\#\text{params})_l \times b_l, \tag{2}$$

where l is a layer index and b_l is the bit width corresponding to the weight parameters at the l-th layer. #params is the number of weight parameters, and it is calculated as

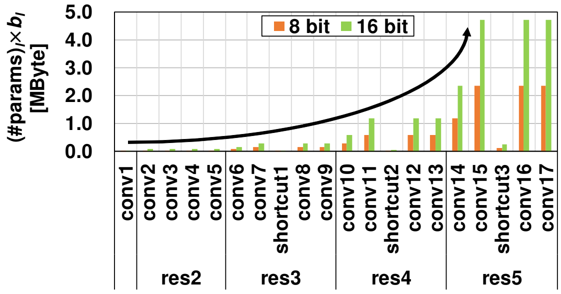
$$\#\text{params} = C_\text{o} C_\text{i} K_\text{h} K_\text{w}, \tag{3}$$

where C_o and C_i are the numbers of output and input channels, respectively, and K_h and K_w are the height and width, respectively, of kernels (i.e., filters).

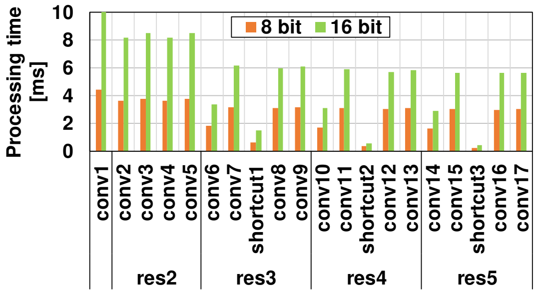
Eqs. (2) and (3) do not include the size of input images or output feature maps at each layer, nor do they directly consider the number of iterations in which the weight parameters are used for forward operations during inference. Therefore, Uhlich et al.'s conventional method of using the model size (or memory footprint) for regularization is effective at obtaining compact models that achieve high accuracy with limited memory consumption, but it is not always effective for optimizing computational costs associated with inference, such as the inference time.

Here, we present measurement results demonstrating the lack of a correlation between the model size and inference time. Fig. 1 shows the values of $(\#\text{params})_l \times b_l$ from Eq. (2) and the processing time at each layer when the inference of the ResNet-18 model [27] is executed at 8 and 16 bits on an NVIDIA Tesla T4 GPU. For this measurement, images in the ImageNet dataset [28] were used, and the batch size was set to 500.

*For example, if a CNN comprises 10000 filters and the bit width can be set from 1 to 8 bits, then the search space will be 8^{10000}.



(a) Products of (#params)$_l$ and b_l for each layer.



(b) Processing time for each layer.

Fig. 1: (a) Product of the number of parameters (#params)$_l$ and bit width (b_l) obtained by Eq. (2), and (b) the processing time at each layer when inference of the ResNet-18 model is executed with 8 and 16 bits on an NVIDIA Tesla T4 GPU. Later layers result in larger products but not a longer processing time. (©2021 IEEE [21])

As shown in Fig. 1(a), the later the layer, the larger the values of (#params)$_l$ $\times$ b_l in the ResNet-18 model. In contrast, as shown in Fig. 1(b), the processing time does not increase in the later layers. Fig. 2 shows the relation between (#params)$_l$ $\times$ b_l and the processing time based on the results shown in Figs. 1(a) and 1(b). The correlation coefficient is 0.16, which indicates that the processing time is uncorrelated with (#params)$_l \times b_l$ or the model size, which accumulates those product values for all layers as given in Eq. (2). These results demonstrate that the conventional method [20] of using the model size for regularization does not optimally reduce the inference time.

2.2 Computational complexity-aware regularization methods

In Section 3, we introduce a computational-complexity metric (MAC×bit) that is correlated with inference time, and in Section 4, we propose a computational complexity-aware regularization method and optimiza-

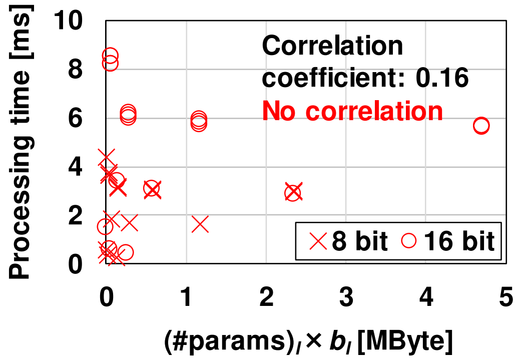


Fig. 2: Relation between the processing time and product of (#params)$_l$ and b_l at each layer when inference of the ResNet-18 model is executed with 8 and 16 bits on an NVIDIA Tesla T4 GPU. (©2021 IEEE [21])

tion flow using this metric. Similar to our approach, previous and concurrent works have introduced computational-complexity metrics, such as BOPs or BitOPs (Bit Operations), which consider both bit precision and the number of MAC operations [29]–[34]. For example, Wu et al. [29] and Cai et al. [30] have used such a metric as a cost function in their mixed-precision network search. Baalen et al. [31] and Yang et al. [32] have attempted to regularize computational complexity by using the metrics of BOPs and BitOPs, where the impacts of the sizes of the output feature maps are incorporated, similar to our introduced metric. They have employed these metrics as a penalty term in their computational complexity-aware regularization, which is qualititatively the same as our approach presented in Section 4.3. Furthermore, recent studies have explored optimizing bit allocations based on input images [35], [36] or improving the efficiency of searching for optimal bit allocation [37] by expanding upon the idea of computational complexity-aware regularization.

For the optimization of bit allocations in quantized CNNs, it is straightforward to introduce a computational-complexity metric that considers both the bit precision and the number of MAC operations, as introduced in the aforementioned works. However, these works do not provide sufficient quantitative analysis regarding the relationship between the metric and computational costs, such as inference time. One of the main objectives of this paper is to reveal the limitations of the conventional regularization method [20], which uses model size, by conducting a quantitative analysis based on its correlation with inference time. We also aim to quantitatively analyze the improvement in correlation achieved using our introduced computational-complexity metric compared to the correlation with model size. Furthermore, we empirically show the optimization transition towards a target by following the optimization flow, in addition to the improvement in

Table 1: Performance summary of hardware supporting multibit precision computing

Computational performance [TOPS*]	8bit	4bit	1bit
NVIDIA Tesla T4 GPU [7]	130	**260**	–
NVIDIA A100 GPU [8]	624	**1248**	**4992**
Dedicated accelerator [9]	0.69	**1.38**	**7.37**
Dedicated accelerator [10]	19.7	**39.3**	–

*Tera Operations per Second

recognition accuracy on image classification datasets as demonstrated in the previous and concurrent works.

2.3 Hardware supporting multibit precision computing

Recently, both general-purpose and dedicated hardware supporting multibit precision computing have been widely released [7]–[10], [26], [38]. Such hardware demonstrate linear improvement in the computational performance with lower bit precision, as given in Table 1. However, the advantages of such hardware cannot be fully exploited for inference by layer-wise or filter-wise quantized CNNs unless the optimal bit allocation is found. In this paper, we focus on the hardware characteristics that linearly decrease the computational time with lower bit precision for determining the optimal bit allocation.

3. Computational-complexity metric for quantized CNNs

CNNs have a very high arithmetic intensity**, so the computational time is often the bottleneck rather than the memory access time [40], [41]. Thus, the inference time is mainly determined by the computational complexity of the inference and computational performance of the hardware. For CNNs, the computational complexity is typically evaluated by the number of MAC operations or floating-point operations (FLOPs) [42]–[44]. However, such metrics do not properly reflect the effect of bit precision on the computational cost for inference by a quantized CNN with weights represented by various bit widths.

For hardware supporting multibit precision computing [7]–[10], [26], [38], lowering the bit precision linearly increases the computational speed, as summarized in Table 1. In other words, the computational time is linearly proportional to the bit precision on such hardware. Among the hardware options for multibit precision computing, our primary focus in this paper is the dedicated accelerator proposed by Maki et al. [26]. Based on the characteristics of this hardware, we can define a metric for estimating the computational cost of inference by a quantized CNN as the sum of the products of the number of MAC operations (#MAC) and bit precision, which we call MAC×bit:

$$\mathrm{MAC{\times}bit} = \sum_{l} (\#\mathrm{MAC})_l \times b_l, \tag{4}$$

where l is a layer index and b_l represents the bit width for weight parameters at the l-th layer, which is the same as in Eq. (2). For the MAC operations of a typical CNN, weight parameters are repeatedly used $O_\mathrm{h} \times O_\mathrm{w}$ times, where O_h and O_w are the height and width, respectively, of the output feature maps. Then, #MAC in Eq. (4) can be calculated and transformed with Eq. (3) to give

$$\begin{aligned} \#\mathrm{MAC} &= O_\mathrm{h} O_\mathrm{w} C_\mathrm{o} C_\mathrm{i} K_\mathrm{h} K_\mathrm{w} \quad (5)\\ &= O_\mathrm{h} O_\mathrm{w} (\#\mathrm{params}). \quad (6)\end{aligned}$$

Here, we show the correlation between the inference time and MAC×bit with measurement results using the ResNet-18 model. Fig. 3 shows the values of

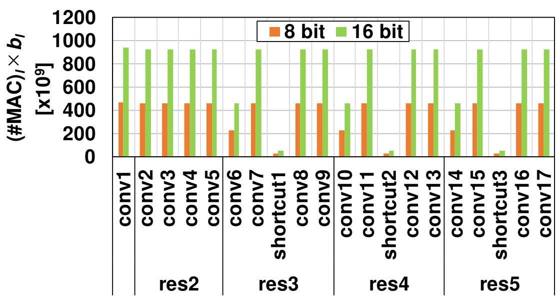


Fig. 3: Product of the number of MAC operations $(\#\mathrm{MAC})_l$ and bit width (b_l) obtained by Eq. 4 at each layer when inference of the ResNet-18 model is executed with 8 and 16 bits on an NVIDIA Tesla T4 GPU.

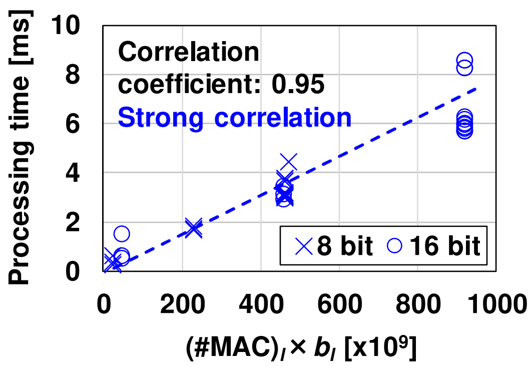


Fig. 4: Relation between the processing time and product of the number of MAC operations $(\#\mathrm{MAC})_l$ and bit width (b_l) at each layer when inference of the ResNet-18 model is executed with 8 and 16 bits on an NVIDIA Tesla T4 GPU. (©2021 IEEE [21])

**For instance, this metric is measured by the number of MAC operations per byte of data transferred from memory [39].

$(\#\mathrm{MAC})_l \times b_l$ obtained from Eq. (4) at each layer, and Fig. 4 shows the relation between the processing time and those values based on the results shown in Figs. 1(b) and 3. The measurement conditions were the same as in Figs. 1 and 2. The correlation coefficient was 0.95, indicating a strong correlation between the processing time and $(\#\mathrm{MAC})_l \times b_l$ or MAC×bit, which accumulates those products for all layers as given in Eq. (4).

Metrics such as MAC×bit that considers both the computational complexity and bit precision have previously been used to evaluate the performance of quantized CNNs or dedicated accelerators for mixed-precision computing [11], [24], [25], [33], [34], [45]. In this paper, we use MAC×bit not only to simply evaluate the computational cost but also as a regularization method. We propose an optimization flow for bit allocation to obtain a quantized CNN with high accuracy and a short inference time.

The metric of MAC×bit incorporates the impact of the number of weight parameters, as indicated in Eqs. (4) and (6). Although the proposed method could be integrated with pruning techniques [46]–[48], which accelerate CNN inference by reducing the number of weight parameters, our main focus in this paper is on quantization techniques themselves. Therefore, in this paper, we primarily evaluate the effectiveness of the proposed method without including pruning techniques on quantized CNNs using the metric of MAC×bit.

4. Proposed method

Our proposed method for obtaining the optimal bit allocation involves three steps. First, we evaluate the hardware characteristics in terms of the MAC×bit values and inference times of various quantized CNNs. Based on the characteristics, we can estimate a target value of MAC×bit to obtain the target inference time. Second, a CNN is fully pre-trained at full precision (i.e., floating-point 32 bits) without quantization. Third, the pre-trained model is optimized to improve the recognition accuracy within the target MAC×bit and inference time. Each step is described in detail below.

4.1 Evaluation of MAC×bit characteristics

We evaluate the hardware characteristics in terms of the MAC×bit values and inference time for various quantized CNNs with different numbers of layers, channels, and bit widths. We use the dedicated accelerator for mixed-precision computing proposed by Maki et al. [26] and evaluate its characteristics.

Fig. 5 shows the relation between MAC×bit values and the simulated inference time for various quantized CNNs, executed with a range of bit widths (1–8 bits) on the dedicated accelerator. We used ResNet-18, ResNet-50, and MobileNetV2 [49] as CNNs and images in the ImageNet dataset for inference. The inference time is strongly correlated with MAC×bit with a correlation coefficient of 0.99, and the relationship is approximately linear. Then, we can estimate a target MAC×bit value from the target inference time. For example, in Fig. 5, if the inference time is less than 150 ms, the target MAC×bit value should be less than 11×10^9. We can then use this estimated target MAC×bit value for the optimization flow described in Section 4.3. In addition, prior to optimization, the CNN architecture to be used for inference is determined based on the target inference time.

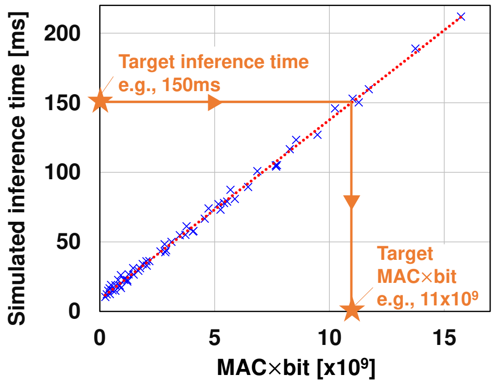


Fig. 5: Relation between MAC×bit and the simulated inference time on a dedicated accelerator [26] for CNNs with various #MAC and bit widths. The target MAC×bit value can be estimated from the target inference time.

4.2 Pre-training

For the gradient descent–based approach to optimal bit allocation (described in Section 2.1), weights can be initialized with random values [4], [5] or pre-trained at full precision [14], [15], [18], [19]. Uhlich et al. [20] empirically demonstrated that quantized CNNs achieve better accuracy when they use pre-trained weights at full precision rather than randomly initialized values. Thus, our proposed method requires the CNN to be fully pre-trained before optimization.

4.3 Optimization

4.3.1 Quantization procedures and calculations

When the weight $\mathbf{W}$ is quantized with the quantization step size s, the quantized weight $\mathbf{W}_\mathrm{q}$ is expressed as

$$\mathbf{W}_\mathrm{q} = \lfloor \mathbf{W}/s \rceil, \tag{7}$$

where $\lfloor \cdot \rceil$ is a rounding function that rounds the argument to the nearest integer, and the division operator ($/$) represents an element-wise operation that performs division on each element of a matrix or tensor. The dequantized weight $\mathbf{W}_{\mathrm{dq}}$ is applied to forward and backward operations, and it is calculated and transformed with Eq. (7) as

$$\mathbf{W}_{\mathrm{dq}} = \mathbf{W}_{\mathrm{q}} \times s \tag{8}$$
$$= \lfloor \mathbf{W}/s \rceil \times s, \tag{9}$$

where the multiplication operator ($\times$) represents an element-wise operation that performs multiplication on each element of a matrix or tensor.

Here, we consider the bit width required to represent the quantized weight $\mathbf{W}_{\mathrm{q}}$ that is used to calculate MAC×bit in Eq. (4). The granularity for the required bit width depends on the quantization method and the hardware specifications used for inference. For example, if we employ layer-wise/filter-wise quantization with dedicated hardware [9], [26], the weights in each layer/filter are quantized with the same bit precision. Then, the required bit width differs for each layer/filter and needs to be calculated for each. Here, we describe the required bit width for filter-wise quantization, which is more granular than layer-wise quantization.

The required bit width is mainly determined by the range of quantized weight values (i.e., the maximum and minimum values). For simplicity, if we consider the weight distribution to be approximately symmetrical across the positive and negative ranges, the approximate range can be obtained by taking the maximum of the absolute values of the quantized weights. By doubling the maximum value and taking the binary logarithm, the bit width (b_f) required to represent the quantized weights in the f-th filter ($\mathbf{W}_{\mathrm{q},f}$) can be calculated and transformed with Eq. (7) as follows:

$$b_f(\mathbf{W}_f, s_f) = \lceil \log_2(\max_f(\mathrm{abs}(\mathbf{W}_{\mathrm{q},f})) \times 2) \rceil \tag{10}$$
$$= \lceil \log_2(\max_f(\mathrm{abs}(\lfloor \mathbf{W}_f/s_f \rceil))) + 1 \rceil, \tag{11}$$

where $\lceil\cdot\rceil$, $\max(\cdot)$, and $\mathrm{abs}(\cdot)$ are the ceiling, max, and absolute value functions, respectively. To calculate MAC×bit using Eq. (4), b_l is determined by averaging the bit widths ($b_{f,l}$) at the l-th layer using $b_l = 1/N_l \sum_f b_{f,l}$, where N_l is the total number of filters in the l-th layer.

In the gradient descent–based approach [18]–[20], $\mathbf{W}$ and s are set as learnable parameters and are iteratively updated by the SGD algorithm to minimize the loss $\mathcal{L}$. The optimized $\mathbf{W}$ and s can then be used in Eq. (11) to derive the optimal bit allocation. To update $\mathbf{W}$ and s, we need to calculate gradients such as $\partial\mathcal{L}/\partial\mathbf{W}$ and $\partial\mathcal{L}/\partial s$. With $\mathbf{W}_{\mathrm{dq}}$ used for forward operations, $\partial\mathcal{L}/\partial\mathbf{W}_{\mathrm{dq}}$ can be obtained directly by backpropagation. With the backpropagated $\partial\mathcal{L}/\partial\mathbf{W}_{\mathrm{dq}}$, we can then approximate $\partial\mathcal{L}/\partial\mathbf{W}$ based on a straight-through estimator (STE) [50]:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}} \approx \frac{\partial\mathcal{L}}{\partial\mathbf{W}_{\mathrm{dq}}}. \tag{12}$$

Additionally, we can use the chain rule to obtain $\partial\mathcal{L}/\partial s$:

$$\frac{\partial\mathcal{L}}{\partial s} = \frac{\partial\mathcal{L}}{\partial\mathbf{W}_{\mathrm{dq}}} \cdot \frac{\partial\mathbf{W}_{\mathrm{dq}}}{\partial s}. \tag{13}$$

Esser et al. [18] proposed approximating $\partial\mathbf{W}_{\mathrm{dq}}/\partial s$ by using an STE-based approximation for the rounding function $\lfloor\cdot\rceil$ in Eq. (9) with the product rule:

$$\frac{\partial\mathbf{W}_{\mathrm{dq}}}{\partial s} = \frac{\partial}{\partial s}(\lfloor \mathbf{W}/s \rceil \times s) \tag{14}$$
$$= \frac{\partial}{\partial s}\lfloor \mathbf{W}/s \rceil \times s + \lfloor \mathbf{W}/s \rceil \times \frac{\partial s}{\partial s} \tag{15}$$
$$\approx \frac{\partial}{\partial s}(\mathbf{W}/s) \times s + \lfloor \mathbf{W}/s \rceil \tag{16}$$
$$= -\mathbf{W}/s + \lfloor \mathbf{W}/s \rceil. \tag{17}$$

Using the backpropagated gradient $\partial\mathcal{L}/\partial\mathbf{W}_{\mathrm{dq}}$ with Eqs. (13) and (17), $\partial\mathcal{L}/\partial s$ can be calculated.

For the regularization of MAC×bit, we need to calculate the gradients of MAC×bit with respect to $\mathbf{W}$ and s. In Eq. (4), MAC×bit is expressed as the sum of the products of #MAC (constant) and the bit width $b(\mathbf{W}, s)$. Thus, we focus on the gradients of $b(\mathbf{W}, s)$. Here, we consider the gradient of b_f with respect to $\mathbf{W}_f$ (i.e., $\partial b_f/\partial\mathbf{W}_f$) and apply the STE-based approximation to the ceiling and rounding functions in Eq. (11), which gives

$$\frac{\partial b_f}{\partial\mathbf{W}_f} = \frac{\partial}{\partial\mathbf{W}_f}\lceil \log_2(\max_f(\mathrm{abs}(\lfloor \mathbf{W}_f/s_f \rceil))) + 1 \rceil \tag{18}$$
$$\approx \frac{\partial}{\partial\mathbf{W}_f}(\log_2(\max_f(\mathrm{abs}((\mathbf{W}_f/s_f)))) + 1). \tag{19}$$

By using the automatic differentiation included in the PyTorch library [51], we can execute the backward operations for standard functions such as $\log_2(\cdot), \max(\cdot)$, and $\mathrm{abs}(\cdot)$, and we can obtain $\partial b_f/\partial\mathbf{W}_f$ from Eq. (19). $\partial b_f/\partial s_f$ can also be calculated by using the same approximation as in Eq. (19). With the PyTorch library, we can define custom backward operations related to the STE-based approximations in Eqs. (12), (17), and (19). Then, Eqs. (7)–(19) can be used to execute the processes from quantization to updating $\mathbf{W}$ and s.

4.3.2 Optimization flow

Our proposed regularization method includes an optimization flow for the weight $\mathbf{W}$ and quantization step size s so that MAC×bit and the inference time are less than the specified target values. To impose a constraint related to the computational cost of inference, we add the MAC×bit value calculated by Eq. (4) as a penalty term to the loss $\mathcal{L}$ to give

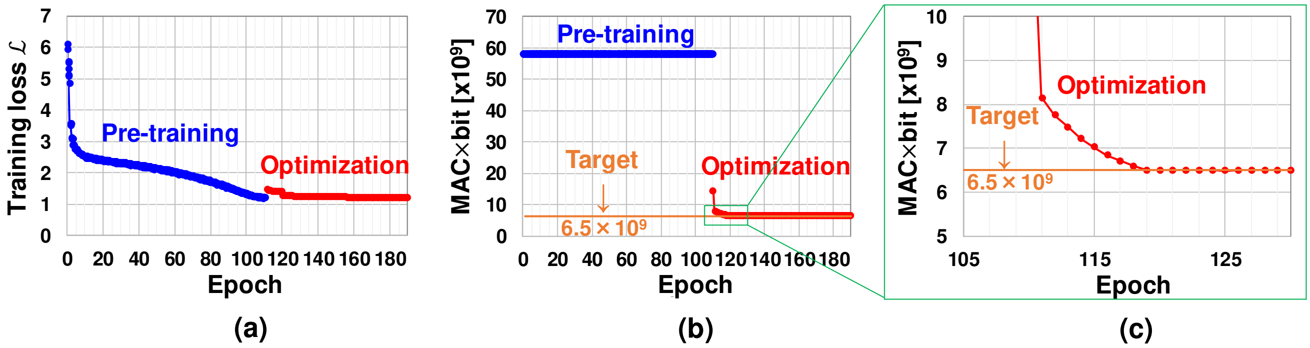


Fig. 6: Transition curves for (a) the training loss $\mathcal{L}$ and (b) MAC×bit during pre-training (blue plots) and optimization (red plots); (c) an enlarged graph of the MAC×bit curve from 105 to 130 epochs. The ResNet-18 model and ImageNet dataset were used to obtain these results. As shown in (c), MAC×bit decreased to the target value (6.5×10^9 indicated by the orange line) with optimization.

$$\mathcal{L} = \mathcal{L} + \lambda(\mathrm{MAC}\times\mathrm{bit}), \tag{20}$$

where λ is a hyperparameter that determines the regularization effect, which is the same as the conventional regularization based on model size in Eq. (1). We can adjust λ so that the penalty term is approximately the same magnitude as the initial loss value before optimization. For instance, if $\mathcal{L} \approx 1$ and MAC×bit $\approx 10^9$ before optimization, then λ can be set to 1×10^{-9}.

Algorithm 1 shows the optimization flow with the regularization in Eq. (20). As described in Sections 4.1 and 4.2, the relation between MAC×bit and the inference time is used to estimate the target MAC×bit value from the target inference time. The pre-trained weights at full precision are used as the initial values for optimization.

The initial value for the quantization step size (s_{init}) is determined by using the distribution of pre-trained weights:

$$s_{\mathrm{init}} = 2\mathrm{mean}(\mathrm{abs}(\mathbf{W}))/\sqrt{Q_{\mathrm{p}}}, \tag{21}$$

where mean$(\cdot)$ returns the mean value of the arguments and Q_{p} denotes the number of quantization bins in the positive region of the weight values. Q_{p} is given by

$$Q_{\mathrm{p}} = 2^{b_{\mathrm{init}}-1} - 1, \tag{22}$$

where b_{init} is the initial bit width before optimization (e.g., 8 bits). This initialization for the quantization step size, as given by Eqs. (21) and (22), was introduced heuristically by Esser et al. [18]. Larger values of b_{init} produce larger Q_{p}, smaller s_{init}, and smaller differences between the de-quantized and original weights (i.e., a smaller initial quantization error).

In forward operations, the convolutional layers are executed with de-quantized weights while other layers such as the batch normalization, ReLU, and fully connected layers are executed at full precision, as described in lines 6–13 of Algorithm 1. To precisely update $\mathbf{W}$ and s, $\mathbf{W}$ and s are stored at full precision during optimization. The optimization iteratively updates $\mathbf{W}$ and s so that MAC×bit becomes less than the target value, as described in lines 16–20 of Algorithm 1. The optimal bit allocation is obtained from the optimized $\mathbf{W}$ and s by using Eq. (11).

As an example, Fig. 6 shows transition curves for the training loss $\mathcal{L}$ and MAC×bit during pre-training and optimization using the ResNet-18 model and ImageNet dataset. The model was pre-trained for 110 epochs and optimized for 80 epochs. The blue plots indicate the curves for pre-training with floating-point 32 bits, and the red plots indicate those for optimization. The orange line shows the target MAC×bit value (i.e.,

Algorithm 1 Optimization with computational-complexity-aware regularization

Input: Initial $\mathbf{W}$ and b, target MAC×bit value, and training dataset (input images and labels)
Output: Optimized $\mathbf{W}$, s, and b

```
1:  // Description for variables
2:  // I is the number of optimization iterations.
3:  // L is the number of layers.
4:  Initialize s with initial W and b                     ▷ Eqs. (21) & (22)
5:  for i = 1 to I do
6:      for l = 1 to L do
7:          if Function of the l-th layer is convolution then
8:              Quantize W_{f,l} by s_{f,l}                   ▷ Eqs. (7)–(9)
9:              Compute output values with de-quantized values
10:         else
11:             Compute output values with full-precision values
12:         end if
13:     end for
14:     Compute loss L
15:     Compute MAC×bit                              ▷ Eqs. (4), (5) & (11)
16:     if MAC×bit is larger than the target value then
17:         Add λMAC×bit to L                                   ▷ Eq. (20)
18:     end if
19:     Backward L and calculate gradients
20:     Update W and s
21: end for
22: Compute b with optimized W and s                           ▷ Eq. (11)
```

6.5×10^9 here). Fig. 6(c) shows an enlarged graph of the MAC×bit curve from 105 to 130 epochs in Fig. 6(b). MAC×bit decreased to the target value of 6.5×10^9 with our proposed optimization.

5. Experiments

5.1 Experimental setting

We conducted experiments to evaluate the performance of our proposed method. We employed the PyTorch library to implement our proposed method. We used VGG7 [1], ResNet-18, ResNet-50, and MobileNetV2 as CNNs and evaluated their performance on the STL-10 [52] and ImageNet datasets. To evaluate the inference time, we used the dedicated accelerator for mixed-precision computing proposed by Maki et al. [26]. For comparison, we also evaluated the performance of models optimized by Uhlich et al.'s conventional regularization method based on model size [20].

The bit width for weights was set to be variable from 1 to 8. Based on the hardware specifications [26], the bit width for activation was fixed at 8. We referred to Sasaki et al. [11] and discretized the activations into 256 ($= 2^8$) values by fixing the quantization step size for activation (s_a) at

$$s_a = (V_{\max} - V_{\min})/2^8, \tag{23}$$

where $V_{\max}$ and $V_{\min}$ are the maximum and minimum values, respectively, of activation in the first mini-batch.

The CNNs were pre-trained in 32 bits by using SGD with momentum. Using the pre-trained models, we set $b_{\text{init}} = 8$ and initialized the quantization step size for weights with Eqs. (21) and (22). We used the target MAC×bit value estimated from the target inference time to determine the optimal bit allocation as given in Algorithm 1. Table 2 summarizes the hyperparameter settings for the pre-training and optimization. The hyperparameters were set to typical values in the literature.

5.2 Experimental results

Fig. 7 shows the relation between the top-1 accuracy for the ImageNet dataset and the simulated inference time of the ResNet-18 and ResNet-50 models optimized by both the proposed and conventional methods. We evaluated the simulated inference time when inference of the optimized models was executed on the dedicated accelerator [26]. Models optimized by the proposed method achieved a shorter inference time with the same level of top-1 accuracy as models optimized by the conventional method (e.g., 21.0% reduction of inference time at a top-1 accuracy of 70%, as shown in Fig. 7(a)) and a better top-1 accuracy at the same level of inference time (e.g., 0.77-point improvement in accuracy at an inference time of less than 350 ms, as shown in Fig. 7(b)).

We compared the average bit allocations of the

Table 2: Summary of pre-training and optimization settings (©2021 IEEE [21])

Pre-training

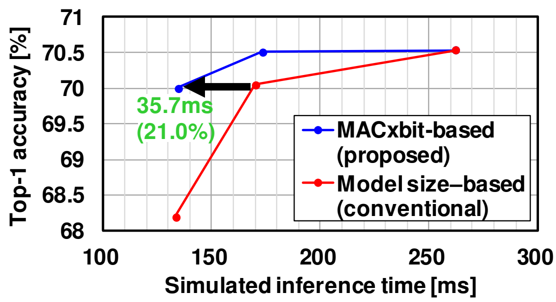
Dataset	STL-10	ImageNet		
Network Architecture	VGG7	ResNet -18	ResNet -50	MobileNet V2
Optimizer	Momentum SGD (momentum: 0.9)	Momentum SGD (momentum: 0.9)		
Initial LR*1	0.1	0.1		
LR Schedule	Divided by 10 at 200, 300th epoch	Decayed with cosine annealing without restart[53]		
Weight decay	5×10^{-4}	1×10^{-4}		
Batch size	128	256	64	256
Epoch	400	110		

*1 LR: Learning Rate

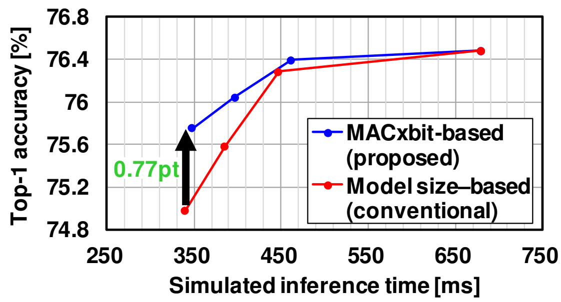
Optimization

λ	1×10^{-10}	1×10^{-10}		
Optimizer	Momentum SGD (momentum: 0.9)	Momentum SGD (momentum: 0.9)		
Initial LR*2	0.001	2×10^{-5}		
LR Schedule	Divided by 10 at 100, 150th epoch	Divided by 10 at 40, 60th epoch		
Weight decay	5×10^{-4}	1×10^{-4}		
Batch size	128	256	64	256
Epoch	200	80		

*2 equivalent to the final LR value during the pre-training

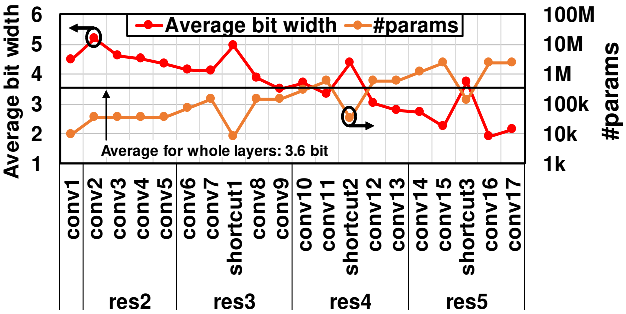


(a) ResNet-18

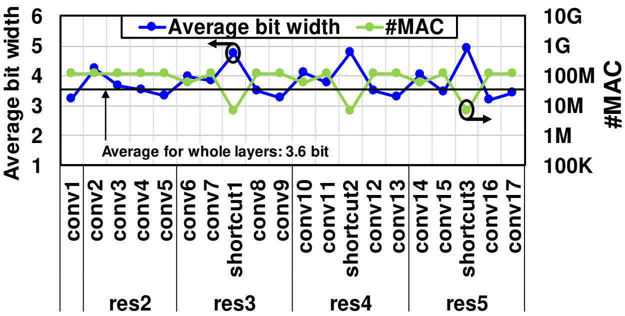


(b) ResNet-50

Fig. 7: Top-1 accuracy plotted against the simulated inference time for (a) ResNet-18 and (b) ResNet-50 models optimized by the proposed method based on MAC×bit and conventional method based on model size. (©2021 IEEE [21])



(a) Average bit widths and #params for each layer.



(b) Average bit widths and #MAC for each layer.

Fig. 8: Relation between the average bit width of weights and (a) #params and (b) #MAC for each layer of the optimized ResNet-18 models. The inference times were reduced to 135 ms by (a) the conventional optimization method based on model size and (b) the proposed optimization method based on MAC×bit.

models optimized by the proposed and conventional methods at the same level of inference time. Fig. 8 shows the average bit widths of weights for each layer of the optimized ResNet-18 models for which the inference time was reduced to 135 ms. Figs. 8(a) and 8(b) show #params and #MAC, respectively, for each layer. Although the inference times of both models were almost identical at 135 ms, the bit allocations differed depending on the optimization method.

The conventional optimization method (Eq. (1)) reduced the bit width for layers with larger #params to reduce the model size, as given in Eq. (2). As a result, smaller bit widths (close to 2 bits) were allocated to the later layers with large #params, except for the shortcut layers for the residual paths of the ResNet-18 model.

On the other hand, our proposed optimization method (Eq. (20)) reduced the bit width for layers with larger #MAC to reduce MAC×bit, as given in Eq. (4). In the ResNet-18 architecture, because #MAC was almost uniform except for the shortcut layers, as shown in Fig. 8(b), the resultant bit allocations were almost uniform (3–4 bits).

Table 3: Performance comparison (©2021 IEEE [21])

STL-10 **VGG7**	Test accuracy [%]	Average bit width	MAC×bit [$\times 10^9$]	Simulated inference time [ms]
Baseline	84.2	8.0	11.0	114.9
Model size–based [20]	83.6	2.5	3.5	46.0
MAC×bit-based (**ours**)	**83.8**	**2.2**	**3.0**	**41.7**
ImageNet **ResNet-18**	Top-1 accuracy [%]	Average bit width	MAC×bit [$\times 10^9$]	Simulated inference time [ms]
Baseline	70.5	8.0	14.5	261.9
Model size–based [20]	70.1	5.0	9.0	169.9
MAC×bit-based (**ours**)	70.0	**3.6**	**6.5**	**134.2**
ImageNet **ResNet-50**	Top-1 accuracy [%]	Average bit width	MAC×bit [$\times 10^9$]	Simulated inference time [ms]
Baseline	76.5	8.0	32.7	679.7
Model size–based [20]	75.6	4.3	17.5	385.5
MAC×bit-based (**ours**)	**75.8**	**3.6**	**14.8**	**325.9**
ImageNet **MobileNetV2**	Top-1 accuracy [%]	Average bit width	MAC×bit [$\times 10^9$]	Simulated inference time [ms]
Baseline	70.8	8.0	2.13	50.2
Model size–based [20]	69.5	4.5	1.20	35.1
MAC×bit-based (**ours**)	**70.0**	**4.3**	**1.14**	**33.9**

Baseline indicates the performance of models without the optimizations.

Table 3 summarizes the performances of the optimized models, including test/top-1 accuracy, average bit width, MAC×bit, and simulated inference time. Models optimized by the proposed method achieved the same level of accuracy as models optimized by the conventional method with shorter inference times and achieved better accuracy at the same level of inference time. For example, the ResNet-50 model optimized by the proposed method achieved a better accuracy of 75.8% with a 15.5% reduction in inference time of 325.9 ms compared to when it was optimized by the conventional method.

6. Conclusion

The conventional gradient descent–based method for the optimization of quantized CNNs focuses on the model size, but it does not directly consider computational costs such as the inference time. In this paper, we demonstrated that the conventional method is not always ideal for reducing inference time, based on the measurement results showing no correlation between model size and inference time. We presented the computational-complexity metric MAC×bit, which is strongly correlated with inference time. We used this metric to propose a regularization method for optimizing the bit allocation of quantized CNNs considering both the inference time and recognition accuracy. We empirically showed that models optimized by the proposed method achieved better accuracy with a shorter inference time than those optimized by the conventional method based on model size.

Acknowledgment

The authors thank Asuka Maki, Shinichi Sasaki, Radu Berdan, and Yasuto Hoshi for their support.

References

[1] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," International Conference on Learning Representations (ICLR), 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.

[3] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," IEEE/CVF International Conference on Computer Vision (ICCV), pp.2980–2988, 2017.

[4] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," arXiv:1602.02830, 2016.

[5] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks," European Conference on Computer Vision (ECCV), pp.373–390, 2018.

[6] Y. Zhou, S. Moosavi-Dezfooli, N. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," AAAI Conference on Artificial Intelligence, pp.4596–4604, 2018.

[7] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation," arXiv:2004.09602, 2020.

[8] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," IEEE Micro, vol.41, no.2, pp.29–35, 2021.

[9] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision," IEEE Journal of Solid-State Circuits, vol.54, no.1, pp.173–185, 2019.

[10] J.S. Park, C. Park, S. Kwon, H.S. Kim, T. Jeon, Y. Kang, H. Lee, D. Lee, J. Kim, Y. Lee, S. Park, J.W. Jang, S. Ha, M. Kim, J. Bang, S.H. Lim, and I. Kang, "A Multi-Mode 8K-MAC HW-Utilization-Aware Neural Processing Unit with a Unified Multi-Precision Datapath in 4nm Flagship Mobile SoC," IEEE International Solid- State Circuits Conference (ISSCC), pp.246–248, 2022.

[11] S. Sasaki, A. Maki, D. Miyashita, and J. Deguchi, "Post Training Weight Compression with Distribution-based Filter-wise Quantization Step," IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), pp.1–3, 2019.

[12] L. Zeng, Z. Wang, and X. Tian, "KCNN: Kernel-wise Quantization to Remarkably Decrease Multiplications in Convolutional Neural Network," International Joint Conference on Artificial Intelligence (IJCAI), pp.4234–4242, 2019.

[13] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M.W. Mahoney, and K. Keutzer, "ZeroQ: A Novel Zero Shot Quantization Framework," IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.13166–13175, 2020.

[14] Z. Dong, Z. Yao, A. Gholami, M.W. Mahoney, and K. Keutzer, "HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision," IEEE/CVF International Conference on Computer Vision (ICCV), pp.293–302, 2019.

[15] Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M.W. Mahoney, and K. Keutzer, "HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks," Advances in Neural Information Processing Systems (NeurIPS), pp.18518–18529, 2020.

[16] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.8604–8612, 2019.

[17] Q. Lou, F. Guo, M. Kim, L. Liu, and L. Jiang., "AutoQ: Automated Kernel-Wise Neural Network Quantization," International Conference on Learning Representations (ICLR), 2020.

[18] S.K. Esser, J.L. McKinstry, D. Bablani, R. Appuswamy, and D.S. Modha, "Learned Step Size Quantization," International Conference on Learning Representations (ICLR), 2020.

[19] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks," in Proceedings of Machine Learning and Systems (MLSys), pp.112–128, 2020.

[20] S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshiyama, J.A. Garcia, S. Tiedemann, T. Kemp, and A. Nakamura, "Mixed Precision DNNs: All you need is a good parametrization," International Conference on Learning Representations (ICLR), 2020.

[21] K. Nakata, D. Miyashita, J. Deguchi, and R. Fujimoto, "Adaptive Quantization Method for CNN with Computational-Complexity-Aware Regularization," IEEE International Symposium on Circuits and Systems (ISCAS), pp.1–5, 2021.

[22] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in Advances in Neural Information Processing Systems (NeurIPS), 2019.

[23] A.T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "ReLeQ : A Reinforcement Learning Approach for Automatic Deep Quantization of Neural Networks," IEEE Micro, vol.40, no.5, pp.37–45, 2020.

[24] Asuka Maki and Daisuke Miyashita and Shinichi Sasaki and Kengo Nakata and Fumihiko Tachibana and Tomoya Suzuki and Jun Deguchi and Ryuichi Fujimoto, "Weight compression mac accelerator for effective inference of deep learning," IEICE Transactions on Electronics, vol.E103-C, no.10, pp.514–523, 2020.

[25] K. Nakata, D. Miyashita, A. Maki, F. Tachibana, S. Sasaki, J. Deguchi, and R. Fujimoto, "Quantization Strategy for Pareto-optimally Low-cost and Accurate CNN," IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp.1–4, 2021.

[26] A. Maki, D. Miyashita, K. Nakata, F. Tachibana, T. Suzuki, and J. Deguchi, "FPGA-based CNN Processor with Filter-Wise-Optimized Bit Precision," IEEE Asian Solid-State Circuits Conference (A-SSCC), pp.47–50, 2018.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," arXiv:1603.05027, 2016.

[28] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.248–255, 2009.

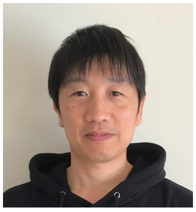
[29] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," arXiv:1812.00090, 2018.

[30] Z. Cai and N. Vasconcelos, "Rethinking differentiable search for mixed-precision neural networks," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.2346–2355, 2020.

[31] M. van Baalen, C. Louizos, M. Nagel, R.A. Amjad, Y. Wang, T. Blankevoort, and M. Welling, "Bayesian bits: Unifying quantization and pruning," Advances in Neural

Information Processing Systems (NeurIPS), pp.5741–5752, 2020.
[32] L. Yang and Q. Jin, "Fracbits: Mixed precision quantization via fractional bit-widths," AAAI Conference on Artificial Intelligence, pp.10612–10620, 2021.
[33] C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A.M. Bronstein, and A. Mendelson, "Uniq: Uniform noise injection for non-uniform quantization of neural networks," ACM Trans. Comput. Syst., vol.37, no.1–4, 2021.
[34] B. Hawks, J. Duarte, N.J. Fraser, A. Pappalardo, N. Tran, and Y. Umuroglu, "Ps and qs: Quantization-aware pruning for efficient low latency neural network inference," Frontiers in Artificial Intelligence, vol.4, 2021.
[35] Z. Liu, Y. Wang, K. Han, S. Ma, and W. Gao, "Instance-aware dynamic neural network quantization," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.12434–12443, 2022.
[36] C. Hong, S. Baik, H. Kim, S. Nah, and K.M. Lee, "Cadyq: Content-aware dynamic quantization for image super-resolution," European Conference on Computer Vision (ECCV), pp.367–383, 2022.
[37] C. Tang, K. Ouyang, Z. Chai, Y. Bai, Y. Meng, Z. Wang, and W. Zhu, "Seam: Searching transferable mixed-precision quantization policy through large margin regularization," ACM International Conference on Multimedia, p.7971–7980, 2023.
[38] Y. Umuroglu, L. Rasnayake, and M. Sjalander, "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing," International Conference on Field Programmable Logic and Applications (FPL), pp.307–3077, 2018.
[39] Y. Wang, G. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU Platforms for Deep Learning," arXiv:1907.10701, 2019.
[40] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), p.1–12, 2017.
[41] A. Castello, M.F. Dolz, E.S. Quintana-Orti, and J. Duato, "Theoretical Scalability Analysis of Distributed Deep Convolutional Neural Networks," IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp.534–541, 2019.
[42] A. Canziani, A. Paszke, and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," arXiv:1605.07678, 2016.
[43] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures," IEEE Access, vol.6, pp.64270–64277, 2018.
[44] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for All: Train One Network and Specialize it for Efficient Deployment," International Conference on Learning Representations (ICLR), 2020.
[45] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," European Conference on Computer Vision (ECCV), pp.544–560, 2020.
[46] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," IEEE International Conference on Computer Vision (ICCV), pp.2755–2763, 2017.
[47] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H.P. Graf, "Pruning Filters for Efficient ConvNets," International Conference on Learning Representations (ICLR), 2017.
[48] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," International Conference on Learning Representations (ICLR), 2020.
[49] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.4510–4520, 2018.
[50] Y. Bengio, N. Léonard, and A.C. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," arXiv:1308.3432, 2013.
[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch," NIPS 2017 Workshop on Autodiff, 2017.
[52] A. Coates, A. Ng, and H. Lee, "An Analysis of Single-Layer Networks in Unsupervised Feature Learning," International Conference on Artificial Intelligence and Statistics (AISTATS), pp.215–223, 2011.
[53] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Restarts," arXiv:1608.03983, 2016.

Kengo Nakata received the B.E. degree in electrical and electronic engineering and the M.E. degree in physical electronics from the Tokyo Institute of Technology, Tokyo, Japan, in 2014 and 2016, respectively. In 2016, he joined the Center for Semiconductor Research and Development, Toshiba Corporation, Kawasaki, Japan, where he was involved in the design of analog circuits for wireless communications. In 2017, he joined Kioxia Corporation, Yokohama, Japan. His current research interest includes efficient hardware architecture and algorithm for machine learning applications.



Daisuke Miyashita received the B.S. and M.S. degrees in electronic engineering from the University of Tokyo, Tokyo, Japan, in 2001 and 2003, respectively. In 2003, he joined Center for Semiconductor Research and Development, Toshiba Corporation, Kawasaki, Japan, where he was engaged in the design of RF and analog circuits for wireless communications. He is now with System Technology Research & Development Center, Kioxia Corporation, Yokohama, Japan. His research interests include efficient mixed-signal/digital hardware and system for AI applications and algorithms designed for such hardware and system. From 2015 to 2016, he was a visiting scholar at Stanford University, Stanford, CA, USA, where he has engaged in the research on the efficient implementation of deep learning algorithms on hardware.

Jun Deguchi received the B.E. and M.E. degrees in machine intelligence and systems engineering and the Ph.D. degree in bioengineering and robotics from Tohoku University, Sendai, Japan, in 2001, 2003, and 2006, respectively. In 2004, he was a Visiting Scholar at the University of California, Santa Cruz, CA, USA. In 2006, he joined Toshiba Corporation, and was involved in design of analog/RF circuits for wireless communications, CMOS image sensors, high-speed I/O, and accelerators for deep learning. From 2014 to 2015, he was a Visiting Scientist at the MIT Media Lab, Cambridge, MA, USA, and was involved in research on brain/neuro science. In 2017, he moved to Kioxia Corporation (formerly Toshiba Memory Corporation), and is serving as the group manager of two research teams: one is working on circuit designs of high-speed I/O for Flash Memory and SSD, and the other is working on technology for AI from algorithms to circuit designs. Dr. Deguchi has served as a member of the international technical program committee (TPC) of IEEE Asian Solid-State Circuits Conference (A-SSCC) since 2017. He has also served as a TPC member of IEEE International Solid-State Circuits Conference (ISSCC) from 2016 to 2023, a Far-East chair of IEEE ISSCC 2023, a TPC vice-chair of IEEE A-SSCC 2019, a guest editor of IEEE Journal of Solid-State Circuits (JSSC) for the special issues on IEEE A-SSCC 2020, IEEE ISSCC 2020 and IEEE ISSCC 2021. He has also been a review committee member of IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) 2020.

Ryuichi Fujimoto (Senior Member) received his B.E., M.E., and Dr. Eng. Degrees from Waseda University, Tokyo, Japan, in 1988, 1990, and 2003, respectively. He joined the Mobile Communication Laboratory, Corporate Research and Development Center, Toshiba Corporation, Kawasaki, Japan, in 1991. Since then, he has been engaged in the research and development of analog integrated circuits and device models for wireless communication systems. In 2005, he was transferred to Wireless & Multimedia LSI Development Department, Toshiba Corp. Semiconductor Company. From 2009 to 2011, he was on loan to Semiconductor Technology Academic Research Center (STARC). Currently, he is with System Technology Research & Development Center, Kioxia Corporation. Dr. Fujimoto was an Associate Editor of IEICE Transactions on Electronics from 2001 to 2004, IEICE Electronics Express (ELEX) from 2003 to 2008, and IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences from 2005 to 2009. He was the secretary of the Japan Chapter of IEEE Circuits and Systems Society from 2008 to 2009 and the chair of Japan Chapter of IEEE Solid-State Circuit Society from 2019 to 2020. He is an organizing committee co-chair of A-SSCC 2024. He is a member of the IEEE, IEEJ, JIEP and JAAS.