IEICE TRANS. ELECTRON., VOL.E103–C, NO.10 OCTOBER 2020

PAPER *Special Section on Analog Circuits and Their Application Technologies*

Weight Compression MAC Accelerator for Effective Inference of Deep Learning

Asuka MAKI[†a)], **Daisuke MIYASHITA**[†], **Shinichi SASAKI**[†], **Kengo NAKATA**[†], **Fumihiko TACHIBANA**[†], **Tomoya SUZUKI**[†], **Jun DEGUCHI**[†], *Nonmembers*, *and* **Ryuichi FUJIMOTO**[†], *Senior Member*

SUMMARY Many studies of deep neural networks have reported inference accelerators for improved energy efficiency. We propose methods for further improving energy efficiency while maintaining recognition accuracy, which were developed by the co-design of a filter-by-filter quantization scheme with variable bit precision and a hardware architecture that fully supports it. Filter-wise quantization reduces the average bit precision of weights, so execution times and energy consumption for inference are reduced in proportion to the total number of computations multiplied by the average bit precision of weights. The hardware utilization is also improved by a bit-parallel architecture suitable for granularly quantized bit precision of weights. We implement the proposed architecture on an FPGA and demonstrate that the execution cycles are reduced to 1/5.3 for ResNet-50 on ImageNet in comparison with a conventional method, while maintaining recognition accuracy.

key words: *deep learning, convolutional neural network, quantization, variable bit width, post-training, inference, accelerator, processor, FPGA*

1. Introduction

Deep convolutional neural networks (CNN) have proven effective in a wide variety of applications, including image recognition [2], [5], [7], object detection [12], and semantic segmentation [4]. With the rapid increase in research on algorithms for various applications, the importance of hardware-related research is also growing, because energy efficient hardware is essential for executing algorithms involving huge numbers of computations under practical power consumption and execution times.

Algorithms employing quantization of weights and activations have attracted attention for improving the energy efficiency for CNN inference. These algorithms are best suited to massively parallel computing because they reduce the number of computations without worsening computational intensity, unlike approaches such as pruning or separable convolution [6]. Although 1 bit [11], 2 bit [3], and other extremely low-bit quantization approaches are attractive in terms of energy efficiency, those approaches significantly degrade recognition accuracy. For example, compared with the full-precision AlexNet [7] with top-5 accuracy of 83% on ImageNet [2], accuracy degradations of 13.8% and 9.9% have been reported for 1-bit and 2-bit quantization, respectively. More than 8-bit precision is generally required in order to maintain accuracy with a uniform bit width.

Recent works have revealed that layer-wise optimized quantization is highly effective for reducing the total number of bits [9], [15]. In practice, however, the required bit width for computations that maintain accuracy varies not only among layers but also among pixels (i.e. spatial positions), channels, and CNN filters. Therefore, we propose an algorithm that further reduces the number of computations by applying filter-wise bit precision, i.e. granularly controlling bit widths of weights filter by filter.

Although 4-bit or lower quantization usually requires retraining or quantization-aware training [22]–[24] to maintain higher accuracy, it is sometimes difficult to obtain a labeled dataset for retraining due to security and/or privacy reasons [19]. In this paper, therefore, we also propose a post-training quantization scheme that quantizes weights to 4- or fewer bits while maintaining test accuracy as high as possible without a labeled dataset for retraining.

Dedicated hardware is required to exploit the granularly optimized bit precision. Ideally, energy efficiency is improved or the number of execution cycles is reduced in proportion to the product of the number of bits and the number of operations. However, this cannot be achieved by using conventional hardware architectures. Therefore, we propose a hardware architecture suited to filter-wise quantization. To improve the energy efficiency for CNN inference, we co-designed an algorithm and its supporting hardware architecture.

The remainder of this paper is organized as follows. Section 2 proposes some techniques to improve energy efficiency while maintaining recognition accuracy, and presents details of the proposed quantization algorithm and other techniques for reducing computations. Section 3 describes a proposed hardware architecture that is suited to filter-wise bit precision. Sections 4 and 5 present the FPGA implementation and experimental results, respectively, and Sect. 6 provides our conclusion.

2. Techniques to Improve Energy Efficiency with Maintaining Accuracy

We propose a post-training quantization scheme that quantizes weights into 4 bits or lower while maintaining the highest possible test accuracy without use of a labeled dataset for retraining.

Manuscript received November 22, 2019.
Manuscript revised February 19, 2020.
Manuscript publicized May 15, 2020.

[†]The authors are with Kioxia Corporation, Kawasaki-shi, 212–8520 Japan.
a) E-mail: asuka.maki@kioxia.com

DOI: 10.1587/transele.2019CTP0007

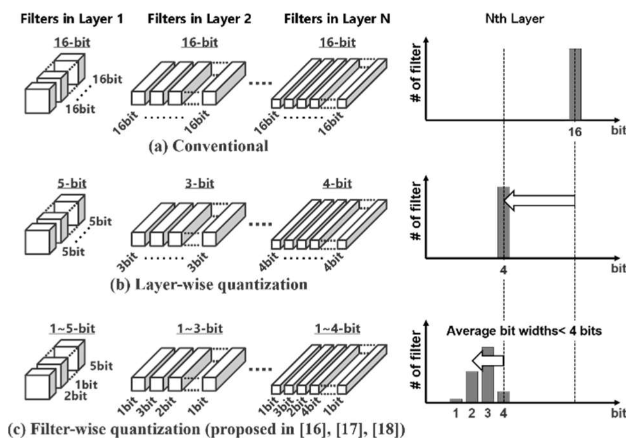
Filters in Layer 1
Filters in Layer 2
Filters in Layer N
Nth Layer
(a) Conventional
(b) Layer-wise quantization
(c) Filter-wise quantization (proposed in [16], [17], [18])
# of filter
bit
Average bit widths< 4 bits

Fig. 1 Weight-quantization techniques (©2019 IEEE [18]).

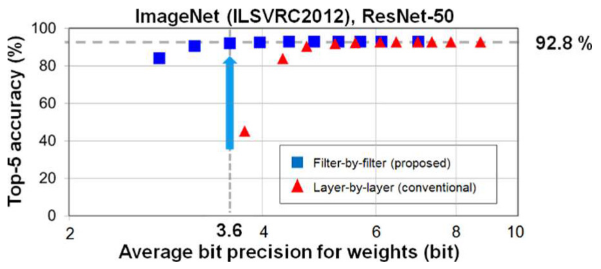

Fig. 2 Top-5 accuracy of ResNet-50 on ImageNet vs. average bit precision when all convolutional layers are quantized. Activations are 8-bit quantized (©2018 IEEE [16]).

2.1 Proposed Filter-Wise Quantization

2.1.1 Concept of Filter-Wise Quantization

We propose a filter-wise quantization that quantizes weights with variable bit precision according to the filter. Using this quantization, we can further reduce the number of computations while maintaining recognition accuracy.

CNNs have many layers, and each layer has many filters as shown in Fig. 1. Figure 1 (a) shows a conventional quantization where all weights are quantized to 16 bits, meaning all calculations are executed with 16-bit precision. Figure 1 (b) shows a layer-wise, or layer-by-layer, quantization. In this case, the weights are quantized with variable bit precision depending on layers, thus reducing the number of computations. To further reduce the number of computations, we proposed a filter-wise quantization algorithm [16]–[18], [21] that assigns variable bit precision for weights filter by filter, as shown in Fig. 1 (c).

The right side in Fig. 1 shows distributions of bit precision in one layer. In the case of layer-wise quantization, since the bit precision in any one layer is uniform, all the weights in the layer are represented with, for instance, 4-bit precision, as shown in the right graph in Fig. 1 (b). On the other hand, the bit precision distribution in a layer with filter-wise quantization spreads as shown in Fig. 1 (c). For quantitatively comparing differences in the bit precision distributions of the quantization methods shown in Fig. 1 (a)–(c), we define the average bit precision as

$$\text{Average bit precision} = \frac{\sum_{l=0}^{L-1}\sum_{m_l=0}^{M_l-1} b_{m_l} \times \widehat{N}_{ml}}{\sum_{l=0}^{L-1}\sum_{m_l=0}^{M_l-1} N_{ml}}, \tag{1}$$

where L is the number of layers, M_l is the number of filters in layer l, b_{ml} is the bit precision for weights in filter m_l, $\widehat{N}_{ml}$ is the actual number of computations for filter m_l, and N_{ml} is the original number of computations for filter m_l. $\widehat{N}_{ml}$ can be smaller than N_{ml} by applying Winograd convolution as will be explained in Sect. 2.2.2. For the quantization methods in Fig. 1 (a) and (b), the average bit precision for Nth layer is 16-bit and 4-bit, respectively. However, the average bit precision under the filter-wise quantization shown in Fig. 1 (c) is smaller than the 4 bits of the layer-wise quantization method shown in Fig. 1 (b).

Figure 2 shows the relationship between top-5 accuracy for ResNet-50 [5] on ImageNet and average bit precision. The horizontal dashed line at 92.8 % shows the accuracy when all weights are quantized to 16-bit precision, which is actually the same accuracy as before quantization. The average bit precisions for the layer-wise and filter-wise quantization methods are respectively shown as red and blue symbols in Fig. 2. The average bit precision for the filter-wise quantization is reduced to 3.6 bits with the top-5 accuracy of 92.8 %, whereas the accuracy obtained using layer-wise quantization is decreased to below 50 % at 3.6-bit precision. These results verify that the filter-wise quantization can reduce the average bit precision while better maintaining accuracy than does layer-wise quantization.

We applied the proposed filter-wise quantization to other neural networks for CIFAR-10 and ImageNet to confirm the applicability of our proposed scheme (Fig. 3). We can apply our proposed method to quantization-aware training like LQ-Net [23] and ABC-Net [24], but this is left to future work. Here, we apply our filter-wise quantization to post-training for comparison. In each evaluation, filter-wise quantization compresses the average bit precisions much more than does layer-wise quantization. Figure 2 shows the results for ResNet-50 on ImageNet. As Fig. 3 (a) shows, Wide ResNet in particular maintains recognition accuracy at lower bit precisions, indicating that the Wide ResNet structure is redundant for CIFAR-10. We can thereby estimate an appropriate network size.

2.1.2 Readjustment of Quantization Step Δ

We introduce a distribution-based filter-wise quantization scheme for maintaining accuracy. The quantization step Δ is readjusted for effective use of the reduced bit width.

As shown in Fig. 4 (a), we assume the quantization step Δ was initially set such that Filter 1 weights are 4-bit quantized. When applying layer-wise quantization, the weights for all other filters in the same layer are also covered within 16 bins. This is adequate for the Filter1 in Fig. 4 (a), but weights in Filters 2 and 3 may sit within only 6 and 4

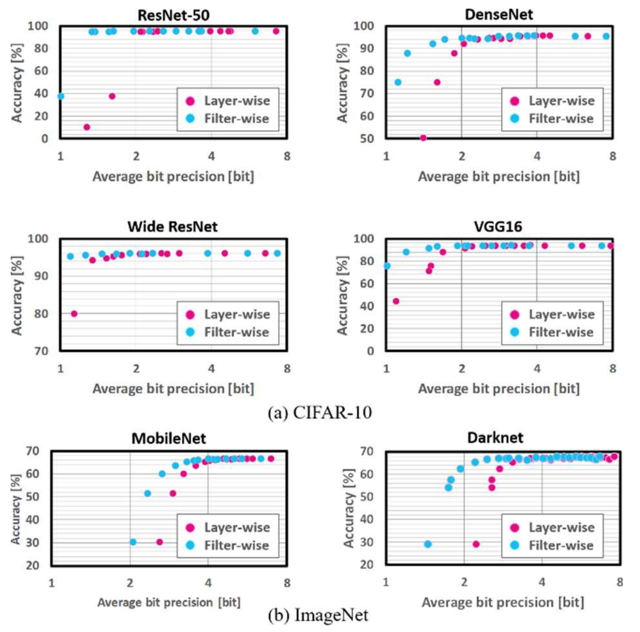

Fig. 3 Top-1 accuracy vs. average bit precision of several neural networks on (a) CIFAR-10 and (b) ImageNet (©2019 IEEE [18]).

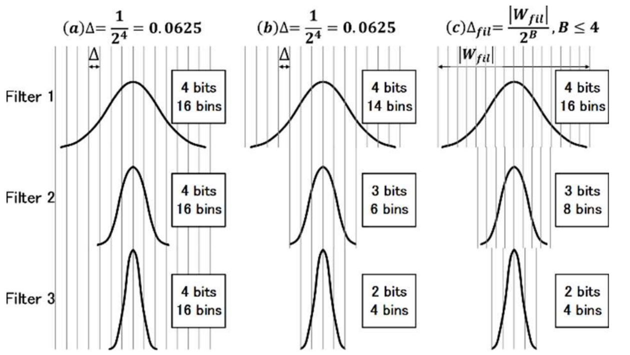
$(a)\Delta=\frac{1}{2^4}=0.0625$
$(b)\Delta=\frac{1}{2^4}=0.0625$
$(c)\Delta_{fil}=\frac{|W_{fil}|}{2^B}, B\le 4$
$|W_{fil}|$
Filter 1
Filter 2
Filter 3
4 bits 16 bins
4 bits 14 bins
3 bits 6 bins
2 bits 4 bins
3 bits 8 bins

Fig. 4 (a) Layer-wise quantization, (b) filter-wise quantization, and (c) after readjustment (©2019 IEEE [17]).

consecutive bins, respectively. In that situation, we individually assign appropriate bit widths for weights in each filter, using the filter-wise quantization described in Sect. 2.1.1. As shown in Fig. 4 (b), we can represent the weights for Filters 2 and 3 with 3 and 2 bits, respectively, instead of 4. The bit width for each filter is determined in this manner according to the weight distribution, and the resultant average bit precision becomes smaller than 4 bits.

After determining the bit precision of each filter by the procedure described above, we readjust Δ to eliminate the underutilization. In the middle of Fig. 4 (b), for example, the weights are in 6 bins but represented with 3 bits. To eliminate this wastefulness and fully exploit the 3-bit representation for higher accuracy, we readjust Δ to divide the weight distribution in the Filter 2 into 8 bins, as shown in Fig. 4 (c).

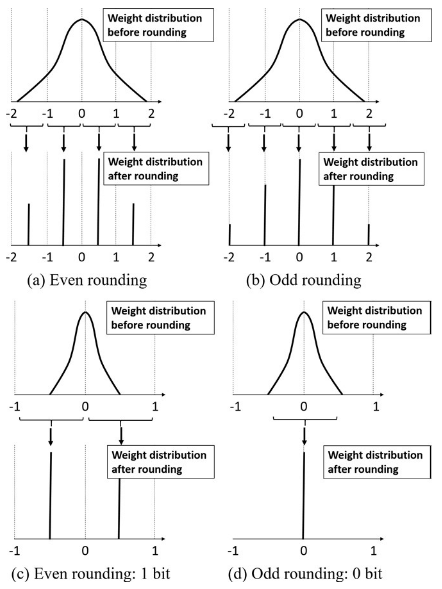

Fig. 5 Even and odd rounding for weight quantization.

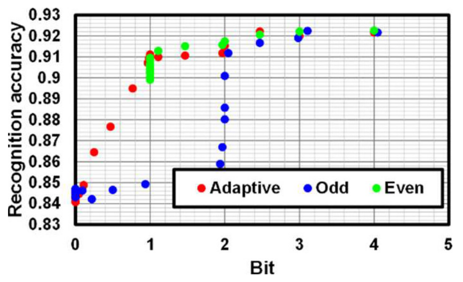

Fig. 6 Rounding type for quantization. Recognition accuracy vs. average bit precision for the 18th layer of ResNet-20 on CIFAR-10. Filter-wise quantization is applied only to the 18th layer. Weights of other layers are not quantized.

2.1.3 Adaptive Rounding Method

For adequate quantization of the weights, we introduce a rounding method we call “adaptive rounding.” Both even and odd rounding, respectively shown by Eqs. (2) and (3), are used for adaptive rounding, depending on the relationship between the quantization step Δ and the distribution range of the weights.

$$Q(W,\Delta) = floor\left(\frac{W}{\Delta}\right) + 0.5\ (even\ rounding) \tag{2}$$

$$Q(W,\Delta) = floor\left(\frac{W}{\Delta} + 0.5\right)\ (odd\ rounding) \tag{3}$$

Here, W is the weight value, Δ is the quantization step, and $Q(W,\Delta)$ is the normalized weight value after rounding. Examples of the even and odd rounding methods are shown in Fig. 5 (a) and (b), respectively.

Figure 6 shows the relationship between the bit precision of various rounding methods and recognition accuracy, calculated under various quantization steps Δ. Filter-wise quantization is applied to the 18th layer in ResNet-20 on

```
Input: $\Delta$ | $Quantization\ step$
       $M = \{W\}$ | $Full\ precision\ DNN\ model$
Output: $m_q = \left\{\frac{W_q}{\Delta_{adj}}\right\}$ | Normalized and quantized model
Algorithm: Distribution-based filter-wise quantization:
  1. $\widehat{W} \leftarrow Q(W, \Delta)$   // Pre-quantize
  2. $W_{val} \leftarrow max(\widehat{W}) - min(\widehat{W}) + 1$   // # of bins
  3. $B \leftarrow ceil(log_2(W_{val}))$   // B | Bit width of each filter
  4. $\Delta_{adj} \leftarrow \frac{max(W) - min(W)}{2^B}$   // Readjusted quantized step
  5. $\frac{W_q}{\Delta_{adj}} \leftarrow Q(W, \Delta_{adj})$   // Normalized and quantized model
Adaptive rounding:
```

$$Q(W,\Delta) = \begin{cases} floor\left(\frac{W}{\Delta}\right) + 0.5, & \Delta < max(W) - min(W) \\ floor\left(\frac{W}{\Delta} + 0.5\right), & \Delta \geq max(W) - min(W) \end{cases}$$

Fig. 7 Proposed quantization algorithm.

CIFAR-10, and weights in the other layers are not quantized. The horizontal axis in Fig. 6 indicates the average bit precision of the weights in the 18th layer.

When the weight distribution is sufficiently larger than Δ, that is, when the number of bits is large, recognition accuracies using either the odd or even rounding method are almost the same; since the quantization step Δ is sufficiently small, differences in the two rounding methods are not dominant for recognition accuracy. With odd rounding, recognition accuracies rapidly degrade at around 2 bits. However, even rounding requires at least two bins (Fig. 5 (c)), so it cannot represent less than 1 bit, as shown in Fig. 6.

To achieve adequate rounding at a wide range of bit precisions, we developed adaptive rounding that uses even rounding when Δ is smaller than the weight distribution and the odd rounding when Δ exceeds the weight distribution. As shown in Fig. 6, even rounding achieves higher accuracy in the 1-2 bit range, so we adopt even rounding when Δ is smaller than the weight distribution. Because even rounding cannot represent less than 1 bit, as previously mentioned, we adopt odd rounding as shown in Fig. 5 (d) when Δ exceeds the weight distribution. As Fig. 6 shows, using adaptive rounding achieves high recognition accuracy over a wide range of bit precisions.

2.1.4 Overall Algorithm for Filter-Wise Quantization

Figure 7 shows overall algorithm of the filter-wise quantization. Input parameter is the quantization step Δ and the weight values before quantization. In the step 1, the weights are quantized and normalized using the adaptive rounding method. The quantization step Δ is readjusted from the step 2 to step 4, and the weights are quantized and normalized again using the readjusted quantization step.

In actual multiply-accumulate (MAC) computations in this work, the quantized and normalized weights m_q are used, and the results of the MAC computations are de-quantized by multiplied by the readjusted quantization step Δ_{adj}.

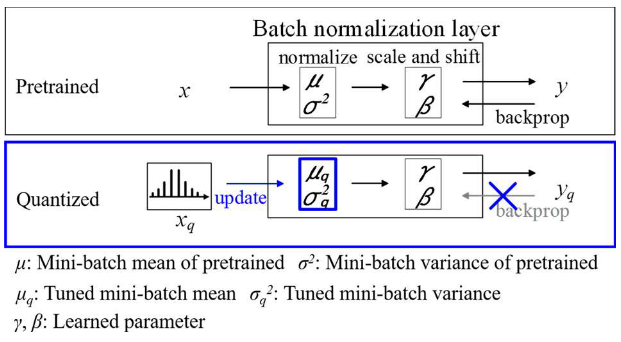
Batch normalization layer
normalize
scale and shift
Pretrained
x
μ
σ^2
γ
β
y
backprop
Quantized
x_q
update
μ_q
σ_q^2
γ
β
y_q
backprop
μ: Mini-batch mean of pretrained σ^2: Mini-batch variance of pretrained
μ_q: Tuned mini-batch mean σ_q^2: Tuned mini-batch variance
γ, β: Learned parameter

Fig. 8 Label-free fine tuning by renormalization (©2019 IEEE [17]).

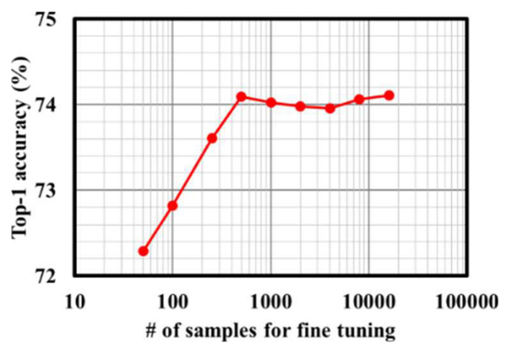

Fig. 9 Required number of samples for fine tuning (©2019 IEEE [17]).

2.2 Other Techniques for Improving Energy Efficiency while Maintaining Accuracy

2.2.1 Label-Free Fine Tuning by Re-Normalizing Activations

We introduce label-free fine tuning that does not require backpropagation training using labeled datasets to recover accuracy degraded by quantization. Figure 8 shows our fine-tuning approach [17]. We update only normalization parameters of μ and σ^2 in batch normalization [20] layers for fine tuning. Typically, when updating parameters in a neural network, backpropagation training using labeled datasets is required to compute gradients. However, the normalization parameters of μ and σ^2 can be updated by just computing the mean and variance of the data passing through batch normalization layers during forward propagation. The proposed fine-tuning method therefore recovers accuracy with no labeled dataset and backpropagation computations. By adjusting normalization parameters of μ and σ^2 in batch normalization layers, misalignment of the weight distribution due to the quantization can be corrected, recovering accuracy.

We evaluated our method using ResNet-50 on the ImageNet dataset, investigating how many image samples are required for the proposed label-free fine tuning. We split 50,000 samples from the ImageNet validation data into 16,000 images for fine tuning and 34,000 images for evaluating accuracy. Figure 9 shows the recognition accuracies over numbers of images randomly sampled from among the

16,000 images and utilized for the proposed fine tuning. This experiment shows that label-free tuning using 500 samples sufficiently recovers degraded accuracy due to quantization, since the accuracies saturate when the number of samples exceeds 500 (Fig. 9).

2.2.2 Reduction of Computations by Winograd Convolution

To reduce the number of computations we use the Winograd convolution [8], which can reduce the number of computations to $16/36 = 1/2.25$ while increasing the weight parameters by 16/9. This reduces $\widehat{N}_{ml}$ (the actual number of computations for filter m_l) in the numerator of Eq. (1), meaning the average bit precision is also reduced. In the Winograd convolution, preprocessing is executed before MAC computation and postprocessing is executed after. Although pre- and postprocessing induces overhead, it is usually negligible because the number of MAC computations is sufficiently large.

Figure 10 shows the relationship between accuracy and average bit precision when the weights in all the convolution layers are quantized at various average bit precisions and activations are quantized at a fixed 8 bits. Applying distribution-based filter-wise quantization and a readjusting scheme (blue line), the average bit precision is reduced by around 1 bit in comparison with layer-wise quantization (gray line). The improvement from the blue line to the green line shows the effectiveness of the Winograd convolution. Accuracy is also recovered by applying the proposed

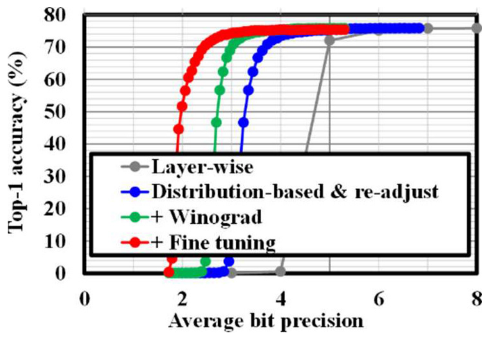

Fig. 10 Top-1 accuracy with ResNet-50 on ImageNet (©2019 IEEE [17]).

Table 1 Comparison of the state-of-the-art post-training quantization on ImageNet (©2019 IEEE [17]).

Method	Top-1 Accuracy (%)	Weights	Activations
[19]	76.1	32	32
	74.2	4	8
	72.6	4	4
Ours	75.8	32	32
	75.3	3.99	8
	74.3	**3.04**	8
	71.3	2.54	8
	72.8	4.00	4

label-free fine-tuning (red line). Here, we achieved 74.3% top-1 accuracy with 3.04 average bits.

Table 1 shows a comparison with the overall performance of other methods. Top-1 accuracy of ImageNet is 1.1% better than the results in [19] when the bit widths of weights and activations are almost the same at 4 bits and 8 bits respectively. The accuracy with 4-bit weights and 4-bit activations is also better. Furthermore, the proposed techniques can further reduce the average bit precision without noticeable accuracy loss, e.g., 71.3% with 2.54-bit weights and 8-bit activations.

3. MAC Processor for Filter-Wise Variable Bit Precision

Figure 11 conceptually describes the strategy for improving hardware utilization. The hardware architecture has eight processing units PE0-PE7 operating in parallel as MAC processors. Figure 11 also shows PE utilization, with colored tiles indicating processing PEs and gray tiles indicating idle PEs. The bit-serial technique, where multiplication is executed bit by bit using multiple cycles as shown in Fig. 11 (a) and (b), is often used to realize variable bit precision. For example, if the weights have 5-bit precision, multiplication requires 5 cycles. With layer-wise quantization, since bit precisions of weights are the same among the same-layer filters, computations for same-layer filters are densely assigned and can be effectively executed in parallel, as shown in Fig. 11 (a).

On the other hand, when the proposed filter-wise quantization is employed with the bit-serial technique, the number of execution cycles varies depending on the bit precision for each filter, as shown in Fig. 11 (b). Using conventional bit-serial and parallelized hardware architectures does not improve throughput or processing times, due to the deterioration of PE utilization despite the reduced number of computations. In other words, conventional bit-serial and parallelized hardware architecture cannot fully exploit the advantages of filter-wise quantization.

We propose a bit-parallel architecture that can resolve this mismatch between the proposed filter-wise quantization and conventional hardware architectures. In the

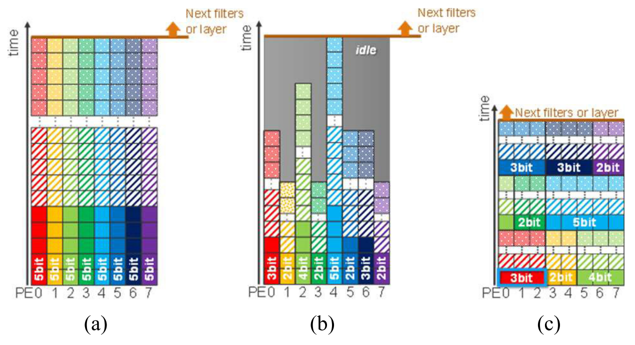

Fig. 11 Concept of the proposed parallel multiply-accumulate (MAC) processor with variable bit precision. (a) Bit-serial and layer-wise, (b) bit-serial and filter-wise, (c) bit-parallel and filter-wise (©2018 IEEE [16]).

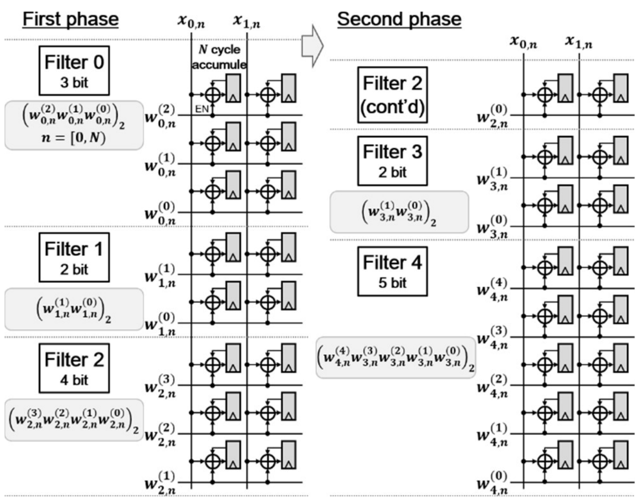

Fig. 12 Example of the proposed parallel multiply-accumulate (MAC) processor with variable bit precision (©2018 IEEE [16]).

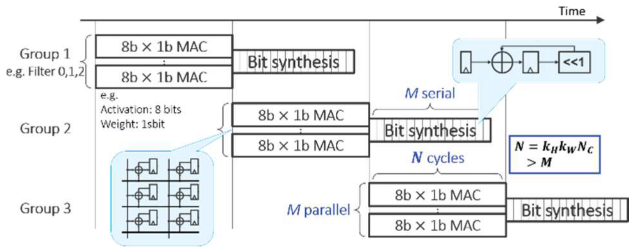

Fig. 13 Order of accumulation and bit synthesis (©2018 IEEE [16]).

proposed bit-parallel architecture, MAC computations for each weight bit are assigned to several PEs in parallel instead of over multiple cycles, as shown in Fig. 11 (c). By employing this architecture, we can improve throughput and processing times due to the increased PE utilization.

Figure 12 shows details of the proposed bit-parallel MAC processor with filter-wise variable bit precision. This architecture is designed to execute the following algorithm. A signed B-bit weight is represented as

$$w_n = -2^{B-1}w_n^{(B-1)} + 2^{B-2}w_n^{B-2} + \cdots + 2^1 w_n^{(1)} + 2^0 w_n^{(0)}, \tag{4}$$

where $w_n^{(b)}$ is a binary value (0 or 1). Using this representation, a MAC computation is

$$\sum_{n=0}^{N-1} w_n x_n = \sum_{n=0}^{N-1}\left(-2^{B-1}w_n^{(B-1)}x_n + \cdots + 2^0 w_n^{(0)}x_n\right), \tag{5}$$

which can be rewritten as

$$= -2^{B-1}\sum_{n=0}^{N-1} w_n^{(B-1)}x_n + \cdots + 2^0\sum_{n=0}^{N-1} w_n^{(0)}x_n. \tag{6}$$

Here, N is the number of accumulations, which is the product of the number of channels and the kernel height and kernel width for normal convolution computations, or the number of channels for Winograd convolution computations [8]. Therefore, a MAC computation with multiple-bit weights can be computed by summing properly bit-shifted results of MAC computations by each 1-bit weight as Eq. (6).

The following describes concrete processes of the proposed MAC processor shown in Fig. 12. There are 8 vertical MAC units and 2 horizontal MAC units, with each MAC unit comprising an accumulator and a register. There are 5 filters respective bit widths of 3, 2, 4, 2, and 5 bits. We unroll each bit width into 1-bit weights for the MAC computation units in parallel. These 1-bit weights are inputs to an enable signal (EN) port of the accumulator. The accumulator is enabled and accumulates input activation values only when the weight value is 1. After completing N accumulation cycles, we sum properly bit-shifted results of MAC computations shown in Eq. (6), a procedure we call "bit synthesis." In each phase, N accumulation cycles are executed in each unit without storing and/or loading intermediate results (output stationary in [1]), the memory bandwidth can be reduced.

As shown in Fig. 12, input activations $x_{0,n}$ and $x_{1,n}$ are vertically broadcasted and reused in 8 MAC units for multiplying with 8 different 1-bit weights. Weights are horizontally broadcasted and reused in 2 MAC units for multiplying with 2 different input activations. This 2-D reuse strategy minimizes the amount of weight loading and buffer activations for executing a certain number of MAC computations [10]. In the example shown in Fig. 12, since we have only 8 MAC units in a column, only 8-bit widths are calculated: 3 bits for Filter 0, 2 bits for Filter 1, and the upper 3 of 4 bits for Filter 2 are calculated in the first phase. In the second phase, MAC computations for the remaining 1 bit for Filter 2, 2 bits for Filter 3, and 5 bits for Filter 4 are executed. After accumulation, bit-synthesis for Filter 2 is executed with the registered intermediate result computed in the previous phase. Here, we have a simple codebook for indicating a delimiting point for the bit widths of each filter, so we can synthesize the registered MAC results of the previous phase. Bit synthesis with a simple codebook allows the proposed MAC processor to greedily utilize PE.

Figure 13 shows the bit-synthesis schedule. The first above described phase indicates group 1 and the second indicates group 2. Since the MAC computations take many more cycles than does bit synthesis, even when bit synthesis is executed in series on a single circuit, the latency due to bit synthesis is completely hidden behind the next MAC computations (Fig. 13). Therefore, throughput is maximized to a value determined by only the MAC array.

We perform bit synthesis after accumulation instead of accumulation after bit-synthesis which is employed in [13], this computation order enables to reduce the required dynamic range of each accumulator as well as to reduce the number of additions from "$B \times N$" to "$B + N$". Note that the proposed computation order is based on Eq. (6) and is equivalent to Eq. (5).

Consequently, the proposed architecture both maximizes the number of simultaneously executed filters and also keeps resource utilization close to 100%, even if the number of filter bit-widths is misaligned with the number of implemented MAC units. The proposed MAC processor with bit-parallel architecture is thus better suited to parallel computing with the filter-wise optimized bit precision than are bit-serial architectures [9].

4. FPGA Implementation

4.1 Architecture of an FPGA-Based MAC Processor

Figure 14 shows details of the proposed MAC processor implemented on an FPGA. The implemented MAC processor has 16×16 MAC units with 4 processors on the FPGA. The bit synthesizer is also implemented on the FPGA, placed in each column.

To reduce the number of computations, as described in Sect. 2.2.2 our implementation supports the Winograd convolution. Weights are preprocessed for the Winograd convolution in advance and stored in memory. The Winograd preprocessor converts 4×4 input activations to 4×4 preprocessed input activations on the fly. These 16 preprocessed input activations are multiplied with 16 different 1-bit weights and accumulated in parallel (represented as the 16 columns in Fig. 14). The Winograd pre- and postprocessors are placed in each plane in Fig. 14. As described in the previous section, we employ further 2D parallelism, reusing 16 weights for input activation (the 16 rows in Fig. 14) and 4 input activations for reused weights (the 4 planes in Fig. 14). Furthermore, we can reduce the bandwidth for loading activations, i.e. by exploiting that parts of input activations are used in multiple planes, we load only 40 activations instead of $16 \times 4 = 64$ activations per cycle. The sliding window overlap operations of convolutions permit such reuse of the input activations.

Figure 15 shows the pipeline strategy employed in the proposed MAC processor. All operations are pipelined to increase throughput. At first, input activations are preprocessed for the Winograd convolution and fed to the proposed MAC array. Weights are also preprocessed for the Winograd convolution in advance and stored in a block RAM (BRAM) buffer, then directly fed to the MAC array. After completing MAC computations, outputs are synthesized to obtain the multiple-bit weight using a codebook with additional bit-widths information, also loaded from an on-chip buffer. Finally, the postprocessing for the Winograd convolution is applied. BN and ReLU are not implemented in this work.

The bottom of Fig. 15 shows the throughput and parallelism. For example, since there are $16 \times 16 \times 4$ MAC units in this MAC processor, its parallelism is expressed as $16P_xP_W$, as shown at the bottom of Fig. 15. P_x is the degree of parallelism in the depth direction, which as shown in Fig. 14 is 4. P_W is the vertical parallelism, which is 16 in this work. The bit synthesis has $16 \times P_x$ MAC units and its parallelism is expressed as $16P_x$. As mentioned above, since MAC computation results are output only N times, throughput is reduced to $1/N$. The following Winograd postprocessing can thus be

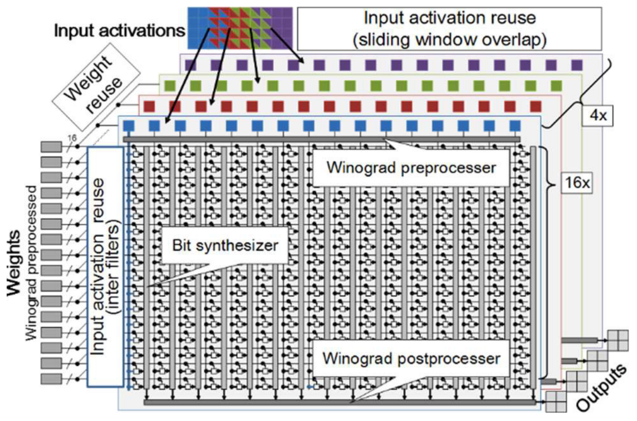

Fig. 14 Detailed FPGA architecture of the implemented MAC processor with filter-wise variable bit precision (©2018 IEEE [16]).

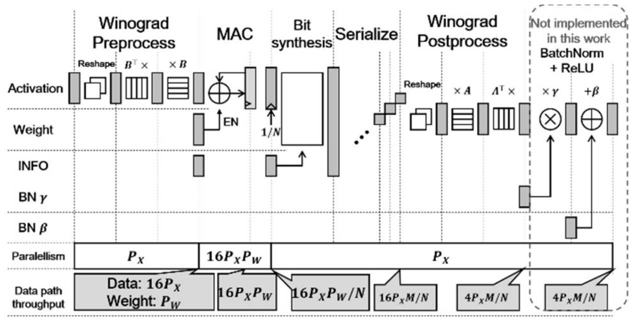

Fig. 15 Pipeline strategy of the proposed CONV engine. $P_x = 4$ and $P_w = 16$ in this work. N and M are the number of input channels and the number of filters, respectively (©2018 IEEE [16]).

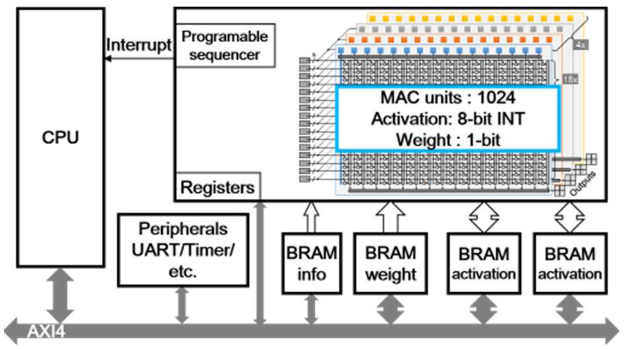

Fig. 16 System architecture of the proposed MAC processor, implemented on an FPGA (©2018 IEEE [16]).

sequentially executed, reducing the dimension of circuits. When 1×1 filters are employed in convolutional layers, we cannot exploit the Winograd convolution. In this case, the Winograd operation can be skipped.

4.2 FPGA System Architecture

We implemented the proposed architecture on a Xilinx ZCU102 Zync UltraScale+ MPSoC board. Figure 16 shows the system architecture implemented in this work. Weights and activations input to the proposed MAC processor are loaded from external DRAM to BRAM buffers in advance by a DMA controller, and a codebook is loaded as well. Weights and activations are then fed to the MAC processor in the proper order under control of the programmable sequencer. Although our design includes 1024 parallel 1-bit MAC processors, FPGA gate utilization is only around 20%, and it is notable that DSP48 instances are not used at all.

5. Experimental Results

To validate the effectiveness of the proposed architecture, we measured execution times of the proposed implementation and a conventional implementation of fixed 16-bit weights. We first quantize the pretrained weights of ResNet-50 on ImageNet for the filter-wise optimized bit precision. As shown in the upper panel of Fig. 17, compared with layer-wise quantization, the product of the number of MAC operations and the number of bits for weights ("MAC × bit," below) for the filter-wise optimized bit precision is significantly reduced under the same penalty of top-5 accuracy.

In the lower graph in Fig. 17, the gray solid line shows the theoretical lower limit on execution cycles assuming hardware with 64 MAC units for 16-bit weights, with execution cycles ideally proportional to MAC × bit. As shown in the figure, the execution cycles measured in our implementation are very close to the theoretical limits, with average utilization of the MAC units exceeding 97%. Compared with the case of 16-bit fixed bit precision and the case of a layer-wise optimized implementation, numbers of execution cycles are reduced (i.e. the throughput is increased) by 5.3x and 1.4x, respectively. Note that the throughput itself is not very large compared to e.g. [25] due to the limited number of MAC units and low clock frequency of 100 MHz and the improvement of them is our future work.

Figure 18 shows a comparison of the numbers of execution cycles for each layer under the proposed architecture and a conventional architecture. Numbers of execution cycles are significantly reduced under the proposed architecture. Our design fully supports convolutional computation with filter-wise quantized weight-bit precision, which reduces computations with no accuracy penalty.

Figure 19 shows the experimental setup. We ran ResNet-50 for the image classification task on the ZCU102 board. All 52 convolutional layers, except for the first layer with a 7×7 kernel, are executed on the FPGA where the proposed architecture is implemented. Other operations such as element-wise add in ResNet-50 and data movement between external DRAM and the FPGA are executed on a Zynq CPU. The screen in Fig. 19 shows an image of a rabbit to be classified, and the bar plot indicates that the image is correctly inferred as "wood rabbit, cottontail, cottontail rabbit."

6. Conclusion

We proposed filter-wise quantization of weights to reduce the number of computations for convolutional neural-network inference while maintaining high accuracy. Specifically, we introduced distribution-based filter-wise quantization as a concrete quantization method, readjusting the quantization step Δ to maintain accuracy, applying a rounding method to adequately reduce bit precision, and

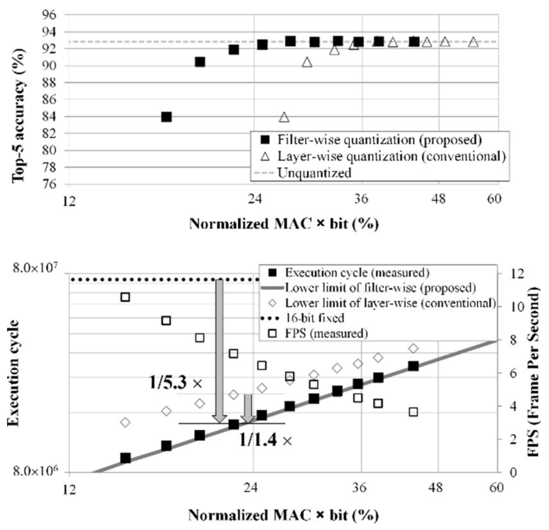

Fig. 17 Top: top-5 accuracy vs. normalized MAC × bit. The horizontal axis shows the product of the number of computations and the number of weight bits normalized to that number before applying our quantization method (16-bit weight). Bottom: number of execution cycles vs. normalized MAC × bit (©2018 IEEE [16]).

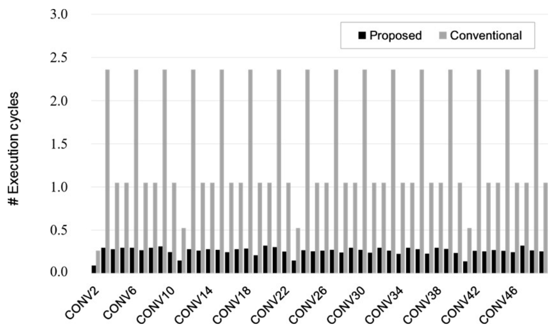

Fig. 18 Comparison of numbers of execution cycles under the proposed architecture vs. a conventional architecture, assumed as a 64-bit parallel architecture for a 16-bit-fixed MAC that consumes hardware resources similar to the proposed one (©2018 IEEE [16]).

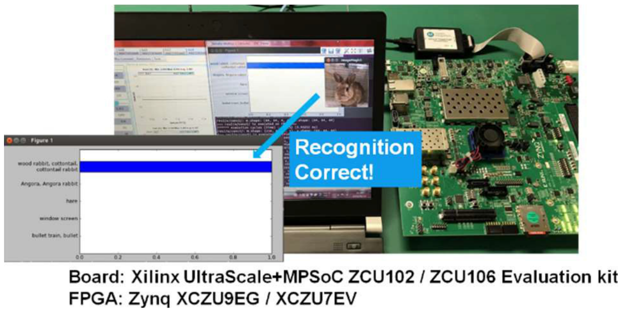
wood rabbit, cottontail, cottontail rabbit
Angora, Angora rabbit
hare
window screen
bullet train, bullet
Recognition Correct!
Board: Xilinx UltraScale+MPSoC ZCU102 / ZCU106 Evaluation kit
FPGA: Zynq XCZU9EG / XCZU7EV

Fig. 19 Experimental setup (©2018 IEEE [16]).

fine-tuning method without backpropagation using a labeled dataset. Furthermore, we proposed a bit-parallel architecture that fully exploits the reduced bit precision, reducing execution time in proportion to the number of operation bits. We implemented the proposed architecture on an FPGA and demonstrated that applied in the case of ResNet-50 runnning on ImageNet, the number of execution cycles is reduced by 5.3x without penalty in top-5 accuracy in comparison with the 16-bits fixed case.

Acknowledgments

The authors would like to thank Takahisa Kaihotsu, Takeshi Kumagaya, and Yoshio Masubuchi for their supports.

References

[1] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," 43rd Annual International Symposium on Computer Architecture (ISCA), pp.367–379, 2016.
[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," CVPR09, 2009.
[3] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang, "Accelerating low bit-width convolutional neural networks with embedded FPGA," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), 2017.
[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," ArXiv:1703.06870, March 2017.
[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," ArXiv:1512.03385, Dec. 2015.
[6] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," ArXiv:1704.04861, April 2017.
[7] A. Krizhevsky, I. Sutskever, and H.E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, pp.1097–1105, 2012.
[8] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Netwo Networks," ArXiv:1509.09308, Sept. 2015.
[9] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," 2018 IEEE International Solid – State Circuits Conference - (ISSCC), Feb. 2018.
[10] B. Murmann, D. Bankman, E. Chai, D. Miyashita, and L. Yang, "Mixed-signal circuits for embedded machine-learning applications," 2015 49th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, pp.1341–1345, 2015. doi: 10.1109/ACSSC.2015.7421361
[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," Computer Vision ECCV, pp.525–542, 2016.
[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," ArXiv:1506.01497, June 2015.
[13] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, pp.764–775, 2018. doi: 10.1109/ISCA.2018.00069
[14] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," Proc. 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp.161–170, 2015.
[15] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive Quantization for Deep Neural Network," ArXiv:1712.01048, Dec. 2017.
[16] 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 Circuit Conference (A-SSCC), pp.47–50, 2018. A-SSCC2018.
[17] S. Sasaki, A. Maki, D. Miyashita, and J. Deguchi, "Post Training Weight Compression with Distribution-based Filter-wise Quantization Step," Cool Chips, 2019.
[18] J. Deguchi, D. Miyashita, A. Maki, S. Sasaki, K. Nakata, and F. Tachibana, "Can in-memory/analog accelerators be a silver bullet for energy-efficient inference?," 2019 IEEE International Electron Devices Meeting (IEDM), 2019.
[19] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, "Post training 4-bit quantization of convolution networks for rapid-deployment," arXiv:1810.05723v2, Jan. 2019.
[20] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv:1502.03167v3, March 2015
[21] K. Nakata, A. Maki, D. Miyashita, F. Tachibana, T. Suzuki, and J. Deguchi, "Live Demonstration: FPGA-based CNN Accelerator with Filter-Wise-Optimized Bit Precision," IEEE International Symposium on Circuits and Systems (ISCAS), 2019.
[22] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," arXiv:1806.08342v1, June 2018.
[23] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks," The European Conference on Computer Vision (ECCV), Sept. 2018.
[24] X. Lin, C. Zhao, and W. Pan, "Towards Accurate Binary Convolutional Neural Network," Advances in Neural Information Processing Systems 30 (NIPS 2017), pp.343–353, 2017.
[25] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek, and K. Keutzer, "Synetgy: Algorithm-hardware Co-design for ConvNet Accelerators on Embedded FPGAs," Proc. 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'2019), Feb. 2019.

Asuka Maki received the B.E. and M.E. degrees in electronic engineering from the Tokyo University of Science, Tokyo, Japan, in 2000 and 2002, respectively. In 2002, she joined Research and Development Center, Toshiba Corporation, Kawasaki, Japan, where she has been engaged in the design of RF and analog circuits for wireless communications. In 2017, she joined Kioxia Corporation, Kawasaki, Japan. She has been engaged in research and develop of efficient hardware and algorithms for deep learning.

Daisuke Miyashita received the B.E. and M.E. 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 the Institute of Memory Technology Research & Development, Kioxia Corporation, Kawasaki, Japan. His research interests include efficient mixed-signal/digital hardware for machine learning applications and algorithms designed for such hardware. 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.

Shinichi Sasaki received the B.E. and M.E. degrees in electrical engineering from Kyushu University, Fukuoka, Japan, in 2007 and 2009, respectively. In 2009, he joined the Center for Semiconductor Research and Development, Toshiba Corporation, Kawasaki, Japan, where he was engaged in SRAM and RF analog circuit design. He has been joined Institute of Memory Technology Research and Development, Kioxia Corporation, Yokohama, Japan, since 2017. His current research interests include computer architecture and energy-aware signal processing algorithm for deep learning.

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, Kawasaki, Japan. His current research interest includes efficient hardware for machine learning applications.

Fumihiko Tachibana received the B.E. and M.E. degrees in electronics engineering from the University of Tokyo, Tokyo, Japan, in 2003 and 2005, respectively. In 2005, he joined the Center for Semiconductor Research and Development, Toshiba Corporation, Kawasaki, Japan, where he was engaged in research and development of low-power digital circuits, embedded SRAMs, image sensors, and high speed I/O. From 2013 to 2014, he was a Visiting Scholar with Stanford University, Stanford, CA, USA, where he was involved in research on energy efficient image sensors. In 2017, he joined Kioxia Corporation, Kawasaki, Japan, where he has been engaged in research and development of efficient hardware and algorithms for machine learning applications, and data converter for high speed I/O. His current research interests include data converter for high speed I/O.

Tomoya Suzuki received the M.E. from Nara Institute of Science and Technology, Japan in 2006. From 2006 to 2017, he worked at Center for Semiconductor Research and Development in Toshiba Corporation to design hardware and software for wireless communications. In 2017, he moved to Institute of Memory Technology Research and Development in Kioxia Corporation, where he is currently working to design new computing systems using emerging memories.

Jun Deguchi received the B.E. andM.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 ToshibaMemory Corporation), and has been a Research Lead of an advanced circuit design team working on high-speed I/O, deep learning/neuromorphic accelerators and quantum annealing. Dr. Deguchi has served as a member of the technical program committee (TPC) of IEEE International Solid-State Circuits Conference (ISSCC) since 2016, and IEEE Asian Solid-State Circuits Conference (A-SSCC) since 2017. He has also served as a TPC vice-chair of IEEE A-SSCC 2019, and a review committee member of IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS) 2020.

Ryuichi Fujimoto 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 integrated circuits and device models. 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). After that, he was with Center of Semiconductor Research & Development, Semiconductor & Storage Products Company, Toshiba Corporation. Currently, he is with Institute of Memory Technology Research and Development in 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 a secretary of the Japan Chapter of IEEE Circuits and Systems Society from 2008 to 2009. Currently, he is a chair of the Japan Chapter of IEEE Solid-State Circuits Society. He is a member of the IEEE, IEEJ and JAAS.