International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 11, Number 2, pages 338–353, July 2021

ProgressiveNN: Achieving Computational Scalability with Dynamic Bit-Precision Adjustment by MSB-first Accumulative Computation

Junnosuke Suzuki, Tomohiro Kaneko, Kota Ando, Kazutoshi Hirose, Kazushi Kawamura,
Thiem Van Chu, Masato Motomura, and Jaehoon Yu
Tokyo Institute of Technology
Yokohama, Kanagawa, 226-8503, Japan

Received: February 15, 2021
Revised: May 3, 2021
Accepted: May 23, 2021
Communicated by Ikki Fujiwara

Abstract

Computational scalability allows neural networks on embedded systems to provide desirable inference performance while satisfying severe power consumption and computational resource constraints. This paper presents a simple yet scalable inference method called *ProgressiveNN*, consisting of bitwise binary (BWB) quantization, accumulative bit-serial (ABS) inference, and batch normalization (BN) retraining. ProgressiveNN does not require any network structure modification and obtains the network parameters from a single training. BWB quantization decomposes and transforms each parameter into a bitwise format for ABS inference, which then utilizes the parameters in the most-significant-bit-first order, enabling progressive inference. The evaluation result shows that the proposed method provides computational scalability from 12.5% to 100% for ResNet18 on CIFAR-10/100 with a single set of network parameters. It also shows that BN retraining suppresses accuracy degradation of training performed with low computational cost and restores inference accuracy to 65% at 1-bit width inference. This paper also presents a method to dynamically adjust the bit-precision of the ProgressiveNN to achieve a better trade-off between computational resource use and accuracy for practical applications using sequential data with proximity resemblance. The evaluation result indicates that the accuracy increases by 1.3% with an average bit-length of 2 compared with only the 2-bit BWB network.

Keywords: deep neural network, bit-wise quantization, progressive inference, batch normalization retraining, dynamic bit-precision

1 Introduction

The availability of neural networks on edge devices provides a promising solution to privacy, network connectivity, and real-time responsiveness problems when applying neural networks in healthcare, robotics, vehicle design, and industries of unpopulated areas. Because neural networks require high computation cost, deployment on the end user-edge device platforms has not been feasible. Presently, the arrival of more powerful and low-energy-consumption edge devices affords more options. However, the key problem is that neural networks require more computational resources than edge devices can afford.

Under severe constraints of computational resources and power consumption on edge devices, computational complexity reduction is critical in exploiting the benefits of neural networks. Quantization is the most widely used technique for this purpose. Using low bit-width activations/parameters enables the edge devices to satisfy the constraints in exchange for sacrificing accuracy. To improve this trade-off, many researchers

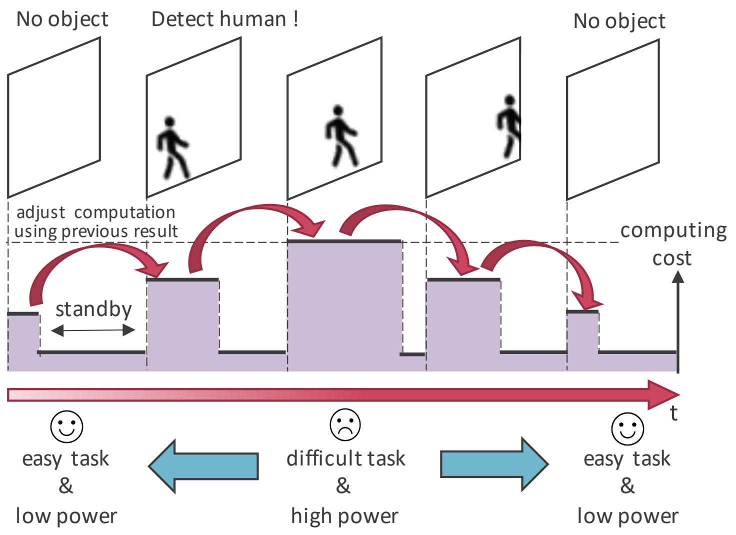


Figure 1: Advantage of computational scalability on edge devices. Inference difficulty depends on tasks, and low computational inference is sufficient for undemanding tasks. For applications using sequential data such as audio and video, it is possible to determine proper computational cost based on previous inference results. Edge devices can reduce energy consumption by adaptively adjusting computational cost.

have proposed binary, ternary, and other low bit-width quantization methods [1–9]. These methods optimize network models to perform their best with a specific bit-width and format. There is no doubt that they can achieve outstanding results, but there is room for improvement in edge-device inference.

When performing inference tasks, edge devices can reduce power consumption without any decrease in accuracy by adjusting their computational resource use according to task difficulty, i.e., less computational resource use for simple tasks and more for complex tasks, as shown in Figure 1. Adaptive inference, as the name suggests, is a method that provides computational scalability to machine inference processes. However, the conventional methods [10–13] require alteration of network structure, causing a high implementation cost for existing network models. Bit-flexible networks [14, 15] provide an alternative solution without requiring structural alteration. They train network parameters in a bit-by-bit manner: training a binary network followed by training a 2-bit network concatenating the 2nd bit to the binary network, and so on. Their use of bit-serial parameters enables computational scalability for inference tasks; however, the training process remains time-consuming and too complex to achieve stable performance.

To solve this problem, we propose a simple and scalable inference method called ProgressiveNN, consisting of three parts: bitwise binary (BWB) quantization, batch normalization (BN) retraining, and accumulative bit-serial (ABS) inference [16]. ProgressiveNN is a bit-flexible network, and its training process is simple and applicable even to pre-trained networks. Besides, its network parameters can be obtained in only a single training. The name, "ProgressiveNN" was coined by analogy with progressive JPEG. As shown in Fig. 2, progressive JPEG improves "user experience" by displaying gradually. ProgressiveNN can optimize the trade-off between computational cost minimization and accuracy, aiming at a wide range of potential applications. This paper presents the details of ProgressiveNN and describes how the trade-off can be dynamically adjusted in sequential data.

The remainder of this paper is structured as follows. Section 2 introduces related methods. Section 3 describes the details of ProgressiveNN. Then, Section 4 provides the evaluation results on the CIFAR-10 and CIFAR-100 datasets. Finally, Section 5 concludes this paper.

2 Related Work

This section describes three categories of related methods: low-bit quantization, bit-flexible networks, and training only BN. The studies on low-bit quantization [3–9] provide dedicated quantization approaches for

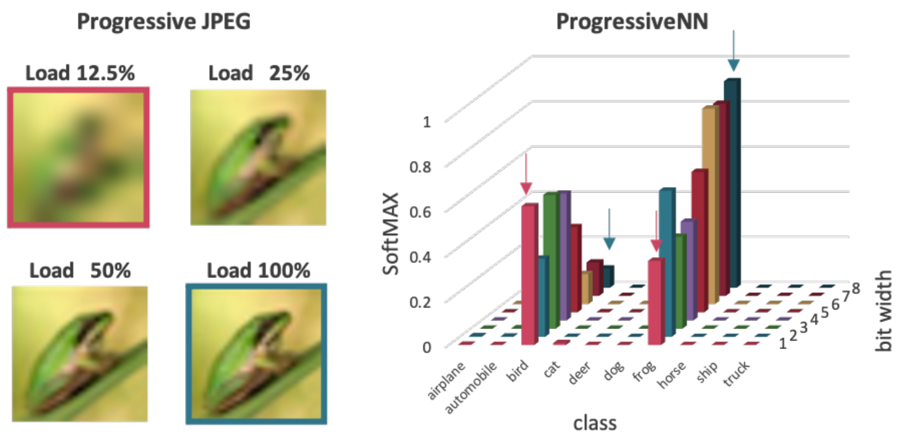


Figure 2: Analogy of progressive JPEG and ProgressiveNN: progressive JPEG gradually enhances the resolution of an image according to loaded data amount, and ProgressiveNN gradually improves the inference accuracy according to the bit-width of weights.

extremely low-bit expressions such as binary and ternary. Although these approaches cannot adjust their computational cost, they clarify the inference potential of networks using low-bit expressions. The studies on bit-flexible networks [14, 15] provide approaches that support computational scalability without network alteration. ProgressiveNN belongs to this category. Finally, training only BN [17] was proposed to investigate the expressive power of BN, which ProgressiveNN exploits as a countermeasure against accuracy degradation at low bit-width.

Low-bit quantization. Low-precision networks are a well-studied research area. Binary connect [3] replaces weights with their sign and uses it for inference and training. Binary weight networks [4] introduce a weight scale factor in binarizing and use the same binarization function. XNOR-Net [5] uses efficient binary approximations to improve accuracy. Ternary weight networks [6] introduce ternary weights with zero added to improve accuracy. These networks can significantly reduce computational complexity, but they cannot provide computational scalability of inference tasks. Our ProgressiveNN presents a solution to this problem.

Bit-flexible networks. FlexNet [14] applies a bit-flexible network for resolving the issue that low-bit convolutional neural networks (CNNs) cannot adjust the inference time and employs a corresponding training method, called bit-progressive training. This training method gradually learns weights from the most significant bit (MSB) to the least significant bit (LSB). In addition, a progressive scaling approach was proposed [15] by introducing a scaling factor to improve low-bit precision in inference. This method improves accuracy by training and adequately using scaling factors for various bit-widths. The problem is that its training cost is considerable because this method needs to train weights in a bitwise manner. Therefore, we propose a more straightforward training method with BWB quantization. Hardware support is indispensable in exploiting the benefits of bit-flexible networks, including ProgressiveNN. For example, in [18], an architecture, called bit fusion, was proposed to dynamically change the bit-widths in inference. Although the bit-widths of the network models for bit fusion are fixed in advance, we can expect that this type of basic framework to work for bit-flexible networks.

Training only BN. BN is an indispensable component of modern neural networks, but its expressive power is not fully understood. To improve the understanding on BN, the inference performance achieved by training BN and freezing other parameters has been investigated [17]. As reported in [17], training only BN with sufficiently deep ResNets reached 82% test accuracy on CIFAR-10 and 32% top-5 accuracy on ImageNet. These results are much higher than those of networks with random parameters. In this study, we exploit the knowledge obtained from [17]. ProgressiveNN recovers its accuracy when using low bit-width weights by retraining BN for each bit-width.

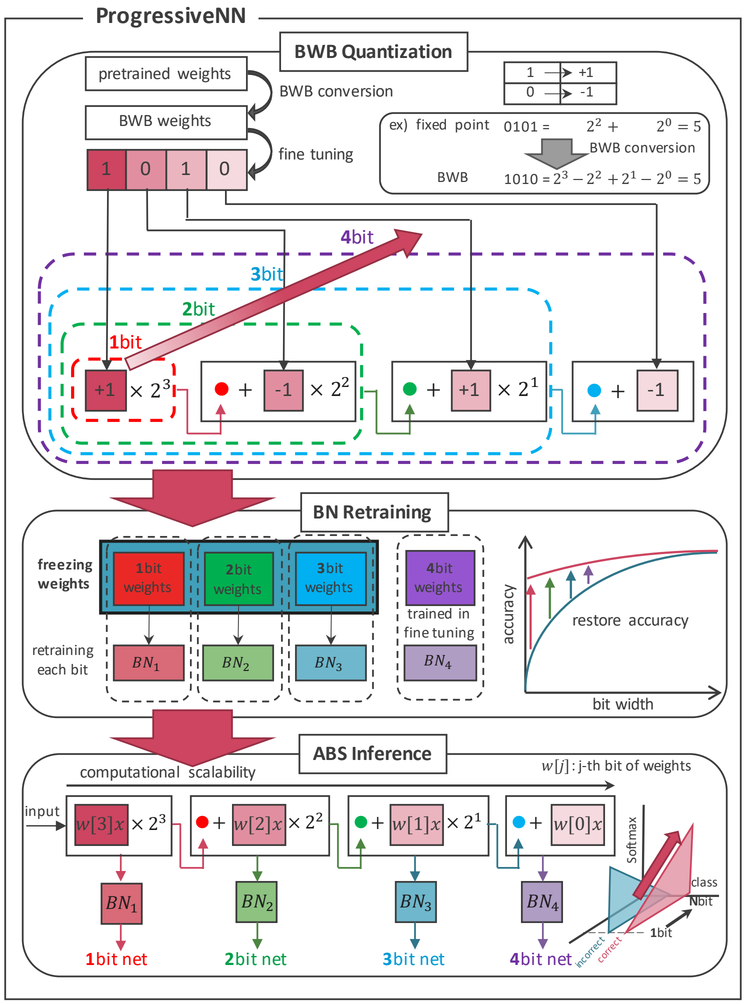


Figure 3: Processing flow of ProgressiveNN, which consists of three processes: BWB quantization, BN retraining, and ABS inference. BWB quantization converts trained floating-point weights to BWB format through fine-tuning. Inference accuracy degradation caused by low bit-width BWB weights is alleviated by BN retraining. Then, ABS inference adjusts the computational cost of inference by accumulating inner products with BWB weights in each bit-width.

3 ProgressiveNN

ProgressiveNN is a bit-flexible network consisting of three processes: BWB quantization, BN retraining, and ABS inference. A significant difference from conventional methods is that ProgressiveNN does not require any unusual training method and is applicable even for pretrained networks. As shown in Figure 3, BWB quantization is a bitwise binary representation that can express each weight in various resolutions. BN retraining is a countermeasure against accuracy degradation caused by using low bit-width weights. ABS inference performs the role of providing computational scalability to inference. This section also presents a method for estimating the proper bit-precision based on low-bit inference results and a use case scenario to achieve a better computational cost–accuracy trade-off. The following subsections describe the details of the aforementioned techniques.

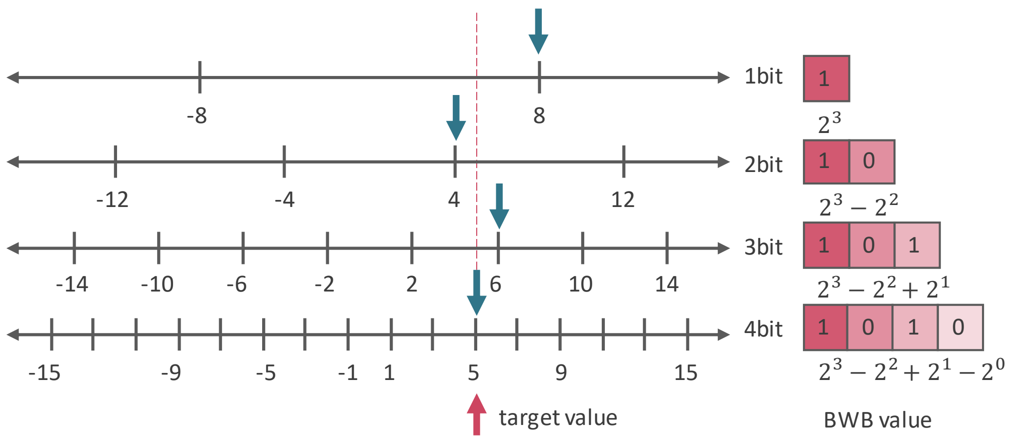


Figure 4: Example of BWB quantization. BWB interprets zero as -1 and one as +1 and multiplies its place value. Therefore, given that the target value is 5, 4-bit BWB expression is "1010" instead of "0101." In each bit-width, the BWB expression indicated by a blue arrow is the nearest value to the target value.

3.1 BWB Quantization

The concept of BWB quantization has been proposed in [14] and [15]. As shown in Figure 4, BWB quantization considers the nearest value to the quantization target as the quantized value in the corresponding bit resolution using targeted N-bit representation. The distinct feature of BWB quantization is that an expression of a target value includes its other expressions with lower bit width. Therefore, BWB quantization allows for a progressive numeric representation with one network weight. BWB quantization interprets zero as -1 and one as +1 in each binary digit, multiplies its "place value," and then accumulates it. For example, the expression of $5 \in \mathbb{I}$ in 4-digit binary is "1010" instead of "0101." One in the 4th digit means +8, which is the nearest value to $5 \in \mathbb{I}$ in 1-bit resolution. Then, adding -4, zero in the 3rd digit, to +8 makes $4 \in \mathbb{I}$, which is the nearest value to $5 \in \mathbb{I}$ in 2-bit resolution, and so on. N-bit BWB quantization expresses the values from $-(2^N - 1)$ to $+(2^N - 1)$ at intervals of 2. ProgressiveNN obtains all BWB values simultaneously by converting trained floating-point weights to a BWB expression through fine-tuning: several training epochs start from the floating-point weights with a very low learning rate. Although BWB quantization has no expression for $0 \in \mathbb{I}$, we allow zero weights because they are implementable as pruned weights with skip instruction in accelerators.

3.2 BN Retraining

We use BN retraining as a countermeasure against accuracy degradation caused by low bit-width expressions. As mentioned earlier, BWB quantization is a convenient method that extracts 1-bit to N-bit BWB expressions simultaneously. The problem, however, is that low bit-width BWB weights do not perform well in inference because there is no dedicated quantization for low bit-width networks. As mentioned in the previous section, training only BN shows high accuracy even with frozen random weights. We believe that retraining BN is also useful against the accuracy drop in low-bit-width BWB expressions. When the maximum BWB expression is N-bit width, ProgressiveNN retrains only BN parameters for 1 to $(N-1)$-bit-width expressions by freezing other parameters. The retraining cost for this is negligible compared with [14] and [15].

3.3 ABS Inference

ABS inference is a progressive inference using BWB weights. For convenience, we describe the details of ABS inference in the form of general neural networks, but ABS inference can be extended to CNNs in a straightforward manner. The inner product value, z, is calculated by

$$z = \sum_{m=1}^{M} w_m x_m, \tag{1}$$

where w_m is the m-th weight, and x_m is the corresponding input activation. Let $w_m[n]$ be the n-th bit of the m-th BWB weight, i.e. $w_m[n] \in \{-1, +1\}$, where the 0-th bit is the LSB. Then, z can be rewritten as follows:

$$z = \sum_{m=1}^{M} \sum_{n=1}^{N} 2^{n-1} w_m[n-1] x_m. \tag{2}$$

We can interchange the order of double summation:

$$z = \sum_{n=1}^{N} \sum_{m=1}^{M} 2^{n-1} w_m[n-1] x_m. \tag{3}$$

Using (3) in the MSB-first order enables progressive inference. For calculating z_l of a l-bit network, the equation is written as

$$z_l = \sum_{n=N-l+1}^{N} \sum_{m}^{M} 2^{n-1} w_m[n-1] x_m \quad (1 \le l \le N). \tag{4}$$

Therefore, the calculation for each bit-width can be written as follows:

$$\begin{aligned} z_1 &= \sum_{m}^{M} 2^{N-1} w_m[N-1] x_m \\ z_2 &= z_1 + \sum_{m}^{M} 2^{N-2} w_m[N-2] x_m \\ &\vdots \\ z_N &= z_{N-1} + \sum_{m}^{M} 2^{0} w_m[0] x_m. \end{aligned} \tag{5}$$

As expressed in (5), because the calculation is recursive, ABS inference needs to calculate only the incremental portion for higher accuracy. In addition, because it uses a single set of BWB weights for 1-bit to N-bit networks, this technique can reduce memory usage to achieve progressive inference.

3.4 Dynamic Bit-Precision Adjustment

We propose a technique to determine the optimal bit-precision, achieving equivalent accuracy of inference results with the less computational cost. In sequential data, such as video frames, there is a resemblance between consecutive inputs. The dynamical bit-precision adjustment exploits this feature. ProgressiveNN conducts a certain bit-precision inference on the current input and decides to increase or decrease the bit precision for the next input. We constructed an adjustment criterion based on the entropy of output distribution from inference. Our technique is inspired by [19], which explores the combination of supermasks by minimizing the entropy of output distribution.

Figure 5 shows the core concept of the criterion for bit-precision adjustment. We consider this as an optimization problem that has a trade-off between computational cost and inference loss, as shown in the left figure. We assume that the computational cost is linearly proportional to the bit-width, and the inference loss is inversely proportional. The objective function is the sum of both terms, as in (6).

$$O(n) = L(n) + \alpha C(n), \tag{6}$$

where $L(n)$ is the n-bit validation loss, $C(n)$ is the n-bit computation cost, and α is the scale factor. The Pareto solution, n^*, is defined as follows:

$$n^* = \underset{1 \le n \le N}{\arg\min}(O(n)). \tag{7}$$

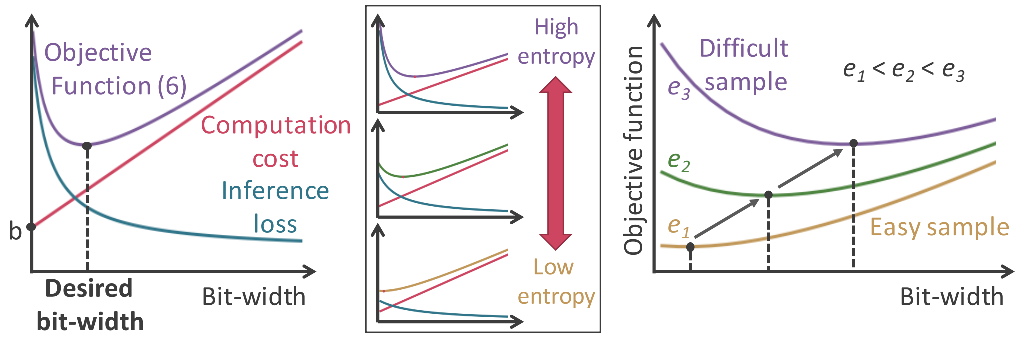


Figure 5: Objective function using computational cost and inference loss (left), transition of inference loss and objective function with the change in entropy threshold (center), and superposition of objective functions in the center panel (right). b in the left figure is the computational overhead. The distribution on the right figure shows objective function when only samples with entropy less than e_i are processed.

Dataset	Bit-width							
	1	2	3	4	5	6	7	8
CIFAR-10/w/o BN Retraining + Dynamic Quantization	14.6	69.5	92.6	94.5	94.8	94.9	94.9	94,9
CIFAR-10/w/ BN Retraining	90.7	92,3	94.0	94.7	95.0	95.0	95.0	95.0
CIFAR-10/w/ BN Retraining + Dynamic Quantization	90.5	94.3	94.9	95.0	94.9	94.9	95.0	94.9
CIFAR-100/w/o BN Retraining + Dynamic Quantization	1.4	36.8	71.4	76.4	77.8	77.9	77.9	78.0
CIFAR-100/w/ BN Retrainig	65.1	68.2	70.3	72.8	75.6	77.3	78.1	78.0
CIFAR-100/w/ BN Retraining + Dynamic Quantization	64.6	75.7	77.4	78.0	78.1	78.2	78.1	78.1

Table 1: Top-1 accuracy of each bit-width on ResNet18 using the CIFAR-10/100 testing datasets.

Because $C(n)$ is proportional to the bit-width, it can be defined as follows:

$$C(n) = an + b, \tag{8}$$

where a is the proportional constant and b is the overhead of computation the cost. The output distribution P is the output of the SoftMax function used in neural networks for classification, and the entropy of the output distribution is defined as follows:

$$e = -\sum_i p_i \log_2 p_i \tag{9}$$

where p_i is the i-th dimension of P. Given a threshold, the bit-precision adjustment algorithm only processes the samples when the entropy of output distribution is below the threshold. We call it the entropy threshold in this paper. Then, we consider the case where the entropy threshold is varied. As shown in the center panel of Figure 5, the higher the entropy threshold, the more significant the difference between high bit-width loss and low bit-width loss. This difference is because samples showing the high entropy are difficult to infer, especially in low bit-width. Also, because the inference loss is associated with the entropy threshold, the objective function changes its shape for each entropy threshold, as shown in the right panel of Figure 5. Then, the bit-width of the minimum value switches after a certain entropy threshold. We set the entropy at which the bit width switches as the threshold for processing with that bit-width. For each bit-width, bit-precision adjustment algorithm processes the samples with entropies below the corresponding entropy threshold.

4 Experiment

To confirm the validity of ProgressiveNN, we applied it to ResNet18 and evaluated its performance. This section describes the performance of ABS inference with BWB quantization and the influence of BN retraining

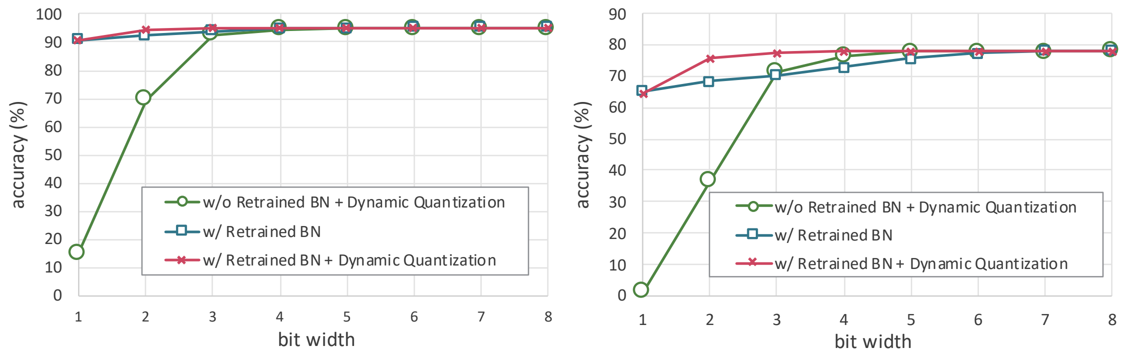


Figure 6: Top-1 accuracy of each bit-width on ResNet18 using the CIFAR-10 (left) and CIFAR-100 (right) testing datasets. The horizontal and vertical axes represent the bit-width of the BWB weights and top-1 accuracy, respectively. The red and blue lines denote the accuracies with BN retraining, red lines show the case of setting the appropriate dynamic range for each BWB layer, and blue lines show the case of setting a constant range for all BWB layers. The green lines indicate the accuracy without BN retraining under the same quantization as the red line.

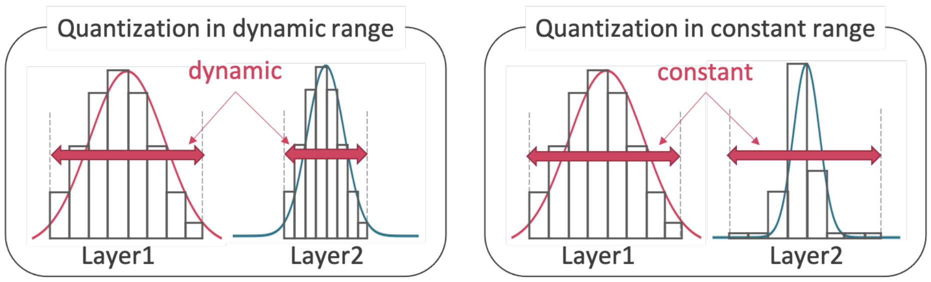


Figure 7: Weight distribution when quantized with 3 bits. The left figure shows the case where each layer is quantized in the range of approximately three times the standard deviation, and the right figure shows the case where all layers are quantized in the constant range, and values outside the range are clipped.

on accuracy. Subsequently, we evaluated the computational cost reduction with dynamic bit-precision. After that, we estimated hardware resource requirement for ProgressiveNN. Finally, we discuss a further reduction in the computational cost by using activation prediction based on ABS inference.

4.1 Experimental Settings

We evaluated the inference accuracy of ProgressiveNN on classification tasks. The network model used was ResNet18 [20], implemented with the PyTorch framework [21]. Two CIFAR datasets [22], CIFAR-10 and CIFAR-100, were used for training and testing, and these contained 50,000 images and 10,000 images, respectively. For training, we adopted the standard data augmentation described in [20]. We used the stochastic gradient descent with momentum as the optimizer and trained the network model with a batch size of 128 for 200 epochs. The initial learning rate was set to 0.1 and was divided by 5 every 60 epochs. The threshold for dynamic bit-precision adjustment was obtained by cross-validation, and 10,000 images were randomly extracted from the CIFAR-100 dataset as validation data.

4.2 Evaluation using ResNet18 on CIFAR-10/100

We adopted 8-bit BWB quantization of the weights of both convolutional and fully connected layers and 8-bit fixed-point quantization of activations and other parameters. Figure 6 shows the top-1 accuracy changes according to the bit-width; the blue lines and red lines represent the results with BN retraining, and the green

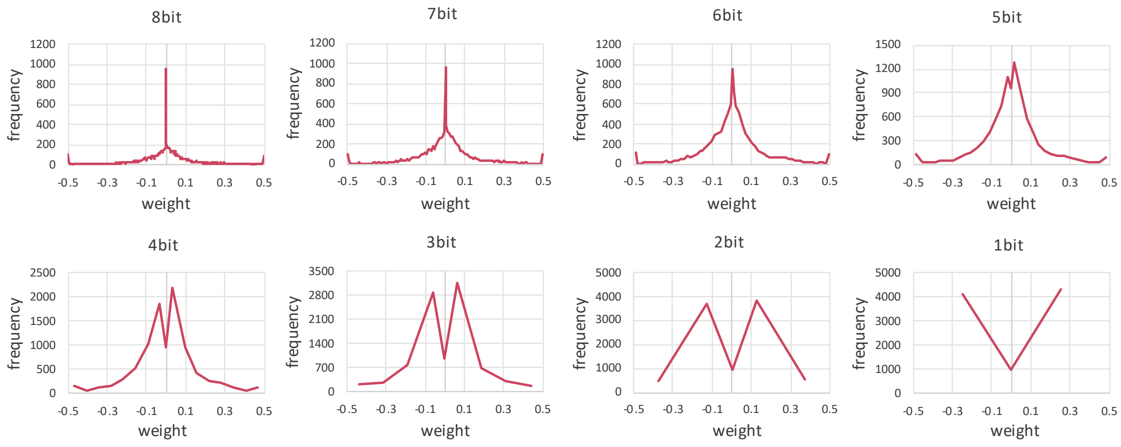


Figure 8: Distribution of each bit-width of BWB weights in the 1st convolution layer of ResNet18. From the top-left to the bottom-right, each graph shows the weight distribution in bit-width order. The top-left, 8-bit weights, shows a unimodal distribution, and narrowing the bit-width distorts the weight distributions when the bit-width is less than 5.

lines represent the result without BN retraining. Furthermore, the red and green lines represent the result with quantization in the range of approximately three times the standard deviation for each BWB layer, as shown on the left of Figure 7; the blue lines represent the result with quantization in the constant range for all BWB layers, as shown on the right of Figure 7. Figure 7 shows an example of 3-bit quantization, clipping out-of-range values. In our previous work [16], constant quantization was used with a low-resolution layer, whereas in this study, dynamic quantization improved the accuracy. Table 1 summarizes the results shown in Figure 6. From Figure 6, we can confirm that narrowing the bit-width of BWB expressions degrades the top-1 accuracy. However, it is also possible to confirm that BN retraining recovers the accuracy drop shown in the result without BN retraining: top-1 accuracy of 1-bit weight network on CIFAR-10 and CIFAR-100 improved to 91% and 65%, respectively. We can also confirm that quantization in the appropriate range for each BWB layer improves accuracy at low bit-widths.

We found a significant variation between the weight distribution of each bit-width through further analysis. As shown in Figure 8, 8-bit BWB weights have a unimodal distribution with a sharp peak at zero, but narrowing the bit-width distorts weight distributions significantly, especially when the bit-width is less than 5. Note that we allow zero weights by assuming skip instruction, as mentioned in Section 3.1, although BWB quantization has no expression for zero. We considered that this distortion is the possible cause of the significant deterioration in low-bit BWB expressions. To resolve this problem, we focused on BN's expressive power. In [17], it is clarified that training only BN enables us to obtain high accuracy even when other parameters are frozen at randomly initialized values. We straightforwardly extended this idea and retrained BN parameters for each bit-width less than 8. The result that the red lines represent proves that the straightforward extension of BN training is useful.

4.3 Evaluation with Dynamic Bit-Precision Adjustment

We evaluated the dynamic bit-precision adjustment based on the entropy of the output distribution. Figure 9 shows the relationship between accuracy and entropy, resulting from experiments using the validation set, with the horizontal axis representing entropy, and vertical axis denoting the accuracy with processing samples below the entropy of the horizontal axis. Based on this figure, we can confirm that the accuracy is improved by increasing the bit-width. We can also confirm that the low entropy output result is confident because of the accuracy of samples with high entropy being low. Results from 5-bit to 8-bit widths are omitted because they are almost identical to the result of the 4-bit widths. Based on this result, the average bit length was reduced by assigning easy tasks with low entropy output and difficult tasks with high entropy output to low and high bits, respectively. The criterion for judging which bits to assign is the maximum entropy that minimizes the

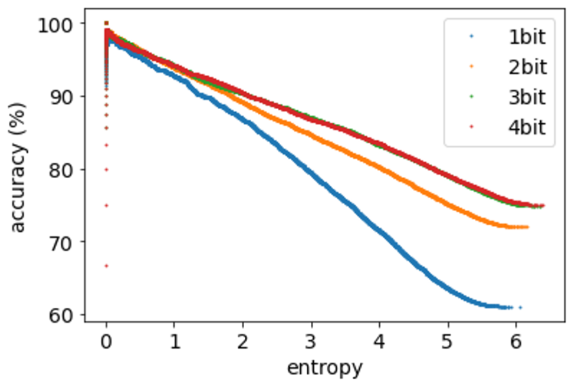


Figure 9: Relationship between entropy and accuracy for each bit-width. The vertical axis shows the accuracy when processing samples below the entropy threshold of the horizontal axis with the corresponding bit-width, excluding the remainder.

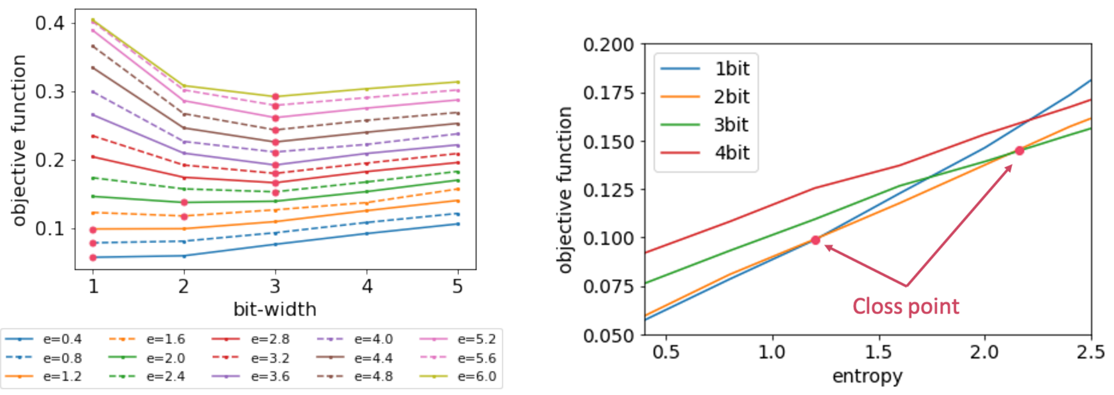
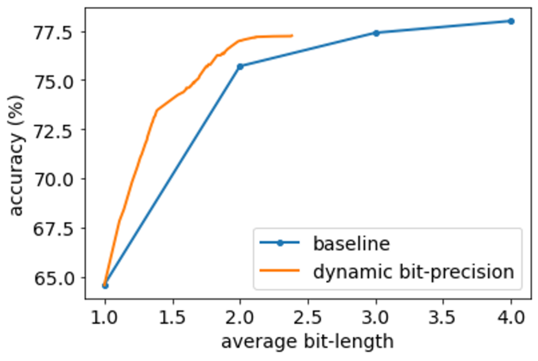


Figure 10: Relationship between objective function and the entropy of each bit-width (left), and objective function when the entropy changes (right). These figures are for α of 1/70. We can confirm that the bit-width showing the minimum value changes when the entropy is approximately 1.2 and 2.2, and these entropies are set as the threshold values.

objective function, as explained in Section 3.4, for each bit-width.

Figure 10 shows the case where α is set to 1/70 in (6). The left figure shows the changes in th objective function based on the accuracy below each entropy, and the right figure shows the changes in the objective function for each bit width. From this figure, we can confirm that the bit width showing the minimum values changes at the entropy of approximately 1.2 and 2.2, and these cross points are set as threshold values which go up to the next bit, down to the previous bit, or remain the same. The left figure in Figure 11 shows the accuracy with dynamic bit-precision and the baseline accuracy on CIFAR-100. The right one shows the changes in the number of samples processed by each bit-width, where the vertical axis represents the number of samples, and the horizontal axis represents the average bit-length. We can confirm that assigning samples with high entropy to high bits achieves higher accuracy than using only specific bits. Table 2 lists the accuracy, average bit length, and the number of samples processed with each bit-width when the scale factor, α, is 1/25, 1/50, and 1/100. When α was 1/25, all samples were processed with 1-bit or 2-bit, whereas when α was reduced to 1/50 or 1/100, the percentage of 3-bit increased. We found that dynamic bit-precision improves the accuracy per average bit-length over the baseline. For example, the accuracy with an average bit length of approximately 2 was 1.3% better than the baseline accuracy with 2-bit weights. From these results, we can confirm that as the scale factor decreases, the average bit-length and accuracy improve progressively, and the ratio of the high bits increases. In situations where the amount of computational resources is limited, selecting an appropriate scale factor according to the condition can achieve the desired trade-off.



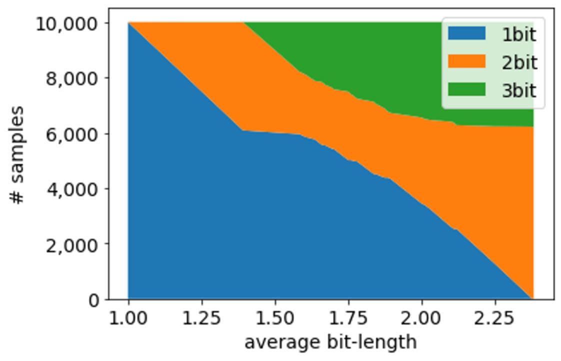


Figure 11: Relationship between the average bit-length and the accuracy with dynamic bit-precision adjustment (left) and the number of each bit as the average bit length changes (right).

scale factor	accuracy	average bit-length	1bit	2bit	3bit	4bit
$\alpha = 1/25$	72.9	1.36	6,440	3,560	0	0
$\alpha = 1/50$	75.7	1.75	5,011	2,475	2,514	0
$\alpha = 1/100$	77.0	2.01	3,407	3,092	3,501	0
baseline	1bit: 64.6, 2bit: 75.7, 3bit: 77.4, 4bit: 78.0					

Table 2: Accuracy with dynamic bit-precision when α is 1/25, 1/50, and 1/100. We can confirm that adjusting α can yield an arbitrary trade-off between accuracy and computational complexity.

Figure 12 shows the accuracy of the top 15% of the output with high entropy. This figure shows that even a 4-bit weighted network has an accuracy of approximately 33% when the entropy of the output is high, whereas a 1-bit weighted network has an accuracy of approximately 23%. In other words, 67% of inputs can be wrong for both high and low bits. We can further reduce the computational resources by assigning such inputs that are wrong in both conditions to low-bits.

4.4 Hardware Resource Requirement for ProgressiveNN

In inference, ProgressiveNN's computation amount is not much different from a counterpart implementation based on MAC operations using a fixed-point multiplier because it changes only the calculation order but not the calculation itself. Also, the additional computation cost for the bit-precision adjustment is negligible compared with the entire computation amount. The problem is that ProgressiveNN requires a specially designed processing unit. This section describes a design of processing units for ProgressiveNN, showing

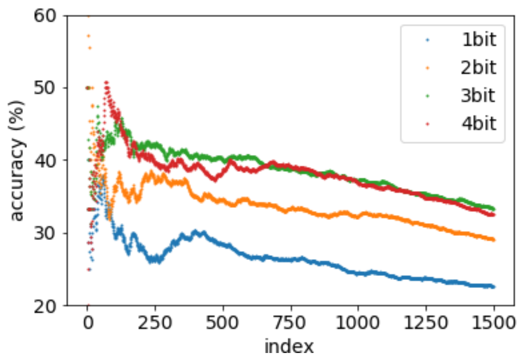


Figure 12: Accuracy of the instances of the top 15% entropy.

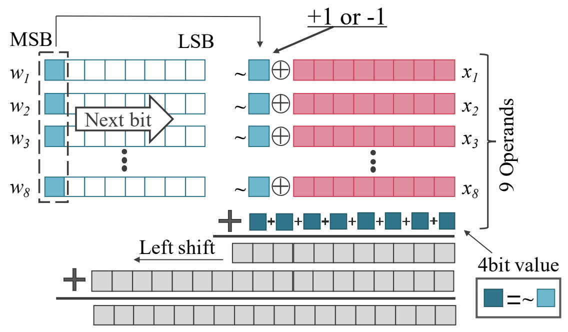


Figure 13: Arithmetic procedure of ProgressiveNN, combining eight 8-bit MAC operations, where w_i represents the weights and x_i represents the corresponding activation. Red boxes represent the bits of activation values, and blue boxes represent each weight's bit used in the current calculation. When the weight bit is 0, bit inversion occurs in the corresponding activation value, and the +1 for the complementary conversion is added as the ninth operand.

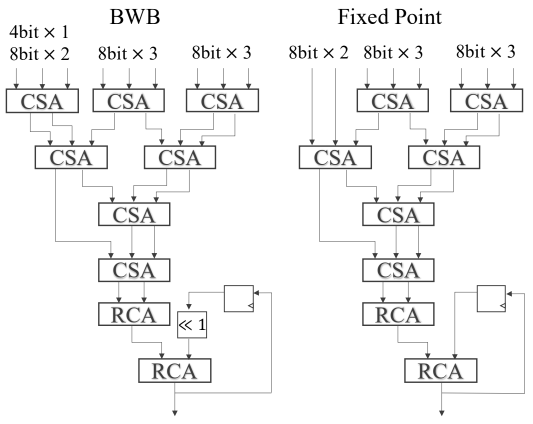


Figure 14: Implementation example of accumulation operations with Wallace Tree using CSA. The left figure shows the accumulation of 9 operands in BWB representation, and the right figure shows the accumulation of partial products of 8 operands in 8-bit fixed-point multiplication.

	BWB		Fixed-point	
Level	FA	HA	FA	HA
1	20	4	12	4
2	16	0	14	3
3	8	0	7	3
4	9	0	8	2
5	9	1	9	1
6	18	0	18	0
Sum	71	5	68	13

Table 3: Number of FAs and HAs used at each level in Figure 14.

that its necessary hardware resources are equivalent to multiplier-based implementations.

Figure 13 depicts an example of the arithmetic procedure in ProgressiveNN, combining the eight 8-bit multiply-accumulate (MAC) operations in MSB-first order. Red boxes represent the bits of activations, and blue boxes represent each weight's bit used in the current calculation. Because 1 and 0 represent +1 and -1 in BWB-quantized weights, multiplying a 1-bit weight to an activation value is a sign inversion only when the weight bit is 0. Considering the sum of +1 for each two's complement conversion is the ninth operand, we can build the MAC calculator for ProgressiveNN based on Wallace tree with eight 8-bit activations and one 4-bit value resulting from eight 1-bit summations. Also, for summing results up from MSB to LSB, the processing unit needs an accumulator and a left shifter.

The BWB-based MAC calculator mentioned above has a pretty similar structure to a fixed-point MAC calculator. Figure 14 compares the core parts of both hardware implementations: the left side is a BWB-based implementation, and the right side is a fixed-point implementation. In Figure 14, we assume that each implementation uses carry-save adders (CSAs) and ripple-carry adders (RCAs) to simplify comparison, and the left and right implementation are signed and unsigned configuration, respectively. Table 3 details the numbers of full adders (FAs) and half adders (HAs) used in each level of both Wallace trees. As shown in Table 3, required hardware resources are almost equivalent in both cases. Also, the BWB-based calculator

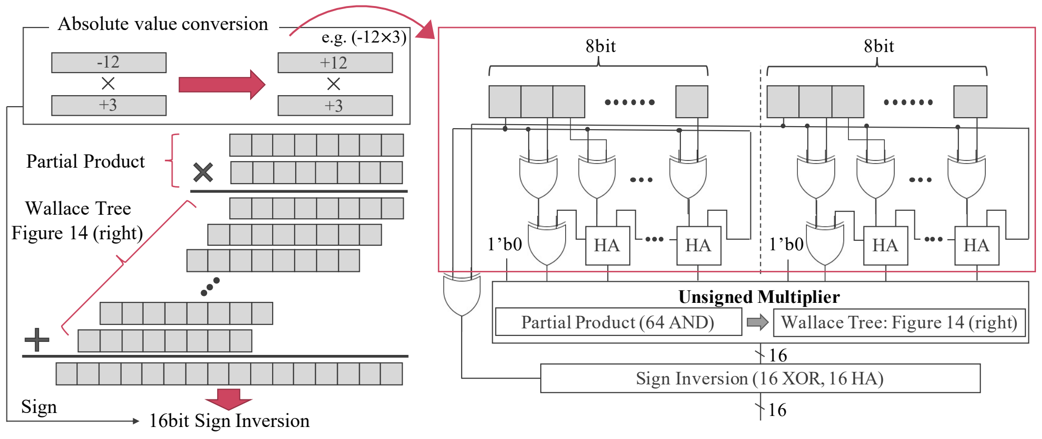


Figure 15: The naive fixed-point signed multiplier. This multiplier calculates the two inputs' absolute values in advance and performs sign inversion after unsigned multiplication.

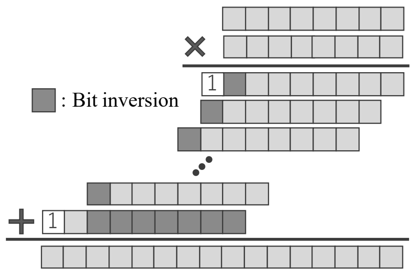


Figure 16: 8-bit fixed-point signed multiplication using the Baugh-Wooley algorithm.

Operation	BWB Number	BWB NAND	Fixed-point Number	Fixed-point NAND	Fixed-point w/ Baugh-Wooley Number	Fixed-point w/ Baugh-Wooley NAND
AND	0	0	64	96	64	96
NOT	8	4	0	0	14	7
XOR	64	128	33	66	0	0
FA	71	603.5	68	578	68	578
HA	5	17.5	41	143.5	13	45.5
Sum		753.0 (1.00×)		883.5 (1.17×)		726.5 (0.96×)

Table 4: Comparison of the number of operations used by 8-bit MAC operations. A comparison is made for the NAND gate.

requires extra 64 XOR and 8 NOT gates for the sign inversion, but the naive fixed-point calculator, as shown in Figure 15, requires 64 AND, 33 XOR, and 28 HA gates to handle signed values as well. This naive calculator utilizes the unsigned multiplication by calculating the two inputs' absolute value in advance. The unsigned multiplication feeds the outputs of the partial product to Wallace Tree shown in the right panel in Figure 14, and the sign inversion converts the output's sign based on the two inputs' sign. However, this comparison is not fair enough because using the Baugh-Wooley algorithm provides more efficiency for signed fixed-point multiplication [23]. Figure 16 shows an 8-bit fixed-point signed multiplication using the Baugh-Wooley algorithm. In this case, signed multiplication requires sign inversion only for 14 bits, which is smaller than the implementation that considers absolute values. As a result, the efficient fixed-point MAC calculator requires 64 AND and 14 NOT gates.

Table 4 summarizes the hardware resource requirements and compares them in the number of NAND gates. From this table, we can confirm that the hardware resource requirements for the BWB-based calculator are equivalent to fixed-point calculators. Furthermore, ProgressiveNN, based on this BWB-based calculator, has more flexibility than neural networks implemented with fixed-point calculators. It can adjust computation amount when needed and even reduce unnecessary computation in the early-stage by activation prediction described in the following section.

4.5 Activation Prediction

In this section, we discuss further reduction in computation by using activation prediction with ProgressiveNN. ABS inference adjusts the computational cost by calculating each inner product value, z, in the MSB-first order. Therefore, once the calculation with 1-bit BWB weights is complete, we can predict with high probability

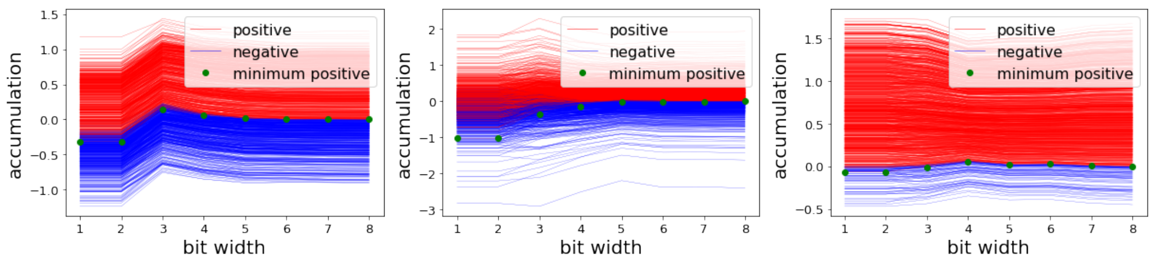


Figure 17: Accumulation of each bit-wise inner product observed from four channels of the 1st convolution layer of ResNet18. The red and blue lines are observed from five images of CIFAR-10. Blue lines represent the final values that are negative at 8-bit, and red lines represent the final value that is positive at 8-bit. Green circles denote the minima of all red lines, and the blue lines below the green circles can be omitted for computation reduction.

whether the inner product takes a negative or positive value in multiple bit-widths. If the activation function is the rectified linear unit, predicting the sign of the final value in advance can save computational cost for negative values by immediately stopping calculation. Figure 17 shows the accumulation of each bit-wise inner product observed from four channels of the first convolutional layer, where each line is obtained from five input images of the CIFAR-10 dataset. Red and blue lines represent the values that take positive and negative values at 8-bit, respectively. Green circles represent the minima of all red lines, and we can stop the calculation for the blue lines below the green circles immediately. Because we examined a small amount of data in this study, it is not easy to define a general expectation of the effectiveness of activation prediction using ProgressiveNN. From this point of view, [24] provides helpful information that shows the activation density of various layers and networks. Activation density is typically higher in early layers but can be as low as 30% in late layers. Assuming 50% activation density on average, we can reduce the computation for negative values to 1/8, which is 44% of the entire calculation.

5 Conclusion

In this paper, we proposed ProgressiveNN and its dynamic bit-precision adjustment, which enables computationally scalable inference, consisting of BWB quantization, BN retraining, and ABS inference. By selecting the least bit-width of weights, ABS inference can successfully obtain satisfactory inference accuracy with the minimum computational cost. We showed that retraining BN for each bit-width of BWB expression suppresses the accuracy drop of ABS inference for low-computational-cost applications. We also analyzed the accuracy drop before BN retraining and confirmed the distortion of the weight distributions in low-bit BWB expressions. Then, we showed that the dynamic bit-precision adjustment could achieve the desired trade-off between accuracy and computational cost. Additionally, we indicated hardware resource requirement between BWB-based calculator and a conventional fixed-point calculator are almost equivalent. Moreover, we introduced the concept of computation reduction using activation prediction based on ABS inference. Under the constraint of computational resources and power consumption, the proposed method can achieve high accuracy while satisfying the constraints. In addition, users can adjust this trade-off for their own purposes. We expect that the proposed method will help reduce the power consumption of edge devices. The proposed method can also be applied to existing networks without any network alteration; therefore, we consider this to be applicable in various fields. In future research, we intend to apply ProgressiveNN to real-time visual-object-recognition systems on edge devices.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Numbers JP18H05288 and JP19H04079.

References

[1] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *Proceedings of Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[2] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. arXiv:1502.02551, 2015.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. arXiv:1511.00363, 2015.

[4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. arXiv:1602.02830, 2016.

[5] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. arXiv:1603.05279, 2016.

[6] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. arXiv:1605.04711, 2016.

[7] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. arXiv:1612.01064, 2016.

[8] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. arXiv:1702.03044, 2017.

[9] Cong Leng, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with ADMM. arXiv:1707.09870, 2017.

[10] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of European Conference on Computer Vision*, pages 3–18, 2018.

[11] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of International Conference on Machine Learning*, pages 527–536, 2017.

[12] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. SkipNet: Learning dynamic routing in convolutional networks. In *Proceedings of European Conference on Computer Vision*, pages 409–424, 2018.

[13] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. arXiv:1908.06294, 2019.

[14] Yu-Shun Hsiao, Yun-Chen Lo, and Ren-Shuo Liu. FlexNet: Neural networks with inherent inference-time bitwidth flexibility. In *Proceedings of International Symposium on Microarchitecture*, 2018.

[15] Yun-Nan Chang and Yu-Tang Lin. Scaling bit-flexible neural networks. In *Proceedings of International SoC Design Conference*, pages 253–254, 2019.

[16] Junnosuke Suzuki, Kota Ando, Kazutoshi Hirose, Kazushi Kawamura, Thiem Van Chu, Masato Motomura, and Jaehoon Yu. ProgressiveNN: Achieving computational scalability without network alteration by MSB-first accumulative computation. In *Proceedings of International Symposium on Computing and Networking*, pages 215–220, 2020.

[17] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. Training batchnorm and only batchnorm: On the expressive power of random features in CNNs. In *Proceedings of International Conference on Learning Representations*, 2021.

[18] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *Proceedings of IEEE International Symposium on Compututer Architecture*, pages 764–775, 2018.

[19] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In *Proceedings of Advances in Neural Information Processing Systems*, pages 15173–15184, 2020.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[23] Charles R. Baugh and Bruce A. Wooley. A two's complement parallel array multiplication algorithm. In *IEEE Transactions on Computers*, volume 22, pages 1045–1047, 1973.

[24] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of IEEE International Symposium on Compututer Architecture*, pages 27–40, 2017.