

RESEARCH ARTICLE

Heuristic Compression Method for CNN Model Applying Quantization to a Combination of Structured and Unstructured Pruning Techniques

DANHE TIAN¹, SHINICHI YAMAGIWA², (Senior Member, IEEE), AND KOICHI WADA², (Member, IEEE)

¹Doctoral Program in Computer Science, University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan

²Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan

Corresponding author: Shinichi Yamagiwa (yamagiwa@cs.tsukuba.ac.jp)


This work was supported in part by JST SPRING under Grant JPMJSP2124, in part by JST PRESTO under Grant JPMJPR203A, and in part by JST AIP Acceleration Research under Grant JPMJCR24U4.

ABSTRACT Model Compression is an actively pursued research field in recent years with the goal of deploying state-of-the-art deep neural networks. It is targeted to implementations which are based on power constrained and resource limited devices as the reduced model achieves without significant accuracy loss, but with effective resource size reduction. The network pruning and the weight quantization techniques are well-known model compression methods. Our previous work successfully demonstrated significant reductions regarding the network model size by applying a managed combination of the structured and unstructured pruning methods. In order to achieve further reduction of the model, this paper introduces new heuristic methods that employ a weight quantization technique with both structured and unstructured pruning methods as those keep a given target accuracy. We experimentally demonstrate the performance evaluations of the proposed method by applying it to the actual state-of-the-art CNN models of VGGNet, ResNet and DenseNet under well-known CIFAR-10 dataset. In the best case during our experimental outcomes, the proposed method achieves the reduction of 28 times less model size and 76 times less compression processing time compared to the brute-force search method.

INDEX TERMS Convolutional neural network, model compression, quantization, structured pruning, unstructured pruning.

I. INTRODUCTION

Convolutional neural networks (CNNs) have achieved remarkable success in many cognitive tasks such as computer vision [1], [2], speech recognition [3], [4] and autonomous driving [5], [6]. However, in order to achieve higher inference accuracy, recent state-of-the-art CNN networks tend to organize a quite deep architecture that consists of many layers and parameters [7]. These deep CNNs require significant amount of implementation resources such as computational power, amount of memory, and electric power for the

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

processing platform. Especially, we need to consider those aspects when we deploy them on IoT environments such as edge devices where the computing resources are limited.

Recently, to overcome the resource problem, model compression is an active research area in the last decade. The extensive works have been proposed to compress large-scale CNN models with obtaining a required accuracy [8], [9]. Then, the techniques allow us to reduce the implementation resources and to bring compact implementation. In software implementation of a CNN-based inference, calculation overhead will be reduced by less nodes or channels of a network model. And in hardware implementation of the inference, computing resources will be reduced, and we can

implement it in a compact integrated circuit. The model compression techniques can be categorized into two major types: quantization and network pruning. Quantization is often employed to implement models on hardware such as Field Programmable Gate Array (FPGA). Quantization is a technique to compress a model by reducing the number of bits required to represent parameters such as weights or activations. On the other hand, the network pruning method can be also categorized into two methods: unstructured and structured ones. The unstructured one tries to delete individual weights or neurons in fully connected layers while the structured one does entire network structures such as channels or layers. We combined the unstructured and the structured methods to reduce network model [10]. The technique compresses a given CNN model with achieving a given target inference accuracy. However, we did not care about the reduction technique focusing on shrinking data type of weight parameters, which is the quantization. We have performed consistently our combined method by applying 32bit floating point to weight parameters. If we can reduce the number of bits for the parameters, we expect to derive more reduced model. However, the quantization often raises a problem of significant accuracy degradation. Furthermore, it is always intuitive to determine the reduction ratio when we apply the quantization technique by combining the structured and the unstructured pruning methods. Therefore, there is no concrete algorithm that promises reducing the network model size by using three compression techniques above. This paper proposes a novel heuristic method that combines quantization, structured and unstructured pruning techniques to derive a minimal network model as promising the target accuracy.

The main contributions of this paper are summarized as follows:

- 1) First, we found a method for compressing CNN model by combining three compression techniques; quantization, structured and unstructured pruning methods.
- 2) We developed an algorithm that achieved more improved compression ratio than our previous approach in [10] as promising a given target accuracy.
- 3) We have demonstrated efficiency of our proposed method according to evaluations with six state-of-the-art CNN models and validated efficiency of our proposed algorithm.
- 4) Finally, our proposed algorithm significantly shrinks computational time for finding minimal compressed model, comparing to the case of the brute-force-search method.

The rest of this paper is organized as follows. The next section introduces background and state-of-the-art model compression techniques. The section will also explain our previous work. Section III will describe our proposed heuristic method based on examples. Section IV will show experimental evaluations by applying our proposed method to actual CNN models. Finally, we will conclude the paper and describe our future plans.

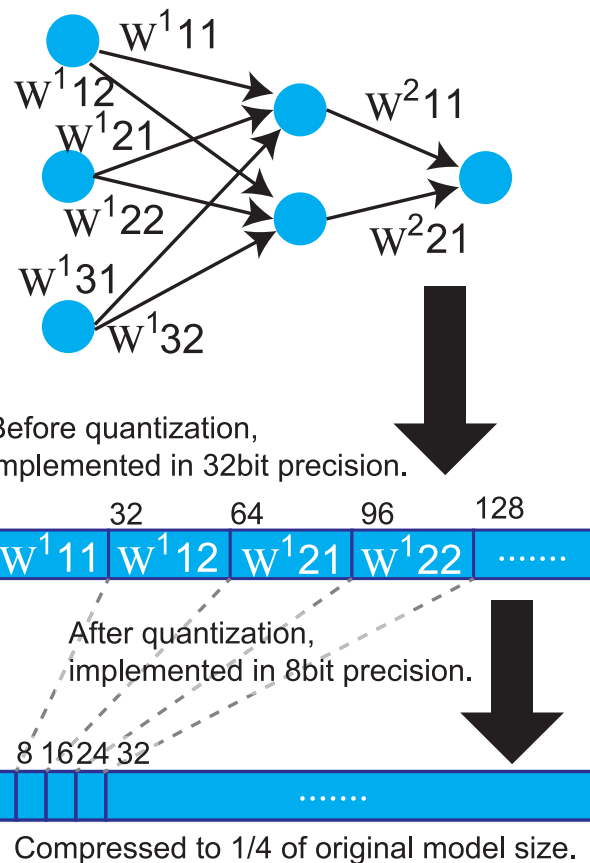


FIGURE 1. Integer quantization example. According to 8bit precision representation for weights, the model size of 32bit can be reduced to 1/4 of its original size.

II. BACKGROUNDS AND DEFINITIONS

A. COMPRESSION TECHNIQUES FOR CNN NETWORK MODELS

Network pruning and quantization techniques are known as common and effective approaches for model compression techniques. The network pruning techniques reduce the size of CNN models by eliminating redundant network components. According to part of the network component to be removed, the techniques can be categorized into two types: structured and unstructured. The structured pruning technique tries to remove available components by scanning the entire network such as channels or filters based on a predetermined criterion. The approach reduces the model size while maintaining the overall structure of the network. On the other hand, the unstructured one does individual weights or connections in the network by focusing on less important weights or connections based on a certain criteria or a given threshold of the pruning ratio. The structured and the unstructured pruning techniques are respectively known as network slimming proposed by Liu et al. [11], and deep compression that the weight pruning method is used [12]. The former one identifies redundant channels in convolutional layers of a network and prunes them. The latter one reduces the weights under a specified threshold of the pruning ratio.

On the other hand, quantization represents weight parameters and activations in low-precision data types like 8bit integer (widely represented as `int8`) instead of a wide range of floating point such as `float32/64`. The reduction of the number of bits brings less memory requirement and also causes less energy consumption without modifying the structure of the network itself. The literature [13] reported resource impact of limited precision regarding 16bit floating point on deep learning. We can also find advantages especially in hardware implementation from survey literature [14]. Figure 1 shows a typical example of reducing the number of bits in weight parameters by applying the quantization technique. It shows a 75% reduction of the total resources in bits while the type of weights is changed from 32bit floating point expression to 8bit integer.

The quantization techniques are categorized to floating-point-based and integer-based techniques. Settle et al. [15] introduced a low-precision floating-point quantization method. Through applying 16bit or 8bit floating-point quantization, the network can be reduced to 50% or even 25%. However, as multiply-accumulators (MACs) based on floating-point need much more hardware resources than the ones based on integer, the authors show the effectiveness on some slim CNNs only such as GoogleNet [16] and MobileNet [17]. He [18] proposed a neural network for image classification based on 8bit integer quantization to reduce the model size. The network model improved response time of inference process. Recently, many effective automated quantization techniques have been proposed due to rapid development of reinforcement learning [19]. Elthakeb et al. [20] proposed ReLeQ framework for automating determination of quantization levels regarding different network components (channels, layers etc.) through deep reinforcement learning technique. Wang et al. [21] employed HAQ (Hardware Aware Quantization) method that leverages the reinforcement learning to simulate the target hardware architecture. It also automatically determines an “optimal” mixed-precision configuration by considering the hardware-specific constraints such as memory size limitation and computational capability. The techniques based on the reinforcement learning achieves good compression ratio by dynamically adapting and determining optimal quantization strategies. However, it needs numerous iterations and adjustments for deriving compressed model. And the computational requirements can be substantial due to increasing compressing time and demanding significant computational resources for the reduction process.

B. MODEL COMPRESSION USING STRUCTURED AND UNSTRUCTURED PRUNING TECHNIQUES

Our previous work [10] effectively reduced network model size by combining the network slimming and the deep compression as keeping a given target inference accuracy. In order to control the balance between the target accuracy and the reduced model size, we employed a *margin*

TABLE 1. Comparisons of reduction ratios when five models are compressed by two execution orders (NS→DC and DC→NS), where 90% target accuracy is specified.

Model	NS→DC	DC→NS
VGG-19	92.31%	90%
ResNet-110	68.75%	67.74%
DenseNet-40	78.72%	77.78%
DenseNet-121	94.12%	92.86%
DenseNet-202	96.30%	95.65%

calculation to adjust the reduction ratio of these pruning methods. We experimentally determined execution orders of those pruning methods. Applying five well-known CNNs, we investigated orders of NS→DC (this means the network slimming is first invoked and the deep compression is done later) and DC→NS (this means vice versa) under a condition of 90% target accuracy. The reduction ratios (i.e. calculated by (1-compressed network size/original network size)) are shown in Table 1. The sequence of NS→DC consistently achieved a better reduction ratio than the one of DC→NS. Remarkably, for large network models (VGG-19, DenseNet-121, and DenseNet-202), both sequences achieved reduction ratios of more than 90%. Therefore, we confirmed that the sequence of NS→DC performed better than the one of DC→NS. Especially for the DenseNet-202, NS→DC achieved 96.30%. Even when smaller models that are not larger than 5MB (ResNet-110 and DenseNet-40) are applied, NS→DC slightly degrades than DC→NS. Thus, we have heuristically chosen NS→DC as a suitable execution order between two pruning methods.

However, we still have more chance to reduce the network model size with respect to the weight parameters by applying the quantization technique. This raises another problem of the order among two pruning techniques and the quantization. Although the number of bits in a weight parameter can be reduced statically, it is not clear how the quantization affects the inference accuracy of the reduced network before/after individual pruning techniques.

C. DISCUSSION

The combination of structured and unstructured pruning techniques can achieve a better compression ratio in the order of NS→DC than the one when each technique is individually applied to a network model. The quantization can be another possible method for further compression with the pruning methods. We expect that the model size can be reduced by combining the integer quantization with the pruning methods.

Here, we need to consider the best timing to apply the integer quantization that maps floating point parameters to integer-based ones regarding the representation of weights in a model. For instance, if the integer quantization is applied before the pruning methods, the subsequent model compression process is performed based on integer. This lets us concern significant accuracy loss from the beginning of the sequence. If the quantization is performed between the network slimming (in short, NS) and the deep compression

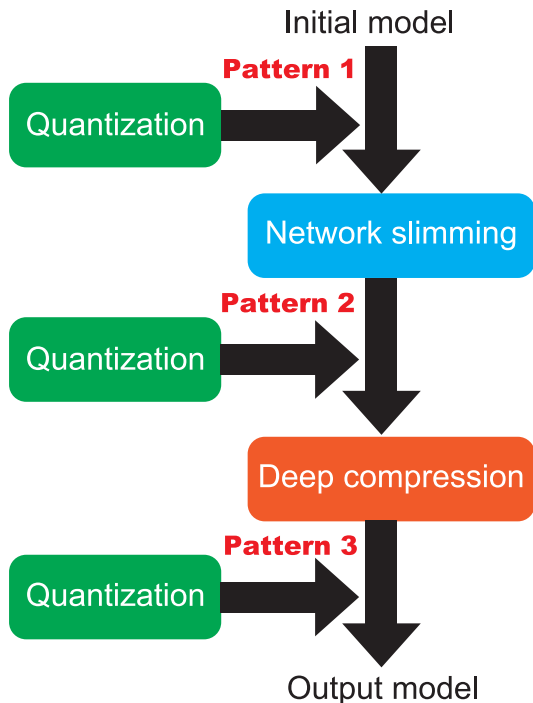


FIGURE 2. Three possible sequences among NS, DC and the integer quantization where the order of NS→DC is preserved.

(in short DC), it can affect to the pruning performance of DC, and the impact of the reduction on the fully connected layers cannot be predicted. If the quantization is applied after the pruning methods, it can cause a sharp drop of inference accuracy because the precision of weight parameters degrades. Thus, these three methods influence each other. Our main aim in this paper is to find the best sequence order of the methods and the best reduction control with a given target accuracy. Thus, in this paper, we will propose a method that heuristically and automatically finds the best order among the network slimming, the deep compression, and the integer quantization methods.

III. HEURISTIC COMPRESSION METHOD FOR CNN MODEL APPLYING INTEGER QUANTIZATION

A. STRATEGY FOR SIZE REDUCTION USING QUANTIZATION AND PRUNING METHODS

Since we have confirmed that NS→DC is the best execution order regarding the pruning techniques, there are three possible combination patterns with the integer quantization as illustrated in Figure 2. The “Pattern 1” in the figure begins from the quantization right before NS→DC sequence. It first reduces the model size by shrinking the number of bits of weight parameters and tries the subsequent compression steps. In the sequence, the network slimming reduces convolutional layers and then, deep compression reduces redundant weight parameters. The “Pattern 2” performs the quantization between NS and DC. Beginning with the network slimming, the sequence can preserve initial network

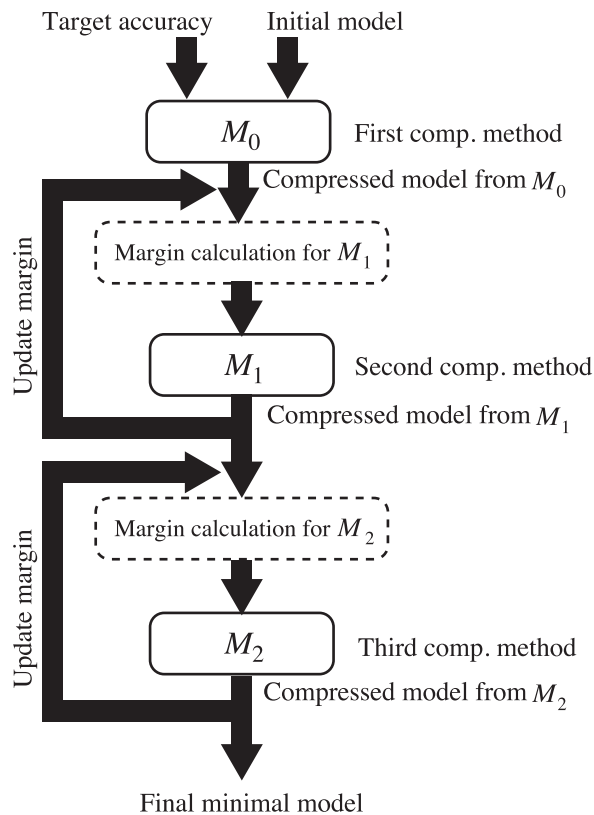


FIGURE 3. The flowchart of the proposed method.

architecture and maintains a given target accuracy. However, it can limit the deep compression to remove more parameters. And then, it can result accuracy drop for inference due to the potential reduced precision by the quantization. Finally, the “Pattern 3” performs the quantization after NS→DC. Our prior research concluded that the sequence employing NS→DC achieved the best compression ratio. Applying the integer quantization after this sequence offers additional compression trial. However, the improvements can be limited. After the deep compression removes unimportant weight parameters, the sequence invokes the quantization. This would bring a significant accuracy loss.

According to the available patterns mentioned above, we apply the following outline for our combined compression method. In order to find a reduced minimal model that satisfies a given target accuracy, a sequence invokes the first compression method as increasing the compression ratio. At every compression ratio, it checks inference accuracy of the given model. The compression ratio is a percentage given to NS or DC. It is called *pruned rate* for NS and *compress rate* for DC in the literatures [11] and [12] respectively. The pruned rate specifies a reduction percentage of channels to be removed from a given model. Similarly, the compress rate specifies a reduction percentage of weight parameters. Once the best compression ratio is found in the current compression method, the compressed model is passed to the subsequent method for the further compression. Here, in the

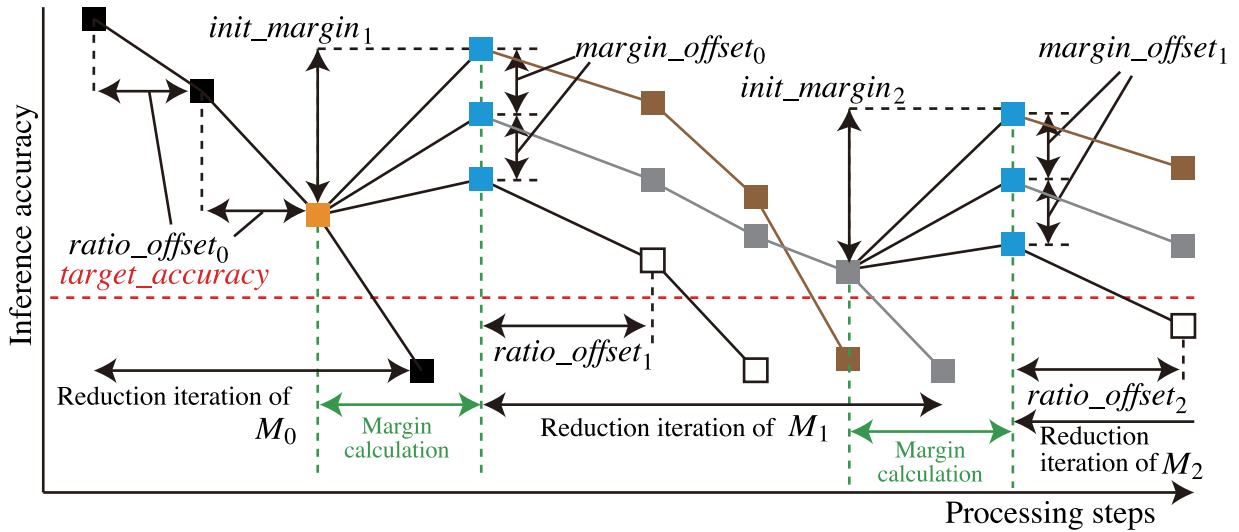


FIGURE 4. An example of the margin calculation with multiple iterations for finding a minimal compressed model.

case of the integer quantization method, the compression ratio is equivalent to the number of bits reduced in a weight parameter. The sequence increases every bit while checking inference accuracy. We apply 32bits as the initial number of bits. In the explanation below, we consider the compression ratio defined by $(32 - \text{the_number_of_reduced_bits})/32 \times 100$. And then, the sequence will find a minimal number of bits that still meets the target accuracy.

Here, we have to consider the optional treatment for specifying a compression ratio after a method in a sequence. When a compression method has been performed by the best ratio, the subsequent method may not have any availability for further compression. To avoid this situation, we introduce *margin calculation* as shown by the dotted boxes in Figure 3 when a sequence switches from a compression method to another. The margin calculation restores the minimal model derived from a step in a given compression ratio before. Assume M_i is a compression method of the integer quantization, the network slimming and the deep compression where $0 \leq i \leq 2$. Each M_i tries to compress a model as maintaining a given target accuracy by repeating M_i with increasing the compression ratio $ratio_j$. At every iteration of j , M_i stores the model m_j if the accuracy is more than or equal to a target accuracy acc_target . When the accuracy becomes less than acc_target , the model m_{j-1} is chosen as the minimal model. Here, note that m_k is passed to the subsequent method M_{i+1} , which is derived from a compression ratio $ratio_{j-1} - init_margin_{i+1}$. The $init_margin_{i+1}$ is given as parameters for M_{i+1} to control the margin calculation, where $0 \leq i \leq 1$. This operation maintains compression availability for the subsequent compression method because the compression ratio is restored to the previous *margin* percentage. The sequence additionally iterates the margin calculation with decrementing $margin_offset_{i+1}$ from the margin value. Figure 4 illustrates an example of the

margin calculation and how M_i passes the model m_k to the subsequent compression method M_{i+1} . The figure also illustrates examples of accuracy changes observed in each execution during this procedure. First, the sequence iterates M_0 by applying the initial model until the accuracy reaches beneath the given target accuracy (dotted red line). The iteration is performed by increasing the compression ratio with a ratio offset $ratio_offset_0$. When the accuracy reaches beneath the target accuracy, we store the compressed model m_{j-1} (yellow square) as a minimal model at j -th iteration. Before passing this compressed model to M_1 , the sequence applies a margin calculation applying an initial margin $init_margin_1$ and passes the stored model m_k that is derived from the compression ratio of $ratio_{j-1} - init_margin_1$ to the subsequent compression method M_1 . The model (blue square) can be further compressed by M_1 . We can find a minimal model when the accuracy falls beneath the target accuracy. Here, the margin is increased by the margin offset $margin_offset_1$. Again, the model derived from M_0 is restored, and the iteration to reduce the model is performed. The figure shows the iterations related to the margin calculation during three margin values. The sequence repeats the margin calculation while the margin is positive value. After the iteration of the margin calculation breaks, M_1 selects a minimal model from the multiple reduced models stored at every margin calculation. The subsequent compression method M_2 performs also the same as the margin calculation as M_1 . Finally, the initial CNN model is compressed to a minimal model size that maintains a given target inference accuracy.

As considering the combinations of the integer quantization, the network slimming and the deep compression with obtaining the order NS→DC, we have three available sequences for deriving a minimal model: IQ→NS→DC, NS→IQ→DC and NS→DC→IQ. In the following sections,

let us investigate the best sequence for deriving an effective minimal model with a given target inference accuracy.

B. DETAILED STEPS FOR SEQUENCES DERIVING MINIMAL CNN MODEL

Now, let us summarize the compression flow of the proposed method in Algorithm 1. Each of M_0 , M_1 , and M_2 represents one of IQ, NS and DC. The given target accuracy for inference is acc_target in the algorithm. The algorithm first receives the pre-trained model from the input parameter $init_model$. Additionally, initial compression ratios ($init_ratio_i$ where $0 \leq i \leq 2$) are given by the input parameters. In the case of IQ, the ratio refers to the number of bits to be reduced. For the other methods, it can be set to zero (i.e. no compression). Regarding NS, the ratio is equivalent to the *pruned ratio* as defined in the literature [11], which is a percentage of channels in a given model reduced by the method. Regarding DC, the one is equivalent to the *compress rate* as defined in the literature [12]. It is also a percentage of weight parameters in a given model reduced by the method. Here, note that the ratios of three methods affect to the corresponding parts of a given model. A model derived from one of the methods is reduced to mostly *ratio %*. However, the inference accuracy after the reduction is not predictable. At performing every method by the function $M_i(model, ratio)$, the accuracy must be confirmed if it satisfies the given target accuracy acc_target or not. The ratio is decreased by $ratio_offset_i$ that is an offset of the compression ratios for the next iteration. For the integer quantization method, the offset represents the number of bits. The compression trial of the first method M_0 is performed (lines 3-7) until the inference accuracy acc becomes lower than the target accuracy by checking it at every compression iteration with updating the compression ratio $ratio_0$ (line 4). Here, the $ratio_0$ derived after the iteration of line 3-7 is the minimal compressed model by M_0 . But, the ratio is restored by reducing $ratio_offset_0$ because the ratio in the previous iteration results an actual minimal model. By using the ratio, the minimal model is derived from the iteration (line 12) and set it to the model used in the next method (line 13). Then, the margin calculation is performed. First, an initial margin is set (line 8). Here, the unit of the margin is as well as the initial ratio parameter. The $ratio_0$ is restored by decreasing the margin value (line 22), and the model is updated (line 17) according to the ratio. With applying M_1 to the reduced model resulted from M_0 , the method reduces the model size as increasing the $ratio_1$ (line 15-19). Similar to the case of M_0 , the minimal model is derived by M_1 . The model size is compared with the min_model that was derived from the previous margin iteration (line 20-21). When all margin offsets are investigated, the min_model is passed to the next method M_2 . The iterations regarding M_2 are equivalent to the ones of M_1 as applying the similar margin calculation. Finally, the minimal model is found (line 40).

We can expect the reduction performance of the algorithm depends on the parameters of $init_ratio_i$, $ratio_offset_i$,

Algorithm 1 Model Compression Flow of Proposed Method

Input: $init_model$: initial model, $init_ratio_i$: initial reduction ratios, $init_margin_i$: initial margin values, acc_target : target accuracy, $ratio_offset_i$: reduction ratio offsets, $margin_offset_i$: margin offsets, where $0 \leq i \leq 2$.
Output: $final_model$: a minimal model after the sequence.

```

1:  $ratio_i \leftarrow init\_ratio_i$ 
2:  $model \leftarrow init\_model$ 
3: do ▷ Begin compression process for  $M_0$ 
4:  $prev\_ratio \leftarrow ratio_0$ 
5:  $(acc, model) \leftarrow M_0(model, ratio_0)$ 
6:  $ratio_0 \leftarrow ratio_0 + ratio\_offset_0$ 
7: while  $acc \geq acc\_target$ 
8:  $margin \leftarrow init\_margin_1$ 
9:  $ratio_0 \leftarrow prev\_ratio$ 
10: repeat
11:  $ratio_0 \leftarrow ratio_0 - margin$  ▷ Margin calculation for  $M_1$ 
12:  $(acc, model) = M_0(init\_model, ratio_0)$ 
▷ Compression Method switches to  $M_1$ 
13:  $min\_model \leftarrow model$ 
14:  $min\_ratio \leftarrow 100$  or  $B$ 
15: do ▷ Accuracy check
16:  $(prev\_model, prev\_ratio) \leftarrow (model, ratio_1)$ 
17:  $(acc, model) \leftarrow M_1(model, ratio_1)$ 
18:  $ratio_1 \leftarrow ratio_1 + ratio\_offset_1$ 
19: while  $acc \geq acc\_target$ 
20:  $\{min\_model, min\_ratio\} \leftarrow$ 
21:  $\min(\{min\_model, min\_ratio\}, \{prev\_model, prev\_ratio\})$ 
22:  $margin \leftarrow margin - margin\_offset_1$ 
23: until  $margin < 0$ 
24:  $margin \leftarrow init\_margin_2$ 
25:  $ratio_1 \leftarrow prev\_ratio$ 
26: repeat
27:  $ratio_1 \leftarrow ratio_1 - margin$  ▷ Margin calculation for  $M_2$ 
28:  $(acc, model) = M_1(init\_model, ratio_1)$ 
▷ Compression Method switches to  $M_1$ 
29:  $min\_model \leftarrow model$ 
30:  $min\_ratio \leftarrow 100$  or  $B$ 
31: do ▷ Accuracy check
32:  $(prev\_model, prev\_ratio) \leftarrow (model, ratio_2)$ 
33:  $(acc, model) \leftarrow M_2(model, ratio_2)$ 
34:  $ratio_2 \leftarrow ratio_2 + ratio\_offset_2$ 
35: while  $acc \geq acc\_target$ 
36:  $\{min\_model, min\_ratio\} \leftarrow$ 
37:  $\min(\{min\_model, min\_ratio\}, \{prev\_model, prev\_ratio\})$ 
38:  $margin \leftarrow margin - margin\_offset_2$ 
39: until  $margin < 0$ 
40:  $final\_model \leftarrow min\_model$ 
41: return  $final\_model$ 

```

$init_margin_i$ and $margin_offset_i$. These should be decided heuristically. We can apply simply 0 to $init_ratio_i$. However, we need to choose the $init_margin_i$ carefully because it

affects to reduction availability of given model for the subsequent compression methods. This is a heuristic issue that we need knowledge from experimental evaluation. Let us discuss it in the evaluation section later. On the other hand, $ratio_offset_i$ and $margin_offset_i$ bring computing overheads because those are granularity parameters of investigations for deriving a minimal model. To know how the overhead is expected, let us consider the complexity of our algorithm.

C. COMPLEXITY CONSIDERATIONS

Before we consider complexities of three sequences of IQ, NS and DC, let us consider a general complexity of Algorithm 1. The total elapsed time is defined by

$$Elapsed_time = \sum_{i=0}^2 Elapsed_time(M_i).$$

Assume $R_{full}(i)$ is the full percentage of reduction ratio regarding M_i (i.e. 100 for NS and DC, or B for IQ). The compression iteration for M_0 is expressed in the following equation;

$$\begin{aligned} Elapsed_time(M_0) &= \sum_{j=0}^{k_0-1} T_{M_0} \\ &= T_{M_0} \times k_0 = T_{M_0} \times \frac{R_{full}(0) - init_ratio_0}{ratio_offset_0} \end{aligned}$$

where T_{M_0} is the average elapsed time of each iteration in M_0 during the iteration, and k_0 is the number of iterations in the worst case. The minimal model should be derived by the smaller number of iterations than k_0 . The elapsed times of M_1 and M_2 include the margin calculation. The equations for both methods are the same. The elapsed time of M_i ($i = 1, 2$) is expressed as follows;

$$\begin{aligned} Elapsed_time(M_i) &= \sum_{n=0}^{m_i-1} \left\{ T_{M_{i-1}} + \sum_{j=0}^{k_i-1} T_{M_i} \right\} \\ &= m_i \times \left\{ T_{M_{i-1}} + k_i \times T_{M_i} \right\} \\ &= \frac{init_margin_i}{margin_offset_i} \times \left\{ T_{M_{i-1}} + \frac{R_{full}(i) - init_ratio_i}{ratio_offset_i} \times T_{M_i} \right\} \end{aligned}$$

where T_{M_i} is the average elapsed time of each iteration in M_i during the iteration, k_i is the number of iterations in the worst case, and m_i is the number of iterations of the margin calculation.

Here, let us consider the worst case of the elapsed time. When both the $margin_offset_i$ and $ratio_offset_i$ are 1, the calculation resolution becomes the highest. Here, the numbers of iterations for reduction by the methods (i.e. line 3-6, 13-16 and 27-30 in Algorithm 1) depends on the effects of the compression methods. Therefore, k_i in the equations above are predictable. Those numbers are depending on the offsets specified by $ratio_offset_i$. The numbers of iterations for the margin calculations are similar. The $margin_offset_i$ affects the numbers of iterations. As we

TABLE 2. The original model size and accuracies of CNN models used in experiments.

model	accuracy	Model size
VGG16	93.69%	58.9 MB
VGG19	93.99%	80.34 MB
ResNet-110	94.59%	4.61 MB
ResNet-560	95.47%	34.38 MB
DenseNet-121	95.51%	42.15 MB
DenseNet-201	95.99%	117.24 MB

discussed above, the complexity of our proposed method is derived as $O(k_i \times m_i)$ where i is 1 or 2, and decided by $\max(T_{M_1}, T_{M_2})$.

To evaluate the efficiency of our method, let us compare the computing overhead with the case when the brute-force search method is applied to find the minimal model. Assume the number of weight bits B , the elapsed time is defined as follows:

$$\begin{aligned} Elapsed_time &= \frac{B}{ratio_offset_{IQ}} \times T_{IQ} \\ &\times \frac{100}{ratio_offset_{NS}} \times T_{NS} \times \frac{100}{ratio_offset_{DC}} \times T_{DC}. \end{aligned}$$

The $ratio_offset_{\{IQ, NS, DC\}}$ are the offsets to increase the reduction ratios of IQ, NS and DC respectively. When the offsets are all 1, the worst case occurs. The equation becomes below;

$$Elapsed_time = B \times 100 \times 100 \times T_{IQ} \times T_{NS} \times T_{DC}.$$

Even if we consider the worst case of our method, k_i and m_i are 100 respectively. Therefore, the elapsed time depends on $10000 \times M_i$. It is obvious that the elapsed time of our proposed method is smaller than the one of the brute-force search method. Thus, our proposed method can achieve less elapsed time to find the minimized network model compared to the brute-force search method. However, the aspect of the inferred accuracy from the resulted minimal model is not defined in this section. This issue should be discussed heuristically using realistic models. The next section will show the evaluations focusing on the accuracy and the elapsed time of our method where three sequences of IQ→NS→DC, NS→IQ→DC and NS→DC→IQ are applied. Moreover, let us discuss which sequence is the most efficient to find a minimal model.

IV. EXPERIMENTAL EVALUATION

A. EXPERIMENTAL SETUP

We evaluate compression performances of our proposed algorithms using various state-of-the-art CNN models, which are VGGNet [22], ResNet [23] and DenseNet [24]. During the following experimental analysis, we utilize VGG-16 with 13 convolutional layers and a fully connected layer, and VGG-19 with 16 convolutional layers and a fully connected layer. Regarding the ResNet, we apply a 110-layer pre-activation ResNet with a bottleneck structure, denoted

TABLE 3. Reduced model sizes and reduction ratios of six CNN models derived from three sequences.

model	baseline	IQ→NS _{int} →DC _{int}	NS _{int} →IQ→DC _{int}	NS _{fp} →IQ→DC _{int}	NS _{int} →DC _{int} →IQ	NS _{fp} →DC _{fp} →IQ
VGG-16	58.9 MB	5.97 MB (90%)	8.34 MB (86%)	8.28 MB (86%)	7.52 MB (87%)	6.65 MB (89%)
VGG-19	80.34 MB	5.90 MB (93%)	8.05 MB (90%)	7.24 MB (91%)	9.13 MB (89%)	7.02 MB (91%)
ResNet-110	4.61 MB	1.08 MB (77%)	1.30 MB (72%)	1.30 MB (72%)	1.58 MB (66%)	1.58 MB (66%)
ResNet-560	34.38 MB	5.04 MB (86%)	7.91 MB (77%)	7.31 MB (79%)	10.5 MB (70%)	7.36 MB (79%)
DenseNet-121	42.15 MB	2.13 MB (95%)	3.47 MB (92%)	2.81 MB (93%)	7.10 MB (83%)	3.83 MB (91%)
DenseNet-202	117.24 MB	4.07 MB (97%)	8.53 MB (93%)	5.93 MB (95%)	9.96 MB (92%)	7.50 MB (94%)

TABLE 4. Minimal model sizes of CNN models derived from the sequence IQ→NS→DC, individual IQ, NS, DC and the our previous result of NS→DC.

model	baseline	IQ	NS	DC	NS→DC	IQ→NS→DC
VGG-16	58.9 MB	14.73 MB (75%)	13.94 MB (76%)	7.66 MB (87%)	6.58 MB (89%)	5.97 MB (90%)
VGG-19	80.34 MB	15.06 MB (81%)	10.2 MB (87%)	7.23 MB (91%)	6.11 MB (92%)	5.90 MB (93%)
ResNet-110	4.61 MB	1.44 MB (69%)	3.87 MB (16%)	1.46 MB (68%)	1.44 MB (69%)	1.08 MB (77%)
ResNet-560	34.38 MB	11.6 MB (67%)	10.3 MB (70%)	6.88 MB (80%)	6.34 MB (82%)	5.04 MB (86%)
DenseNet-121	42.15 MB	10.54 MB (75%)	8.72 MB (79%)	2.90 MB (93%)	2.49 MB (94%)	2.13 MB (95%)
DenseNet-202	117.24 MB	32.97 MB (72%)	15.09 MB (87%)	4.69 MB (96%)	4.32 MB (96%)	4.07 MB (97%)

TABLE 5. The margin values after NS ($margin_{NS}$) and DC ($margin_{DC}$) during the sequence IQ→NS→DC is performed.

model	$margin_{NS}$	$margin_{DC}$	$final_model_size$
VGG-16	4	10	5.97 MB
VGG-19	3	25	5.90 MB
ResNet-110	1	27	1.08 MB
ResNet-560	4	12	5.04 MB
DenseNet-121	2	27	2.13 MB
DenseNet-202	3	8	4.07 MB

TABLE 6. The elapsed times in cases of the sequence IQ→NS→DC and the brute-force search method.

model	IQ→NS→DC Hours(# of checks)	Brute-force search method Hours
VGG-16	8.6 (6042)	228
VGG-19	13.5 (7614)	240
ResNet-110	3.5 (4723)	266
ResNet-560	22.3 (7004)	1018
DenseNet-121	12 (6923)	896
DenseNet-202	63.7 (7214)	2960

as ResNet-110, and a 560-layer with a deeper network architecture denoted as ResNet-560. Regarding DenseNet, we apply two variations: DenseNet-121 (121 layers) and DenseNet-201 (201 layers). We adopt the network models with batch normalization layer according to [25] among all the CNN models. Table 2 provides those model sizes and inference accuracies derived from the original network models. Regarding DenseNet-121 (121 layers) and DenseNet-201 (201 layers), we adopt the network models with batch normalization layer according to [25] among all the CNN models. Table 2 provides those model sizes and inference accuracies derived from the original network models.

We use CIFAR-10 dataset [26] in the experiments. The dataset consists of 50K 32×32 color images in the training set and 10K images in the testing set. As the baseline models compared with our method, we show compression performances that IQ, NS and DC are individually applied to the same initial network models.

We investigate the compression efficiency based on the given target accuracy among three sequences; IQ→NS→DC,

NS→IQ→DC and NS→DC→IQ. We use a common $margin_offset_i = 1$ among three methods. The $init_margin_i$ are different among three methods. The unit of value depends on the sequence. In the margin calculation after IQ, the $init_margin$ is the number of bits to be restored. In the other cases, the unit is compression ratios of NS or DC. We apply 50 for $init_margin$ after NS and DC. This can cause underflow (i.e. less than 0%) of the ratio. When the margin calculations cause the underflow, the value is rounded up to 0. And we use 20 for the $init_margin$ of IQ. This value is intuitively decided from our experiments. This also causes underflow when the margin calculation results the ratio less than 0. In this case, the ratio is also rounded up to 0.

We perform the experiments in a GPU-based environment where an Intel Xeon E5-2698 V4 CPU with 512GB RAM works with a NVIDIA Tesla V100 with 32GB VRAM via PCI Express bus. On the CPU, Ubuntu 18.04 Linux operating system is working. Our proposed algorithm is implemented using the PyTorch framework [27].

B. PERFORMANCE COMPARISONS AMONG SEQUENCES

First, we perform experiments to find the most effective combination pattern among three sequences. Table 3 shows the results when the target accuracy is set to 90% for all experiments. Regarding the sequences of NS→IQ→DC and NS→DC→IQ, we have performed both patterns of DC and NS based on weight parameters of 32bit integer (shown as NS_{int}, DC_{int}) and 32bit floating point (shown as NS_{fp}, DC_{fp}). From the table, the results of the sequence IQ→NS→DC show that both VGG-16 and VGG-19 achieve reduction ratios more than 90%. Especially the one of VGG-19 achieves the ratio of 93%. It is higher than the ratios of the other sequences. In the case of ResNet models, the sequence IQ→NS→DC achieves 77% for ResNet-110 and 86% for ResNet-560. Additionally, the sequence IQ→NS→DC achieves the best reduction ratio of 72% for ResNet-110 among three sequences while the other sequences do the ones of 79% for ResNet-560. Focusing on the results from DenseNet models, we confirmed significant reduction ratios

based on the sequence $IQ \rightarrow NS \rightarrow DC$, which are 95% for DenseNet-121 and 97% for DenseNet-202. Thus, we have confirmed that the sequence brings the best compression ratio for six CNN models. According to the observations above, the sequence $IQ \rightarrow NS \rightarrow DC$ achieves the best performance. This means that the integer quantization before the network slimming derives smaller model size.

Let us compare performances among individual methods, the outcome from our previous research (i.e. $NS \rightarrow DC$) and $IQ \rightarrow NS \rightarrow DC$. During the experiments, the target accuracy is defined as 90%. Table 4 shows that the sequence $IQ \rightarrow NS \rightarrow DC$ reduces from 77% to 97% of the baseline model size. Its reduction ratios show more effective for larger models. As discussed above, we confirmed that the proposed method in the sequence $IQ \rightarrow NS \rightarrow DC$ reduces effectively the sizes of CNN models across various popular network architectures. It also performs better than individual compression methods as well as our previous method of the $NS \rightarrow DC$ sequence. Thus, we confirmed that the sequence $IQ \rightarrow NS \rightarrow DC$ achieves stable reduction performance for various CNN models and the reduction performance is robust.

Finally, let us discuss the most crucial pre-defined parameter in our proposed method, which is *init_margin*. It relates to the iterations for the margin calculation. When the *init_margin* is configured to a large value, the number of iterations for the margin calculation increases while the availability for reducing the model received from the previous method remains. On the other hand, when *init_margin* is small, the subsequent method cannot compress much because the margin calculation starts from a model adequately compressed by the previous compression method while the iteration of the margin calculation is small. We summarized the margin values after NS and DC in Table 5 when the sequence $IQ \rightarrow NS \rightarrow DC$ achieves the minimal model size. Note that the margin values during the iterations are maintained less than the $init_margin_i$ that we gave as the initial parameters. Therefore, we need to assign larger values for $init_margin_i$ than the ones when the minimal model size is derived. The largest margin numbers during the phases of NS and DC are 4 and 27 respectively. Therefore, in this paper, we heuristically assigned 20 and 50 to the *init_margins* for NS and DC phases respectively. These adjusted values will speed up the sequence to reach a minimal model. Based on the experiments above, the values we have used in this section can be good references when the proposed method is first applied to another CNN model.

C. PERFORMANCE EVALUATION REGARDING ELAPSED TIME

We further analyze the execution times of our proposed method. Table 6 shows the elapsed times of the sequence $IQ \rightarrow NS \rightarrow DC$ when the given CNN models are applied with 90% target accuracy. The time consists of mainly accuracy checks by comparing inference accuracies at every reduced model. The table also shows the numbers of the checks in the brackets. For the comparison, the table shows elapsed times

of the brute-force search method discussed in section III-C, which needs 320K accuracy checks to find a minimal model. As we can see in the results, we confirmed that our proposed method significantly reduces the number of accuracy checks comparing to the 320K times in the case of brute-force search method. In the case of ResNet-110, our method reduces 99% of the number of accuracy checks compared to the brute-force-search method, and thus, results 76x faster processing time to find a minimal model.

According to the experimental evaluations above, we have found that $IQ \rightarrow NS \rightarrow DC$ is the best sequence when we combine the integer quantization to our previous outcome. The compression performance shows significant achievement as increasing the given model size. Additionally, the sequence reduces the computational cost for finding the minimal model. Thus, we have proved that the sequence provides a good solution to reduce required computational resources when the minimal model is implemented on a CNN-based application.

V. CONCLUSION

This paper proposed a heuristic method for reducing the size of CNN models based on a combination of integer quantization, network slimming and deep compression, as a given target accuracy is maintained in the reduced model. We have also proposed an algorithm that combines three methods by using the margin calculation approach. Through experimental evaluations with six state-of-the-art CNN models, we found the best sequence was $IQ \rightarrow NS \rightarrow DC$ invoked in the proposed algorithm. We also found the sequence achieves stably a minimal size by applying our proposed algorithm with a given target accuracy. Furthermore, our method significantly reduces the processing time needed for finding a minimal model compared with the brute-force search method. In our proposed algorithm, we have some heuristic parameters. However, even if the values for the parameters are decided eventually, the proposed algorithm finds a minimal model in a less processing time than the case we apply the brute-force-search method for the combination of three compression methods.

For the future research, we plan to apply our method to larger and more complex CNN models in order to find a way that the best values can be assigned to the heuristic parameters. We will also evaluate the resource size of the minimized model required in hardware implementation such as an FPGA. Furthermore, we are focusing on effect of our method to decrease power consumption of not only software-based but hardware-based implementations of the reduced model compared to the cases when the original model. We will evaluate the effect by using GPU for software-based and FPGA for hardware-based implementations.

REFERENCES

- [1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

- [2] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, Dec. 2020.
- [3] A. Mehrish, N. Majumder, R. Bhardwaj, R. Mihalcea, and S. Porla, "A review of deep learning techniques for speech processing," 2023, *arXiv:2305.00359*.
- [4] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech, Lang., Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.
- [5] A. Agnihotri, P. Saraf, and K. R. Bapnad, "A convolutional neural network approach towards self-driving cars," 2019, *arXiv:1909.03854*.
- [6] X. Du, M. H. Ang, and D. Rus, "Car detection for autonomous vehicle: LiDAR and vision fusion approach through deep learning framework," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 749–754.
- [7] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.
- [8] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017, *arXiv:1710.09282*.
- [9] A. Berthelot, T. Chateau, S. Duffner, C. Garcia, and C. Blanc, "Deep model compression and architecture optimization for embedded systems: A survey," *J. Signal Process. Syst.*, vol. 93, no. 8, pp. 863–878, Aug. 2021.
- [10] D. Tian, S. Yamagiwa, and K. Wada, "Heuristic method for minimizing model size of CNN by combining multiple pruning techniques," *Sensors*, vol. 22, no. 15, p. 5874, Aug. 2022.
- [11] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2755–2763.
- [12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
- [13] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.
- [14] J. Lee, L. Mukhanov, A. S. Molahosseini, U. Minhas, Y. Hua, J. Martinez del Rincon, K. Dichev, C.-H. Hong, and H. Vandierendonck, "Resource-efficient convolutional networks: A survey on model-, arithmetic-, and implementation-level techniques," *ACM Comput. Surveys*, vol. 55, no. 13s, pp. 1–36, Dec. 2023.
- [15] S. O. Settle, M. Bollavaram, P. D'Alberto, E. Delaye, O. Fernandez, N. Fraser, A. Ng, A. Sirasao, and M. Wu, "Quantizing convolutional neural networks for low-power high-throughput inference engines," 2018, *arXiv:1805.07941*.
- [16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
- [17] 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," 2017, *arXiv:1704.04861*.
- [18] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.
- [19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
- [20] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmailzadeh, "ReLeQ: A reinforcement learning approach for automatic deep quantization of neural networks," *IEEE Micro*, vol. 40, no. 5, pp. 37–45, Sep. 2020.
- [21] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 8604–8612.
- [22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
- [23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2261–2269.
- [25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
- [26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Master's thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.
- [27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, and G. Chanan, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32. Red Hook, NY, USA: Curran Associates, 2019, pp. 8024–8035.



implementation in hardware, such as on FPGA.

DANHE TIAN received the B.S. degree in computer science from Liaoning Normal University, China, in 2016, and the M.S. degree in engineering from the University of Tsukuba, Japan, in 2018, where she is currently pursuing the Ph.D. degree in computer science. Her current research interests include applications and fundamental technologies of the edge computing field, especially compression technology for network models for CNN, edge applications using image recognition, and its



research interests include applications and fundamental technologies of the edge computing field, especially stream-based data compression technology, embedded systems, cloud computing, implementation techniques for digital systems using FPGA and LSI, computer architecture, and artificial intelligence focusing on sports sciences.

SHINICHI YAMAGIWA (Senior Member, IEEE) received the B.S. degree in information engineering, the M.S. degree in engineering, and the Ph.D. degree in engineering from the University of Tsukuba, Japan, in 2002.

He was a Postdoctoral Researcher and a Ph.D. Researcher with INESC-ID Lisbon, Portugal. Since 2012, he has been an Associate Professor with the Institute of Systems and Information Engineering, University of Tsukuba. His current



currently a Professor Emeritus with the University of Tsukuba. His research interests include high-performance system architecture, edge AI systems, and parallel and distributed computing.

KOICHI WADA (Member, IEEE) received the M.E. and Ph.D. degrees in computer science from Kobe University, Japan, in 1981 and 1984, respectively. From 1984 to 1987, he was an Assistant Professor with Kobe University. In 1987, he was a Lecturer with the University of Tsukuba, Japan. In 1999, he was a Professor with the Department of Computer Science, University of Tsukuba. He has been a Visiting Professor with the University of Victoria, Canada, in 2000. He is