Memory-Reduced Network Stacking for Edge-Level CNN Architecture With Structured Weight Pruning

Seungsik Moon, *Student Member, IEEE*, Younghoon Byun, *Student Member, IEEE*, Jongmin Park, *Student Member, IEEE*, Sunggu Lee, *Member, IEEE*, and Youngjoo Lee, *Member, IEEE*

***Abstract*— This paper presents a novel stacking and multi-level indexing scheme for convolutional neural networks (CNNs) used in energy-limited edge-level systems. Basically, the proposed scheme offers multiple accuracy modes by adopting a structured weight pruning method that enables a CNN to be trained once with multiple pruning ratios and thereby allows for adaptive energy-accuracy trade-offs. The memory overhead required to store several different networks is kept to a minimum by adopting a novel method for including smaller lower-accuracy networks as subnetworks of larger higher-accuracy networks and by using a unique multi-level indexing scheme that can effectively store compressed weight data for the proposed stacked-CNN architecture. Experimental results show that the proposed method successfully reduces the memory footprint by up to 33% when compared to a baseline CNN architecture. An FPGA-based multi-mode CNN accelerator that implements the proposed scheme has been designed. Energy usage analysis with a case study shows that the inference energy required for on-device CNN processing can be reduced by up to 1.94 times over the baseline design.**

***Index Terms*— Convolutional neural network, edge-level device, memory-reduced architecture, structured weight pruning.**

I. Introduction

IN THE past decade, the convolutional neural network (CNN) has been continuously improved in order to enhance its performance for images classification tasks [1]. Based on recent hardware-level innovations that helped achieve high-speed massive-parallel computations [2], a number of CNN algorithms have been adopted in practical real-time recognition systems [3]–[5], but only with limited capabilities. In order to achieve a high level of recognition accuracy, contemporary CNN architectures normally require an excessive number of computations, and are thus mainly implemented on high-performance computing servers [6]–[8]. Several studies have reported that CNN processing can require frequent access to power-hungry external memories in order to support the large and complex network structures used, thereby dissipating a huge amount of energy [9], [10]. Due to their limited computing power and energy budget, however, it is impractical to directly apply such CNN algorithms to edge-level devices, which are regarded as the next big market for the realization of on-device neural network processing [11]. When targeting edge-level solutions, therefore, what is urgently required is the development of aggressive optimization schemes for reducing the computational costs and memory requirements, which can make implementation of complex algorithms more realistic.

In general, an on-device CNN accelerator must be designed to minimize the energy consumption while achieving the target level of accuracy [12]–[14]. In addition, since the required accuracy level can vary depending on the usage scenario, it would be desirable to be able to exploit energy-accuracy trade-offs in edge-level systems [15]. There are also clearly instances where it would be desirable for an application to be able to use multiple network options on a single device. For example, the noise-adaptive CNN processing in [16] changes the network complexity to reduce the energy consumption when clear images are detected. The energy-efficient speech recognition module from [17] provides two different processing options for energy-performance trade-offs. As reported in [18], the energy consumption of action recognition tasks can also be remarkably reduced by dynamically switching the processing mode depending on the current confidence level.

Although there exist several architecture-level works that support different CNN modes without degrading processing efficiency [2], [19]–[21], algorithm-level techniques for edge-level CNN applications have not been sufficiently studied. This work introduces novel algorithm-hardware schemes for utilizing memory-reduced multiple accuracy CNN processing, which can be easily applied with existing on-device CNN accelerators in order to enable attractive energy-accuracy trade-offs [16]. For a given CNN architecture, the proposed CNN stacking scheme generates multiple processing modes by applying the well-known pruning technique [22] with multiple pruning ratios. Given a specific pruning ratio $p(<1)$, a traditional pruning approach involves randomly removing $p \times 100\%$ of the links in a CNN and undergoing training to determine the weights for the surviving links that result in the highest image classification accuracy. Such an approach

Manuscript received July 30, 2019; revised September 29, 2019 and October 31, 2019; accepted November 3, 2019. Date of publication November 7, 2019; date of current version December 12, 2019. This work was supported by the Samsung Research Funding and Incubation Center of Samsung Electronics under Project SRFC-TB1703-07. This article was recommended by Guest Editor T. Delbruck. *(Seungsik Moon and Younghoon Byun contributed equally to this work.) (Corresponding author: Youngjoo Lee.)*

The authors are with the Department of Electrical Engineering, Pohang University of Science and Technology (POSTECH), Pohang 37673, South Korea (e-mail: youngjoo.lee@postech.ac.kr).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2019.2952137

TABLE I
DETAILED STRUCTURE OF TARGET NETWORKS

Network	CNN structure
MobileNetV2 [23]	CONV + BOTTLENECK $\times$ 7 + CONV + AVGPOOL + FC
VGG-16 [1]	(CONV $\times$ 2 + MAXPOOL) $\times$ 2 + (CONV $\times$ 3 + MAXPOOL) $\times$ 3 + FC $\times$ 3
ResNet-18 [24]	(CONV + MAXPOOL) + CONV $\times$ 16 + FC

TABLE II
EFFECT OF QUANTIZATION ON VARIOUS NETWORKS

Network	MobileNetV2	VGG-16	ResNet-18
Dataset	CIFAR-100 [25]	Tiny ImageNet [26]	
Floating	70.53%	54.10%	57.52%
10bit	70.41%	54.14%	57.44%
9bit	70.19%	53.99%	57.49%
8bit	70.12%	53.93%	57.48%
7bit	69.80%	53.54%	57.21%
6bit	69.25%	49.40%	56.40%

normally uses a different set of links and retraining every time the pruning ratio p is changed because the network behaves differently with different sets of links.

In the proposed network stacking approach, multiple networks with different p ratios can be stored in a compressed manner (referred to as *network stacking*) and trained using an efficient iterative method. First, the most energy-conserving network is trained by using the highest permitted pruning ratio p. Then, a more accurate network (with a lower p value) is constructed by reviving some pruned positions and retraining while fixing the links (and weight values for those links) from the previous network. Continuing in this manner, networks with different pruning ratios (and thus different energy usage and classification accuracy levels) can be formed in a *stacked* manner, with a lower-accuracy network forming a subnetwork of higher-accuracy networks.

By using this method, all networks (and the weights used in those networks) can be stored in external memory by just storing the highest-accuracy network and a list of indices for the weights in each stacked network. Also, an efficient multi-level indexing method based on our previous work in [27], with a correspondingly efficient retrieval method for each network, is used to effectively *compress* the highest-accuracy network before it is stored. Simulation results are used to show that the proposed approach requires a significantly smaller memory footprint than existing comparable methods. Experimental results using an FPGA-based on-device CNN accelerator design also show that the proposed approach definitely provides more attractive energy-accuracy trade-offs than existing methods for edge-level devices.

The rest of this paper is organized as follows. Section II describes the related previous works, and the proposed network stacking scheme is presented in Section III. The structured pruning method for relaxing the indexing complexity is described in Section IV. Section V analyzes the simulation and implementation results for various multi-mode CNN solutions, including the proposed network stacking approach. Finally, concluding remarks are presented in Section VI.

II. RELATED WORKS

A. CNN Architectures for Edge-Level Devices

It is widely known that the complexity of CNN architecture is mostly dominated by two types of computations, i.e., convolution (CONV) and fully-connected (FC) operations [9]. Table I summarizes the structures of several recent CNN architectures, which are mainly used for case studies throughout this paper. Note that the BOTTLENECK layer of MobileNetV2 in Table I represents the inverted residual layers, including convolution operations, whose internal structures are reported in [23]. As there are more than 10 processing layers associated with millions of weight variables, it is natural that the CNN processing becomes memory-centric, requiring large internal and external memories even for edge-level devices [28]. In general, memory accessing operations consume much more energy than internal computations in a typical CNN architecture [22]. Thus, when implementing a CNN on an edge-level device, which will have limited computational and energy usage capabilities, conservation of memory usage is of paramount importance.

As all the weights of a traditional CNN are normally trained using floating-point numbers, conversion to fixed-point numbers is a well-known approach for reducing memory overhead [29]. Using two popular data sets, i.e., CIFAR-100 [25] and Tiny ImageNet [30], Table II depicts how the linear quantization process affects the recognition accuracy for different types of networks. Several recent works have revealed that aggressive quantization methods can be applied to further reduce memory overhead by utilizing specialized processing units [12] or a nonlinear quantization process [31].

In the approach proposed in this paper, linear quantization is used and previous approaches are adopted as baseline networks upon which the proposed methods can be applied. Note that the baseline networks in this paper are obtained by applying 7b linear quantization to all of the weights, which allows for an accuracy drop of less than 1% when compared to the original float-point based networks. Even though the baseline networks have reduced the required memory overhead by adopting quantization, the resulting memory overhead is still quite significant, resulting in a huge amount of energy for memory accesses [23].

Pruning is another popular technique for reducing network complexity [32]. By eliminating less important weights, it is reported that the conventional pruning approach can remove around 90% of all of the weights in an unpruned CNN without degrading the accuracy [32]. However, additional indexing information must be stored to denote the positions of the surviving weights, as illustrated in Fig. 1 [27]. Using the

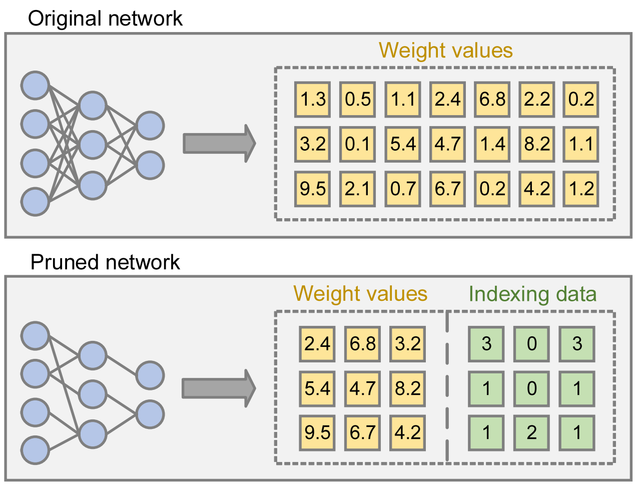


Fig. 1. Indexing overheads for storing the pruned network.

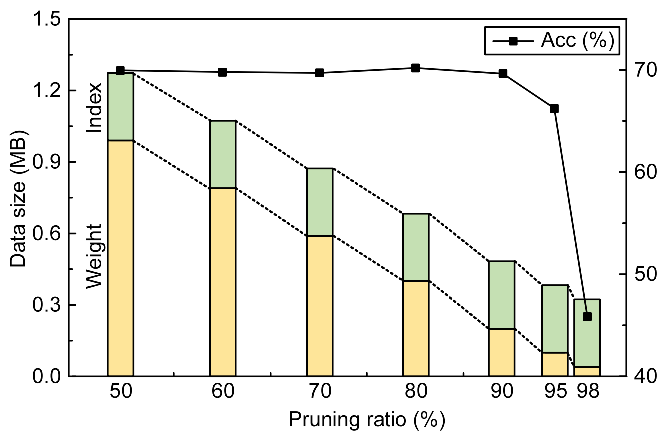


Fig. 2. Accuracy and memory size of pruned MobileNetV2 network.

7b-quantized MobileNetV2 for CIFAR-100 data [25] as a concrete example, Fig. 2 shows how pruning can reduce the overall memory overhead by sacrificing recognition accuracy.

After performing the retraining process [33], the accuracy of the pruned network tends to be similar to that of the baseline network even for highly-pruned architectures. However, note that the accuracy starts to decrease rapidly starting from a specific pruning ratio. For a resource-limited edge-level system, it is necessary to select the most pruned network that still provides an adequate level of accuracy [22]. For the case of the MobileNetV2 architecture, for example, selecting a pruning ratio of 0.9 can be a reasonable candidate since it results in an accuracy drop of only 0.89%, which is widely considered to be acceptable in practical case studies [34].

It is interesting to note that the portion of memory required for indexing information also increases, and at a rapid pace, as the pruning ratio is increased, as depicted in Fig. 2. For a lightweight CNN architecture, therefore, an efficient indexing scheme for highly-pruned networks would be highly desirable.

B. Indexing Overhead for Pruned Networks

To denote the location of surviving weight values, as depicted in Fig. 3(a), a straightforward on-off indexing scheme can be used by storing 1 or 0 for surviving or pruned positions, respectively [19]. However, the on-off indexing scheme is inefficient since it has to allocate one bit for each potential weight position. As a state-of-the-art CNN contains an extremely large number of weights, unfortunately, the storage demands of on-off indexing are unacceptable.

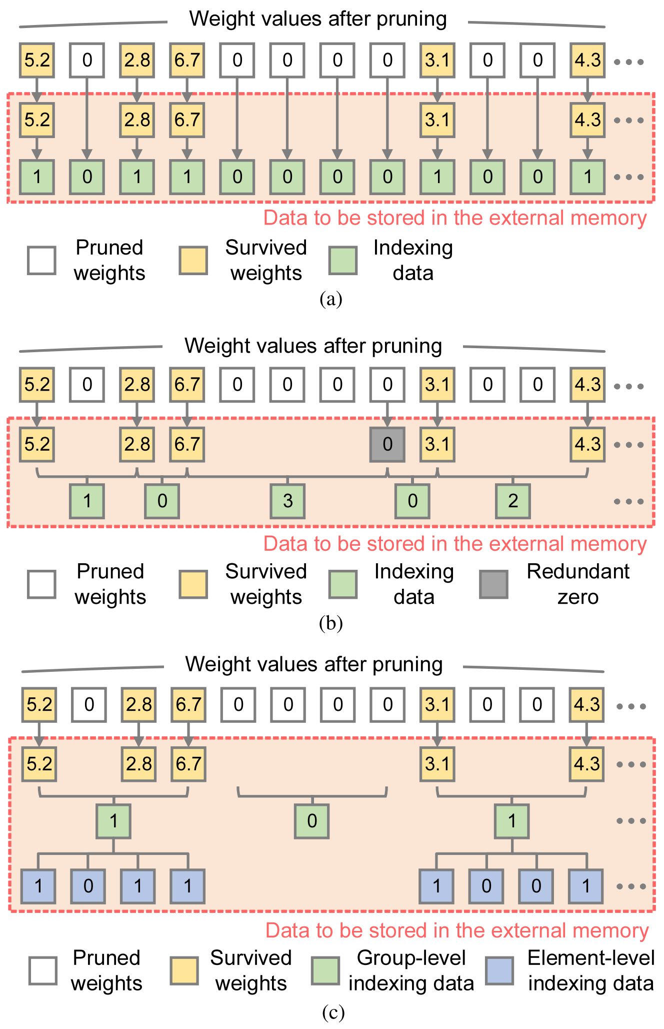

Weight values after pruning
Data to be stored in the external memory
Pruned weights
Survived weights
Indexing data
(a)
Weight values after pruning
Data to be stored in the external memory
Pruned weights
Survived weights
Indexing data
Redundant zero
(b)
Weight values after pruning
Data to be stored in the external memory
Pruned weights
Survived weights
Group-level indexing data
Element-level indexing data
(c)

Fig. 3. Previous indexing schemes using (a) the on-off method [19] (b) the SME-based approach [22] and (c) the multi-level method [27].

As a low-complexity indexing scheme, the sparse matrix encoding (SME) method is widely used in practical applications. The SME method records the distance between surviving weight positions, as shown in Fig. 3(b) [22]. Due to the limited bitwidth for representing the distance, however, SME-based indexing sometimes stores redandant zeros to cut the long distance between two adjacent surviving weights. Using the 2b representation of SME-based indexing, for example, we need to insert a redundant zero weight at the gray-colored positions in Fig. 3(b). Thus, due to the enlarged gaps between two adjacent surviving weights, the previous SME-based indexing becomes more inefficient as the pruning ratio increases.

Our recent work in [27] reveals that the indexing overheads can be greatly relaxed by adopting a multi-level representation as exemplified in Fig. 3(c). After performing the pruning

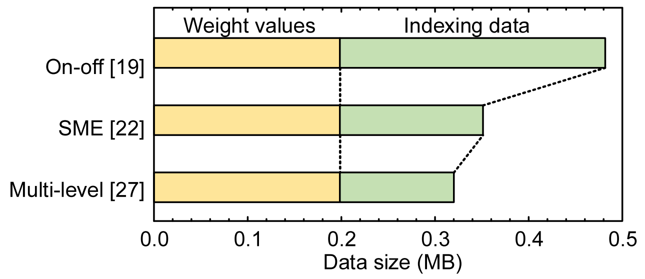


Fig. 4. Memory footprints of pruned MobileNetV2 networks with different indexing schemes.

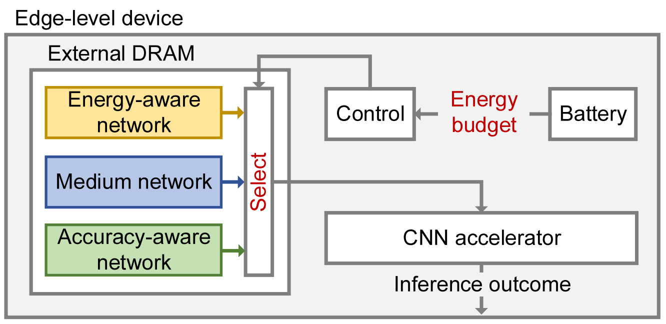


Fig. 5. Utilizing the energy-accuracy trade-off for on-device CNN processing with multiple network options.

operation, the multi-level indexing method of [27] constructs weight groups by collecting consecutive weight positions in the unpruned network. Then, group-level indexing is used with only one bit per group to denote whether each group includes at least one surviving weight or not. For the surviving groups, element-level indexing, implemented in an on-off manner, is applied to locate the exact positions of the surviving weights.

Fig. 4 analyzes the memory footprints of different indexing schemes for the 7b-quantized MobileNetV2 architecture where the pruning ratio is set to 0.9. Note that our multi-level indexing requires the smallest memory size, but the amount of indexing data is still considerable compared to that of the surviving weights. Therefore, it is necessary to consider the indexing overhead in order to design a compact CNN algorithm for edge-level systems. By extending the multi-level indexing method of [27] to stacked networks, referred to as *structured pruning*, indexing overhead is significantly reduced while storing multiple CNNs for variable levels of accuracy.

III. Proposed Network Stacking Method

In energy-limited edge devices, it is important to select the proper algorithm to solve the given problem, which in this case implies the selection of an algorithm that consumes the minimum processing energy while providing the required level of image recognition performance. Previous works have presented circuit-level techniques for implementing energy-accuracy trade-offs in CNNs [20], [35].

As shown in Fig. 5, in this work, we basically offer an algorithm-level scheme that considers the hardware complexity at runtime, i.e., achieving the energy-accuracy trade-off by preparing several CNN options with different computational costs and levels of recognition performance. The edge-level device simply selects a less accurate but more energy-conserving network when the energy budget is tight, and selects a higher energy-usage network when the energy budget is relaxed or when a higher level of accuracy is essential. Note that this approach can be easily combined with the previous hardware-level innovations, further improving the overall processing efficiency. For resource-limited edge-level devices, it is especially important that multiple CNN models, enabling runtime energy-accuracy trade-offs, be implemented and stored in an efficient a manner as possible.

A. Basic Methods for Realizing Energy-Accuracy Trade-Offs

To provide multiple CNN options with varying levels of recognition performances, the most basic strategy is to prepare completely different networks with different topologies. For example, we can use the simple AlexNet structure [1] for an energy-efficient mode with low recognition accuracy, and change the network to a ResNet-like structure [24] for higher recognition accuracy accompanied by higher energy usage. This trivial approach simply provides an energy-accuracy trade-off with contemporary CNN accelerators. However, when considering the tight energy constraints of edge-level devices, it is necessary to use a more efficient method for storing multiple CNN architectures [9]. For example, the embedded device-oriented MobileNetV2 architecture uses several millions of weight values [36], resulting in difficulties when uploading entire filters in on-device memories even for state-of-the-art hardware accelerators [2].

Another possible approach for providing multiple CNN modes with acceptable hardware efficiency is to use a single network with different pruning ratios. Increasing the pruning ratio gradually results in the elimination of weights without changing the original CNN structure; thus, the same computing procedures can be used to run various pruned networks. The energy-accuracy trade-off is naturally achieved by changing the pruning ratio, i.e., the more pruned network consumes less energy by sacrificing the accuracy as depicted in Fig. 2. However, storing multiple pruned networks also comes with a great burden for the resource-limited devices as the retraining process for accuracy compensation changes the weight values [33]. Like to the basic method that prepares different types of CNN architectures, each pruned network has different weights and has to be stored separately in external memory, thereby increasing the required memory size in a manner directly proportional to the number of CNN options. To relax the resource burden for edge-level devices, this work proposes the concept of network stacking, which can provide energy-accuracy trade-offs with negligible hardware-level overhead.

B. Network Stacking for Resource-Limited Devices

In contrast to a naive method that offers energy-accuracy trade-offs by defining multiple CNN options with different pruning ratios without any restrictions, the proposed stacked-CNN constructs multiple weight sets where the upper sets always include the weights of lower ones as shown

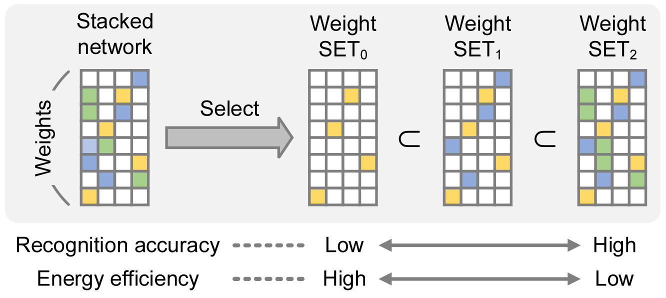


Fig. 6. Concept of the proposed network stacking approach.

in Fig. 6. As in the naive method, we can provide different recognition accuracies by adjusting the number of surviving weights, i.e., the upper sets are designed to realize higher accuracy networks while the lower sets result in lower energy-usage networks. However, note that the proposed stacking concept is suitable for intelligent edge-level devices as it only stores the weights of the most accurate network to support different operating modes and thereby enable online energy-accuracy trade-offs.

As the proposed stacking concept limits the freedom to choose the pruned weights, it is important to develop a systematic training methodology that can preserve the recognition accuracy when compared to a conventionally pruned network with a similar pruning ratio. Thus, an incremental training method for constructing a stacked-CNN architecture is proposed as illustrated in Fig. 7. As shown in the figure, before starting the training sequence, we first define a set of weights to represent the initial network, which is denoted as $\mathrm{SET}_{\mathrm{SEED}}$. In fact, the weights in $\mathrm{SET}_{\mathrm{SEED}}$ are nothing but the initial random values from a Gaussian distribution [37], which will be reused in the following retraining steps.

Based on this $\mathrm{SET}_{\mathrm{SEED}}$, the set of baseline weights ($\mathrm{SET}_{\mathrm{BASE}}$) is constructed by training all of the weights to maximize the validation accuracy [38]. The total number of weights in the baseline network is denoted as N_{BASE}. From this dense network with the $\mathrm{SET}_{\mathrm{BASE}}$, we construct the most energy-conserving network whose weight set is denoted as SET_0. For example, with MobileNetV2, we can perform conventional pruning with the aggressive pruning ratio of $p_0 \geq 0.9$ (below which accuracy drops precipitously, as shown in Fig. 2). Then, the resulting SET_0 contains only $N_0 = (1 - p_0)N_{\mathrm{BASE}}$ weights of $\mathrm{SET}_{\mathrm{BASE}}$, which supports an energy-limited CNN with a small level of accuracy degradation. Note that only the N_0 surviving weights are retrained.

To construct the next, more accurate CNN structure, the missing weights, relative to $\mathrm{SET}_{\mathrm{SEED}}$, are filled in from $\mathrm{SET}_{\mathrm{SEED}}$ as depicted in Fig. 7. Then, the rejuvenated positions are retrained while fixing the values of the surviving weights from SET_0 to make a dense network again. Next, the pruning operation is performed again with the next pruning ratio $p_1(< p_0)$, thereby removing $p_1 N_{\mathrm{BASE}}$ weights. More precisely, N_0 weights in SET_0 are fixed during this pruning step, and thus we only eliminate the weights among the refilled ones to generate the stacked-CNN structure as depicted in Fig. 7. In other word, SET_1 includes $N_1 = (1 - p_1)N_{\mathrm{BASE}}$ non-zero weights, where N_0 of them belong to SET_0.

As depicted in Fig. 7, this incremental training method is repeated until we construct the $(L_{\mathrm{MAX}}-1)$-th network with the smallest pruning ratio, whose weights constitute the uppermost set $\mathrm{SET}_{L_{\mathrm{MAX}}-1}$. Note that the CNN structure associated with SET_i is always generated by stacking more weights on top of the previous SET_{i-1}. Thus, with this method, all possible L_{MAX} CNN options can be saved by only storing $\mathrm{SET}_{L_{\mathrm{MAX}}-1}$, which is the super-set of all lower-level networks. Experimental results, which will be presented in Section V, show that this new approach significantly reduces the memory overhead required while supporting an online energy-accuracy trade-off capability.

IV. Structured Pruning for Network Stacking

Although the network stacking method described above allows the minimum memory footprint for the storage of multiple CNN modes with different pruning ratios, there is an indexing problem that must solved in order to enable the user to quickly access the specific CNN mode desired at any given time instant. Based on the multi-level indexing method from our previous work [27], a more advanced pruning scheme for a stacked-CNN architecture is presented in this section.

A. Reducing the Indexing Complexity With Structured Pruning

It is obvious that the effectiveness of a multi-level indexing scheme is highly dependent on the patterns of pruned positions [27]. To minimize the indexing overheads, in this work, we introduce a new pruning methodology that can generate structured pruning patterns that can be compressed efficiently using a multi-level indexing scheme. For the pruning ratio of p, the conventional value-aware pruning method just eliminates the less important weights based on their absolute values, and only the surviving weights are retrained [32].

Although this pruning scheme can effectively reduce the number of weights with negligible accuracy loss, the resulting patterns of surviving positions are usually irregular and unpredictable. As the multi-level indexing scheme utilizes group-level information, even a single surviving weight in a group requires element-level information, necessitating $g + 1$ bits of indexing data, which includes the group-level indexing bit, as shown in Fig. 3(c). Therefore, the improvement possible with the multi-level indexing method is obviously limited by the irregular patterns created during pruning. Thus, in this work, the group size is considered, in addition to the weight magnitude, when selecting the weights that are less important, and can thus be pruned.

As in multi-level indexing, the proposed structured weight pruning method consists of two phases, i.e., group-level pruning followed by element-level pruning. To achieve the target pruning ratio p, for expository purposes, we define two internal variables p_{G} and p_{E} for the pruning ratios of the group-level and element-level eliminations, respectively. These two variables are determined such that $p = 1 - (1 - p_{\mathrm{G}})(1 - p_{\mathrm{E}})$. In the group-level pruning process, as depicted

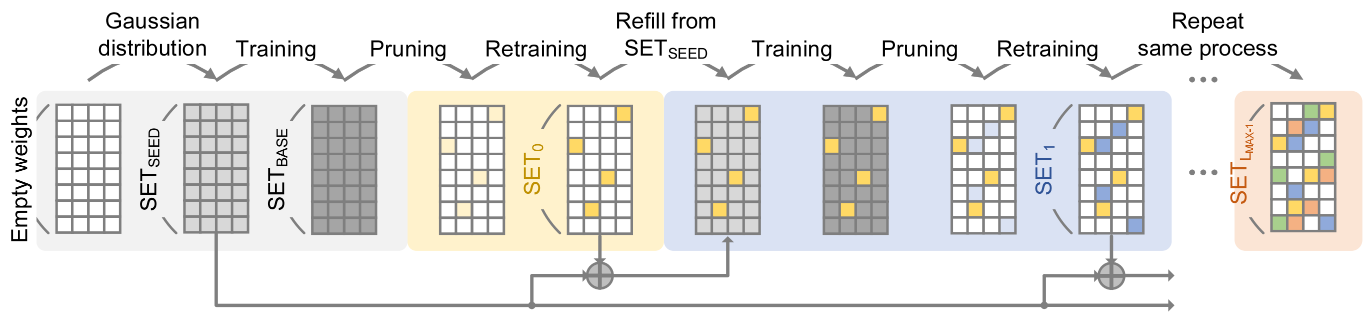


Fig. 7. Incremental training procedure for the proposed network stacking approach.

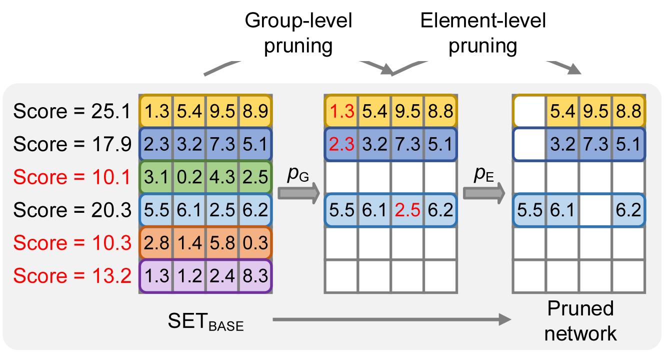


Fig. 8. The proposed structured pruning method.

in Fig. 8, we construct the weight groups such that there are g consecutive weights in the baseline architecture. Then, an importance score for each group is calculated by accumulating all of the absolute values of weights in the each group. Unlike the conventional value-aware pruning method that eliminates each weight independently, in the proposed structured pruning method, we remove an entire set of weights in less important groups. As a result, after performing the group-level elimination, the network contains $(1 - p_G)N_{BASE}$ weights, where the pruned positions are structured as depicted in Fig. 8.

Then, element-level pruning with the pruning ratio of p_E is performed (for fine tuning) by removing weights within the surviving groups. As shown in Fig. 8, element-level pruning is basically identical to conventional pruning, but is applied to only those elements in surviving groups. Thus, after both phases are complete, the pruned network has $(1 - p_G)(1 - p_E)N_{BASE}$ weights remaining, which is the same ratio of surviving weights that would result from conventional pruning with pruning ratio $p = 1-(1-p_G)(1-p_E)$. However, the main difference is that the proposed pruning method results in a structured surviving weight pattern that can be summarized with high efficiency by using multi-level indexing.

To reduce indexing overhead, one possible strategy is to make g as large as possible, which would result in a very large number of consecutive pruned positions. However, using extremely large group sizes can cause severe accuracy drops since important weights that are relatively close to a several unimportant weights could be pruned. Therefore, an appropriate g value must be selected that reduces the indexing complexity while providing an acceptable level of recognition accuracy. In addition, it is helpful to consider the hardware accelerator architecture, which normally accesses external memory to load a series of $k \times k$ filters at a time [39]. In this work, therefore, we select the group size to be multiples of k so that the existing hardware can easily support the proposed structured pruning method. Similarly, the two pruning ratios, i.e., p_G and p_E, are adjusted (through experimentation) to provide highly memory-efficient indexing information while achieving acceptable recognition accuracy.

B. Low-Cost Indexing Method for Network Stacking

To minimize the overall memory requirements for the proposed stacked-CNN architecture, a low-cost indexing method that accommodates the proposed stacked-CNN method and the structured pruning method is required. As summarized in Fig. 9, structured pruning is applied to create each SET$_i$, and each network constructed in such a manner has a structured pruning pattern that heightens the efficiency of multi-level indexing.

In order to maintain a structured pruning pattern with network stacking, during the retraining step used to generate the next CNN level in Fig. 9, it is important to fix the surviving *groups* of the previous network whereas the original scheme described in Section III maintains only the surviving *elements*. Notably, it was found that this type of "structured stacking" can result in more attractive energy-accuracy trade-offs when compared to the original stacked-CNN approach.

For example, assume that we provide four different CNN modes for supporting energy-accuracy trade-offs in intelligent edge-level devices, i.e., $L_{MAX} = 4$. Then, the original stacked-CNN architecture adopting the conventional SME-based indexing scheme requires four different sets of indexing data to denote the corresponding weight sets as depicted in Fig. 10(a). Even if we reduce the indexing overhead by applying our multi-level indexing scheme [27], four different indexing sets must still be prepared since each weight group may contain elements from different weight sets. Therefore, the total memory size required is still considerable.

If the proposed structured pruning scheme shown in Fig. 9 is applied to construct the stacked-CNN architecture, we can significantly reduce the indexing complexity by sharing element-level indexing information. More precisely, Fig. 10(b) depicts how we store the indexing information for the 4-level

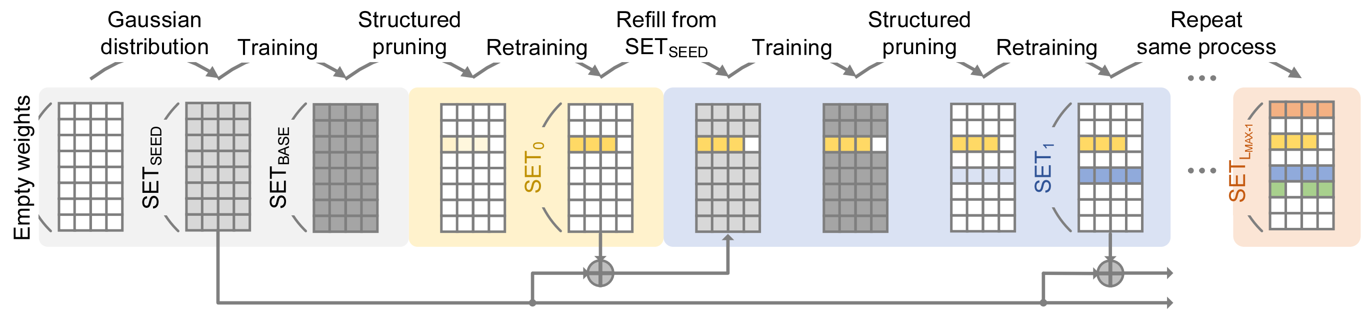


Fig. 9. Incremental training procedure adopting the proposed structured pruning.

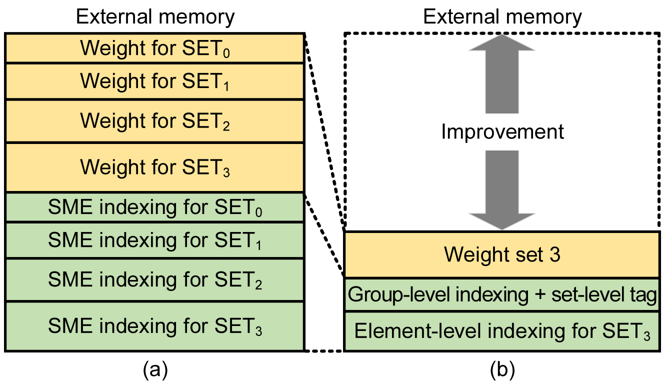


Fig. 10. Storage overheads for storing the stacked-CNN architectures generated by (a) the conventional pruning method and (b) the proposed structured pruning scheme.

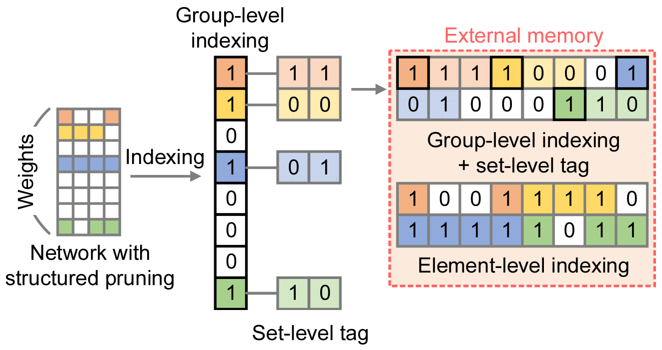


Fig. 11. Indexing information with the proposed set-level tags.

stacked-CNN with structured pruning. For denoting the surviving elements, note that we only store one element-level index data item in SET_3, which represents the most accurate network. To access the lower-level networks, we only select a subset of the element-level indexing data by checking the surviving groups in a given target level as all the groups in the lower-level CNNs are already included in the upper-level networks. Therefore, it is sufficient to additionally record the corresponding CNN level for each surviving group, which is denoted as a set-level tag in Fig. 10(b). As a result, the proposed scheme naturally generates memory-optimized indexing information by fully sharing element-level surviving weight information between CNN levels.

Fig. 11 shows how the additional set-level tag identifies the surviving groups for a given target CNN level. By utilizing $\log_2 L_{\text{MAX}}$ bits for each surviving group, the tag represents the lowest level that includes the corresponding group. When SET_i is selected for the current CNN architecture, it is possible to construct the target network by simply ignoring the surviving groups whose set-level tags are greater than i. Note that the additional memory overhead for representing multiple networks can be negligible even compared to the indexing data for a single CNN architecture, since we use these set-level tags only for the surviving groups, as shown in Fig. 11.

In the example shown in Fig. 11, note that we utilize only two bits for each set-level tag. Hence, the total size of the indexing information for representing four different CNN modes is almost similar to that for the single CNN architecture SET_3, whereas the conventional indexing approach would require four independent indexing data sets. By minimizing the indexing complexity with the proposed structured pruning method, the proposed network stacking approach provides an attractive energy-accuracy trade-off capability, enabling its use for memory-reduced CNN solutions in edge-level devices.

V. Experimental Results

This section summarizes the performance of several alternative methods for enabling energy-accuracy trade-offs in edge-level devices. The multiple accuracy methods implemented and compared include: 1) the naive method that constructs completely different networks with different pruning ratios, 2) the stacked-CNN method with unstructured pruning, and 3) the proposed fully-optimized stacked-CNN method with structured pruning. To check the overall memory overhead, we also implement several indexing schemes such as the previously proposed SME-based method and our multi-level indexing methods. For the case studies, note that we use three different networks, i.e., MobileNetV2 [23], VGG-16 [1], and ResNet-18 [24], whose structures are shown in Table I. Considering the practical applications, two datasets are used, i.e., CIFAR-100 [25] and Tiny ImageNet [26].

A. Algorithm-Level Performance

Table III summarizes the algorithm-level performance of various methods supporting multiple CNN options with different pruning ratios. Regardless of the network structure,

TABLE III
SIMULATION RESULTS FOR ALGORITHM-LEVEL PERFORMANCES

Network	Baseline accuracy	Technique	Accuracy options			Memory size
			SET_0	SET_1	SET_2	
MobileNetV2 for CIFAR-100 (p_0 = 0.95, p_1 = 0.85)	69.91%	Naive method with conventional pruning	66.20%	69.83%	-	0.74MB
		Network stacking with conventional pruning	66.20%	68.95%	-	0.57MB
		Network stacking with structured pruning	66.23%	67.50%	-	0.51MB
VGG-16 for Tiny ImageNet (p_0 = 0.98, p_1 = 0.95, p_2 = 0.90)	54.22%	Naive method with conventional pruning	47.78%	52.34%	53.39%	4.36MB
		Network stacking with conventional pruning	47.78%	52.31%	53.57%	3.05MB
		Network stacking with structured pruning	45.66%	52.08%	53.46%	2.36MB
ResNet-18 for Tiny ImageNet (p_0 = 0.98, p_1 = 0.95, p_2 = 0.90)	54.33%	Naive method with conventional pruning	49.20%	52.41%	54.33%	3.21MB
		Network stacking with conventional pruning	49.20%	52.65%	53.91%	2.69MB
		Network stacking with structured pruning	50.36%	52.88%	53.09%	1.77MB

the proposed stacking approach provides multiple accuracy modes while significantly reducing the overall memory overhead. Similar to the naive method managing multiple pruned networks individually, the proposed work can offer more accurate recognition process with larger weight set. For the case of pruned MobileNetV2 architecture, for example, accurate network SET_1 of the proposed stacked-CNN provides most accurate accuracy. Due to the memory-efficient stacking method, overall memory overhead can be relaxed by 23% with conventional pruning scheme when compared to the naive approach, since we store only SET_1 for supporting two different processing modes. If the structured pruning is applied during the incremental training process, as summarized in Table III, we can further reduce the overall memory overhead by 31%, while we experience some accuracy drops as the group-level eliminations may remove several important weights that are surrounded by unimportant weights.

However, the amount of accuracy degradation may be acceptable for practical edge-level applications if we consider the improvements in terms of memory footprint possible due to the resulting structured pruning patterns. When targeting the MobileNetV2 architecture, note that the stacked-CNN architecture with structured pruning uses only 0.51MB for storing two different network levels, which is 1.45 and 1.12 times less than the memory overheads required by the naive method and stacked-CNN with conventional pruning, respectively. Fig. 12 shows how the proposed optimization schemes can gradually relax the memory footprint by reducing the storage costs for both weight values and indexing data. Note that

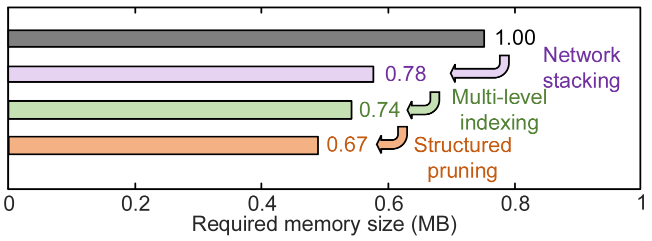


Fig. 12. Reducing the memory requirements by adopting the proposed methods to MobileNetV2 network for CIFAR-100 dataset.

network stacking first reduces the total size of the weights by adopting an incremental training process, wherein weights of upper level sets contain the weights used in lower level sets. Introducing the multi-level indexing method from our previous work [27] further reduces the memory overhead as it provides a more efficient way for representing highly-pruned networks as compared to the conventional on-off or SME-based indexing scheme. The fully-optimized stacked-CNN architecture exploits the smallest memory footprint by applying structured pruning, which significantly relaxes the overhead of the multi-level indexing method. Even compared to the single-mode CNN architecture with $p = 0.85$, it is noticeable that the proposed stacked-CNN architecture requires smaller memory space even though it supports extra pruning options for energy-accuracy trade-off.

To verify the effects of the proposed schemes, different experimental results with various networks such as VGG-16

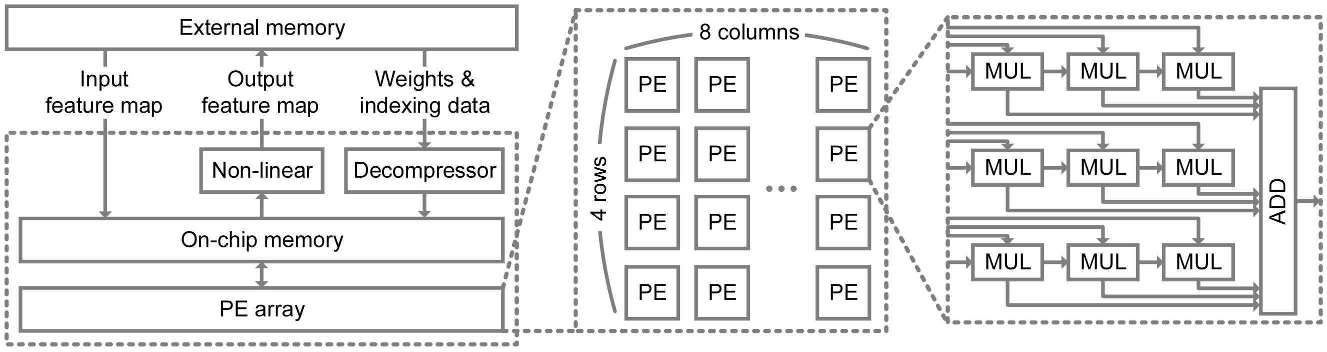


Fig. 13. Conceptual diagram of the FPGA-based hardware accelerator for case study.

and ResNet-18 are also listed in Table III. As depicted in the table, the proposed stacked-CNN architecture provides similar improvements as in the case study for MobileNetV2, i.e., offering the different algorithm-level performance by using multiple weight sets with different pruning ratios. When compared to the naive approach, the proposed method can significantly reduce the memory requirements while preserving a similar level of accuracy. More specifically, the proposed scheme can reduce the overall memory footprint by up to 45.9%, and 44.9% for the VGG-16 and ResNet-18 networks, respectively. Therefore, it is clear that the proposed network stacking method provides an effective memory-reduced solution for edge-level CNN processing that efficiently utilizes energy-accuracy trade-offs during runtime.

B. Case Study for VGG-16 Accelerator

In order to demonstrate the hardware-level efficiency of the proposed energy-accuracy trade-off, we design the dedicated CNN accelerator for VGG-16 network as a case study. Including the dedicated accelerator, the verification platform is realized by using the commercialized FPGA-based development board, which includes Intel's Stratix-V FPGA device [40]. Note that the verification platform utilizes an external SDRAM for storing the multiple CNN options based on the VGG-16 architecture [41], and the energy-accuracy trade-offs can be naturally exploited by selecting an appropriate CNN at runtime as shown in Fig. 5. Note that the proposed work mainly focuses on reducing the memory overheads of storing the multiple networks with the structured pruning patterns for improving the overall efficiency. Similar to the works in [3], [39], and [42], therefore, the internal processing modules in this work are based on a dedicated 3×3 filter size to maximize the hardware efficiency by considering the architecture of VGG-16 network. For different network, the accelerator can easily process with various filter sizes by using the recent techniques including zero padding or filter decomposition [3]. It is also possible to realize the dedicated hardware designs for further enhancing the processing efficiency of other CNN architectures with different filter sizes, such as ResNet or MobileNet adopting 1×1 convolutions.

Fig. 13 shows the overall architecture of the proposed CNN accelerator, which includes on-chip memory, a processing element (PE) array, and a decompressor for supporting the pruned networks. The size of the on-chip memory is selected to support the weight-reusing data flow [3], where all the weights in a layer are loaded only once from the external memory. During CNN processing, the decompressor first accesses the external memory to get the pruned layer, i.e., the weight values and the indexing information, and then converts them to initialize parallel PEs. Using the input feature-map and partial sums, each PE then performs the weighted sum operations. Depending on the network structure, the final outputs of each layer can be transferred to nonlinear operators [3].

TABLE IV
IMPLEMENTATION RESULTS OF FPGA-BASED ACCELERATORS

System		Naive	Proposed
Network		VGG-16 for Tiny ImageNet	
Frequency		100MHz	
DSP blocks		324	
ALM complexity		15,789	15,813
Processing latency*		16.73ms	8.27ms
Total amount of DRAM accessing*	Feature-map	1.24MB	1.24MB
	Weight	1.44MB	1.06MB
Energy consumption*		9.94mJ	5.11mJ

* For operating the energy-efficient option with SET_0.

By changing the network information stored in the external DRAM, we quantitatively analyze the amount of energy consumed. For a fair comparison, we first realize the naive approach, which independently trains multiple pruned networks represented by the previous SME-based indexing method. The architecture proposed in this work is constructed by using the proposed incremental training method to support different networks, each of which is based on the structured pruning and multi-level indexing schemes described above. Table IV summarizes the implementation results for each processing mode. To implement multiplications with parallel PEs, we use 324 DSP blocks running at a speed of 100MHz. To show the complexity of the logical processing parts, the number of adaptive logic modules (ALM) is used in this comparison [40]. As depicted in Table IV, the fully-optimized stacked-CNN architecture is superior to a naive

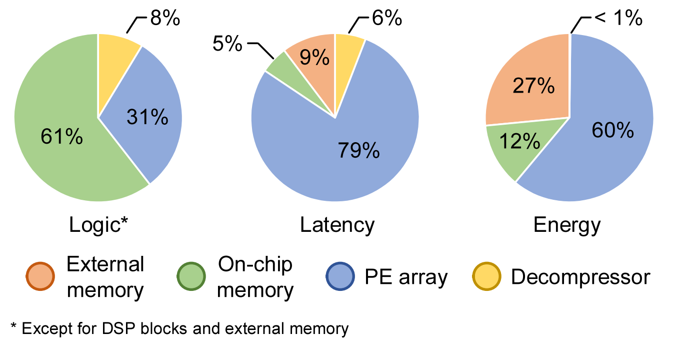


Fig. 14. Breakdown of the fully-optimized FPGA-based accelerator in terms of logic complexity, latency, and energy consumption.

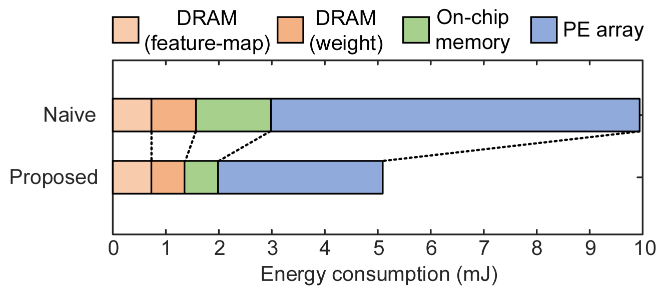


Fig. 15. Energy-level breakdowns for different architectures.

implementation in terms of the energy consumption as well as the processing speed as it significantly improves the overall accelerator efficiency by introducing the structured weight patterns with reduced memory overheads. One may be concerned about the increased hardware complexity caused by the new indexing scheme. However, as shown in Fig. 14, the additional overhead produced by decompressor logic is negligible, which shows about 1% of energy consumption in total energy consumption.

Fig. 15 shows the detailed energy-level breakdowns for two accelerator designs. Note that the proposed fully-optimized version reduces the energy consumption of all units. As the hardware architecture of this case study reconstructs 3×3 filters that contain at least one surviving weight value, moreover, the proposed structured pruning definitely provides more compact processing steps by reducing the number of survived filters when compared to the naive architecture with the conventional unstructured pruning. As shown in Fig. 15, as a result, we can save half of the energy consumption caused by the on-chip memory accesses as well as the convolution operations.

The energy-accuracy trade-offs achieved by the FPGA-based accelerator are illustrated in Fig. 16 by considering the recognition accuracy figures shown in Table III. Since the proposed architecture significantly reduces the overall energy consumption, it results in a more energy-efficient recognition method than a naive approach for achieving a similar level of recognition accuracy. For example, when using the most energy-conserving network, the proposed method reduces the inference energy by 46% compared to the naive architecture. In general, note that the proposed fully-optimized solution always provides more energy-efficient

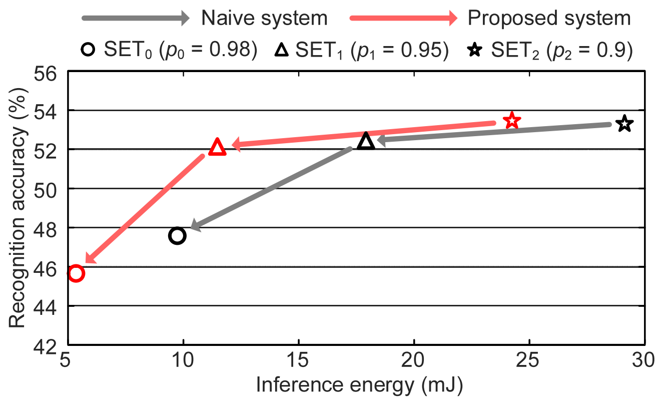


Fig. 16. Energy-accuracy trade-off provided by different techniques.

options, as shown in Fig. 16. Moreover, the overall memory footprint of the proposed system is also smaller than that of the naive system. As a result, the proposed stacked-CNN architecture with advanced indexing and pruning methods provides an attractive energy-accuracy trade-off method for realizing on-device CNN processing in resource-constrained edge-level devices.

VI. Conclusion

In this paper, we have presented several optimization techniques for reducing the memory overhead required with a multiple accuracy mode CNN architecture. We first develop the concept of the stacked-CNN architecture, which constructs the upper-level accuracy-aware CNNs by re-utilizing the existing weights in lower-level energy-aware CNNs. Compared to a naive multiple-pruning-level CNN ensemble, selected as the baseline architecture, the amount of memory required to store all of the necessary weight values is significantly reduced by storing only the weights for the most accurate CNN model. In addition, a novel structured pruning method for efficiently indexing the surviving weights in a pruned CNN is presented. The total memory required to support multiple CNN architectures with different accuracy levels is significantly reduced by combining the proposed network stacking and structured pruning techniques. As demonstrated with an FPGA-based case study, this enables a practical method for offering energy-accuracy trade-offs in intelligent edge-level systems.

References

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

[2] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.

[3] L. Du *et al.*, "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 198–208, Jan. 2018.

[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2961–2969.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[6] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," 2018, *arXiv:1802.01548*. [Online]. Available: https://arxiv.org/abs/1802.01548

[7] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 1–7.

[8] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[9] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[10] A. Wong, "NetScore: Towards universal metrics for large-scale performance analysis of deep neural networks for practical on-device edge usage," in *Proc. Int. Conf. Image Anal. Recognit.*, 2019, pp. 15–26.

[11] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1396–1401.

[12] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Oct. 2018.

[13] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "MALOC: A fully pipelined FPGA accelerator for convolutional neural networks with all layers mapped on chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2601–2612, Jul. 2018.

[14] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 10, pp. 1415–1419, Aug. 2018.

[15] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Proc. Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2016, pp. 1–8.

[16] Y. Byun, M. Ha, J. Kim, S. Lee, and Y. Lee, "Low-complexity dynamic channel scaling of noise-resilient CNN for intelligent edge devices," in *Proc. Design Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 114–119.

[17] S. Oh *et al.*, "An acoustic signal processing chip with 142-nW voice activity detection using mixer-based sequential frequency scanning and neural network classification," *IEEE J. Solid-State Circuits*, vol. 54, no. 11, pp. 3005–3016, Nov. 2019.

[18] J. Kung, D. Kim, and S. Mukhopadhyay, "Dynamic approximation with feedback control for energy-efficient recurrent neural network hardware," in *Proc. Int. Symp. Low Power Electron. Design*, 2016, pp. 168–173.

[19] A. Aimar *et al.*, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2019.

[20] J. Song *et al.*, "An 11.5 TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8 nm flagship mobile SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 130–132.

[21] S. Yin *et al.*, "An energy-efficient reconfigurable processor for binary- and ternary-weight neural networks with flexible data bit width," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 1120–1136, Apr. 2019.

[22] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: https://arxiv.org/abs/1510.00149

[23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 4510–4520.

[24] 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.

[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[26] Stanford University Online. (2019). *CS231n: Convolutional Neural Networks for Visual Recognition*. [Online]. Available: http://cs231n.stanford.edu/

[27] J. Park, S. Moon, Y. Byun, S. Lee, and Y. Lee, "Multi-level weight indexing scheme for memory-reduced convolutional neural network," in *Proc. Int. Conf. Artif. Intell. Circuit Syst. (AICAS)*, Mar. 2019, pp. 284–287.

[28] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 13–19.

[29] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 1131–1135.

[30] *Tiny Imagenet Challenge*. Accessed: Jul. 10, 2019. [Online]. Available: https://tinyimagenet.herokuapp.com

[31] S. Jung *et al.*, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4350–4359.

[32] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1135–1143.

[33] S. Han *et al.*, "DSD: Regularizing deep neural networks with dense-sparse-dense training flow," vol. 3, no. 6, 2016, *arXiv:1607.04381*. [Online]. Available: https://arxiv.org/abs/1607.04381

[34] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: https://arxiv.org/abs/1602.07360

[35] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 27–40.

[36] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861

[37] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*. [Online]. Available: https://arxiv.org/abs/1803.03635

[38] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*, 2010, pp. 177–186.

[39] J. Yue *et al.*, "A 3.77 TOPS/W convolutional neural network processor with priority-driven kernel optimization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 2, pp. 277–281, Feb. 2019.

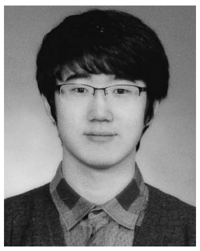
[40] Altera. *DE5-Net FPGA Development Kit*. Accessed: Sep. 30, 2019. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=526

[41] K. T. Malladi, F. A. Nothaft, K. Periyathambi, B. C. Lee, C. Kozyrakis, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile DRAM," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 37–48.

[42] S. Moon *et al.*, "FPGA-based sparsity-aware CNN accelerator for noise-resilient edge-level image recognition," in *Proc. Asian Solid-State Circuits Conf.*, Nov. 2019, pp. 205–208.

Seungsik Moon (S'18) received the B.S. degree in electrical engineering from POSTECH, Pohang, South Korea, in 2018, where he is currently pursuing the M.S. degree.

His current research interests include advanced error-correction codes, next-generation communication algorithms, and architectures for digital systems.

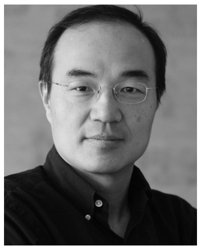


Younghoon Byun (S'18) received the B.S. degree in electrical engineering from POSTECH, Pohang, South Korea, in 2018, where he is currently pursuing the M.S. degree.

His current research interests include low power deep learning architecture, deep learning hardware accelerator, and embedded systems.

Jongmin Park (S'18) received the B.S. degree in electrical engineering from POSTECH, Pohang, South Korea, in 2019, where he is currently pursuing the M.S. degree.

His current research interests include AI algorithms and optimized architecture for mobile embedded systems.



Sunggu Lee (M'88) received the B.S.E.E. degree (Hons.) from the University of Kansas, Lawrence, KS, USA, in 1985, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 1987 and 1990, respectively.

He was an Assistant Professor with the Department of Electrical Engineering, University of Delaware, Newark, DE, USA. From 1997 to 1998, he was a Visiting Scientist with the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He was a Visiting Researcher with the DREAM Laboratory, University of California at Irvine, Irvine, CA, USA, from 2005 to 2006. He is currently a Professor with the Department of Electrical Engineering, Pohang University of Science and Technology (POSTECH), Pohang, South Korea. His current research interests include wireless sensor networks, cloud computing, real time computing, parallel computing, and fault-tolerant computing.

Youngjoo Lee (M'14) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2008, 2010, and 2014, respectively.

Prior to joining Pohang University of Science and Technology (POSTECH), Pohang, South Korea, he was with the Interuniversity Microelectronic Center (IMEC), Leuven, Belgium, from 2014 to 2015, where he researched reconfigurable SoC platforms for software-defined radio systems. From 2015 to 2016, he was with the faculty of the Department of Electronic Engineering, Kwangwoon University, Seoul, South Korea. Since 2017, he has been an Assistant Professor with the Department of Electrical Engineering POSTECH. His current research interests include the algorithms and architectures for embedded processors, intelligent mobile systems, and energy-efficient communication systems.