This ACCV 2024 paper, provided here by the Computer Vision Foundation, is the author-created version.
The content of this paper is identical to the content of the officially published ACCV 2024
LNCS version of the paper as available on SpringerLink: https://link.springer.com/conference/accv

Designing Extremely Memory-Efficient CNNs for On-device Vision Tasks

Jaewook Lee, Yoel Park, and Seulki Lee

Ulsan National Institue of Science and Technology (UNIST)
jaewook.lee@unist.ac.kr, joel157@unist.ac.kr, seulki.lee@unist.ac.kr

Abstract. In this paper, we introduce a memory-efficient CNN (convolutional neural network), which enables resource-constrained low-end embedded and IoT devices to perform on-device vision tasks, such as image classification and object detection, using extremely low memory, *i.e.*, only 63 KB on ImageNet classification. Based on the bottleneck block of MobileNet, we propose three design principles that significantly curtail the peak memory usage of a CNN so that it can fit the limited KB memory of the low-end device. First, 'input segmentation' divides an input image into a set of patches, including the central patch overlapped with the others, reducing the size (and memory requirement) of a large input image. Second, 'patch tunneling' builds independent tunnel-like paths consisting of multiple bottleneck blocks per patch, penetrating through the entire model from an input patch to the last layer of the network, maintaining lightweight memory usage throughout the whole network. Lastly, 'bottleneck reordering' rearranges the execution order of convolution operations inside the bottleneck block such that the memory usage remains constant regardless of the size of the convolution output channels. The experiment result shows that the proposed network classifies ImageNet with extremely low memory (*i.e.*, 63 KB) while achieving competitive top-1 accuracy (*i.e.*, 61.58%). To the best of our knowledge, the memory usage of the proposed network is far smaller than state-of-the-art memory-efficient networks, *i.e.*, up to 89x and 3.1x smaller than MobileNet (*i.e.*, 5.6 MB) and MCUNet (*i.e.*, 196 KB), respectively.

1 Introduction

As an increased number of vision applications are moving towards low-end devices, a variety of on-device CNNs have been devised in an efficient and lightweight manner. Among many of them, MobileNet [23] has become the de facto standard for many resource-constrained embedded, mobile, and IoT devices thanks to its efficient network architecture, especially the simple yet effective bottleneck block. Although MobileNet [23] enables low-end devices to run various vision tasks(*e.g.*, image classification [5] and object detection [9,19]), the stacked bottleneck block structure significantly contributes to its peak memory (RAM) requirement(*i.e.*, 5.6 MB for ImageNet [5]), which is still far exceeds the limited memory capacity of millions of low-end devices having only several hundreds of KB of RAM(*e.g.*, 256 KB), as shown in Fig. 1. Considering that 1) embedded

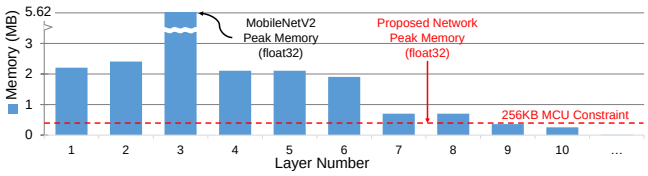

Fig. 1: Peak memory usage across MobileNetV2 layers on ImageNet 224x224, reaching 5.62MB at the third. The red line indicates our network's 256KB peak memory in fp32.

cameras are becoming cheaper, smaller, more powerful, and low-powered [1, 2], and 2) an explosive number of embedded cameras are being installed on low-end devices at a fast pace, an increasing number of low-end devices and cameras are expected to handle advanced vision tasks on the device.

The bottleneck block of MobileNet is the key enabler for the rapid deployment of CNNs onto resource-constrained low-end devices. Based on an inverted residual structure where the skip connections are between the bottleneck layers, it efficiently learns features and representations by applying point-wise and depth-wise convolutions. However, since it deals with large-sized images with the bottleneck block that expands the intermediate features with the point-wise (expansion) convolution, its memory usage soars (5.6 MB) beyond the memory capacity of many low-end devices having only KB of memory, such as ARM core-based IoT devices [24, 25]. Therefore, to meet the growing demand for on-device vision applications running on low-end devices, the memory usage of MobileNet, composed of multiple bottleneck blocks, should be reduced substantially.

In this paper, we introduce an extremely memory-efficient CNN, which dramatically reduces the memory usage of the bottleneck block [23], making MobileNet's learning capability accessible to memory-constrained low-end devices for various computer vision tasks(*e.g.*, image classification [3, 5] and object detection [9,19]). We propose three design principles that collectively construct the proposed memory-efficient(and memory-aware) CNN, with peak memory usage can be flexibly adjusted to fit the limited memory budget of low-end devices. **First**, 'input segmentation' divides an input image into a set of patches, reducing the memory requirement of large input images at the beginning of the network. To compensate for the possible performance degradation caused by input segmentation, we make patches partially overlap with each other along their edges and add a central patch to capture crucial features in the center of the image. **Next**, each segmented input patch constructs a dedicated network path, consisting of multiple bottleneck blocks, all the way through the whole network independent of each other, which we call 'patch tunneling', keeping the memory usage of each patch under the memory budget. At the last layer of the network, they are combined to cooperate for the target task(*e.g.*, image classification and object detection). **Lastly**, 'bottleneck reordering' maintains the memory usage of each bottleneck to be constant by rearranging execution orders of the point-wise and depth-wise convolution operations inside the bottleneck. Unlike existing approaches that execute convolution operations layer by layer without manag-

ing the peak memory usage, it reorders them across layers in a memory-efficient manner based on the computational properties of the convolution.

We implement the proposed network and deploy it onto a real low-end device, *i.e.*, STM32H7 [25]. We conduct experiments with two classes of vision tasks, *i.e.*, image classification (ImageNet [5] and VWW [3]) and object detection (PASCAL-VOC [9] and MS-COCO [19]). The results show that the proposed network enables on-device vision tasks with extremely low memory while providing competitive model performance, *e.g.*, 61.58% and 63.84% top-1 accuracy on ImageNet using only 63 KB and 254 KB of memory, respectively, validating the efficacy of the proposed network for memory-constrained low-end devices.

The contributions of this paper are summarized as follows.

- We introduce an extremely memory-efficient CNN for memory-limited low-end devices, taking only 63 KB of memory for ImageNet classification, which is 89x and 3.1x lower than state-of-the-art memory-efficient CNNs, *i.e.*, MobileNet (5.6 MB) [23] and MCUNet (196 KB) [16], respectively.
- We propose three memory-aware design principles, *i.e.*, 'input segmentation', 'patch tunneling', and 'bottleneck reordering', which construct an extremely memory-efficient CNN based on the memory budget of low-end devices.
- We implement and deploy the proposed network onto a real embedded device (*i.e.*, STM32H7 [25]), demonstrating that it enables extremely memory-efficient on-device vision tasks, *i.e.*, image classification and object detection, achieving 61.58% top-1 accuracy on ImageNet using only 63 KB memory.

2 Related Work

General Techniques for Making Networks Efficient: Various methods have been proposed to improve the efficiency of neural networks. Quantization [10,14] simplifies the precision of number formats used in a network, usually from float32 to int8, int4, int2, or even binary bit [13], intending to reduce the amount of computation. Although it can decrease memory usage along with computation, the memory reduction ratio is bounded to the number of bits quantized, usually 4x (from float32 to int8), and the model performance tends to deteriorate as more aggressive quantization is applied [18]. Pruning [26, 27] is another technique that removes unnecessary weight parameters in a network based on their importance [11]. After pruning, a network becomes streamlined, improving resource efficiency, including the memory requirement. However, it deforms the original network architecture, causing several negative impacts, such as model performance degradation, unexpected model behavior, and repeated training procedures. Even worse, the memory usage may not be reduced after pruning if the pruned weight parameters are replaced with zeros, leaving the memory usage for the related operations the same. NAS (Neural Architecture Search) [6, 8, 12] seeks the optimal network architecture, which can be applied to find a memory-efficient network by imposing a memory constraint during the search [15]. Although it automatically searches for a potentially memory-efficient network, it is not guaranteed to find the best architecture that satisfies both the

model performance and memory constraint. Moreover, the search process usually takes tens or hundreds of GPU days [30], which is exorbitant in practice. Unlike those techniques orthogonal to the proposed methods, the three design principles introduced in this paper 1) construct a CNN with flexible memory reduction, not restricted by some upper bound, unlike quantization, 2) do not severely degrade the model performance, 3) readily apply to various tasks without modifying the network architecture, and 4) do not entail a long search time nor repeated training processes as NAS. Since they are orthogonal to the proposed methods, we apply one of them, *i.e.*, quantization, to our CNNs.

MCUNet [16,17] is a memory-efficient CNN proposed to perform ImageNet [5] classification under 242 KB of memory on the STM32F board [24], achieving 60.3% and 64.9% top-1 classification accuracy in their first and second implementation, respectively. The memory requirement is reduced with the in-place depth-wise convolution and patch-based inference, executed by a run-time library called TinyEngine on the device. The network architecture that fits the target memory is searched by NAS [6,8,12] via the joint neural architecture and inference scheduling search. Although MCUNet is the first network that classifies ImageNet with the tight memory budget, it relies on existing techniques such as NAS and quantization, along with a significant embedded systems engineering procedure, entailing a large amount of time, computation, and human endeavors to find the network that fits the target memory budget. However, unlike MCUNet, the proposed network does not require quantization as mandatory and still achieves a similar level of memory usage, *i.e.*, 254 KB with the float32 format, which can be further reduced to 63 KB when applying int8 quantization. Also, given a memory budget, the proposed network can be easily and flexibly constructed for various vision tasks, *e.g.*, image classification and object detection, by applying the proposed three design principles accordingly, without requiring significant time, computation, and engineering efforts.

3 Method: Design Principles

Fig. 2 depicts an overview of the proposed network, in which the peak memory usage of convolutional operations is flexibly adjusted under the memory budget of the target low-end device. It is achieved by the three memory-aware design principles, *i.e.*, 'input segmentation', 'patch tunneling', and 'bottleneck reordering', of which details are described in the following subsections.

3.1 Input Segmentation

As the size of images fed into a CNN is usually too large to fit the limited memory of many low-end devices, we decrease the dimension of the raw input image by dividing it into several patches through a process called 'input segmentation' (Fig. 3). Given an input image of dimension $[h \times w \times c]$, where h is the height, w is the width, and c is the number of channels, the input image is segmented

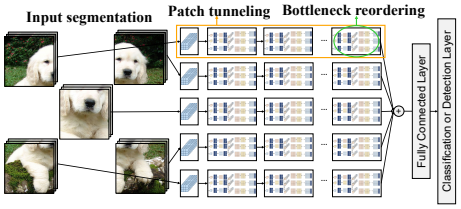

Fig. 2: An overview of the proposed memory-efficient CNN constructed by the three memory-aware design principles: 'input segmentation', 'patch tunneling', and 'bottleneck reordering'.

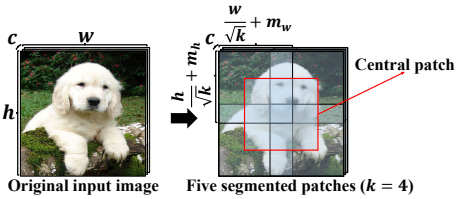

Fig. 3: The proposed 'input segmentation' splits the input image into k patches, along with the central patch. Which results in a reduction of the initial memory requirement of the network into $\frac{1}{k}$.

into k patches of dimension $[\frac{h}{\sqrt{k}} \times \frac{w}{\sqrt{k}} \times c]$. For instance, the images in ImageNet [5] (*i.e.*, $[224 \times 224 \times 3]$), are segmented into four patches of the dimension $[112 \times 112 \times 3]$ when $k = 4$, reducing the memory space required for the raw image by $\frac{1}{k} = \frac{1}{4}$ by processing each patch separately. Although such a simple segmentation can mechanically reduce the memory usage to $\frac{1}{k}$, it may degrade the model performance due to the segregation of features, making it difficult for the segmented patches to have a common view of the entire image. To compensate for such potential performance loss likely to be caused by segmentation, we add some margins along patch edges so that they can overlap slightly with each other. With the vertical and horizontal margins, denoted as m_h and m_w, respectively, the dimension of patches then becomes:

$$[(h/\sqrt{k} + m_h) \times (w/\sqrt{k} + m_w) \times c] \tag{1}$$

which makes the overlapped areas of $2m_h$ and $2m_w$ between two neighboring patches over the vertical and horizontal axis, respectively. For simplicity, by assuming that an image is square, *i.e.*, $h = w$, and $m_h = m_w$, the ratio between the original image $[h \times w \times c]$ and the patch in Eq. (1) becomes $\frac{1}{k} + \frac{2m_h}{h\sqrt{k}} + \frac{m_h^2}{h^2}$, which becomes smaller when k and h increase, converging to $\frac{1}{k}$ when $h \gg m_h$. For further improvement, we add the $(k+1)$-th patch on top of k patches, which we call the central patch, to learn critical information located at the central part of the image likely to affect the model performance significantly, resulting in a total of $k+1$ patches. It is based on the observation that objects tend to be situated at the central area of the image so it is worth having a dedicated patch for that region. The central patch overlaps entirely with other k patches, learning important features in the central area at the cost of having an additional convolution kernel for it. However, since each patch is executed independently, adding the central patch can be a useful way to enhance the model performance without the peak-memory increase as shown in the experiment (Sec. 4), which can be regarded as a good trade off between the huge model performance improvement and the relatively small increase in the network size.

3.2 Patch Tunneling

The idea of 'patch tunneling' is that features presented at each $k+1$ individual patches divided by 'input segmentation' can be learned separately all the way through isolated layers of lightweight (memory-efficient) convolutions, which we call a patch tunnel, and then be effectively combined at the last layer of the network. It is different from conventional approaches that apply the convolution operation to the non-segmented entire image with a large number of convolution kernels (filters), which requires a huge memory space for storing the input and output of the convolution, instead of learning a smaller patch with a relatively small number of kernels taking a small amount of memory. With $k+1$ patches divided by 'input segmentation' at the first layer of the network, 'patch tunneling' constructs $k+1$ independent patch tunnels consisting of multiple layers of lightweight convolutions in the form of bottleneck [23], starting from each patch, such that the memory requirement of the convolution operation of each layer does not exceed the memory budget m_b until the last layer of the network. Given the input feature of dimension $[h \times w \times c]$ and the convolutional kernel (filter) as $[k_h \times k_w \times c_{in} \times c_{out}]$ at an arbitrary layer of a patch tunnel, the output dimension of the convolutional operation becomes $[\big(\frac{h-k_h+2p_h}{s}+1\big)\times\big(\frac{w-k_w+2p_w}{s}+1\big)\times c_{out}]$, where s is the stride, and p_h and p_w is the vertical and horizontal padding, respectively. Since both the input and output should be stored in memory during the convolution operation, the summation of the input and output dimension, while ignoring the minute memory usage required for the convolution computation, should be maintained at most the target memory budget m_b as:

$$\underbrace{h \times w \times c}_{\text{Input}} + \underbrace{\big(\frac{h-k_h+2p_h}{s}+1\big)\times\big(\frac{w-k_w+2p_w}{s}+1\big)\times c_{out}}_{\text{Output}} \leq m_b \qquad (2)$$

which first starts from the segmented patches in Eq. (1) as the initial input $[h \times w \times c]$ at the first layer of the network and computes the corresponding convolution output that will be used as the input for the next convolution layer. By adjusting s, k_h, w_h, p_h, p_w, and c_{out} accordingly, Eq. (2) becomes satisfied with m_b for all convolution layers of each patch tunnel. Since the dimension of each patch is approximately reduced to $\frac{1}{k}$ of the original image with some margins as in Eq. (1), it is possible to find s, k_h, w_h, p_h, p_w, and c_{out} satisfying Eq. (2) without degrading the model performance substantially. For example, the summation of the input and output in Eq. (2) becomes under $m_b = 256$ KB (the size of SRAM on STM32F [24]), by setting $s = 1, k_h = 3, w_h = 3, p_h = 0, p_w = 0$, and $c_{out} = 3$, given the input patch of ImageNet [5] as $[\big(\frac{h}{\sqrt{k}}+m_h\big)\times\big(\frac{w}{\sqrt{k}}+m_w\big)\times c]$ with $h = 224, w = 224, c = 3, k = 4$, and $m_h = m_w = 18$.

At the last layer of the network, the features independently learned and propagated through each patch tunnel are combined as the aggregated feature set with a fully connected layer. We found that summation is sufficient to aggregate features effectively, achieving competitive model performance while taking smaller memory compared to alternative aggregation methods such as concatenation. By keeping the memory requirement of each patch tunnel at most the

memory budget m_b and postponing the aggregation process of features learned in each tunnel until the last layer of the network, a single large convolution consuming a large amount of memory, can be effectively split into $k+1$ smaller patch tunnels, keeping the overall memory usage by almost $\frac{1}{k}$ in the entire network.

3.3 Bottleneck Reordering

Although each patch tunnel can retain its memory usage within the memory budget m_b by adjusting convolution kernel parameters, *i.e.*, s, k_h, w_h, p_h, p_w, and c_{out} in Eq. (2), the extremely low memory budget of many low-end devices (e.g., $m_b = 256$ KB) limits the number of output channels *i.e.*, c_{out}, to a point where feature learning becomes insufficient, potentially degrading model performance. In [23], the expansion ratio t was introduced to expand c_{out} in point-wise (expansion) convolution. Then c_{out} can be expressed in terms of t, such as $c_{out} = c_{in} \times t$. For adequate model performance, however, c_{out} must be increased beyond the constraints of the memory budget m_b.

To tackle this problem, we propose 'bottleneck reordering', which rearranges the operation order of the convolution in the bottleneck [23] such that Eq. (2) is satisfied with a large c_{out}. It executes the convolution operation for each output channel independently one at a time over the three convolution layers in the bottleneck, *i.e.*, the point-wise (expansion), depth-wise, and point-wise (reduction) convolution, until all the output channels are computed, as shown in Fig. 4. Then, the independently computed outputs of each channel are relayed to the last layer of the bottleneck and accumulated into the final output. Since only a single output channel is computed and stored in memory one at a time over the entire bottleneck block, the memory space required to store the outputs of the point-wise (expansion) and depth-wise convolution in Eq. (2) is reduced to $\frac{1}{c_{out}}$ for each. Unlike the conventional way that performs convolutions over all output channels at the same time and stores the final output in a huge memory space, 'bottleneck reordering' simply changes the execution order of convolution

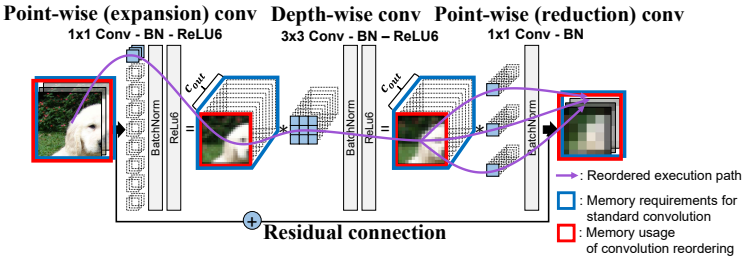

Fig. 4: The proposed 'bottleneck reordering' restricts the memory usage of the bottleneck to be constant irrespective of the output channel size (c_{out}) of the point-wise (expansion) and depth-wise convolution by rearranging their execution order: each convolution output channel is computed one at a time (red box), not all at once (blue box).

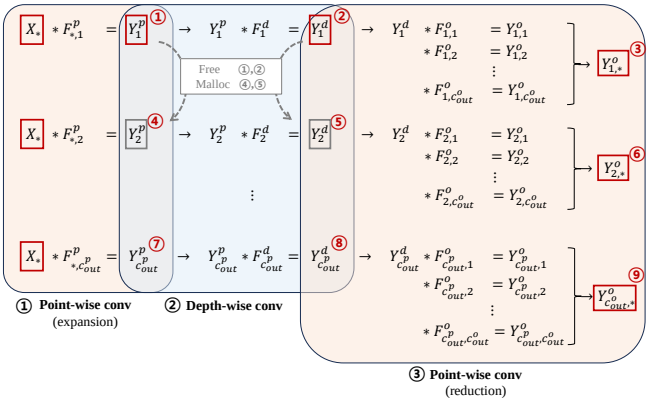

Fig. 5: In a typical bottleneck operation, the black 1, 2, and 3 conv layers are processed sequentially. With bottleneck reordering, the layers are computed in the order of red (1, 2, 3), (4, 5, 6), and (7, 8, 9). After each cycle, two intermediate outputs are freed, and the next outputs (gray boxes and lines) are allocated. This approach reduces the peak memory requirement caused by intermediate outputs to $1/c_{\text{out}}$.

in a channel-wise manner for consecutive convolution layers in the bottleneck, producing the exact same final output using much smaller (constant) memory.

By applying 'bottleneck reordering' over three consecutive convolution layers in the bottleneck, *i.e.*, the point-wise (expansion), depth-wise, and point-wise convolution (reduction), the total memory space for storing the related input and outputs of the bottleneck becomes at most the memory budget m_b as:

$$\underbrace{h_i \times w_i \times c_i}_{\substack{\text{Input of}\\ \text{bottleneck}}} + \underbrace{h_o^p \times w_o^p \times 1}_{\substack{\text{Output of}\\ \text{point-wise conv}}} + \underbrace{h_o^d \times w_o^d \times 1}_{\substack{\text{Output of}\\ \text{depth-wise conv}}} + \underbrace{h_o \times w_o \times c_o}_{\substack{\text{Output of}\\ \text{bottleneck}}} \leq m_b \qquad (3)$$

where $[h_i \times w_i \times c_i]$ and $[h_o \times w_o \times c_o]$ is the memory space for the initial input and the final output of the bottleneck, respectively, $[h_o^p \times w_o^p \times c_{out}^p]$ is the memory space for the output of the point-wise (expansion) conv, and $[h_o^d \times w_o^d \times c_{out}^d]$ is the memory space for the output of the depth-wise conv, with $c_{out}^p = c_{out}^d = 1$. Fig. 5 illustrates the reordering procedure, where X is the input of the bottleneck, F_i^p, F_i^d, F_i^o is the i-th kernel (filter) of the point-wise (expansion), depth-wise, and point-wise (reduction) convolution, Y_j^p, Y_j^d, Y_j^o is the j-th channel output of the point-wise (expansion), depth-wise convolution, and bottleneck, respectively.

Note that the output channels of both the point-wise (expansion) and depth-wise convolution (*i.e.*, $[h_o^p \times w_o^p \times 1]$ and $[h_o^d \times w_o^d \times 1]$) stays as one, *i.e.*, $c_{out}^p = c_{out}^d = 1$, as highlighted as light-gray in Eq. (3), meaning that the required output memory space does not increase over the number of output channels, *i.e.*, c_{out}^p and c_{out}^d, at all. Considering that the point-wise (expansion) convolution of the bottleneck block is usually executed with a large size of output channels, *e.g.*, $c_{out}^p = 100, 200, ...$, to improve the model performance, the proposed 'bottleneck

Table 1: The peak memory usage and classification accuracy (top-1 and top-5) of the proposed network on ImageNet [5], compared with MobileNet [23] and MCUNet [16,17].

Model Name	Peak-mem(KB)	Quant	Res	Top-1(%)	Top-5(%)
Proposed network	254/63	fp32/int8	224	63.84/61.58	84.80/83.26
	533/133	fp32/int8	224	64.68/61.50	85.47/83.19
MCUNetV1	968/242	fp32/int8	160	60.90/60.30	83.30/82.60
MCUNetV2	196	int8	N/A	64.90	86.20
	456	int8	N/A	71.80	90.70
MobileNetV2	5,619	fp32	224	72.83	91.06

reordering' provides an effective way of constraining the large memory space for storing the output channels to a constant, regardless of the size of c^p_{out} and c^d_{out}.

4 Experiment

We implement a series of CNNs based on the proposed three design principles in C language and deploy them onto STM32 development boards [25], featuring an ARM Cortex-M7 processor and 256 to 4,096 KB of SRAM. The networks are evaluated on two key tasks: image classification [3,5] and object detection [9,19].
Datasets. For image classification, we evaluate the proposed networks with ImageNet [5] and VWW (Visual Wake Words) [3]. For object detection, we use PASCAL-VOC [9] and MS-COCO [19] for evaluation.
Network Architecture. By using the three proposed design principles (Sec. 3), we construct CNNs with different memory budgets (*i.e.*, m_b), ranging from 70 to 512 KB (float32) for image classification and from 259 to 1,719 KB (float32) for object detection, and then apply int8 quantization (Quant) to each of them. For image classification, the input image is segmented into four patches (k=4) along with the central patch, where each patch has m_h=m_w=18 and 9 of margins as the default setups, and experiments are conducted by changing them. For object detection, we attach the SSDLite [20] structure after bottleneck blocks.
Baselines. We compare the memory usage and model performance of the proposed networks with the two baselines, *i.e.*, MobileNet [23] and MCUNet [16,17].

4.1 Image Classification

Memory Usage and Model Performance. Tab. 1 and 2 show the proposed networks with different memory budgets (usages) and classification accuracy on ImageNet [5] and VWW [3], compared with the two baselines, *i.e.*, MobileNet [23] and MCUNet [16, 17]. In Tab. 1, the proposed network takes the lowest memory among all, *i.e.*, 63 KB with quantization (int8), 3.1x smaller than the smallest one of the baselines (196 KB), achieving 61.58% and 83.26% top-1 and top-5 accuracy. Even without quantization, it can be seen that the

Table 2: The peak memory usage and classification accuracy of the proposed network, MobileNet [23], and MCUNet [16, 17] on VWW (Visual Wake Words) [3].

Model Name	Peak-mem(KB)	Quant	Res	Accuracy(%)
Proposed network	70/18	fp32/int8	66	82.80/82.63
	254/63	fp32/int8	224	90.67/87.99
MCUNetV1	584/146	fp32/int8	64	87.40/87.30
	648/162	fp32/int8	160	88.90/88.90
	1,244/311	fp32/int8	144	91.70/91.80
MCUNetV2	30	int8	N/A	90.00
	62	int8	N/A	93.00
	118	int8	N/A	94.00

Table 3: The peak memory usage and classification accuracy on ImageNet [5] with various expansion ratios (*i.e.*, t) in the bottleneck. The value of c_{out}, representing the number of output channels after channel expansion, is derived by multiplying t with the number of output channels from the preceding depthwise convolution layer.

t	Peak-mem(KB)	Quant	Top-1(%)	Top-5(%)
2	254/63	fp32/int8	58.82/55.75	81.04/78.97
4	254/63	fp32/int8	62.28/58.98	83.76/81.63
6	254/63	fp32/int8	63.48/60.70	84.98/82.92
8	254/63	fp32/int8	63.84/61.58	84.80/83.26

proposed network still takes the relatively lower memory, *i.e.*, 254 KB (float32), achieving 63.84% top-1 accuracy, comparable to MCUNet requiring 196 KB with quantization (int8) to provide 64.90% top-1 accuracy. In Tab. 2, it also takes the smallest memory, *i.e.*, 18 KB, while achieving comparable classification accuracy, *i.e.*, 82.63%. It shows that the proposed three design principles allow for flexibly building memory-efficient networks that fits extremely tight memory budgets in a memory-aware manner while providing comparable model performance.

Channel Output. Tab. 3 shows the peak memory usage and classification accuracy of ImageNet [5] with different numbers of expansion ratio, *i.e.*, t, of the point-wise (expansion) convolution in the bottleneck. As shown in the table, the peak memory usage does not increase over t, enabled by 'bottleneck reordering', whereas the classification accuracy increases, *e.g.*, 58.82% (t=2) and 63.84% (t=8) top-1 accuracy, entailing a slight increase in the point-wise kernel size. It shows that a flexible trade-off exists between the model performance and kernel size, whereas the peak memory usage remains constant. **Patch Composition.** Tab. 4 shows the peak memory usage and classification accuracy of ImageNet [5] and VWW [3] when removing the central patch tunnel from the network. As shown in the table, the classification accuracy drops for both ImageNet (from 63.84% to 62.32%) and VWW (from 82.80% to 78.97%), substantiating the crucial role of the central patch in classification. Tab. 5 shows the peak memory usage and classification accuracy of ImageNet [5] and VWW [3] over different

Table 4: The experiments on ImageNet [5] and VWW (Visual Wake Words) [3] without the central patch (*i.e.*, having only four patches in the network).

Dataset	# of Patches	Peak-mem(KB)	Quant	Top-1(%)	Top-5(%)
ImageNet	4	254/63	fp32/int8	62.32/84.38	84.38
VWW	4	70/18	fp32/int8	78.97/78.46	N/A

Table 5: The experiments on ImageNet [5] and VWW (Visual Wake Words) [3] with different numbers of patches (*i.e.*, k). The top two rows show the result of the segmented patches, while the remaining rows show the result of resizing the non-segmented images.

ImageNet					VWW			
Patch	Mem	Quant	Top-1	Top-5	Patch	Mem	Quant	Top-1
5	254KB	fp32	63.84%	84.80%	5	70KB	fp32	82.80%
5	63KB	int8	61.58%	83.26%	5	18KB	int8	82.63%
5	254KB	fp32	60.70%	82.59%	5	70KB	int8	81.56%
5	63KB	int8	57.61%	80.80%	5	18KB	fp32	81.39%
4	254KB	fp32	60.83%	82.93%	4	70KB	fp32	81.46%
4	63KB	int8	57.65%	80.16%	4	18KB	int8	81.02%
3	254KB	fp32	59.14%	81.90%	3	70KB	fp32	81.23%
3	63KB	int8	58.54%	81.85%	3	18KB	int8	80.97%
2	254KB	fp32	57.13%	80.40%	2	70KB	fp32	81.10%
2	63KB	int8	55.67%	78.97%	2	18KB	int8	80.12%

numbers of patches (*i.e.*, from $k = 5$ to $k = 2$), which learns the non-segmented entire images resized to [130×130×3] and [42×42×3], respectively, where the two top rows show the cases when learning the proposed five segmented image patches (*i.e.*, 'input segmentation'). It shows that 'input segmentation' helps boost the model performance, while learning the same entire images as multiple patches does not provide similar performance regardless of the number of patches (*e.g.*, 63.84% vs. 60.70% in ImageNet [5]), demonstrating the proposed 'input segmentation' is essential to achieve competitive performance. **Margin Size.** Tab. 6 shows the peak memory usage and classification accuracy of ImageNet [5] and VWW [3] over different sizes of margins, *i.e.*, m_h and m_w, where larger margins tend to achieve better model performances in general. It demonstrates that adding some margin to the patch, leading them to slightly overlap with each other, is crucial to the model performance. However, since a larger margin (*i.e.*, m_h and m_w) increases the memory requirement as in Eq. (1), the proper margin values should be chosen based on the memory budget.

4.2 Object Detection

Memory Usage and Model Performance. Tab. 7 and 8 show the proposed networks with different memory budgets and mAP (mean Average Precision)

Table 6: The experiments on ImageNet [5] and VWW (Visual Wake Words) [3] by varying the patch margin (*i.e.*, m_h and m_w).

ImageNet					VWW			
Margin	**Mem**	**Quant**	**Top-1**	**Top-5**	**Margin**	**Mem**	**Quant**	**Top-1**
18	254KB	fp32	63.84%	84.80%	9	70KB	fp32	82.80%
18	63KB	int8	61.58%	83.26%	9	18KB	int8	82.63%
10	208KB	fp32	62.23%	83.85%	5	70KB	fp32	80.48%
10	52KB	int8	60.00%	82.42%	5	18KB	int8	80.79%
5	195KB	fp32	62.12%	83.91%	2	70KB	fp32	80.05%
5	49KB	int8	59.51%	82.25%	2	18KB	int8	79.65%
0	175KB	fp32	61.31%	83.39%	0	70KB	fp32	80.00%
0	44KB	int8	58.33%	81.33%	0	18KB	int8	79.60%

Table 7: The peak memory usage and mAP (mean Average Precision) of the proposed network, MobileNet [23], and MCUNet [16, 17] on PASCAL-VOC [9].

Model Name	**Peak-mem(KB)**	**Quant**	**Res**	**mAP (%)**
Proposed network	259/65	fp32/int8	130	51.8/50.7
	702/176	fp32/int8	224	63.4/62.1
	1,260/315	fp32/int8	300	67.0/66.3
MCUNetV1	466	int8	224	51.4
MCUNetV2	438	int8	224	68.3
	247	int8	224	64.6
MobileNetV2	10,080	fp32	300	68.6

on PASCAL-VOC [9] and MS-COCO [19], compared with MobileNet [23] and MCUNet [16, 17]. For MS-COCO, all networks are trained with trainval35k and evaluated on minival5k in the MS-COCO 2014 dataset. Similar to image classification, the proposed network takes the lowest memory among all in Tab. 7, *i.e.*, 65 KB with quantization (int8), 155x and 3.8x smaller than MobileNet (10,080 KB) and MCUNet (247 KB), respectively, achieving 50.7% mAP, showing that it effectively trades off the peak memory usage and mAP. However, the network taking the second lowest memory, *i.e.*, 176 KB, achieves 62.1% mAP, comparable to MCUNet taking 247 KB to provide 64.6% mAP. In Tab. 8, the proposed network also takes much smaller memory compared to MobileNet, *e.g.*, 1,719 KB (5.8x smaller) with a reasonable level of mAP (30.0%), verifying that the proposed design principles also effectively apply to object detection. Fig. 6 shows examples of object detection on PASCAL-VOC [9] and MS-COCO.

Channel Output. Tab. 9 shows the peak memory usage and mAP (mean Average Precision) on PASCAL-VOC and MS-COCO with different numbers of expansion ratio, *i.e.*, t, of the point-wise (expansion) convolution in the bottleneck. As shown in the table, the peak memory usage does not increase over t, enabled by 'bottleneck reordering', whereas mAP increase, *e.g.*, from 52.3% to

Table 8: The peak memory usage and mAP (mean Average Precision) of the proposed network, MobileNet [23], and MCUNet [16, 17] on MS-COCO [19].

Model Name	Peak-mem(KB)	Quant	Res	mAP^{val}(%)
Proposed network	446/112	fp32/int8	130	22.0/21.2
	1,210/303	fp32/int8	224	28.9/27.2
	1,719/430	fp32/int8	224	30.0/28.0
MobileNetV2	10,080	fp32	300	31.8

Table 9: The object detection on PASCAL-VOC [9] and MS-COCO [19] with different expansion ratios (*i.e.*, t) in the point-wise (expansion) conv within the bottleneck.

	Pascal-VOC				MS-COCO			
t	Mem(KB)	Quant	Res	mAP(%)	Mem(KB)	Quant	Res	mAP(%)
2	446/112	fp32/int8	130	22.0/21.2	446/112	fp32/int8	130	22.0/21.2
2	210/303	fp32/int8	224	28.9/27.2	446/112	fp32/int8	130	22.0/21.2
2	719/430	fp32/int8	224	30.0/28.0	446/112	fp32/int8	130	22.0/21.2

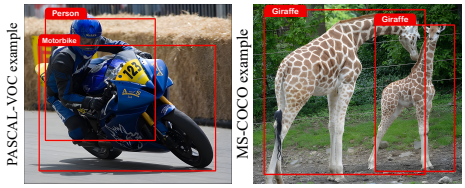

Fig. 6: Examples of object detection performed by the proposed network: (Left) PASCAL-VOC [9] and (Right) MS-COCO [19].

63.4% in PASCAL-VOC, along with a slight increase in the convolution kernel size. Similar to the image classification tasks, the proposed network provides a flexible trade-off between mAP and the size of the kernel for object detection, while keeping the peak memory requirement constant.

5 Limitations and Discussions

Network Size. Although the proposed network takes much smaller memory when compared to state-of-the-art memory-efficient MCUNet [16, 17] and MobileNet [23], *e.g.*, 89x and 3.1x smaller for ImageNet [5], respectively, the network size is not decreased significantly, unlike huge memory reduction. That is because the number of weight parameters of the proposed network for ImageNet, *i.e.*, 3.34 million, is similar to that of MobileNet (*i.e.* 3.2 million) and larger than that of MCUNet (*i.e.* 0.73 million). It can be regarded as a trade-off between memory usage and network size, which can be flexibly adjusted based on the system and/or application requirements. Since the MCUNet architecture was designed using NAS, unlike our network, we expect that NAS could identify

a more efficient architecture in terms of weight parameters while maintaining our three proposed design principles. Alternatively, the weight parameters of k patch tunnels could be shared among them to reduce the total number of weight parameters of the network by k times. Also, it can be considered to combine some patch tunnels at the deeper layer of the network where the memory usage is smaller, as seen in Fig. 1. We will investigate these approaches in future work.
Larger Images. The proposed 'input segmentation' and 'patch tunneling', in principle, can be applied to images larger than the ImageNet data (*i.e.* 224×224) by adjusting the number of patches and the overlapped areas accordingly based on the memory budget. However, as a large image is segmented into an increased number of smaller patches, the amount of information included in a single patch tends to decrease, making it more difficult to look at and learn the image from a global perspective. As each patch tunnel separately learns features available only in an individual local patch, the feature aggregation procedure for the global image learning performed at the last layer of the network is likely to become more difficult, possibly causing model performance degradation. We expect that it can be mitigated by segmenting an image into patches more flexibly, such as making different sizes of patches at various positions of the image which can cover different areas of the image with varying sizes. We also leave it for future work.
Other Tasks and Datasets: In this work, we construct memory-efficient CNNs for two vision tasks, *i.e.*, image classification and object detection, demonstrating its effectiveness. For a more thorough evaluation, however, additional experiments on different tasks and various datasets, such as image segmentation [4,29], super resolution [7, 22], and image generation [21, 28], should be conducted.

6 Conclusion

We introduce a bottleneck-based, extremely memory-efficient CNN that fits the tight memory constraints (*e.g.*, under 256 KB) of embedded and IoT devices. The network is built on three design principles: input segmentation', patch tunneling', and 'bottleneck reordering', allowing flexible adjustment of memory usage based on device capacity. We implement the network on an actual device (STM32H7 [25]), demonstrating that these design principles enable memory-aware CNNs for various vision tasks, such as memory-efficient image classification on ImageNet [5] using 63 KB (61.58% top-1 accuracy) and object detection on PASCAL-VOC [9] using 65 KB (50.7% mAP).

Acknowledgement

This research was supported by the Challengeable Future Defense Technology Research and Development Program through the Agency for Defense Development(ADD) funded by the Defense Acquisition Program Administration(DAPA) in 2022(No.915062201), and Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2020-II201336, Artificial Intelligence Graduate School Program(UNIST)).

References

1. Bigas, M., Cabruja, E., Forest, J., Salvi, J.: Review of cmos image sensors. Microelectronics journal **37**(5), 433–451 (2006)
2. Choi, J., Shin, J., Kang, D., Park, D.S.: Always-on cmos image sensor for mobile and wearable devices. IEEE Journal of Solid-State Circuits **51**(1), 130–140 (2015)
3. Chowdhery, A., Warden, P., Shlens, J., Howard, A., Rhodes, R.: Visual wake words dataset (2019)
4. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding (2016), https://arxiv.org/abs/1604.01685
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
6. Dhar, S., Guo, J., Liu, J., Tripathi, S., Kurup, U., Shah, M.: A survey of on-device machine learning: An algorithms and learning theory perspective. ACM Transactions on Internet of Things **2**(3), 1–49 (2021)
7. Dong, C., Loy, C.C., He, K., Tang, X.: Image super-resolution using deep convolutional networks (2015), https://arxiv.org/abs/1501.00092
8. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. The Journal of Machine Learning Research **20**(1), 1997–2017 (2019)
9. Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes challenge: A retrospective. International Journal of Computer Vision **111**(1), 98–136 (Jan 2015). https://doi.org/10.1007/s11263-014-0733-5, https://doi.org/10.1007/s11263-014-0733-5
10. Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference. In: Low-Power Computer Vision, pp. 291–326. Chapman and Hall/CRC (2022)
11. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
12. He, X., Zhao, K., Chu, X.: Automl: A survey of the state-of-the-art. Knowledge-Based Systems **212**, 106622 (2021)
13. Huang, K., Ni, B., Yang, X.: Efficient quantization for neural networks with binary weights and low bitwidth activations. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 3854–3861 (2019)
14. Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing **461**, 370–403 (2021)
15. Liberis, E., Dudziak, Ł., Lane, N.D.: μnas: Constrained neural architecture search for microcontrollers. In: Proceedings of the 1st Workshop on Machine Learning and Systems. pp. 70–79 (2021)
16. Lin, J., Chen, W.M., Cai, H., Gan, C., Han, S.: Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. arXiv preprint arXiv:2110.15352 (2021)
17. Lin, J., Chen, W.M., Lin, Y., Gan, C., Han, S., et al.: Mcunet: Tiny deep learning on iot devices. Advances in Neural Information Processing Systems **33**, 11711–11722 (2020)
18. Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.M., Wang, W.C., Xiao, G., Dang, X., Gan, C., Han, S.: Awq: Activation-aware weight quantization for on-device llm compression and acceleration. Proceedings of Machine Learning and Systems **6**, 87–100 (2024)

19. Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft coco: Common objects in context (2015)
20. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. pp. 21–37. Springer (2016)
21. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild (2015), https://arxiv.org/abs/1411.7766
22. Magnitskaya, M., Chtchelkatchev, N., Tsvyashchenko, A., Salamatin, D., Lepeshkin, S., Fomicheva, L., Budzyński, M.: Electron and phonon properties of noncentrosymmetric rhge from ab initio calculations (2017), https://arxiv.org/abs/1708.08788
23. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)
24. STMicroelectronics: STM32F. https://www.st.com/en/microcontrollers-microprocessors/stm32f412.html (2024)
25. STMicroelectronics: STM32H7. https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html (2024)
26. Vadera, S., Ameen, S.: Methods for pruning deep neural networks. IEEE Access **10**, 63280–63300 (2022)
27. Xu, S., Huang, A., Chen, L., Zhang, B.: Convolutional neural network pruning: A survey. In: 2020 39th Chinese Control Conference (CCC). pp. 7458–7463. IEEE (2020)
28. Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., Xiao, J.: Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop (2016), https://arxiv.org/abs/1506.03365
29. Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., Torralba, A.: Semantic understanding of scenes through the ade20k dataset (2018), https://arxiv.org/abs/1608.05442
30. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)