International Journal of Computer Vision (2026) 134:91
https://doi.org/10.1007/s11263-025-02688-w

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

Yoel Park[1] · **Jaewook Lee**[1] · **Seulki Lee**[1]

Received: 30 April 2025 / Accepted: 17 September 2025
© The Author(s) 2025

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 and audio tasks, such as image classification, object detection, and audio classification, 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. We also present 'peak memory aware quantization', enabling desired peak memory reduction in actual deployment of quantized network. 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.

Keywords On-device CNN · Peak memory reduction · Image classification · Object detection · Audio classification

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 (Sandler et al., 2018) 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 (Sandler et al., 2018) enables low-end devices to run various vision tasks (*e.g.*, image classification (Deng et al., 2009) and object detection (Everingham et al., 2015; Lin et al., 2015)), the stacked bottleneck block structure significantly contributes to its peak memory (RAM) requirement (*i.e.*, 5.6 MB for ImageNet (Deng et al., 2009)), 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 cameras are becoming cheaper, smaller, more powerful, and low-powered (Choi et al., 2015; Bigas et al., 2006), 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. Additionally, audio-related tasks on low-end devices present significant challenges, yet they are increasingly in demand in applications such as urban noise monitoring (Kumari et

Communicated by Minsu Cho.

✉ Seulki Lee
seulki.lee@unist.ac.kr

Yoel Park
joel157@unist.ac.kr

Jaewook Lee
jaewook.lee@unist.ac.kr

[1] Artificial Intelligence Graduate School, Ulsan National Institute of Science and Technology(UNIST), Ulsan, Republic of Korea

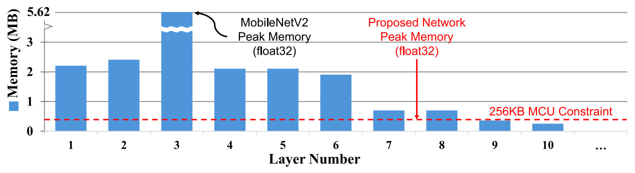

Fig. 1 Peak memory usage across MobileNetV2 layers on ImageNet 224x224, reaching 5.62MB at the third. The red line indicates our network's 256 KB peak memory in fp32

al., 2019) using Internet of Things (IoT) devices. Similar to vision tasks, CNNs can be applied to audio tasks-such as environmental sound classification-by transforming audio signals into two-dimensional spectrograms via the Fourier transform. However, for resource-constrained low-end devices, even allocating memory to store just five seconds of audio data can be infeasible due to their limited memory capacity.

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 and audio data 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 (STMicroelectronics, 2024a, b). Therefore, to meet the growing demand for on-device CNN 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 (Sandler et al., 2018), making MobileNet's learning capability accessible to memory-constrained low-end devices for various computer vision (*e.g.*, image classification (Deng et al., 2009; Chowdhery et al., 2019) and object detection (Everingham et al., 2015; Lin et al., 2015)) and audio tasks (*e.g.*, sound classification (Piczak, 2015; Salamon et al., 2014)). 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 managing the peak memory usage, it reorders them across layers in a memory-efficient manner based on the computational properties of the convolution, while having the total amount of computations remain the same.

To facilitate the efficient deployment of the proposed extremely memory-efficient CNN on real-world low-end devices, we introduce an optimization technique we named Peak Memory Aware Quantization, which sequentially applies de-quantization and re-quantization in conjunction with layer integration. It is specifically designed to ensure that the peak memory reduction originally intended through quantization is effectively realized at the activation level. In contrast to conventional quantization approaches (Wei et al., 2024), which may inadvertently increase peak memory usage upon deployment, the proposed Peak Memory Aware Quantization maintains minimal run-time memory usage by accounting for memory dynamics during the CNN inference process.

We implement the proposed network and deploy it onto a real low-end device, *i.e.*, STM32H7 (STMicroelectronics, 2024b). We conduct experiments with two classes of vision tasks, *i.e.*, image classification (ImageNet (Deng et al., 2009) and VWW (Chowdhery et al., 2019)), object detection (PASCAL-VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015)), and audio classification (ESC-50 (Piczak, 2015) and UrbanSound8k (Salamon et al., 2014)). 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) (Sandler et al., 2018) and MCUNet (196 KB) (Lin et al., 2021), 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 introduce two Peak Memory Aware Quantization methods, *i.e.*, 'Quantized BatchNorm + ReLU6 Integration (QBRI)' and 'Element-wise Requantization Accumulation (ERA)' which ensures that memory benefits are realized for real low-end devices.
– We implement and deploy the proposed network onto a real embedded device (*i.e.*, STM32H7 (STMicroelectronics, 2024b)), demonstrating that it enables extremely memory-efficient on-device vision and audio tasks, *i.e.*, achieving 61.58% top-1 accuracy on ImageNet using only 63 KB memory and 79.1% accuracy on UrbanSound8k using only 42KB.

2 Related Work

General Techniques for Making Networks Efficient: Various methods have been proposed to improve the efficiency of neural networks. Quantization (Gholami et al., 2022; Liang et al., 2021) simplifies the precision of number formats used in a network, usually from float32 to int8, int4, int2, or even binary bit (Huang et al., 2019), 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 (Lin et al., 2024a). Pruning (Xu et al., 2020; Vadera and Ameen, 2022) is another technique that removes unnecessary weight parameters in a network based on their importance (Han et al., 2015). 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) (Dhar et al., 2021; He et al., 2021; Elsken et al., 2019) seeks the optimal network architecture, which can be applied to find a memory-efficient network by imposing a memory constraint during the search (Liberis et al., 2021). 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 (Zoph et al., 2018), 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 (Lin et al., 2020, 2021) is a memory-efficient CNN proposed to perform ImageNet (Deng et al., 2009) classification under 242 KB of memory on the STM32F board (STMicroelectronics, 2024a), 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 (Dhar et al., 2021; He et al., 2021; Elsken et al., 2019) via the joint neural architecture and inference scheduling search. More recently MCUNetV3 (Lin et al., 2024b), has pushed the state-of-the-art further, but its main contribution lies in enabling on-device training under a 256 KB memory budget, introducing techniques like Quantization-Aware Scaling and Sparse Update to reduce the memory cost of backpropagation. As such, MCUNetV3 primarily targets training efficiency rather than inference-time memory reduction, and is therefore not a direct comparison point for our work.

In summary, while the MCUNet framework provides a powerful and holistic co-design methodology for ultra-low-memory deep learning, it heavily relies on hardware-specific optimizations, quantization, NAS, and customized runtime support, all of which entail substantial computational and engineering efforts. In contrast, our approach introduces a general and hardware-agnostic architectural design principle: without mandatory quantization or NAS, our model achieves a comparable memory footprint of 254KB in float32, further reducible to 63KB via post-training int8 quantization. Moreover, given a target memory budget, our design can be flexibly applied across both vision and audio tasks, *e.g.*, image or audio classification and object detection, without requiring significant time, computation, or embedded systems expertise.

3 Method: Design Principles

Figure 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.

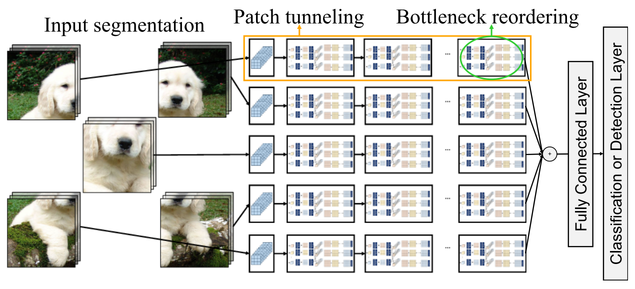

Fig. 2 An overview of the proposed network constructed by the three memory-aware design principles: 'input segmentation', 'patch tunneling', and 'bottleneck reordering'

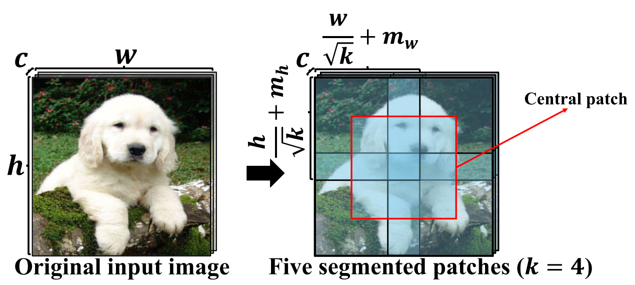
c
w
h
$\frac{w}{\sqrt{k}} + m_w$
$\frac{h}{\sqrt{k}} + m_h$
Central patch
Original input image
Five segmented patches ($k = 4$)

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 $\frac{1}{k}$

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 into k patches of dimension $[\frac{h}{\sqrt{k}} \times \frac{w}{\sqrt{k}} \times c]$. For instance, the images in ImageNet (Deng et al., 2009) (*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:

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

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 Eqation (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 (Section 7), 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 (Sandler et al., 2018), 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 $[(\frac{h-k_h+2p_h}{s}+1) \times (\frac{w-k_w+2p_w}{s}+1) \times c_{out}]$, 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:

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

which first starts from the segmented patches in Equation 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, Equation (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 Equation (1), it is possible to find s, k_h, w_h, p_h, p_w, and c_{out} satisfying Equation (2) without degrading the model performance substantially. For example, the summation of the input and output in Equation (2) becomes under m_b = 256 KB (the size of SRAM on STM32F (STMicroelectronics, 2024a)), 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 (Deng et al., 2009) as $[(\frac{h}{\sqrt{k}} + m_h) \times (\frac{w}{\sqrt{k}} + m_w) \times c]$ 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 Equation (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 Sandler et al. (2018), the expansion ratio t was introduced to expand c_{out} in point-wise (expansion) convolution. Then c_{out}

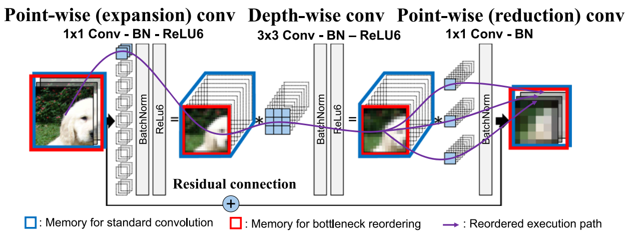
Point-wise (expansion) conv
1x1 Conv - BN - ReLU6
Depth-wise conv
3x3 Conv - BN – ReLU6
Point-wise (reduction) conv
1x1 Conv - BN
BatchNorm
ReLU6
BatchNorm
ReLU6
BatchNorm
Residual connection
□: Memory for standard convolution
□: Memory for bottleneck reordering
→: Reordered execution path

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)

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 (Sandler et al., 2018) such that Equation (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 Equation (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 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:

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

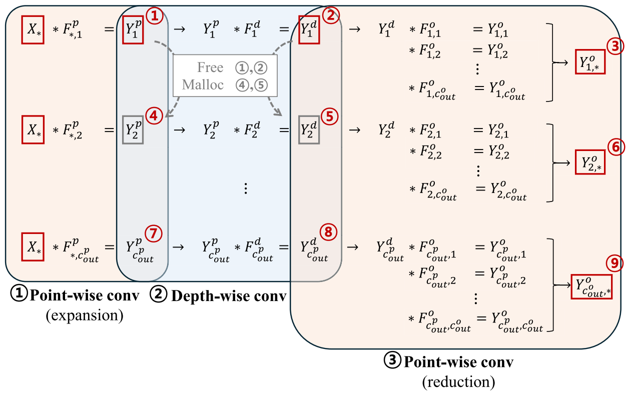

Fig. 5 In a typical bottleneck operation, the black 1, 2, and 3 conv layers are processed sequentially. With bottleneck reordering, the layers are computed in the order of red (1, 2, 3), (4, 5, 6), and (7, 8, 9). After each cycle, two intermediate outputs are freed, and the next outputs (gray boxes and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

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 $c_{out}^p = c_{out}^d = 1$. Figure 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 Equation (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.*, $c_{out}^p = 100, 200, \ldots$, to improve the model performance, the proposed 'bottleneck 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_{out}^p and c_{out}^d.

4 Peak-Memory Aware Quantization

The standard procedure for inference with quantized neural networks involves operating on int8 inputs that are repeatedly dequantized to fp32 for convolution, batch normalization, and activation, and then re-quantized to int8. Despite int8 quantization, most backends still allocate fp32 buffers due to compiler-level integer promotion and scaling operations.

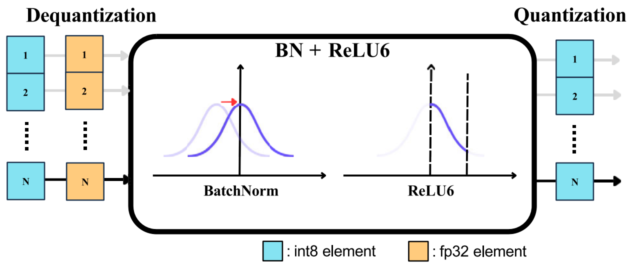

Fig. 6 An illustration of QBRI (Quantized BatchNorm + ReLU6 Integration), which keeps BatchNorm and ReLU6 outputs in int8 format, thereby eliminating fp32 activation storage

Consequently, fp32 activations dominate peak memory consumption, meaning that quantization does not substantially reduce inference memory as expected. To address these challenges, we propose a peak-memory aware quantization implementation that: (1) integrates BatchNorm and ReLU6 operations, and (2) employs element-wise requantization in point-wise (reduction) layers.

4.1 Quantized BatchNorm + ReLU6 Integration (QBRI)

Our target architecture consists of bottleneck blocks (Sandler et al., 2018) followed by a linear classifier. Each bottleneck block is composed of a consistent arrangement of CBR (Convolution-BatchNorm-ReLU6) units, specifically following a CBR-CBR-CB pattern. In these units, the BatchNorm layer is always followed by ReLU6. Conventional backends process both operations in fp32, requiring large intermediate activation buffers. To address this, we propose Quantized BatchNorm + ReLU6 Integration (QBRI). Each activation element is dequantized, normalized, clamped by ReLU6, and immediately re-quantized before moving to the next element. Thus, only int8 tensors are stored, with at most a single fp32 scalar held temporarily. Figure 6 illustrates this element-wise pipeline. When combined with bottleneck reordering, point-wise (expansion) and depth-wise convolutions also benefit: their single-channel outputs can be processed sequentially through QBRI, eliminating fp32 activation storage in these stages.

4.2 Element-wise Requantization Accumulation (ERA)

While QBRI prevents fp32 activations in BatchNorm and ReLU6, reduction convolutions require accumulation across multiple channels. Direct int8 accumulation risks overflow due to limited dynamic range, and thus QBRI alone cannot resolve this issue. To address this, we propose Element-wise Requantization Accumulation (ERA). In ERA, each partial sum is dequantized to higher precision, accumulated, and immediately re-quantized to int8 before proceeding to the next channel. This ensures that output activations remain in int8 throughout computation, without repeated addition of zero-points that could bias results. To further stabilize this process, we adopt Quantization Aware Training (QAT). QAT is a training technique that simulates the effects of quantization during model optimization (Jacob et al., 2018; Gholami et al., 2021). Both weights and activations are quantized in the forward pass, while gradients remain in full precision for backpropagation. One critical component of QAT is the calibration of activation outputs to ensure they fall within the quantization clipping range (e.g., -128 to 127 for int8). This further reduces the risk of overflow by keeping activations within the representable range of the quantization format.

5 Vision Tasks

We describe how the proposed three design principles are applied to two on-device vision tasks, *i.e*., image classification and object detection. The details of experimental results are provided in Section 7.

5.1 Image Classification

Image classification is a fundamental computer vision task that predicts predefined class labels for input images, serving as the foundation for various high-level visual recognition applications. We apply the proposed methods to both the large-scale ImageNet (Deng et al., 2009) dataset for general image classification and the Visual Wake Words (VWW) (Chowdhery et al., 2019) dataset, which is tailored for low-complexity recognition tasks in resource-constrained environments. VWW focuses on binary classification problems like person detection, making it ideal for evaluating network architectures under extreme memory limitations.

While MobileNetV2's Inverted Residual and Linear Bottleneck structures have been widely adopted in lightweight networks like EfficientNet (Tan and Le, 2019) and MnasNet (Tan et al., 2019) due to their computational efficiency, we observe that channel expansion within these structures leads to substantial activation memory usage that could impede applications on low-end devices. The proposed three design principles-input segmentation, patch tunneling, and bottleneck reordering-directly address this limitation by controlling peak memory consumption while preserving the representational power of the original architecture. The resulting network demonstrates remarkable memory efficiency, requiring only 253KB for ImageNet and 70KB for VWW in FP32 precision, while maintaining competitive classification accuracy suitable for deployment on severely resource-constrained low-end devices.

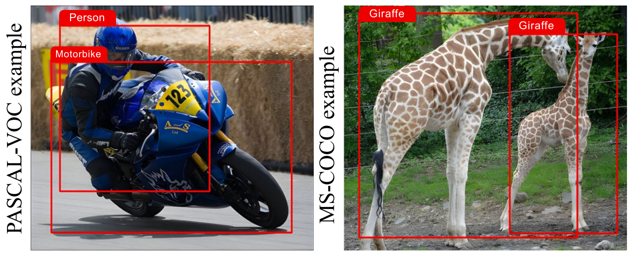

Fig. 7 Examples of object detection performed by the proposed network: (Left) PASCAL-VOC (Everingham et al., 2015) and (Right) MS-COCO (Lin et al., 2015)

5.2 Object Detection

Object detection represents a more complex vision task that requires both localization and classification of multiple objects within a single image. In modern object detection frameworks, both YOLO (Redmon et al., 2016) and SSD (Liu et al., 2016) architectures achieve efficient detection through their inherently low computational complexity. SSDLite (Sandler et al., 2018) extends the standard SSD approach by replacing conventional box prior heads and associated convolutional operations across different feature map sizes with depthwise separable convolutions, significantly enhancing computational efficiency. The proposed bottleneck reordering technique further optimizes memory usage within the SSDLite (Sandler et al., 2018) framework by retaining only a single expanded channel (*i.e*., c_{out}) from the bottleneck layer at any given time. Since SSDLite already employs depthwise separable convolutions, applying bottleneck reordering allows us to compute the detection head operations one channel at a time, rather than storing all expanded channels simultaneously. This modification considerably reduces the peak memory footprint while maintaining detection performance.

We implement the proposed methods on two standard object detection benchmarks: PASCAL VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015). PASCAL VOC contains 20 object categories across challenging real-world scenes and has been widely used to assess detector performance in resource-constrained settings. MS-COCO presents a more demanding benchmark with 80 categories and greater variation in object scale, occlusion, and environmental conditions, providing a comprehensive evaluation of detection capabilities. In our implementation, we observe that peak memory consumption occurs at different points depending on the dataset and input resolution: at the first extra layer for VOC 130×130 inputs, at the second bottleneck block for VOC 224×224 inputs, and at the first convolution layer in the classification header for COCO tasks. The resulting class confidence scores and box locations being deloaded from

SRAM for storage, further optimizing memory usage during inference.

6 Audio Tasks

The proposed memory-efficient CNN is also applicable to CNN-based audio classifiers, enabling audio classification on low-end devices. Audio data is inherently represented as a one-dimensional signal determined by the product of the audio duration and sample rate. However, by applying iterative Short-Time Fourier Transforms (STFTs), the audio signal can be converted into a two-dimensional representation-namely, a spectrogram. This spectrogram can be treated as a single-channel image, enabling the application of Convolutional Neural Networks (CNNs) for audio classification.

6.1 Audio Classification with CNNs

CNN-based Audio Classification. Spectrograms combined with CNNs have demonstrated promising performance in audio classification, and numerous CNN-based approaches have been extensively explored in the literature (Chaturvedi et al., 2022; Palanisamy et al., 2020; Hershey et al., 2017). Several studies have specifically targeted audio classification on embedded devices. For instance, EdgeL^3 (Kumari et al., 2019) proposes compressing a self-supervised L^3 network via pruning, fine-tuning, and knowledge distillation into a 0.814 MB network, achieving minimal performance degradation on edge devices. However, EdgeL^3 requires approximately 12 MB of activation memory in its early layers, making it unsuitable for low-end targets (e.g., STMicroelectronics (2024a, b)). The ACDNet (Mohaimenuzzaman et al., 2023) employs CNNs for acoustic classification using multiple layers of a Convolution-based Spectral Feature Extraction Block (SFEB) and a Temporal Feature Extraction Block (TFEB). To reduce model size for low-end devices, ACDNet combines unstructured and structured pruning (hybrid pruning) with quantization. Notably, ACDNet bypasses the spectrogram conversion process by directly applying 1D convolutions on raw audio data. However, this approach requires activation memory proportional to the input audio length (*i.e.*, L_{in}), leading to increased computational overhead on low-end devices.

Mel-Spectrogram. A single Fourier Transform is applied to a predefined window length of audio to compute the intensity of signals across different frequency bins at a specific time step. Repeating this process with a hop length between windows produces a spectrogram. The Mel-Spectrogram is obtained by transforming the spectrogram into the perceptually relevant Mel scale (Logan et al., 2000; Wyse, 2017). Given an audio input of length L_{in} without padding, hop length L_h, and window length L_w, the spectrogram shape (Time T, Frequency F) is calculated as:

$$(T, F) = \left(\left\lceil \frac{L_{\text{in}} - L_w}{L_h} \right\rceil + 1, \frac{L_w}{2} + 1 \right) \tag{4}$$

After conversion to the Mel scale, the frequency dimension is reduced to a predefined number of Mel bins, typically 64.

Table 1 The shape and size of variables for Mel-Spectrogram extraction. The audio data and Spectrogram (highlighted rows) have memory sizes that exceed the capacity of low-end devices (*e.g.*, STMicroelectronics (2024a, b))

Variable	Shape	Size
Audio Data	160,000	625KB
Spectrogram	200 × 513	401KB
Mel Filter	513 × 64	128KB
Mel-Spectrogram	200 × 64	50KB

Memory Considerations during Feature Extraction.

In a system with less than 256KB of SRAM, it is not feasible to extract a Mel-Spectrogram from the entire audio sample at once. As shown in Table 1, processing the complete audio sample exceeds the available memory constraints. To address this limitation, we propose an iterative approach that sequentially loads small segments of the audio data, each corresponding to the window length required for processing, and constructs the Mel-Spectrogram column by column. Specifically, the algorithm first loads a segment of audio data of length L_{in} and applies the real FFT operation to compute one column of the spectrogram. Subsequently, the Mel scale conversion is performed via vector-matrix multiplication, resulting in one column of the Mel-Spectrogram. Since this process does not store the entire audio data or the intermediate spectrogram in memory simultaneously, it only requires memory to store the Mel filter and the Mel-Spectrogram. Consequently, the complete Mel-Spectrogram can be extracted while maintaining memory usage below 200 KB. Figure 8 illustrates this process.

7 Experiment

We implement a set of CNNs based on the proposed three design principles in C language and deploy them onto STM32 development boards (STMicroelectronics, 2024b), featuring an ARM Cortex-M7 processor and 256 to 4,096 KB of SRAM. The networks are evaluated on two key tasks: image classification (Deng et al., 2009; Chowdhery et al., 2019) and object detection (Everingham et al., 2015; Lin et al., 2015). For efficient real FFT (RFFT) operations, we use the built-in DSP module on the STM32H7 MCU development

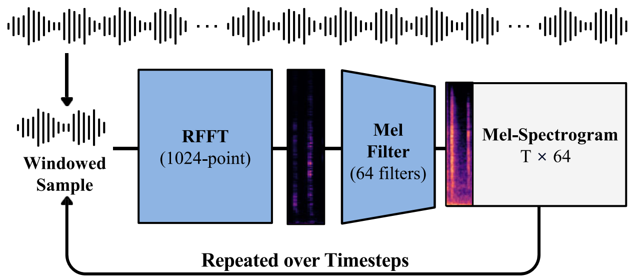

Fig. 8 Our method avoids storing the entire audio or intermediate spectrogram in memory simultaneously. Each 1024-point windowed audio waveform undergoes an RFFT, followed by a 64-filter Mel filterbank, producing a single 64-dimensional frame. By repeating this process over T timesteps, we construct a $T \times 64$ Mel-spectrogram

board, which is optimized for high-speed signal processing. The ARM Cortex-M7 processor (Arm Holdings plc, 2014) features a Floating Point Unit (FPU) and a specialized DSP instruction set, enabling rapid execution of RFFT algorithms. This hardware-based acceleration significantly reduces computational complexity, memory usage, and power consumption compared to software implementations-critical factors for real-time frequency analysis on low-end devices.

Datasets. For image classification, we evaluate the proposed networks with ImageNet (Deng et al., 2009) and VWW (Visual Wake Words) (Chowdhery et al., 2019). For object detection, we use PASCAL-VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015) for evaluation. For audio classification, we use ESC-50 (Piczak, 2015) and Urban-Sound8k (Salamon et al., 2014).

Network Architecture. By using the three proposed design principles (Section 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 (Liu et al., 2016) structure after bottleneck blocks.

Baselines. We compare the memory usage and model performance of the proposed networks for vision tasks with the two baselines, *i.e.*, MobileNet (Sandler et al., 2018) and MCUNet (Lin et al., 2020, 2021). For audio classification tasks, we compared MobileNet and other two baselines, *i.e.*, EdgeL^3 (Kumari et al., 2019) and ACDNet Mohaimenuzzaman et al. (2023).

7.1 Image Classification

Memory Usage and Model Performance. Tables 2 and 3 show the proposed networks with different memory budgets (usages) and classification accuracy on ImageNet (Deng et al., 2009) and VWW (Chowdhery et al., 2019), compared with the two baselines, *i.e.*, MobileNet (Sandler et al., 2018) and MCUNet (Lin et al., 2020, 2021). In Table 2, 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 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 Table 3, 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. Table 4 shows the peak memory usage and classification accuracy of ImageNet (Deng et al., 2009) with different numbers of expansion ratio, *i.e.*, t, of the pointwise (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 pointwise 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. Table 5 shows the peak memory usage and classification accuracy of ImageNet (Deng et al., 2009) and VWW (Chowdhery et al., 2019) 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. Table 6 shows the peak memory usage and classification accuracy of ImageNet (Deng et al., 2009) and VWW (Chowdhery et al., 2019) over different 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 (Deng et al., 2009)), demonstrating the proposed ‘input segmentation’ is essential to achieve competitive performance.

Margin Size. Table 7 shows the peak memory usage and classification accuracy of ImageNet (Deng et al., 2009) and VWW (Chowdhery et al., 2019) over different sizes of mar-

Table 2 The peak memory usage and classification accuracy (top-1 and top-5) of the proposed network on ImageNet (Deng et al., 2009), compared with MobileNet (Sandler et al., 2018) and MCUNet (Lin et al., 2020, 2021)

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

Table 3 The peak memory usage and classification accuracy of the proposed network, MobileNet (Sandler et al., 2018), and MCUNet (Lin et al., 2020, 2021) on VWW (Visual Wake Words) (Chowdhery et al., 2019)

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 4 The peak memory usage and classification accuracy on ImageNet (Deng et al., 2009) 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

gins, *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 Equation (1), the proper margin values should be chosen based on the memory budget.

7.2 Object Detection

Memory Usage and Model Performance. Tables 8 and 9 show the proposed networks with different memory budgets and mAP (mean Average Precision) on PASCAL-VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015), compared with MobileNet (Sandler et al., 2018) and MCUNet (Lin et al., 2020, 2021). For MS-COCO (Lin et al., 2015), all networks are trained with trainval35k (*i.e.*, a union of train data and a subset of validation data with a size of 35k in the COCO 2014 dataset) and evaluated on minival5k in the MS-COCO 2014 dataset. Similar to image classification, the proposed network takes the lowest memory among all for Pascal-VOC (Everingham et al., 2015) in Table 8, *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. For MS-COCO (Lin et al., 2015) in Table 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%),

Table 5 The experiments on ImageNet (Deng et al., 2009) and VWW (Visual Wake Words) (Chowdhery et al., 2019) 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	62.32/61.11	84.38
VWW	4	70/18	fp32/int8	78.97/78.46	N/A

verifying that the proposed design principles also effectively apply to object detection. Figure 7 shows examples of object detection on PASCAL-VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015).

Channel Output. Table 9 shows the peak memory usage and mAP (mean Average Precision) on PASCAL-VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015) 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 63.4% in PASCAL-VOC (Everingham et al., 2015), 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.

7.3 Audio Classification

Memory Usage and Model Performance. Tables 10 and 11 show the proposed networks with different memory budgets and K-fold validation accuracy on ESC-50 (Piczak, 2015) and UrbanSound8k (Salamon et al., 2014), compared with MobileNet (Sandler et al., 2018) , ACDNet (Mohaimenuzzaman et al., 2023), and EdgeL^3 (Kumari et al., 2019). In ESC-50, there are 5 predefined folds, and in UrbanSound8K, there are 10 predefined folds. All K-fold validation accuracies were measured by averaging the validation accuracy across each fold. Table 10 demonstrates that the proposed network takes the lowest memory among all for ESC-50 (Piczak, 2015) in Table 10, *i.e.*, 45 KB with quantization (int8), 32 and 6.7x smaller than MobileNet (1,434 KB) and Micro-ACDNet (303 KB), respectively, achieving 78.8% accuracy, showing that it offers competitive performance while using significantly less memory than other baselines. For UrbanSound8k in Table 10, the proposed network also takes much smaller memory compared to MobileNet, *e.g.*, 42 KB (26x smaller) while having higher accuracy (79.1%), ensuring the proposed network effectively applied not only on vision tasks but also audio tasks. We experimented with different input audio lengths (*i.e.*, L_{input}) to verify our sequantial Mel-Spectrogram extraction strategy. For ESC-50 (Piczak, 2015), we tested 160,000 to 96,000; for UrbanSound8K, we tested 128,000 to 64,000 samples, all at a 32kHz sampling rate. Thanks to the strategy, the peak memory remains relatively stable across different input lengths, *e.g.*, 45 to 40KB for ESC-50 (Piczak, 2015), and 42 to 37KB for UrbanSound8K (Salamon et al., 2014). However, performance drops significantly as L_{input} decreases, *e.g.*, 80.0% to 69.6% on ESC-50 and 80.4% to 75.1% on UrbanSound8K (Salamon et al., 2014). This degradation is mainly due to input audio being increasingly trimmed. In contrast to other approaches that aggressively downsample or truncate audio, our model preserves richer information by avoiding such operations, leading to more robust performance.

Channel Output. Table 11 shows the peak memory usage and K-fold validation accuracy on ESC-50 Piczak (2015) and UrbanSound8k Salamon et al. (2014) 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 accuracy increase, *e.g.*, from 74.6% to 80.0% in ESC-50 Piczak (2015), along with a slight increase in the convolution kernel size. Similar to the vision tasks, the proposed network provides a flexible trade-off between accuracy and the size of the kernel for audio classification, while keeping the peak memory requirement constant.

7.4 Ablation on Peak-Memory Aware Quantization

Table 12 shows the ablation experiments on ImageNet (Deng et al., 2009), to analyze the effect of proposed Peak-Memory Aware Quantization techniques (Section 4), *i.e.*, QBRI and ERA, on peak memory reduction and classification accuracy. Starting from the fp32 baseline, our model requires 254KB peak memory with 63.84% top-1 accuracy. Applying weight quantization to int8 substantially reduces memory to 159KB, while maintaining accuracy (63.42%). Introducing the proposed QBRI layer further decreases peak memory usage to 142 KB without any accuracy loss, demonstrating the effectiveness of eliminating redundant fp32 activations in BatchNorm and ReLU6 operations. Finally, applying ERA to the point-wise reduction layers achieves the most significant memory reduction, bringing peak memory down to only 63 KB-an almost 4× reduction compared to the fp32 baseline. This comes at a minor accuracy drop to 61.58%, which is acceptable given the extreme memory savings. Overall, these results validate that QBRI and ERA complement each other by systematically addressing the memory bottlenecks in quantized inference, enabling extremely memory efficient deployment while preserving competitive accuracy.

8 Limitations and Discussions

Network Size. Although the proposed network takes much smaller memory when compared to state-of-the-art memory-efficient MCUNet (Lin et al., 2020, 2021) and MobileNet (Sandler et al., 2018), *e.g.*, 89x and 3.1x smaller for ImageNet (Deng et al., 2009), 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

Table 6 The experiments on ImageNet (Deng et al., 2009) and VWW (Visual Wake Words) (Chowdhery et al., 2019) 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.

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

Table 7 The experiments on ImageNet (Deng et al., 2009) and VWW (Visual Wake Words) (Chowdhery et al., 2019) by varying the patch margin (*i.e.*, m_h and m_w)

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

Table 8 The peak memory usage and mAP (mean Average Precision) of the proposed network, MobileNet (Sandler et al., 2018), and MCUNet (Lin et al., 2020, 2021) on PASCAL-VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015)

Model Name	Pascal-VOC				MS-COCO			
	Mem(KB)	Quant	Res	mAP(%)	Mem(KB)	Quant	Res	**mAP$^{\mathbf{val}}$**(%)
Proposed network	259/65	fp32/int8	130	51.8/50.7	446/112	fp32/int8	130	22.0/21.2
	702/176	fp32/int8	224	63.4/62.1	1,210/303	fp32/int8	224	28.9/27.2
	1,260/315	fp32/int8	300	67.0/66.3	1,719/430	fp32/int8	224	30.0/28.0
MCUNetV1	466	int8	224	51.4	N/A	N/A	N/A	N/A
MCUNetV2	438	int8	224	68.3	N/A	N/A	N/A	N/A
	247	int8	224	64.6	N/A	N/A	N/A	N/A
MobileNetV2	10,080	fp32	300	68.6	10,080	fp32	300	31.8

Table 9 The object detection on PASCAL-VOC (Everingham et al., 2015) and MS-COCO (Lin et al., 2015) with different expansion ratios (*i.e.*, t) in the point-wise (expansion) conv within the bottleneck

t	Pascal-VOC			MS-COCO		
	Mem(KB)	Quant	mAP(%)	Mem(KB)	Quant	**mAP$^{\mathbf{val}}$**(%)
2	702/176	fp32/int8	52.3/46.6	1719/430	fp32/int8	22.4/13.3
4	702/176	fp32/int8	57.3/53.1	1719/430	fp32/int8	26.6/22.9
6	702/176	fp32/int8	61.5/60.2	1719/430	fp32/int8	28.3/25.5
8	702/176	fp32/int8	63.4/62.1	1719/430	fp32/int8	30.0/28.0

Table 10 The peak memory usage and accuracy of the proposed network, MobileNet (Sandler et al., 2018), EdgeL^3 (Kumari et al., 2019), and ACDNet (Mohaimenuzzaman et al., 2023) on ESC-50 (Piczak, 2015) and UrbanSound8k (Salamon et al., 2014). Model names with asterisk (*) are reproduced by us

Model Name	ESC-50				UrbanSound8k			
	Mem(KB)	Quant	$\mathbf{L}_{\text{input}}$	Acc.(%)	Mem(KB)	Quant	$\mathbf{L}_{\text{input}}$	Acc.(%)
Proposed network	178/45	fp32/int8	160,000	80.0/78.8	168/42	fp32/int8	128,000	80.4/79.1
	168/42	fp32/int8	128,000	73.5/73.2	158/40	fp32/int8	96,000	79.0/77.8
	158/40	fp32/int8	96,000	69.6/68.4	148/37	fp32/int8	64,000	75.1/75.1
EdgeL^3	≈12,000	fp32	48,000	72.3	≈12,000	fp32	48,000	73.9
Micro-ACDNet	1,003/303	fp32/int8	30,225	83.7/71.0	1,003	fp32	30,225	78.3
MobileNetV2*	1,434	fp32	160,000	72.2	1,120	fp32	128,000	77.3

Table 11 The audio classification on ESC-50 (Piczak, 2015) and UrbanSound8k (Salamon et al., 2014) with different expansion ratios (*i.e.*, t) in the point-wise (expansion) conv within the bottleneck

t	ESC-50			UrbanSound8k		
	Mem(KB)	Quant	Acc.(%)	Mem(KB)	Quant	Acc.(%)
2	178/45	fp32/int8	74.6/74.4	168/42	fp32/int8	78.2/76.6
4	178/45	fp32/int8	77.7/76.9	168/42	fp32/int8	79.1/77.3
6	178/45	fp32/int8	78.6/77.4	168/42	fp32/int8	79.6/78.5
8	178/45	fp32/int8	80.0/78.8	168/42	fp32/int8	80.4/79.1

Table 12 The ablation experiments on ImageNet (Deng et al., 2009) for two proposed approaches, Quantized BatchNorm + ReLU6 Integration (QBRI) and Element-wise Requantization Accumulation (ERA), on peak memory Usage and top-1 classificaiton accuracy

Setting	Peak-mem	Quant	Top-1 Accuracy
Baseline	254KB	fp32	63.84%
+ Weight Quantization	159KB	int8	63.42%
+ QBRI	142KB	int8	63.42%
+ ERA	**63KB**	int8	61.58%

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 and audio tasks, *i.e.*, image classification, object detection, and audio classification, demonstrating its effectiveness. For a more thorough evaluation, however, additional experiments on different tasks and various datasets, such as image segmentation (Cordts et al., 2016; Zhou et al., 2018), super resolution (Magnitskaya et al., 2017; Dong et al., 2015), and image generation (Liu et al., 2015; Yu et al., 2016), should be conducted. We plan to apply the proposed methods to various tasks and investigate their applicability and limitations.

9 Conclusion

We introduce a highly memory-efficient and bottleneck-based 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. For efficient deployment of our design, we adopt quantization technique named 'peak memory aware quantization' which ensure the intended peak memory reduction in activation level. We implement the network on an actual low-end device (STM32H7 (STMicroelectronics, 2024b)), demonstrating that these design principles enable memory-aware CNNs for various vision and audio tasks, such as memory-efficient image classification on ImageNet (Deng et al., 2009) using 63 KB (61.58% top-1 accuracy), object detection on PASCAL-VOC (Everingham et al., 2015) using 65 KB (50.7% mAP), and audio classification on Urbansound8k (Salamon et al., 2014) using 42 KB (79.1% accuracy).

Funding Open Access funding enabled and organized by Ulsan National Institute of Science and Technology (UNIST)

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Arm Holdings plc (2014). ARM Coretex-M7. https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m7.

Bigas, M., Cabruja, E., Forest, J., & Salvi, J. (2006). Review of cmos image sensors. *Microelectronics journal,37*(5), 433–451.

Chaturvedi, A., Yadav, S. A., Salman, H. M., Goyal, H. R., Gebregziabher, H., & Rao, A. K. (2022). Classification of sound using convolutional neural networks. (pp. 1015–1019)

Choi, J., Shin, J., Kang, D., & Park, D.-S. (2015). Always-on cmos image sensor for mobile and wearable devices. *IEEE Journal of Solid-State Circuits,51*(1), 130–140.

Chowdhery, A., Warden, P., Shlens, J., Howard, A., & Rhodes, R. (2019). Visual wake words dataset.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). *Imagenet: A large-scale hierarchical image database* (pp. 248–255). Ieee.

Dhar, S., Guo, J., Liu, J., Tripathi, S., Kurup, U., & Shah, M. (2021). A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Transactions on Internet of Things,2*(3), 1–49.

Dong, C., Loy, C. C., He, K., & Tang, X. (2015). Image super-resolution using deep convolutional networks.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research,20*(1), 1997–2017.

Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision,111*(1), 98–136.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2021). A survey of quantization methods for efficient neural network inference.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2022). A survey of quantization methods for efficient neural network inference. *Low-Power Computer Vision* (pp. 291–326). Chapman and Hall/CRC.

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint* arXiv:1510.00149.

He, X., Zhao, K., & Chu, X. (2021). Automl: A survey of the state-of-the-art. *Knowledge-Based Systems,212*, Article 106622.

Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R. J., & Wilson, K. (2017). Cnn architectures for large-scale audio classification. (pp. 131–135)

Huang, K., Ni, B., & Yang, X. (2019). Efficient quantization for neural networks with binary weights and low bitwidth activations. *In Proceedings of the AAAI Conference on Artificial Intelligence,33*, 3854–3861.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference.

Kumari, S., Roy, D., Cartwright, M., Bello, J. P., & Arora, A. (2019). EdgeΓ3: Compressing Γ3-net for mote scale urban noise monitoring. (pp. 877–884)

Liang, T., Glossner, J., Wang, L., Shi, S., & Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing,461*, 370–403.

Liberis, E., Dudziak, Ł., & Lane, N. D. (2021). *mu*nas: Constrained neural architecture search for microcontrollers. (pp. 70–79)

Lin, J., Chen, W.-M., Cai, H., Gan, C., & Han, S. (2021). Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. *arXiv preprint* arXiv:2110.15352.

Lin, J., Chen, W.-M., Lin, Y., Gan, C., Han, S., et al. (2020). Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems,33*, 11711–11722.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., & Han, S. (2024). Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems,6*, 87–100.

Lin, J., Zhu, L., Chen, W.-M., Wang, W.-C., Gan, C., & Han, S. (2024b). On-device training under 256kb memory.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). Microsoft coco: Common objects in context.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer.

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild.

Logan, B., et al. (2000). Mel frequency cepstral coefficients for music modeling. *Ismir* (Vol. 270, pp. 1–11). MA: Plymouth.

Magnitskaya, M., Chtchelkatchev, N., Tsvyashchenko, A., Salamatin, D., Lepeshkin, S., Fomicheva, L., & Budzyński, M. (2017). Electron and phonon properties of noncentrosymmetric rhge from ab initio calculations.

Mohaimenuzzaman, M., Bergmeir, C., West, I., & Meyer, B. (2023). Environmental sound classification on the edge: A pipeline for deep acoustic networks on extremely resource-constrained devices. *Pattern Recognition,133*, Article 109025.

Palanisamy, K., Singhania, D., & Yao, A. (2020). Rethinking cnn models for audio classification.

Piczak, K. J. (2015). *Esc: Dataset for environmental sound classification.* New York, NY, USA: Association for Computing Machinery.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection.

Salamon, J., Jacoby, C., & Bello, J. P. (2014). A dataset and taxonomy for urban sound research. *22nd ACM International Conference on Multimedia (ACM-MM'14)* (pp. 1041–1044). FL, USA: Orlando.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

STMicroelectronics (2024a). STM32F. https://www.st.com/en/microcontrollers-microprocessors/stm32f412.html.

STMicroelectronics (2024b). STM32H7. https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile.

Tan, M., & Le, Q. (2019). In K. Chaudhuri & R. Salakhutdinov (Eds.), *EfficientNet: Rethinking model scaling for convolutional neural networks* (pp. 6105–6114). PMLR.

Vadera, S., & Ameen, S. (2022). Methods for pruning deep neural networks. *IEEE Access,10*, 63280–63300.

Wei, L., Ma, Z., Yang, C., & Yao, Q. (2024). Advances in the neural network quantization: A comprehensive review. *Applied Sciences,14*(17), 7445.

Wyse, L. (2017). Audio spectrogram representations for processing with convolutional neural networks. *arXiv preprint* arXiv:1706.09559.

Xu, S., Huang, A., Chen, L., & Zhang, B. (2020). *Convolutional neural network pruning: A survey* (pp. 7458–7463). IEEE.

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., & Xiao, J. (2016). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop.

Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., & Torralba, A. (2018). Semantic understanding of scenes through the ade20k dataset.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.