Efficient On-device Transfer Learning using Activation Memory Reduction

Amin Yoosefi[*,†], Hamid Mousavi[*], Masoud Daneshtalab[*,§], and Mehdi Kargahi[†]

[*]*Innovation, Design and Engineering, Mälardalen University*, Västerås, Sweden
[†]*School of Electrical and Computer Engineering, University of Tehran*, Tehran, Iran
[§] *School of Computer Systems, Tallinn University of Technology*, Tallinn, Estonia
Emails: a.yoosefi@ut.ac.ir, {seyedhamidreza.mousavi, masoud.daneshtalab}@mdu.se, kargahi@ut.ac.ir

***Abstract*—On-device transfer learning suggests fine-tuning pre-trained neural networks on new input data directly on edge devices. The memory limitation of edge devices necessitates the use of memory-efficient fine-tuning methods to enable on-device transfer learning. Fine-tuning involves two primary phases: the forward-pass phase and the backward-pass phase. The forward-pass phase generates output activations for each layer given the input activations coming from the preceding layer, and the backward-pass phase computes gradients and updates the parameters accordingly. Although the forward-pass phase demands a temporary memory to store the input and output activations of a layer when conducting its inference, the backward-pass phase may call for storing the output activations from all layers to compute gradients. This fact introduces the memory cost of the backward-pass phase as the main contributor to the huge training memory demands of deep neural networks (DNNs), which has been the focus of many previous research studies. However, little attention has been made to how the temporary activation memory involved in the forward-pass phase may also act as the memory bottleneck, which is the main focus of this paper. This paper aims to mitigate this memory bottleneck by pruning unimportant channels from layers that require significant temporary activation memory. These layers are initially identified using a memory usage analysis. Our approach not only reduces the memory footprint for temporary activation memory in the forward-pass phase but also reduces the memory requirements for the backward-pass phase of training. Experimental results demonstrate how the proposed method effectively reduces the memory footprint for on-device transfer learning.**

***Index Terms*—On-device transfer learning, memory efficiency, activation memory, deep neural networks (DNNs)**

I. Introduction

The growing use of IoT devices has led to a continuous stream of new data being generated. This new data can assist in adapting pre-trained neural networks to new task domains, introducing the transfer learning paradigm. [1], [2]. With transfer learning, a pre-trained network is fine-tuned from a primary task to adapt itself to a new task [3], [4]. On-device transfer learning eliminates the need to move data to the cloud and preserves data privacy.

In transfer learning, there are two primary approaches to adjusting a pre-trained network during fine-tuning: 1) utilizing the pre-trained network as a fixed feature extractor and solely fine-tuning the last layer of the network [5], [6], and 2) fine-tuning either the entire network or some specific parts of it [7], [8]. While the first approach is memory-efficient since it does not require storing intermediate activations during fine-tuning, its performance is constrained by a limited transfer capacity [8]. The second approach offers good performance but is not memory-friendly for on-device training as it necessitates storing all intermediate activation values to calculate gradients during fine-tuning.

Network expansion [1], [9]–[11], layer selection [11], [12], and gradient quantization and filtering [2], [13]–[15] methods have been proposed to reduce the memory cost of gradient computations. Network expansion methods employ a lightweight neural network alongside the fixed main backbone. The focus is on fine-tuning the parameters of this smaller network, which helps reduce the memory cost of gradient computations. While network expansion methods effectively reduce the gradient computation memory, they do not address the temporary activation memory used during the forward-pass phase in the fine-tuning process. Layer selection methods aim to identify the crucial layers within the network and only fine-tune those layers. Although layer selection methods reduce the memory cost of gradient computations by storing activations only for some specific layers, the selected layers themselves tend to be computationally demanding. Gradient quantization methods decrease the number of arithmetic operations involved in gradient computation and employ lower-bit representations to minimize memory usage. Furthermore, gradient filtering methods enhance gradient quantization by reducing both memory usage and computational costs. Nevertheless, these methods fail to address the issue of temporary activation memory and lack support for widely used deep neural network (DNN) frameworks like PyTorch [16] and TensorFlow [17].

The primary limitation in previous research studies lies in the temporary activation memory utilized during the forward-pass phase of the fine-tuning algorithm, which poses a significant bottleneck. To tackle this issue, we present a method that seamlessly integrates with other approaches to minimize the temporary activation memory and reduce the overall memory footprint required for transfer learning. More specifically, we

Preprint submitted to International Conference on Fog and Mobile Edge Computing (FMEC 2023)

decrease the temporary activation memory by utilizing channel pruning techniques. First, we identify the layers contributing to increased memory demand during on-device training. Next, we identify and eliminate insignificant channels from activation feature maps, effectively pruning them from the backbone network. To restore the accuracy of the pruned backbone network, we conduct retraining by focusing on the remaining weights within the network. Ultimately, we employ this lightweight backbone network in network expansion methods to minimize the gradient computation memory and facilitate the utilization of transfer learning.

Our method is designed to target both the temporary activation memory during the forward-pass phase and the gradient computation memory involved in the backward-pass phase and reduce them jointly.

Our main contributions are summarized as follows:

1) Investigating the bottleneck caused by temporary activation memory and identifying the layers that contribute to this bottleneck.
2) Incorporating channel pruning into bottleneck layers of the backbone network to decrease temporary activation memory and combining it with network expansion to decrease the memory footprint.
3) Evaluating the effectiveness of the proposed method on datasets such as Cifar-10 and experimentally demonstrating how it successfully reduces the overall memory requirement for on-device transfer learning.

II. Related Work

On-device training methods confront two main challenges: computational and memory constraints inherent in the backpropagation algorithm to compute the gradients. We can categorize on-device training methods into four distinct groups based on the specific solutions devised to tackle these issues.

A. Network Expansion

This solution involves introducing a lightweight neural network alongside the backbone network, where only the parameters of this simplified network are updated. The final decision is made by leveraging the predictions of both networks during the inference phase [1], [9]. TinyTL [9] introduced a lightweight module that can be incorporated alongside the frozen backbone. They updated only the parameters of this smaller module as well as the biases of the original backbone network. In this approach, the lightweight module is developed separately from the backbone network, and there is no direct connection between the two. In order to maintain the linkage between the backbone and lightweight networks, Rep-Net [1] employed feature reprogramming as a means to construct the lightweight network. In particular, Rep-Net introduced an activation connector that serves as a link between the backbone and reprogramming network, enabling mutual benefits for both networks. Nevertheless, these techniques reduce the memory demands during training by simply augmenting the fixed backbone network with a lightweight module without making any alterations to the backbone itself. Furthermore, they have the ability to decrease only the overall gradient computation memory while keeping the temporary activation memory unchanged. In this research paper, we employ channel pruning to modify the backbone network and achieve a reduction in memory usage. This approach effectively lowers both the temporary activation memory and the gradient computation memory.

B. Layer selection

Various methodologies are employed by these techniques to choose specific layers of the network for fine-tuning [11], [12]. More specifically, [12] introduced an important metric to determine the selection of layers, whereas [11] employed an evolutionary algorithm for the same purpose. Nevertheless, the layers chosen by these methods tend to be computationally intensive and do not provide substantial reductions in memory usage during training. Our approach effectively complements these methods by utilizing channel pruning to reduce the computational load of the selected layers.

C. Gradient quantization and Filtering

Gradient quantization has the potential to decrease the computational expenses involved in arithmetic operations during the backpropagation algorithm [13]. Nevertheless, this method has limitations in terms of reducing the total number of operations and achieving significant speed improvements. To address this limitation, the gradient filtering method has been introduced as a solution to decrease both memory usage and computational requirements [2]. Nevertheless, this method solely focuses on reducing the overall gradient computation memory and can be combined with our approach to achieve a reduction in the total activation memory.

III. Preliminaries

Transfer learning methods involve fine-tuning a neural network pre-trained on a large-scale dataset like ImageNet [18] on a new dataset. Fine-tuning involves performing a forward-pass phase to calculate the activations and a backward-pass phase to compute gradients and update the parameters. In the following, we analyse the memory usage during fine-tuning.

The maximum activation memory usage, called peak activation memory, during each round of fine-tuning is represented by M^{act} and computed as follows:

$$M^{\text{act}} = \max_{b_i \in \{b_1, b_2, \ldots, b_L\}} \{M^{\text{act}}(b_i)\}, \tag{1}$$

wherein b_i refers to the i^{th} block; L is the total number of blocks within the network; and $M^{\text{act}}(b_i)$ denotes the peak activation memory for the i^{th} block.

The memory footprint in each block comprises two parts: (1) the temporary activation memory, which is defined as the memory amount temporarily required for storing the input and output activations of a layer when conducting its inference in the forward pass phase, and (2) the cumulative gradient memory, which is defined as the amount of memory required for storing activations of all the previous trainable layers. Therefore, the peak activation memory for the i^{th} block (i.e., $M^act(b_i)$) is computed as follows:

$$M^{\text{act}}(b_i) = \max_{l_{i,j} \in \{l_{i,1}, l_{i,2}, \ldots, l_{i,L_i}\}} \{M^{\text{act}}_{\text{tmp}}(l_{i,j}) + M^{\text{act}}_{\text{cugrd}}(l_{i,j})\},$$
$$i = 1, 2, \ldots, L, \quad (2)$$

in which $l_{i,j}$ denotes the j^{th} layer within the i^{th} block; L_i is the total number of layers within the i^{th} block; $M^{\text{act}}_{\text{tmp}}(l_{i,j})$ and $M^{\text{act}}_{\text{cugrd}}(l_{i,j})$ represent the temporary activation memory and cumulative gradient memory for the layer $l_{i,j}$, respectively. The cumulative gradient memory is computed as follows:

$$M^{\text{act}}_{\text{cugrd}}(l_{i,j}) = \sum_{p=l_{1,1}}^{l_{i,j-1}} \text{grd}(p) \times M^{\text{act}}_{\text{grd}}(p), \quad (3)$$

where $M^{\text{act}}_{\text{grd}}(p)$ refers to the memory requirement for storing activations involved in computing the gradients of the layer p; and $\text{grd}(p)$ is an indicator variable returning 1 if the layer p is trainable and returning 0 otherwise. The total memory cost is calculated by considering the memory required for parameters, denoted by M^{par}, and the batch size, denoted by B, as formulated below:

$$M = M^{\text{act}} \times B + M^{\text{par}}. \quad (4)$$

IV. Research Motivation

Equation (2) introduces the temporary activation memory (i.e., $M^{\text{act}}_{\text{tmp}}$) and the cumulative gradient memory (i.e., $M^{\text{act}}_{\text{cugrd}}$) as two contributors to the peak activation memory, as mentioned previously. Previous research endeavors have aimed to decrease the cumulative gradient memory by fine-tuning a smaller network and keeping the backbone model unchanged. However, in spite of decreasing the cumulative gradient memory, the substantial temporary activation memory in specific blocks continues to have a notable impact on the overall peak memory footprint. In other words, our research is motivated by the idea that peak memory will remain large no matter how small the cumulative gradient memory is as long as the temporary activation memory in some blocks is larger than the cumulative gradient memory.

To verify the idea, we utilize the Rep-Net [1] in conjunction with the MobileNetV2 [19] architecture as the backbone. Rep-Net [1] attempts to mitigate the cumulative gradient memory by allowing only a compact network to fine-tune, while the memory-intensive backbone architecture is frozen. Three types of memory costs are measured for each inverted residual block within the architecture: the peak activation memory calculated using (2), the temporary activation memory, and the cumulative gradient memory. The temporary activation memory for each block b_i is determined by calculating the maximum memory usage among the layers within that block, as formulated below:

$$M^{\text{act}}_{\text{tmp}}(b_i) = \max_{l_{i,j} \in \{l_{i,1}, l_{i,2}, \ldots, l_{i,L_i}\}} \{M^{\text{act}}_{\text{tmp}}(l_{i,j})\}, \quad i = 1, 2, \ldots, L. \quad (5)$$

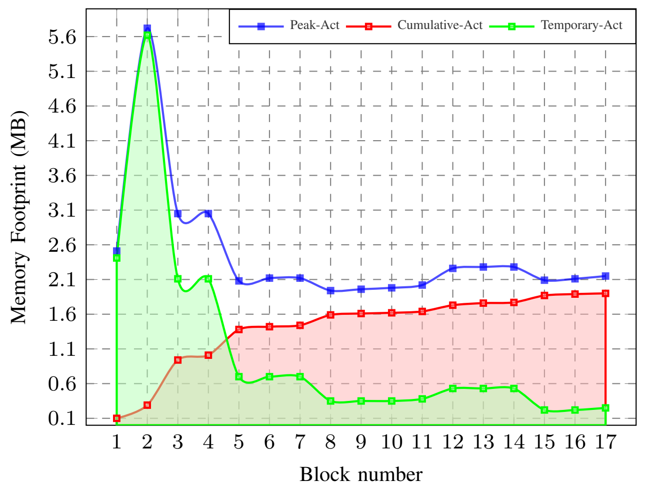

Fig. 1. Illustration of memory cost profiling for Rep-Net [1] with MobileNetV2 as the backbone model.

The cumulative gradient memory for each block is determined by summing up the gradient memory costs of the previous blocks and the current block as follows:

$$M^{\text{act}}_{\text{cugrd}}(b_i) = \sum_{p=l_{1,1}}^{l_{i,j-1}} \text{grd}(p) \times M^{\text{act}}_{\text{grd}}(p), \quad i = 1, 2, \ldots, L. \quad (6)$$

The measured memory costs are depicted in Figure 1. As shown, the second block has the highest peak activation memory among other blocks, and thus it is the block that establishes the peak activation memory for the entire network, which is 5.72 MB. However, the overall cumulative gradient memory obtained by Rep-Net is 1.9 MB. This gap comes from the enormous temporary activation memory within the second block, which is 5.6 MB. This observation implies that no matter how much a memory-reduction technique like Rep-Net decreases the cumulative gradient memory in the backward-pass phase, the temporary activation memory in the forward-pass phase can still act as a memory wall. As a result, memory reduction techniques must affect memory requirements in both phases jointly.

V. Method

Using memory-reduction techniques like Rep-Net [1] to reduce the cumulative gradient memory will not effectively impact the peak activation memory unless the temporary activation memory remains below the reduced amount for the cumulative gradient memory. Therefore, to effectively reduce the overall peak activation memory, we propose adjusting temporary activation memory usage in accordance with the cumulative gradient memory usage. For this, we present a framework to make this adjustment by keeping temporary activation memory usage close to the cumulative gradient memory.

In our framework, blocks causing a memory bottleneck are identified at the first step, and we refer to them as sensitive blocks. These blocks are characterized by having

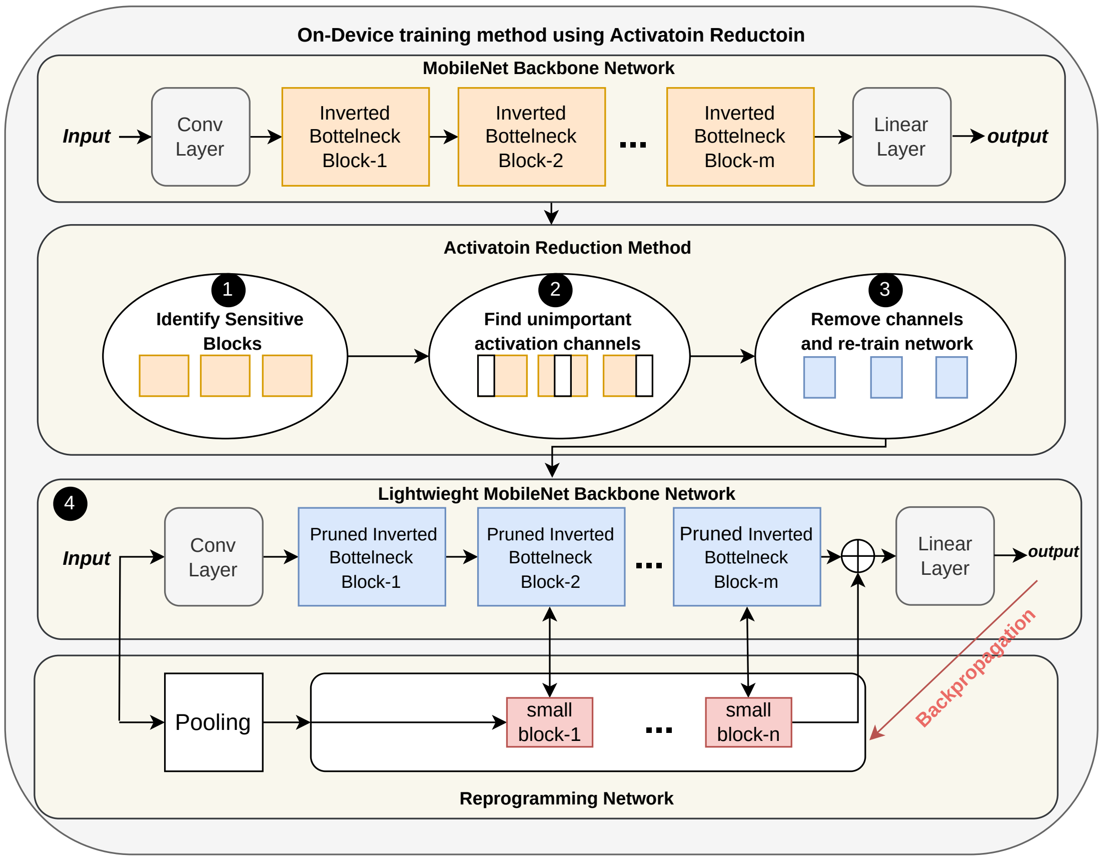
On-Device training method using Activatoin Reductoin
MobileNet Backbone Network
Input
Conv Layer
Inverted Bottelneck Block-1
Inverted Bottelneck Block-2
...
Inverted Bottelneck Block-m
Linear Layer
output
Activatoin Reduction Method
1
Identify Sensitive Blocks
2
Find unimportant activation channels
3
Remove channels and re-train network
4
Lightwieght MobileNet Backbone Network
Input
Conv Layer
Pruned Inverted Bottelneck Block-1
Pruned Inverted Bottelneck Block-2
...
Pruned Inverted Bottelneck Block-m
Linear Layer
output
Backpropagation
Pooling
small block-1
...
small block-n
Reprogramming Network

Fig. 2. The overview of the proposed framework to reduce the overall memory footprint for on-device training.

a high amount of temporary activation memory, larger than the cumulative gradient memory. Next, l_2-norm is used to identify and eliminate unnecessary channels within the identified blocks. In order to improve the performance of pruned blocks, the backbone network is retrained for a few epochs. Ultimately, we utilize the lightweight pruned backbone as the foundational module in Rep-Net [1] to effectively minimize peak activation memory. The overview of our framework is illustrated in Figure 2. In the following, we provide a detailed explanation of each individual step in the framework.

Step1: A lightweight network (such as MobileNetV2) is chosen as the compact backbone for on-device learning. To identify the sensitive blocks within the network, we conduct an analysis of each block using memory cost profiling. For each block, denoted as b_i with $i = 1, 2, \ldots, L$, three metrics are calculated: peak activation memory ($M^{\text{act}}(b_i)$), temporary activation memory ($M^{\text{act}}_{\text{tmp}}(b_i)$), and cumulative gradient memory ($M^{\text{act}}_{\text{cugrd}}(b_i)$), which are computed using equations (2), (5), and (6), respectively. These calculations provide us with detailed information about the memory consumption of each block when using network expansion techniques (such as RepNet [1]) for on-device training. The sensitive blocks are identified by comparing the temporary activation memory with the cumulative gradient memory for each block. If the temporary activation memory is higher than the overall cumulative gradient memory, the block is considered as sensitive. The overal cumulative gradient memory is the cumulative gradient memory for the last block, i.e., $M^{\text{act}}_{\text{cugrd}}(b_L)$. Thus, a block b_i is sensitive if $M^{\text{act}}_{\text{tmp}}(b_i) > M^{\text{act}}_{\text{cugrd}}(b_L)$.

Step2: In this step, l_2-norms of the activation channels within each convolution layer of the identified sensitive blocks are calculated. By analyzing these norms, we can determine the importance of each channel in terms of its contribution to the overall accuracy. Channels with low importance are considered unimportant and are subsequently removed. We carefully select pruning ratios for the identified sensitive blocks to ensure that the temporary activation memory achieved by pruning is lower than the overall cumulative gradient memory before pruning. For the purpose of illustration, in Figure 1, we can apply pruning to blocks 1, 2, 3, and 4 with pruning ratios of 0.3, 0.78, 0.46, and 0.46, respectively, to make the temporary activation memory costs close to the overall cumulative gradient memory (which is 1.9 MB).

Step3: To maintain the performance of the pruned backbone network, the network is retrained for a few epochs after the pruning process. This retraining phase allows the network to adapt and fine-tune its parameters, compensating for any performance loss incurred during channel pruning. By doing so, it is ensured that the network maintains its optimized

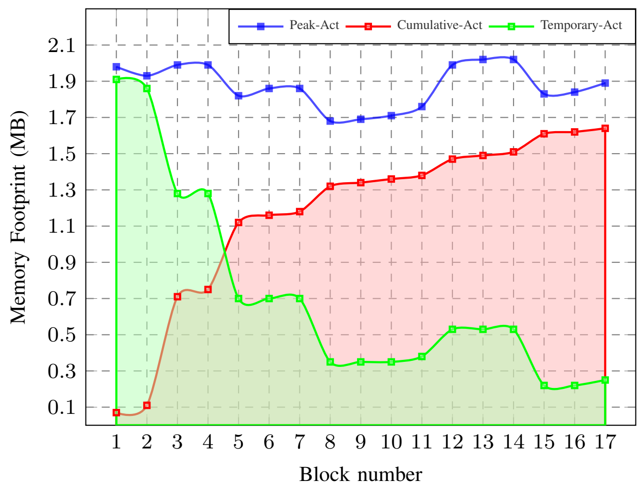

Fig. 3. Illustration of memory cost profiling for Rep-Net [1] with the first, second, third, and fourth blocks from the backbone model pruned with ratios of 0.3, 0.78, 046, and 0.46.

performance while benefiting from the memory reduction achieved through channel pruning.

Step4: The concept of feature reprogramming introduced in RepNet [1] is jointly used with our pruned backbone model to reduce the peak activation memory during on-device training. This involves incorporating a lightweight pruned side-network that operates in parallel with our lightweight backbone network.

VI. Experiments

In this section, we provide the setup for our experiments, and then the experimental results are presented.

A. Setup

1) Datasets and Models: We used MobileNetV2 as the backbone model and followed two training series: (1) Pre-training the backbone model on a large-scale dataset and (2) transfer learning the pre-trained backbone on downstream datasets using Rep-Net. For transfer learning, the pre-trained backbone model was set as frozen, with only Rep-Net being fine-tuned on downstream datasets. Imagenet was used as the large-scale dataset, and CIFAR10 [20], Flowers [21], Pets [22], and CUB [23] were used as the downstream datasets. Rep-Net has six modules corresponding to inverted residual blocks 2, 4, 7, 11, 14, and 17 from the backbone model, following [1].

2) Training details: Two pre-trained versions of the backbone model were obtained, one without pruning and one with pruning. First, the backbone model was trained on ImageNet for 100 epochs, reaching an accuracy of 65.2%. Then, the first four blocks from the pre-trained backbone model were pruned with ratios of 0.3, 0.78, 0.46, and 0.46 in the order of their indices. The pruning ratios were obtained as explained in step 3 of the Method section. The pruned backbone model was then retrained on Imagenet to reach a similar accuracy. In our case, retraining the pruned backbone model reached an accuracy of 64.6%. Fine-tuning involved in transfer learning was performed for five epochs using the Adam optimizer [24] with a batch size of 8 on a single GPU. The initial learning rate for each dataset was adopted from [1], and the cosine schedule was used for learning rate decay.

B. Results

This subsection compares our proposed method with Rep-Net in [1]. The compared methods differ only in the employed backbone model in that Rep-Net [1] uses the pre-trained backbone model with no pruning applied, while our method uses the pre-trained backbone model with blocks 1, 2, 3, and 4 pruned with ratios of 0.3, 0.78, 0.46, and 0.46, respectively. The pruning ratios are chosen so that the temporary activation memory usages of the blocks within the backbone model are adjusted in accordance with the cumulative gradient memory cost, as described in our methodology.

Two series of results are included. The first series emphasizes how our proposed method effectively reduces the peak activation memory cost. The second series highlights the effectiveness of our proposed method in reducing the total memory cost considering overall accuracy.

1) Peak Activation Memory Reduction: Figure 3 illustrates the memory cost profiling results for our proposed method. The following observation is made from Figure 3.

Observation 1. Adjusting the temporary activation memory costs of the blocks in accordance with the cumulative gradient memory cost has made the peak activation memory reduce from 5.72MB in Figure 1 to 2.02MB, reporting a 65% memory reduction in peak activation memory.

This observation demonstrates how the peak activation memory is effectively reduced by applying our proposed methodology, which is adjusting the temporary activation memory usage and the cumulative gradient memory cost in accordance with each other. In other words, using channel pruning to keep the temporary activation memory costs of the blocks close to the cumulative gradient memory cost has provided extensive room for the peak activation memory to reduce.

2) Total Memory Reduction versus Accuracy: Table I reports the total memory costs and accuracy levels for Rep-Net [1] and our proposed method. This table implies the following observation.

Observation 2. Using channel pruning to reduce the temporary activation memory involved in the forward pass phase resulted in a 59% reduction in total memory with an average accuracy loss of 3%.

This observation signifies the predominant role of the temporary activation memory in training memory cost, and it shows how applying channel pruning based on our initial memory cost profiling analysis can lead to more than 50% memory savings. However, the obtained memory saving was subjected to 3% accuracy degradation on average, which can be considered acceptable in that at least around 50% of the weight channels within the initial blocks were pruned in a greedy manner to reduce the memory cost. Pruning the initial blocks may lead to more accuracy drop compared to the others since these blocks typically capture low-level features and provide

TABLE I
REPORTING TRAINING MEMORY COST AND ACCURACY FOR REP-NET [1] AND OUR PROPOSED METHOD.

Method	**Train. Mem.**	**CIFAR10**	**Flower**	**CUB**	**Pets**
Rep-Net [1]	51MB	88.9	78.6	66.5	85.2
Ours	21MB	88.7	74.3	62.0	82.1

a foundation for subsequent blocks to build upon. Pruning these initial blocks might remove crucial information at the early stages of the network, which can have a more significant impact on accuracy compared to pruning later blocks.

VII. CONCLUSION AND FUTURE WORK

In this work, we attempted to address the high memory footprint issue in on-device training algorithms by proposing keeping high temporary activation memory demands close to the cumulative gradient memory cost. We suggested performing a memory cost profiling to identify blocks with high temporary activation memory costs. After identifying these blocks, their high temporary activation memory costs were mitigated by taking a channel-pruning approach. It was then shown how incorporating the backbone model pruned based on our methodology to network expansion methods can further reduce peak activation memory and total memory costs by 65% and 59%, respectively, at the cost of a small drop in accuracy. In future endeavors, we plan to explore the application of quantization methods to further reduce memory usage in our approach. Additionally, we aim to develop and incorporate new learning methods to enhance the accuracy of the model.

REFERENCES

[1] L. Yang, A. S. Rakin, and D. Fan, "Rep-net: Efficient on-device learning via feature reprogramming," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 277–12 286.

[2] Y. Yang, G. Li, and R. Marculescu, "Efficient on-device training via gradient filtering," *arXiv preprint arXiv:2301.00330*, 2023.

[3] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "Spottune: transfer learning through adaptive fine-tuning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4805–4814.

[4] Y. Cui, Y. Song, C. Sun, A. Howard, and S. Belongie, "Large scale fine-grained categorization and domain-specific transfer learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4109–4118.

[5] G. Zhong, S. Yan, K. Huang, Y. Cai, and J. Dong, "Reducing and stretching deep convolutional activation features for accurate image classification," *Cognitive Computation*, vol. 10, pp. 179–186, 2018.

[6] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *arXiv preprint arXiv:1405.3531*, 2014.

[7] M. Sandler, A. Zhmoginov, A. G. Howard, and P. K. Mudrakarta, "Parameter-efficient multi-task and transfer learning," Jun. 13 2023, uS Patent 11,676,008.

[8] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2661–2671.

[9] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce activations, not trainable parameters for efficient on-device learning," *arXiv preprint arXiv:2007.11622*, 2020.

[10] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 711–11 722, 2020.

[11] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," *arXiv preprint arXiv:2206.15472*, 2022.

[12] Y. Lee, A. S. Chen, F. Tajwar, A. Kumar, H. Yao, P. Liang, and C. Finn, "Surgical fine-tuning improves adaptation to distribution shifts," *arXiv preprint arXiv:2210.11466*, 2022.

[13] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, 2017.

[14] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," in *International Conference on Machine Learning*. PMLR, 2018, pp. 560–569.

[15] Z. Hong and C. P. Yue, "Efficient-grad: Efficient training deep convolutional neural networks on edge devices with grad ient optimizations," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 2, pp. 1–24, 2022.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[20] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[21] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, 2008, pp. 722–729.

[22] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3498–3505.

[23] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.