From LLMs to Edge: Parameter-Efficient Fine-Tuning on Edge Devices

Georg Slamanig[*]
Graz University of Technology
Austria
georg.slamanig@student.tugraz.at

Francesco Corti[*]
Graz University of Technology
Austria
francesco.corti@tugraz.at

Olga Saukh
TU Graz / Complexity Science Hub
Austria
saukh@tugraz.at

Abstract

Parameter-efficient fine-tuning (PEFT) methods reduce the computational costs of updating deep learning models by minimizing the number of additional parameters used to adapt a model to a downstream task. While extensively researched in large language models (LLMs), their application to smaller models used on edge devices, such as convolutional neural networks, remains underexplored. This paper benchmarks and analyzes popular PEFT methods on convolutional architectures typically deployed in resource-constrained edge environments. We evaluate LoRA, DoRA, and GaLore for updating standard and depthwise convolutional architectures to handle distribution shifts and accommodate unseen classes. We utilize recently proposed PyTorch profilers to compare the updated model performance and computational costs of these PEFT methods with traditional fine-tuning approaches. With resource efficiency in mind, we investigate their update behavior across different rank dimensions. We find that the evaluated PEFT methods are only half as memory-efficient when applied to depthwise-separable convolution architectures, compared to their efficiency with LLMs. Conversely, when targeting convolutional architectures optimized for edge deployment, adapter-based PEFT methods can reduce floating point operations (FLOPs) during model updates by up to 95%. These insights offer valuable guidance for selecting PEFT methods based on hardware constraints, performance requirements, and application needs. Our code is online[1].

CCS Concepts

• **Computing methodologies** → *Neural networks*; *Transfer learning*.

Keywords

Edge computing, Deep Learning, Parameter-efficient fine-tuning, Resource-constrained devices, Transfer learning

1 Introduction

Edge devices are increasingly used to deploy deep neural network (DNN) models for local data processing [36], enhancing application accessibility and security by reducing latency and preserving user privacy through the elimination of server communication [31]. However, these devices face significant hardware constraints, such as limited memory, computational capacity and dynamic resource constraints [5], which pose challenges for deploying DNNs [22]. Additionally, DNNs deployed at the edge often struggle with distribution shifts in incoming data, leading to degraded performance over time [27]. Maintaining performance under such conditions requires models to be updated efficiently within the hardware constraints specific to each edge device.

Several approaches have been proposed to update models for unseen classes and distribution shifts. These fall into two categories: methods that build robust invariant models, and those that use domain adaptation techniques. The former pre-train models to be resilient to distribution shifts using data augmentations [32], contrastive loss functions [19], and regularization strategies that reduce sensitivity to domain discrepancies [1]. The latter adapt models at deployment time by training a parameterized subspace of configurable networks [30] or fine-tuning pre-trained features via backpropagation [28]. While invariant models are more robust to expected shifts, they may fail to generalize to unseen distributions and can be brittle when adapted via backpropagation [1].

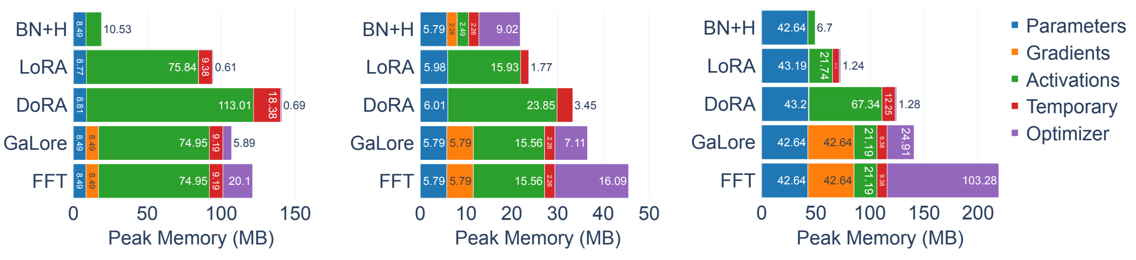
However, on-device adaptation is limited by the computational constraints of edge devices, since standard backpropagation requires at least three times more computation than inference [34]. This challenge is exacerbated by the high resource demands of state-of-the-art models. For instance, Vision Transformers (ViT), despite their strong performance in computer vision [7], have quadratic computational complexity with respect to input size, making them unsuitable for resource-constrained settings [7]. As a result, Convolutional Neural Networks (CNNs), whose complexity scales linearly with input size [15], are typically preferred for edge deployment. Still, updating CNNs via backpropagation remains computationally expensive and often impractical on low-power devices [9]. To address this, MobileNet architectures replace standard convolutions with depthwise-separable convolutions (DSCs), reducing inference computation by a factor of 8 to 9 [15].

To overcome the limitations of updating models on edge devices with constrained resources, parameter-efficient fine-tuning (PEFT) methods have emerged as a promising solution. Assuming updates to a pre-trained model lie in a low-rank subspace [16], PEFT reduces computational and memory demands by restricting gradients or weight updates to low-rank representations [16, 35]. While effective in LLMs [10], PEFT remains underexplored for CNNs in edge vision tasks. Moreover, tools for evaluating PEFT on CNNs in terms of resource usage, complexity, and accuracy are lacking.

This paper addresses the question: *Given the hardware constraints of an edge device and a pre-trained convolutional model, which PEFT method best enables efficient and effective on-device updates?* We propose a novel framework to estimate the efficiency and effectiveness of PEFT methods for specific tasks. Our contributions are:

- We extend PyTorch's FLOPs counter and memory tracker to assess PEFT methods on CNN models typically deployed on edge devices.

[*]Equal contribution.
[1]https://github.com/gslama12/pytorch-model-profiler



a MobileNetV2 b MobileNetV3 c ResNet-18

Figure 1: Peak memory consumption analysis of PEFT methods for different models. Analysis of forward and backward passes for a single 224×224 image. The model architecture influences the total peak memory across the profiled memory groups. For depthwise convolution models, LoRA, DoRA, GaLore, and BN+H show higher peak memory usage compared to standard convolution models due to activations required memory. The peak memory usage of DoRA is 49% to 51% higher than that of LoRA for the MobileNetV2 depthwise architecture. For ResNet18 a larger share of gradient calculation and optimizer state memory enhances the efficiency of LoRA and DoRA PEFT methods.

- We evaluate PEFT performance on pre-trained CNNs, analyzing peak memory usage, FLOPs, and accuracy when updating to unseen classes or handling distribution shifts.
- We benchmark LoRA, DoRA, and GaLore for updating standard and depthwise convolutional architectures to handle distribution shifts and accommodate unseen classes. We show that fine-tuning depthwise convolutional architectures can diminish resource efficiency of PEFT methods.
- We investigate the impact of the rank hyperparameter on the adaptation accuracy.

2 Parameter-Efficient Fine-Tuning

PEFT methods reduce computational and memory costs by minimizing the number of updated parameters. This is achieved by assuming that weight updates lie in a low-rank subspace, leading to lightweight adaptations without sacrificing performance [16]. While extensively studied in LLMs, the application of PEFT methods to CNNs under resource constraints remains underexplored. This section introduces popular PEFT methods evaluated in this work: Low-Rank Adaptation (LoRA), Weight-Decomposed Low-Rank Adaptation (DoRA), Gradient Low-Rank Projection (GaLore), and head-only fine-tuning with batch-normalization (BN+H).

Low-Rank Adaptation (LoRA). LoRA introduces low-rank matrices into the weight update process, enabling fine-tuning with a small number of additional parameters [16]. Specifically, the weight update ΔW is expressed as $\Delta W = AB^T$, where $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{d \times r}$ are low-rank matrices with $r \ll d$. This decomposition significantly reduces the number of parameters and computational complexity, as the gradients are computed and applied only for the low-rank factors A and B. Advantages of LoRA include its simplicity, scalability, and compatibility with various architectures. However, its performance may degrade when the rank r is set too low, particularly for tasks requiring significant adaptation.

Weight-Decomposed Low-Rank Adaptation (DoRA). DoRA initially decomposes the pre-trained weights W_0 into magnitude vector m and direction matrix V and only applies LoRA to V. During training, only m and V are updated. The fine-tuned weights W' can be formulated as $W' = m\frac{V+\Delta V}{V+\Delta V} = m\frac{W_0+BA}{||W_0+BA||_c}$ where $\Delta V = BA$ is the directional update and $|| \cdot ||_c$ represents the vector-wise norm across each column vector [23]. The authors introduce DoRA as an alternative to LoRA, demonstrating that it more closely resembles the training behavior of full fine-tuning when comparing the magnitude and directional updates of the weight matrices of LLMs during training [23]. The trainable magnitude vectors introduce slightly more trainable parameters for DoRA compared to LoRA. Additionally, the weight decomposition in DoRA introduces a more complex computational graph during backpropagation.

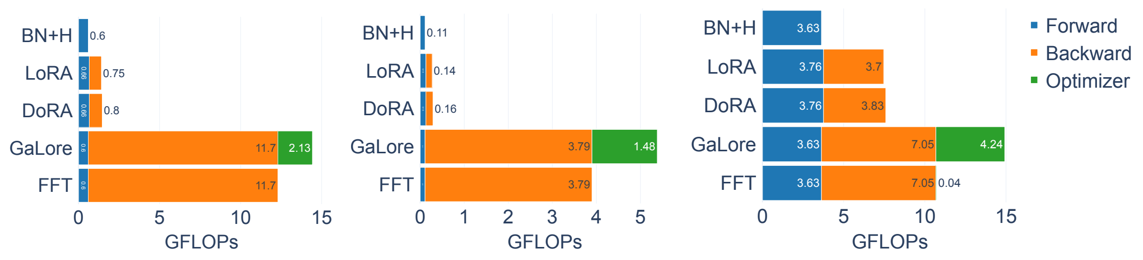
Gradient Low-Rank Projection (GaLore). GaLore combines low-rank approximation with gradient sensitivity to adaptively refine parameter updates [35]. Given the gradient matrix G, GaLore formulates updates as $G \approx \sum_{i=1}^{r} \sigma_i u_i v_i^T$ where u_i and v_i are singular vectors and σ_i are singular values obtained from Singular Value Decomposition (SVD) of G. GaLore dynamically determines the rank r by truncating based on a threshold ϵ, ensuring $\sum_{i=1}^{r} \sigma_i / \sum_j \sigma_j \geq 1 - \epsilon$. This approach ensures an optimal trade-off between computational efficiency and approximation accuracy, adapting resource usage to the task requirements. However, the reliance on SVD and rank truncation can be computationally expensive, especially for high-dimensional gradients.

Head-only Fine-Tuning with Batch-Normalization (BN+H). This fine-tuning approach only updates the final classification layer (*i.e.*, head) using backpropagation and the batch normalization (BN) statistics during the forward pass. The updates of the BN statistics introduce additional flexibility and help to address feature distribution shifts, improving update performance in certain scenarios [8]. While the low number of trainable parameters makes this method highly resource efficient, its effectiveness depends on the degree of distribution shift between the pre-training and target datasets.

We evaluate the practicality of PEFT methods for CNNs on edge devices by analyzing trade-offs in computation, memory, and accuracy, and compare them to head-only with batch normalization updates (BN+H) and full fine-tuning (FFT).

3 Profiling PEFT Methods

Performance Measures. We evaluate the performance of PEFT methods by measuring the number of FLOPs and the peak memory



a MobileNetV2 b MobileNetV3 c ResNet-18

Figure 2: FLOPs analysis of PEFT methods for different models. Analysis of forward and backward passes for a single 224×224 image. Except for GaLore all the PEFT methods reduce the required FLOPs significantly compared to FFT. Depthwise architectures (*i.e.*, MobileNets) report a FLOPs reduction of more than 10× compared to standard convolution.

usage required to update the models to each task during the forward and backward passes. The former serves as the inference latency to estimate the model's execution time on an edge device [21], while the latter is considered the major bottleneck for enabling neural networks on the edge [22]. Using these measurements for each PEFT method configuration, along with the edge device's hardware and latency constraints, our framework offers guidance to estimate the efficiency and effectiveness of PEFT methods for specific tasks.

3.1 Profiling Framework

We modify an existing FLOPs counter[2] to distinguish between the FLOPs used during gradient computation and those used during the optimizer's weight update step in the backward pass. An example usage can be seen in Listing 1.

```
flops_counter = FlopCounterMode(model)

with flops_counter:
    optimizer.zero_grad()
    outputs = model(input_tensor)
    loss = outputs.sum()
    loss.backward()
    flops_counter.reset_module_tracking()
    optimizer.step()
```

Listing 1: Example of profiler usage to compute model's forward-backward FLOPs.

The FLOPs counter uses `__torch_dispatch__` to attach hooks to the tensor level operations of PyTorch [25]. By tracking the operations for tensor convolution, multiplication, addition and batch normalization we calculate the number of required FLOPs for each operation from the operand shapes.

Memory. PyTorch's latest memory tracker[3] can distinguish between the memory groups listed in Table 1. We modify the memory tracker to profile the peak memory usage of each group regardless of when it occurs, with total memory computed as their sum and profiling done using the steps reported in Listing 1.

4 Evaluation

To investigate the capabilities of PEFT methods for models typically deployed on edge devices, we evaluate the performance of LoRA, DoRA, GaLore, BN+H and FFT on MobileNetV2 [29] and MobileNetV3 [14]. These models employ optimized DSCs layers, which reduce the computational cost during inference by up to 9× times [15]. To highlight the performance and efficiency differences of PEFT methods on DSC architectures compared to standard convolution architectures, we also include the results for ResNet-18 [11]. We show accuracy and profiling results for all models, initially pretrained on ImageNet [6], on various downstream tasks, including CIFAR10-C [12] corruptions and the Visual Wake Words [4] dataset (VWW). We choose CIFAR-10-C corruptions of varying difficulty to highlight the task-specific achieved accuracy of the investigated PEFT methods and demonstrate the advantages of LoRA, DoRA, and GaLore over the simpler BN+H approach. Furthermore, we evaluate the effectiveness of the selected PEFT methods on the VWW dataset for the binary classification task of detecting the presence of a person in an image. Additionally, we conduct the same experiments on the MobileNetV2 and ResNet18 models pretrained on CIFAR-10 [20]. For all the experiments we followed the hyperparameter recommendations from [16, 23, 35] to ensure a fair comparison. We use implementations of LoRA and DoRA from the Hugging Face PEFT library[4] and the pre-release implementation of GaLore[5] according to [35] with small adaptions to suit CNN architectures on the edge.

Table 1: PyTorch memory groups analyzed during training. The tracker categorizes memory usage by group, offering detailed insights into resource allocation.

Group	Description
PARAM	Model parameters.
GRAD	Gradients of model weights during backpropagation.
ACT	Intermediate activations stored for the backward pass.
OPT	Optimizer state memory (*e.g.*, momentum buffers).
TEMP	Temporary buffers used in gradient computations (*e.g.*, autograd intermediates).

Memory. In Fig. 1 we analyze the peak memory usage of the models' forward pass, backward pass, and optimizer step for all PEFT methods. We observe that the investigated PEFT methods only reduce memory usage for gradients and optimizer groups. For

[2]https://gist.github.com/soumith/5f81c3d40d41bb9d08041431c656b233
[3]https://github.com/pytorch/pytorch/pull/124688
[4]`https://huggingface.co/docs/peft/index`
[5]`https://github.com/jiaweizzhao/GaLore`

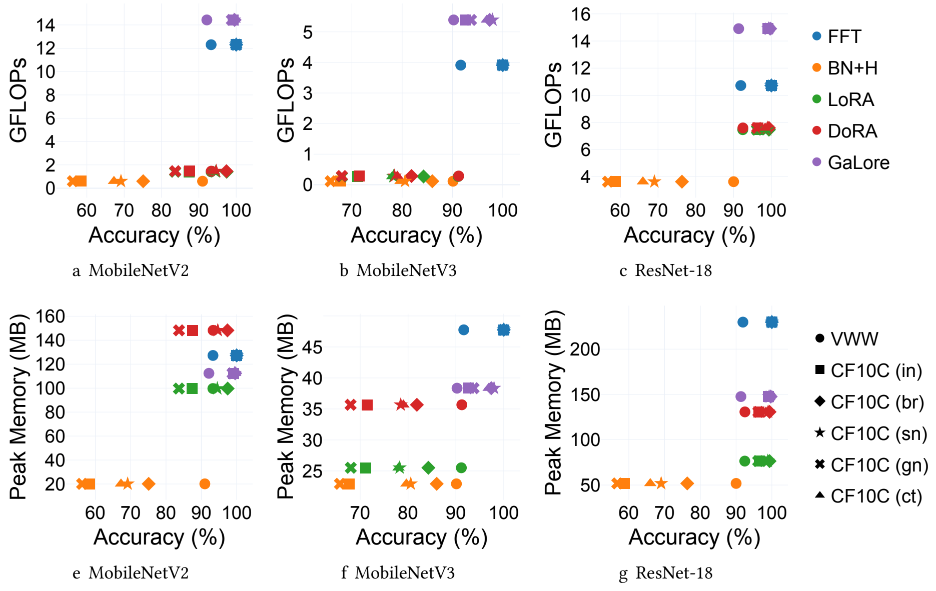


Figure 3: Trade-off between accuracy and resource usage. Profiling and evaluation of the PEFT methods on four different models pre-trained on ImageNet, across different fine-tuning tasks for one training step using a single 224×224 image. While GaLore achieves consistent accuracy results comparable to FFT across different models and fine-tuning tasks, LoRA and DoRA show accuracy variations of up to 20% across tasks on MobileNet architectures. LoRA delivers the best trade-off between accuracy and resource consumption across various tasks on ResNet-18. Accuracies are reported using early stopping after 10 epochs without validation loss improvement.

models that use DSCs, the storage required for activation maps is the primary contributor to total peak memory, limiting the effectiveness of PEFT methods for such model architectures. With minimal trainable parameters, BN+H fine-tuning stands out as the most memory-efficient PEFT method. By avoiding full backpropagation through all layers, it achieves up to 85% total peak memory savings compared to FFT (Fig. 1a). Since the classifier of MobileNetV3 comprises two trainable linear layers, BN+H exhibits reduced memory efficiency on this model compared to MobileNetV2, achieving only a 52% reduction in peak memory usage relative to FFT (Fig. 1b).

We observe LoRA to be the second most memory efficient PEFT method, with a peak memory reduction of up to 67% compared to FFT on ResNet-18 (Fig. 1c), replicating the improvement observed on LLMs [16]. However, for models using DSCs, we observe a smaller peak memory reduction, ranging between 22% on MobileNetV2 (Fig. 1a) and 48% on MobileNetV3 (Fig. 1b), which is notably lower than the improvements observed in LLMs. Standard convolution CNNs, such as ResNet-18, allocate a larger portion of peak memory to optimizer state and gradients compared to models with DSCs, making LoRA more memory-efficient for these architectures (Fig. 1). Although DoRA offers an almost costless alternative to LoRA during inference [23], its more complex computational graph leads to a memory overhead between 29% on MobileNetV3 (Fig. 1b) and 58% on ResNet-18 (Fig. 1c) during training.

Contrary to LoRA and DoRA, GaLore only optimizes the memory used for storing optimizer states [35]. This also implies that standard convolution CNNs, like ResNet-18, benefit more from GaLore than models using DSCs, due to the larger share of optimizer state memory in the total peak memory (Fig. 1c). Similarly, as reported in [35] for LLMs, our results in Fig. 1 demonstrate an average reduction of 65% in optimizer state memory across the tested models. However, for models using DSCs, this only results in an overall peak memory reduction of around 5-10%, as the optimizer state memory is not the primary contributor (Fig. 1a and Fig. 1b). While [35] reports that GaLore is more memory-efficient than LoRA for LLMs, we did not observe this behavior in the CNNs evaluated in this study (Fig. 1). With its significantly smaller number of trainable parameters, LoRA utilizes only about 10% of the optimizer state memory required by GaLore, resulting in higher memory efficiency. We observe similar results to those in Fig. 1 for models pre-trained on CIFAR-10 using 32×32 images, remaining within acceptable limits.

FLOPs. In Fig. 2 we analyzed the FLOPs required by the PEFT methods to perform forward and backward passes. For standard convolution CNNs, the ratio of FLOPs required for the backward pass to those needed for the forward pass is approximately 2:1 [13]. DSCs alter this ratio by splitting the filters of the convolutional layer into groups, where the number of groups equals the number of input channels C_{in}. With each filter only processing a single input channel, the FLOPs during the forward pass of DSC layers are reduced by the factor of $\frac{1}{C_{\text{in}}}$. During the backward pass of DSC layers, the FLOPs include computations for both the input gradient and the weight gradient. While the weight gradient benefits from reduced FLOPs due to filter grouping, the input gradient does not. Consequently, this results in an approximate 20:1 ratio between the FLOPs required for the forward pass and the backward pass during FFT (Fig. 2). This finding underscores the need for further investigation into optimizing backward pass FLOPs for DSC models.

Adapter-based PEFT methods, such as LoRA and DoRA, significantly reduce the 20:1 FLOPs ratio of DSCs to approximately 1.2:1, achieving an overall FLOPs reduction of 80% for MobileNetV3 (Fig. 2b) compared to FFT. For standard convolutional models like ResNet-18, which exhibit the expected 2:1 FLOPs ratio between the forward and backward pass, the FLOPs reduction achieved by LoRA and DoRA is limited to approximately 57% (Fig. 2c). Additionally, we observe that the SVD operation in GaLore's optimizer step introduces a FLOPs overhead of 10% to 30% compared to FFT (Fig. 2), slightly exceeding the 10% overhead reported for LLMs in [35] in certain models.

Accuracy and Performance. Our results in Fig. 3 demonstrate, that the accuracy after fine-tuning with PEFT methods is significantly influenced by the model architecture, the size of the model, and the specific fine-tuning task. While BN+H demonstrates acceptable performance on lightweight adaptions like the CIFAR10-C Brightness (br) corruption and the VWW dataset, LoRA, DoRA and GaLore consistently outperform BN+H fine-tuning on all other tasks, with accuracy differences of up to 40%.

By utilizing full-rank weight updates, GaLore shows the most consistent accuracy across different fine-tuning tasks and model architectures and achieves comparable results to FFT. While LoRA and DoRA achieve similar accuracy scores on ResNet-18, the adapter-based PEFT methods exhibit inconsistent performance across different fine-tuning tasks on MobileNet architectures. Notably, for challenging fine-tuning tasks such as Impulse noise (in) and Gaussian noise (gn), MobileNets pre-trained on ImageNet experience accuracy drops of up to 20% when using LoRA or DoRA compared to GaLore (Fig. 3a and Fig. 3b). GaLore offers greater robustness than LoRA but uses 1.13–2× more memory and 2–20× more FLOPs, depending on the model. In contrast, LoRA requires about 2× more training iterations to converge.

Fig. 3f shows that, unlike results for LLMs reported in [35], LoRA outperforms GaLore by up to 2.5% on some tasks. Similarly, contrary to [23], DoRA shows no accuracy gain over LoRA on edge-optimized CNNs in any experiment (Fig. 3). Results on CIFAR-10 pretraining mirror these trends, with all models improving on corruption tasks and LoRA consistently offering the best accuracy-efficiency trade-off.

Impact of the rank. Theoretically, a PEFT method's rank determines its learning capacity in the low-rank space, with higher ranks better approximating FFT performance [16, 35]. However, as shown in Fig. 4, fine-tuning a pre-trained model for 5 epochs reveals that higher rank does not always improve accuracy.

Unlike LLM results in [35], we find that for tasks where the pre-fine-tuning accuracy is high (Fig. 4a and Fig. 4f), the accuracy of GaLore may degrade with a higher rank setting. In these cases, LoRA and DoRA achieve up to 6% higher accuracy compared to GaLore. We conjecture that for these fine-tuning tasks, a lower rank setting provides a smoother gradient landscape that is easier to optimize, leading to better model performance.

When the pre-trained model performs poorly on a fine-tuning task, adapter-based methods like LoRA and DoRA often struggle to adapt (Fig. 4e). In such cases, GaLore outperforms them by up to 50% at low ranks, benefiting from its full-rank weight updates. We hypothesize that the combined gradient rank of LoRA and DoRA is insufficient for small r, particularly when the fine-tuning task diverges significantly from the pre-training objective. This idea is supported by a similar effect observed in federated learning scenarios in [2], where increasingly diverse data distributions across clients also increased the accuracy gap between LoRA and FFT.

Results with CIFAR-10 pretraining follow the same trends. Although [23] reports DoRA outperforming LoRA by up to 37% for $r < 16$, we observe no such gain, as both perform similarly across tasks. Further analysis is left for future work.

Summary. Overall, the PEFT methods LoRA, DoRA, and GaLore significantly outperform BN+H in terms of accuracy, especially on challenging fine-tuning tasks. GaLore demonstrates robust accuracy and requires, on average, approximately 2× fewer iterations during fine-tuning compared to LoRA, DoRA and BN+H. Although LoRA offers a better trade-off between resources and accuracy, with particularly low FLOPs consumption on models employing DSCs and significantly improved memory efficiency on standard convolution CNNs, it comes at the cost of longer training times and less robust accuracy scores on hard fine-tuning tasks. While DoRA introduces a memory overhead compared to FFT, it does not show improved performance over LoRA in any of our experiments, making it less efficient than LoRA for CNNs optimized for edge devices.

5 Related Work

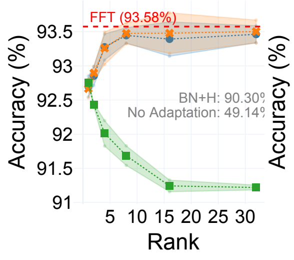
Test-time Adaptation on the Edge. Test-time adaptation (TTA) has emerged as a technique for dynamically adjusting model's parameters to the incoming test data stream to address domain shifts [24, 33]. TTA enables deep models to adjust their predictions based on the characteristics of the incoming test data, which may differ from the training data [17]. While TTA is extensively employed to adapt deep models at the edge, it presents several limitations, including representation collapse due to overfitting on the test data [26] and the inability to handle open set domain adaptations [3]. Furthermore, TTA model's weight update exhibits memory usage similar to full fine-tuning, *i.e.*, infeasible at the edge, since the model needs to backpropagate through its whole architecture to compute the gradient [18] and store it in the optimizer memory buffer. PEFT methods mitigate these limitations by learning a set of low-rank external matrices [16], or by fine-tuning the model with a memory-efficient low-rank projection of the gradient [35] and therefore reducing the required optimizer memory.

Parameter-Efficient Fine-Tuning for LLMs. PEFT methods have drawn attention for fine-tuning large language models (LLMs) without incurring the prohibitive computational or memory costs of standard fine-tuning methods. PEFT methods introduce a small number of additional parameters to be fine-tuned, *i.e.*, through low-rank decomposition and specialized adapters, while keeping the model weights frozen or updating the weights through a low-rank projection of the gradients. While these methods have been studied for LLMs fine-tuning [10], analysis of their performance for on-device deep model updates is lacking. In contrast to previous works, we explore the performance of PEFT methods for optimized deep learning models for edge devices, and compare their performance and computational cost for different downstream tasks.

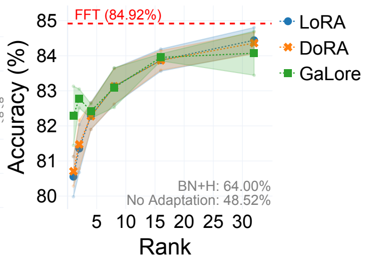
6 Conclusion, Limitations, and Outlook

This study benchmarks parameter-efficient fine-tuning (PEFT) methods on CNN architectures for edge devices, revealing distinct trade-offs between accuracy and resource efficiency. While LoRA achieves the best balance between performance and computational cost in most scenarios, DoRA's additional memory overhead limits its applicability in resource-constrained settings. GaLore demonstrates robust accuracy but incurs higher computational complexity due to SVD-based updates. Across architectures using depthwise-separable convolutions, PEFT methods are only half as memory-efficient as reported for LLMs, with adapter-based methods achieving up to 95% FLOPs reduction compared to full fine-tuning. These findings provide actionable insights into selecting PEFT methods for edge deployments, depending on hardware constraints and application needs.

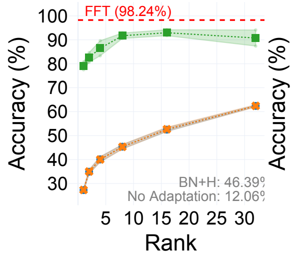
Limitations. This study does not include on-device profiling, limiting realism for specific deployment scenarios. Comparisons are based on fixed hyperparameters without extensive tuning, and quantization is not considered. While focused on CNNs, generalization to other edge-relevant architectures like lightweight transformers remains unexplored. These aspects are left for future work.



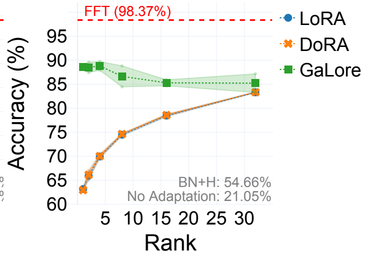
a ImageNet ⟶ VWW



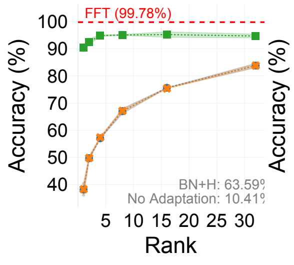
b CIFAR-10 ⟶ VWW



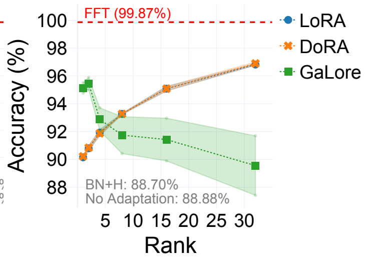
c ImageNet ⟶ CIFAR-10-C (in)



d CIFAR-10 ⟶ CIFAR-10-C (in)



e ImageNet ⟶ CIFAR-10-C (br)



f CIFAR-10 ⟶ CIFAR-10-C (br)

Figure 4: Impact of different ranks on adaptation accuracies. Two pre-trained (*i.e.*, ImageNet and CIFAR10) MobileNetV2 models are fine-tuned for five epochs on three different datasets (*i.e.*, CIFAR10-C Brightness (br), CIFAR10-C Impulse noise (in) and VWW) by varying the PEFT methods ranks. PEFT methods' performance is influenced by the initial task accuracy of each pre-trained model.

Acknowledgments. This research was funded in part by the Austrian Science Fund (FWF) within the DENISE doctoral school (grant number DFH 5). The results presented in this paper were computed using the computational resources of the HLR resources of the Zentralen Informatikdienstes of Graz University of Technology.

References

[1] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz. 2019. Invariant risk minimization. *arXiv preprint arXiv:1907.02893* (2019).
[2] S. Babakniya, A. Elkordy, Y. Ezzeldin, et al. 2023. SLoRA: Federated Parameter Efficient Fine-Tuning of Language Models. https://arxiv.org/abs/2308.06522
[3] P. Busto and G. Juergen. 2017. Open set domain adaptation. In *ICCV*. 754–763.
[4] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes. 2019. Visual wake words dataset. *arXiv preprint arXiv:1906.05721* (2019).
[5] F. Corti, B. Maag, J. Schauer, U. Pferschy, and O. Saukh. 2023. REDS: Resource-Efficient Deep Subnetworks for Dynamic Resource Constraints. *arXiv preprint arXiv:2311.13349* (2023).
[6] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*. IEEE, 248–255.
[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, and *et al.*. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.
[8] J. Frankle and *et al.*. 2020. Training batchnorm and only batchnorm: On the expressive power of random features in CNNs. *arXiv:2003.00152* (2020).
[9] S. Han, X. Liu, H. Mao, and *et al.*. 2016. EIE: Efficient Inference Engine on Compressed DNNs. In *Symposium on Computer Architecture*. IEEE.
[10] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. (2024).
[11] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *IEEE CVPR*.
[12] D. Hendrycks and T. Dietterich. 2019. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261* (2019).
[13] M. Hobbhahn and J. Sevilla. 2021. What's the Backward-Forward FLOP Ratio for Neural Networks? https://epoch.ai/blog/backward-forward-FLOP-ratio
[14] A. Howard, M. Sandler, G. Chu, and *et al.*. 2019. Searching for MobileNetv3. In *ICCV*. 1314–1324.
[15] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, et al. 2017. MobileNets: Efficient CNNs for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861* (2017).
[16] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, et al. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
[17] Y. Iwasawa and Y. Matsuo. 2021. Test-time classifier adjustment module for model-agnostic domain generalization. *NeurIPS* 34 (2021), 2427–2440.
[18] H. Jia, Y. Kwon, A. Orsino, T. Dang, D. Talia, and C. Mascolo. 2024. TinyTTA: Efficient TTA via Early-exit Ensembles on Edge Devices. In *NeurIPS*.
[19] P. Khosla, P. Teterwak, C. Wang, et al. 2020. Supervised contrastive learning. *NeurIPS* 33 (2020), 18661–18673.
[20] A. Krizhevsky, V. Nair, and G. Hinton. 2009. CIFAR-100 and CIFAR-10. http://www.cs.toronto.edu/~kriz/cifar.html MIT License.
[21] E. Liberis, Ł. Dudziak, and N. Lane. 2021. μnas: Constrained neural architecture search for microcontrollers. In *1st Workshop on Machine Learning and Systems*.
[22] J. Lin, L. Zhu, W. Chen, W. Wang, and S. Han. 2023. Tiny machine learning: progress and futures. *IEEE Circuits and Systems Magazine* 23, 3 (2023), 8–34.
[23] S. Liu, C. Wang, H. Yin, P. Molchanov, and *et al.*. 2024. DoRA: Weight-Decomposed Low-Rank Adaptation. arXiv:2402.09353 [cs.CL]
[24] S. Niu, C. Miao, G. Chen, P. Wu, and P. Zhao. 2024. Test-time model adaptation with only forward passes. *arXiv preprint arXiv:2404.01650* (2024).
[25] A. Paszke, S. Gross, F. Massa, A. Lerer, and *et al.*. 2019. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS* 32 (2019).
[26] O. Press, R. Shwartz-Ziv, Y. LeCun, and M. Bethge. 2024. The entropy enigma: Success and failure of entropy minimization. *arXiv:2405.05012* (2024).
[27] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. Lawrence. 2008. *Dataset shift in machine learning*. The MIT Press.
[28] D. Rumelhart, G. Hinton, and R. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.
[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*. 4510–4520.
[30] O. Saukh, D. Wang, X. He, and L. Thiele. 2023. Representing Input Transformations by Low-Dimensional Parameter Subspaces. *arXiv:2305.13536* (2023).
[31] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.
[32] C. Shorten and T. Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019).
[33] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell. 2020. Tent: Fully test-time adaptation by entropy minimization. *arXiv:2006.10726* (2020).
[34] D. Xu, M. Xu, Q. Wang, S. Wang, et al. 2022. Mandheling: Mixed-precision on-device dnn training with dsp offloading. In *Mobile Computing And Networking*.
[35] J. Zhao, Z. Zhang, B. Chen, and *et al.*. 2024. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. https://arxiv.org/abs/2403.03507
[36] Z. Zhou, X. Chen, E. Li, and *et al.*. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.