Parameter-Efficient Fine-Tuning (PEFT) with LoRA on CNNs and ViTs

Frederic Lorenz
KTH, Stockholm
fjlorenz@kth.se

Manuel Voit
KTH, Stockholm
voit@kth.se

Anton Holderied
KTH, Stockholm
holde@kth.se

Abstract

We study how parameter-efficient fine-tuning with Low-Rank Adaptation (LoRA) compares to standard transfer-learning strategies in constrained vision adaptation. Using Oxford-IIIT Pet as the main benchmark, we evaluate linear probing, simultaneous and gradual full fine-tuning, and LoRA on ResNet-50/101, then assess whether the performance trends observed for these configurations persist on multi-label MS COCO and ViT-B/16.

Full fine-tuning remains strongest on CNNs for the full dataset. On ResNet-101, the best held-out test accuracy is achieved by simultaneous and gradual unfreezing, while LoRA remains competitive with less than 5% of the trainable parameters. Under data scarcity, however, full fine-tuning becomes unstable, whereas frozen-backbone methods generalize much more reliably. In this regime, linear probing is the strongest CNN baseline, and LoRA preserves much of this robustness while offering greater adaptation capacity.

LoRA also transfers well beyond the initial setting. When applied to CNNs on MS COCO, it stays close to the best full fine-tuning baselines. On ViTs it becomes especially attractive under strong data scarcity, where it outperforms full fine-tuning. However, parameter efficiency does not automatically imply large memory savings. For CNNs, activation storage dominates VRAM usage and deeper, better-performing injections can offset the savings. Overall, LoRA is a competitive and robust adaptation strategy, especially in low-data, though it does not fully replace full fine-tuning when maximizing final CNN test accuracy is the main goal.

Our implementation is available on GitHub and our trainings on DagsHub.

1 Introduction

Fine-tuning (FT) pre-trained models presents a fundamental optimization trade-off. Full-parameter FT risks catastrophic forgetting of generalized feature extractors, while strictly frozen *Linear Probing (LP)* often lacks the capacity to model the nuanced geometries of a target domain. These challenges are exacerbated by constraints like data scarcity and class imbalance, which expose high-capacity models to overfitting. Furthermore, as architectures scale from *Convolutional Neural Networks (CNNs)* to larger *Vision Transformers (ViTs)*, full-parameter FT is becoming computationally infeasible.
To mitigate these bottlenecks, *Parameter-Efficient Fine-Tuning (PEFT)* methods like *Low-Rank Adaptation (LoRA)* have emerged. LoRA hypothesizes that weight updates during adaptation possess a low intrinsic dimension [1, p. 2]. Rather than updating the full pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA freezes W_0 and injects trainable rank decomposition matrices to represent the update ΔW:

$$W = W_0 + \Delta W = W_0 + BA \tag{1}$$

where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$ and the rank $r \ll \min(d, k)$.
In this work, we present a systematic evaluation of these FT and PEFT strategies under heavily constrained conditions. Using ResNet architectures as a baseline on the Oxford Pets dataset, we

first isolate the dynamics of standard FT, specifically unfreezing depth, *learning rates (LR)*, and regularization, also taking into account data scarcity and class imbalance. Building upon these baselines, we integrate LoRA, systematically analyzing its hyperparameter space (rank r, injection depth and gradual unfreezing schedules) to optimize the trade-off between task performance and training cost. Finally, we validate our optimal configurations by assessing their transferability to ViTs and the more complex MS COCO dataset. Our results show that full FT remains strongest on CNNs under full data, whereas LoRA becomes increasingly attractive under data scarcity and when transferred to MS COCO and ViTs. It can also lead to lower training times due to faster convergence.

2 Related Work

Transfer Learning and Parameter Efficiency. Traditional adaptation via Full Fine-Tuning (FFT) or LP forces a trade-off between expressive capacity and catastrophic forgetting [2], a dynamic fundamentally tied to deep network feature transferability [3]. While scheduling techniques like gradual unfreezing [4] attempt to mitigate this, PEFT architectures, originating from adapter modules [5], offer robust structural solutions. Specifically, LoRA [1] constrains gradient updates to a low intrinsic rank, drastically reducing computational overhead while preserving generalization capability.

Adapting LoRA for Vision and Convolutions. LoRA currently dominates *Large Language Model (LLM)* adaptation [6, p. 1], consistently matching or exceeding FFT performance [1, p. 1]. However, directly applying standard, independent LoRA matrices to CNNs disrupts spatial image features, often yielding unsatisfactory results [7]. To bridge this gap, Zhong et al. [8] integrated convolutional priors into the bottleneck, proving spatial awareness is critical. Similarly, ConvLoRA [9, p. 2] addresses this architectural friction by mathematically treating the pre-trained convolutional weights as 2D matrices, constraining updates through a parallel low-rank decomposition. By confining gradient updates strictly to these tiny factorized matrices rather than the network's massive spatial kernels, the framework drastically reduces the trainable parameter footprint and computational overhead without sacrificing the backbone's capacity to adapt to new target distributions. As PEFT paradigms successfully expand into ViTs [10], these developments depict further ways of drastically reducing trainable parameters in deep architectures.

3 Data

As an initial baseline, we used Oxford-IIIT Pet, an image classification and segmentation dataset of 37 categories of various breeds of dogs and cats, which can be used for multi and binary classification [11]. It contains 7349 images in total with roughly 200 per class. State-of-the-art solutions for classification include YOLOv8 for object detection combined with a ViT, which achieves a test accuracy on the dataset of 96.85% for image classification [12]. Furthermore, we try to generalize our findings by applying the best working configurations to MS COCO. It is an 80-class dataset containing around 330,000 images of everyday objects; however, rather than standard object detection, we formulate the task as multi-label image classification. As reference, a ResNet-101 achieved a classification mAP of 77.3% [13, p. 6]. It should be noted that we scale down the pictures to 224x224px due to restricted compute resources. On validation and test data, this happens without stretching, using Torch's `transforms.Resize` and `transforms.CenterCrop` [14], i.e. the shortest edge gets scaled to 224px, and then a central crop of the image is taken. All data is normalized according to ImageNet's mean and standard deviation: `mean=[0.485,0.456,0.406]` and `std=[0.229,0.224,0.225]`. For Oxford-IIIT Pet, we use the official `test` split for final evaluation and split `trainval` into train and validation with a stratified 80:20 split. For MS COCO, we use `val2017` as test and split `train2017` into train and validation with an 80:20 split via iterative stratification for multi-labeling.

Augmentation	**Hyperparameters**
Random Cropping	`scale=(0.7,1.0)`
Random Horizontal Flip	`p=0.5`
Color Jitter	`brightness=0.15, contrast=0.15, saturation=0.15, hue=0.02`
Random Rotation	Range of `degrees=`$[-10, 10]$
Random Erasing	`p=0.15, scale=(0.02,0.12), ratio=(0.3,3.3), value="random"`

Table 1: Augmentations from Torch's `transforms` [14], applied only to training data on Oxford-IIIT Pet and MS COCO.

4 Methods

Architectures and Framework. We evaluate FT dynamics across two model architectures: CNNs, represented by ResNet-50 and -101 [15], and sequence models, represented by the ViT-B/16 architecture [16]. The computational graphs, forward-pass mechanics and Autograd engines are implemented natively in PyTorch. For reproducibility of the hyperparameter search space across all adaptation strategies, every experimental run is tracked via MLflow, logging model configurations alongside train and validation metrics, including memory, train time, loss, accuracy, F1, precision, recall, etc.

Data Scarcity, Class Imbalance and Regularization. To evaluate adaptation under data scarcity, we artificially constrain the multi-class training subset via a scarcity multiplier $\gamma \in]0, 1]$. Given a total dataset of size N and base split ratio ρ, the effective training sample count is reduced to $N_{\text{train}} = \lfloor \gamma \rho N \rfloor$. Furthermore, to simulate distributional skew, we deliberately induce structural class imbalance by downsampling a targeted subset of classes by a secondary imbalance factor $\beta \in]0, 1]$. To counteract the resulting class imbalances, we employ either oversampling via a *Weighted Random Sampler* [17] and/or a weighted *Cross-Entropy (CE)* loss, scaling class penalties inversely to their sample counts ($w_c \propto 1/N_c$). If no imbalance is applied, these countermeasures are disabled, defaulting to standard uniform sampling and an unweighted CE-loss. Moreover, to prevent overfitting, we always enforce an *Early Stopping*, extracting the optimal parameter state θ^* by halting training if the validation loss $\mathcal{L}_{\text{val}}$ increases by a minimum margin $\Delta_{\text{EarlyStop}} = 0$ over a patience window of $p = 3$ consecutive epochs.

Optimizers and Scheduling. Across all structural paradigms, *Gradient Descent (GD)* is executed via AdamW [18] to properly decouple weight decay regularization from the adaptive LR. For ViTs, the LR η_t requires a linear warmup phase of w epochs to stabilize the randomly initialized classification head. Following this warmup, η_t decays via Cosine Annealing:

$$\eta_t = \begin{cases} \eta_{\max} \frac{t}{w} & \text{if } t \leq w \\ \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\pi \frac{t-w}{T-w}\right)\right) & \text{if } t > w \end{cases} \tag{2}$$

For CNNs, which are more robust to initial gradient spikes, this schedule simplifies to standard cosine annealing evaluated over the full domain $t \in [1, T]$ by setting the warmup duration to $w = 0$. Layer-wise LR decay is applied to ViTs for FFT with a factor of 0.7.

FT Considerations. Since this work aims to compare various FT and PEFT strategies with one another, it is pertinent to define them as part of our methodology. Concretely, we are using pre-trained *Neural Networks (NN)* $f(x; \theta)$, where the parameters θ are partitioned into frozen and trainable sets: $\theta = \theta_{\text{frozen}} \cup \theta_{\text{trainable}}$. When using LP, we set $\theta_{\text{trainable}} = W_{\text{Head}}$, where $W_{\text{Head}} \in \mathbb{R}^{d \times C}$ is the final classification projection. In other FT strategies, we unfroze main weight matrices: Let the total parameters of the pre-trained NN, θ, be partitioned into a stem, four sequential stages S_i, and a linear classification head. For a depth hyperparameter $k \in \{0, 1, 2, 3, 4\}$, we freeze the stem and the first $4 - k$ stages, and train the final k stages together with the classification head. We define a stage in CNNs as a contiguous sequence of residual blocks whose output feature maps share the same spatial resolution and channel dimension – in ResNets, there are 4 of them [15]. In ViTs, specifically with ViT-B/16, we summarize 3 subsequent transformer blocks as a stage. In both cases, a more granular approach would have been possible, however this was disregarded due to the computation overhead of an exponentially larger hyperparameter search space, as well as the diminishing returns we expected. The unfreezing of said stages happened either in a simultaneous or a gradual way. In the latter case, a further hyperparameter τ, defining the unfreeze interval, was introduced – the unfreezing then happened according to Algorithm 1.

Unified LoRA formulation. To adapt pre-trained weights without catastrophic forgetting, we freeze the base network parameters W_0 and introduce a parallel low-rank trainable branch. For a given input x and target layer $\mathcal{F}$, the forward pass is redefined by the rank r and the scaling factor $\alpha = 2r$:

$$h = \mathcal{F}(x) + \frac{\alpha}{r} \mathcal{B}(\mathcal{A}(x)) \tag{3}$$

While this underlying scaling logic remains constant, the architectural realization of $\mathcal{A}$ and $\mathcal{B}$ must diverge fundamentally to respect the spatial versus sequence inductive biases of CNNs and ViTs.

Architectural LoRA implementations.

- **CNNs (Spatial):** To preserve local spatial correlations in CNNs without flattening 4D feature tensors, we implement $\mathcal{A}$ as a convolutional projection with the same kernel size, stride and padding as the original layer, mapping the input from C_{in} channels to rank r. We then implement $\mathcal{B}$ as a pointwise (1×1) convolution that maps the rank r representation to C_{out} output channels.
- **ViTs (Linear):** In ViTs, $\mathcal{F}$ represents dense linear projections within the Multi-Head Self-Attention blocks. Consequently, $\mathcal{A} \in \mathbb{R}^{r \times d_{in}}$ and $\mathcal{B} \in \mathbb{R}^{d_{out} \times r}$ are instantiated as standard 2D parameter matrices targeting the query and value projection weights (W_q, W_v). We restricted ViT LoRA injections to the attention mechanism to avoid the parameter inflation associated with targeting feedforward blocks.

To ensure correct implementation, we verify our LoRA injections maintain forward-pass equivalence on initialization, and isolate gradient updates, ensuring frozen backbone parameters remain unaltered.

LoRA targeting and depth scheduling. We define a LoRA injection configuration by its target module set Λ and its maximum injection depth k, measured from the classification head. The target set Λ specifies exactly which layer types receive low-rank updates (e.g. spatial convolutions in CNNs, or the W_q and W_v projections in ViTs). For temporal scheduling (Gradual LoRA), we adapt the logic of Algorithm 1. In Gradual LoRA, all adapters are inserted at initialization into the modules defined by Λ, but their parameters are unfrozen progressively during training, moving from head-adjacent layers toward layers farther from the classification head, up to depth k. This strategy was motivated by the effectiveness of gradual unfreezing in standard FT and was intended to improve adaptation dynamics by introducing additional trainable capacity in a staged manner.

Computational Cost and Memory Profiling. To evaluate the true viability of these adaptation strategies under constrained environments, we measure the peak GPU memory allocation dynamically via PyTorch's native CUDA memory tracker [19]. The total VRAM footprint V_{total} is composed of:

$$V_{\text{total}} \approx V_{\text{weights}} + V_{\text{gradients}} + V_{\text{optimizer}} + V_{\text{activations}} \tag{4}$$

Here, V_{weights} denotes the stored model parameters, $V_{\text{gradients}}$ the gradient tensors retained for trainable parameters, $V_{\text{optimizer}}$ the optimizer state (e.g. AdamW moments), and $V_{\text{activations}}$ the intermediate forward activations required for backpropagation. PEFT methods such as LoRA reduce $V_{\text{gradients}}$ and $V_{\text{optimizer}}$ by restricting training to a small subset of parameters, while leaving the frozen base weights in memory. However, the overall peak memory usage for the tested model architectures still depends strongly on the activation footprint induced by the full forward pass through the backbone. We therefore report the total observed VRAM usage rather than inferring memory efficiency solely from the trainable parameter count. We evaluate temporal computational cost through mean epoch train time (t_{epoch}) and overall training time (T_{total}). The latter reflects both the hardware-dependent cost per epoch and the number of epochs required for convergence:

$$T_{\text{total}} = t_{\text{epoch}} \times E_{\text{total}} \tag{5}$$

, where E_{total} denotes the total number of epochs executed prior to early stopping. All investigations into memory and time usage are done on the same host and load conditions, to ensure fair results.

Baselines and Transferability. We perform a hyperparameter search for each strategy (LP, simultaneous unfreezing, gradual unfreezing and LoRA) selecting the best ResNet models exclusively by validation accuracy. The test set is used only once for the analysis, after the final configurations are fixed. After establishing baselines on Oxford-IIIT Pet for ResNet-50/101, we assess whether the performance trends observed for these configurations persist on multi-label MS COCO. The same process was repeated for ViT-B/16 to assess the impact of LoRA on a different type of architecture.

5 Experiments

5.1 Baselines for various FT strategies for CNNs

To ensure fair comparisons, we first isolated the optimal configurations for each FT strategy via a systematic hyperparameter search, evaluating varying LRs, L_2, unfreeze depths k and unfreeze frequencies τ. With full data availability ($\gamma = 1.0$), unfreezing the pre-trained weights yields the highest validation performance. Gradual unfreezing with $k = 1$ and $\tau = 3$ as well as simultaneous unfreezing with $k = 1$ achieved a peak validation accuracy of 96.6%, while LP capped at 95.2%.

Final evaluation on the held-out test set preserved the same overall ranking, but with smaller margins: LP reached 92.8% test accuracy and the selected gradual FT configuration achieved the strongest generalization at 93.8%. However, this dynamic inverts under data starvation. At $\gamma = 0.02$, FFT collapses due to overfitting. Gradual unfreezing plummets to 16.6% peak validation accuracy, exhibiting class-collapse (Macro-F1: 13.1%, Precision: 16.8%, Recall: 16.6%). Conversely, LP dominates this strong scarcity, maintaining 67.5% accuracy (Macro-F1: 65.9%). By freezing the backbone, LP acts as a strong structural regularizer and is therefore less prone to overfitting under extreme data scarcity. This highlights one central trade-off motivating LoRA: FFT offers greater adaptation capacity but becomes unstable when data is limited, whereas LP remains robust but lacks the flexibility needed to fully adapt the representation.

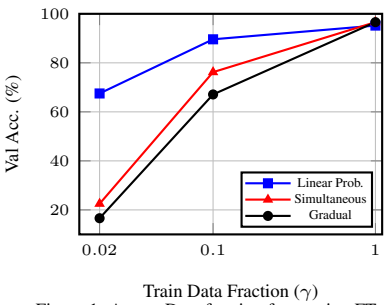

Figure 1: Acc vs Data fraction for varying FT.

5.2 LoRA on CNNs: Mechanics and Trade-offs

LoRA hyperparameter dynamics. To adapt LoRA for CNNs, we searched its corresponding hyperparameter space. On the validation set, the optimal configuration achieved parity with FFT (96.6% val accuracy) utilizing $\eta_{\text{LoRA}} = 0.001$, $r = 8$ and a *targeted* injection with depth $k = 3$. While $r = 1$ achieves strong parameter efficiency with minimal accuracy reduction (95.8% vs. 96.6%), expanding to $r = 8$ increases capacity without computational penalty. Since NVIDIA hardware optimizes matrix dimensions in multiples of 8, smaller ranks are automatically padded [20]. Crucially, the *targeted* strategy (restricting LoRA to 3×3 convolutions) outperformed the *general* baseline on the validation set; freezing the 1×1 channel-mixers regularizes the network against fine-grained overfitting. However, the held-out test set revealed that this validation advantage did not fully transfer: both evaluated LoRA variants reached 92.2% test accuracy, below the selected gradual FT baseline at 93.8%. Hence, on Oxford Pets, LoRA should be interpreted as competitive and substantially more parameter-efficient, but not fully equivalent to the best FFT configuration in final generalization. Gradual unfreezing LoRA adapters degraded performance: The constantly shifting loss landscape introduced gradient instability, making a simultaneous activation of all LoRA injections the better strategy.

Fine-Tuning Strategy	**Trainable Params**	**Val. Acc.**	**Test Acc.**
Linear Probing	75.8 K	95.0%	92.8%
Gradual FT ($k = 1, \tau = 3$)	15.0 M	**96.6%**	**93.8%**
LoRA Gen. ($r = 8, k = 2$)	1.29 M	95.8%	92.2%
LoRA Targ. ($r = 8, k = 3$)	711 K	**96.6%**	92.2%

Table 2: ResNet101: LoRA vs FFT/LP.

Resource paradox and train time. While LoRA successfully competes in accuracy, its application in CNNs exposes a contradiction between parameter efficiency and memory overhead. While trainable parameters can be reduced by 95.3% when comparing optimal configurations (see Table 2), the corresponding VRAM savings are more nuanced. For a fixed injection depth, LoRA can reduce memory usage relative to FFT because it lowers $V_{\text{gradients}}$ and $V_{\text{optimizer}}$. However, these gains remain limited on CNNs because high-resolution feature maps make $V_{\text{activations}}$ the dominant term in eq. 4. In practice, the best-performing LoRA configurations required deeper injections, and these additional adapters forced the Autograd engine to cache more large spatial activations during the forward pass. As a result, the strongest targeted LoRA setting ($k = 3$) increased peak VRAM on ResNet-101 to $V_{\text{total}} = 10,425$MB, compared with $1,764$MB for FFT. Furthermore, the reduction in trainable parameters does not translate to faster per-epoch compute times (t_{epoch}), since GPUs are limited by memory bandwidth when executing many small, sequential operations. However, since LoRA acts as a strong regularizer, it often converges in fewer epochs, i.e. E_{total} is lower (see Fig. 3). Consequently, the total training time (T_{total}) (see Eq. 5) for LoRA can still be shorter despite the higher memory footprint of deeper, better-performing injections.

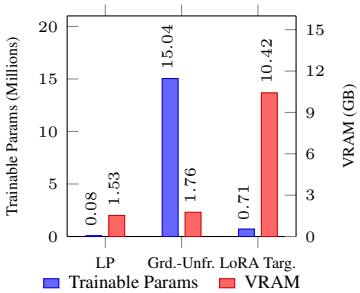

Figure 2: LoRA CNN memory Paradox.

Robustness and data starvation. Lastly, we revisit the data starvation problem. As hypothesized, the parameter efficiency of LoRA lessens the NN's vulnerability to overfitting. Under starvation ($\gamma = 0.02$), FFT strategies suffer catastrophic memorization, with validation accuracy collapsing (see Fig. 1). In contrast, LoRA maintains a robust 67.1% accuracy (Macro-F1: 65.9%), effectively matching the performance of the best low-data baseline, LP (67.5%). Furthermore, the optimal dynamics of LoRA flip in low-data environments. While deeper injections across multiple layers yielded peak performance on $\gamma = 1$, this trend inverts under starvation. At $\gamma = 0.1$ & $\gamma = 0.02$, restricting LoRA to $k = 1$ outperformed deeper injections (val acc.: 88.5% & 67.1%). This indicates that the restricted sample size is insufficient to reliably train deeper representations. Data augmentation's regularizing effect became significant as data scarcity increased. To sum up, LoRA on CNNs is competitive with baseline FT under full data according to validation accuracy, preserves robust performance during extreme starvation, and offers tunable capacity via targeted injection depth, although the final Oxford Pets test set still favors gradual FFT.

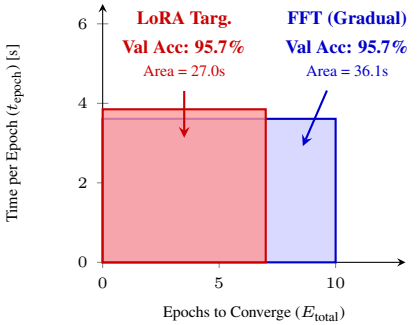

Figure 3: Exemplary T_{total} comparison between two runs with similar validation accuracies

5.3 LoRA generalization to MS COCO and ViTs

MS COCO. On the more complex MS COCO dataset, LP fails to scale effectively, reaching an mAP of 69.83% on ResNet-101. Here, mAP denotes the multi-label image classification metric on the held-out test split, not standard detection mAP over bounding boxes. FFT remains strongest, with simultaneous and gradual unfreezing reaching 77.26% and 77.22%, matching the expected performance [13, p. 6], while LoRA ($r = 8$, $k = 1$) transfers well and reaches 76.26%. The performance degradation of LP indicates underfitting. Conversely, FFT strategies and LoRA maintain the capacity required for complex feature adaptation.

ViTs on Oxford Pet. On ViT-B/16, a separate hyperparameter search was performed to identify the best configurations by validation accuracy for the different strategies. Simultaneous unfreezing ($k = 4$) performed best with a test accuracy of 93.8% followed by gradual unfreezing ($k = 3$, $\tau = 1$) with 93.5%. LoRA ($r = 8$, $k = 3$) reached 93.0%, outperforming LP (92.3%) while converging in less compute time than FFT.

ViTs Under Data Starvation. Although pre-trained ViTs learn powerful representations, their highly flexible architecture remains susceptible to catastrophic forgetting on limited data. At a 2% split, simultaneous and gradual unfreezing collapsed to 59% test accuracy as representations overfit. Conversely, LoRA effectively protected the pre-trained weights, maintaining 80.8% and outperforming linear probing (80.5%).

6 Conclusion

This work evaluated how LP, FFT and LoRA behave under constrained adaptation on CNNs and ViTs. On Oxford-IIIT Pet with full data, the best final results were still obtained by FFT, while LoRA remained competitive with fewer trainable parameters. Under data scarcity, however, FFT became unstable, whereas frozen-backbone methods generalized much more reliably; LP was the strongest CNN baseline and LoRA preserved much of this robustness while offering greater adaptation capacity.

The transfer experiments showed that LoRA remains effective beyond the initial setting. On MS COCO, it stayed close to the best FFT baselines, and on ViTs it became especially attractive under strong data scarcity, where it clearly outperformed FFT. Finally, although LoRA reduced trainable parameters substantially, its VRAM savings on CNNs were modest because activations dominated memory usage, and the best-performing configurations required deeper injections that offset the gain. For improving performance, future work could investigate more architecture-aware LoRA designs for CNNs, since the remaining gap to FFT suggests that standard low-rank adapters may not fully capture the spatial adaptation required in convolutional features. When memory savings are critical, gradient checkpointing appears to be a promising complement to LoRA, since it directly targets the activation memory cost that dominated our experiments.

References

[1] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *CoRR* abs/2106.09685 (2021). arXiv: 2106.09685. URL: https://arxiv.org/abs/2106.09685.

[2] Ananya Kumar et al. *Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution*. 2022. arXiv: 2202.10054 [cs.LG]. URL: https://arxiv.org/abs/2202.10054.

[3] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014. URL: https://arxiv.org/pdf/1411.1792.

[4] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. 2018. URL: https://arxiv.org/pdf/1801.06146.

[5] Neil Houlsby et al. "Parameter-Efficient Transfer Learning for NLP". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2790–2799. URL: https://arxiv.org/pdf/1902.00751.

[6] Yuren Mao et al. "A survey on LoRA of large language models". In: *Frontiers of Computer Science* 19.7 (2025). Cited by: 73; All Open Access, Green Open Access, Hybrid Gold Open Access. DOI: 10.1007/s11704-024-40663-9. URL: https://doi.org/10.1007/s11704-024-40663-9.

[7] Wu Ran et al. "Correlated Low-Rank Adaptation for ConvNets". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025. URL: https://openreview.net/forum?id=3pF7rt9fQM.

[8] Zihan Zhong et al. "Convolution Meets LoRA: Parameter Efficient Finetuning for Segment Anything Model". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=ezscMer8L0.

[9] Sidra Aleem et al. *ConvLoRA and AdaBN based Domain Adaptation via Self-Training*. 2024. arXiv: 2402.04964 [cs.CV]. URL: https://arxiv.org/abs/2402.04964.

[10] Menglin Jia et al. "Visual Prompt Tuning". In: *European Conference on Computer Vision*. Springer. 2022, pp. 709–727. URL: https://arxiv.org/abs/2203.12119.

[11] Omkar M Parkhi et al. "Cats and dogs". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3498–3505. URL: https://www.robots.ox.ac.uk/~vgg/data/pets/.

[12] Noman Khan et al. "Head-Body Guided Deep Learning Framework for Dog Breed Recognition". In: *Computers, Materials & Continua* 85.2 (2025), pp. 2935–2958. DOI: 10.32604/cmc.2025.069058. URL: https://doi.org/10.32604/cmc.2025.069058.

[13] Zhao-Min Chen et al. "Multi-Label Image Recognition with Graph Convolutional Networks". In: *CoRR* abs/1904.03582 (2019). arXiv: 1904.03582. URL: http://arxiv.org/abs/1904.03582.

[14] TorchVision Contributors. *Transforming and augmenting images — TorchVision Documentation*. Accessed: 2026-05-19. 2026. URL: https://docs.pytorch.org/vision/main/transforms.html.

[15] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778. URL: https://arxiv.org/pdf/1512.03385.

[16] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2021). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[17] PyTorch Contributors. *WeightedRandomSampler — torch.utils.data*. PyTorch 2.12 documentation, accessed 2026-05-18. 2025. URL: https://docs.pytorch.org/docs/2.12/data.html#torch.utils.data.WeightedRandomSampler.

[18] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[19] PyTorch Contributors. *torch.cuda — PyTorch Documentation*. Accessed: 2026-05-18. 2024. URL: https://pytorch.org/docs/stable/cuda.html.

[20] NVIDIA Developer Blog. *Optimizing GPU Performance with Tensor Cores*. Accessed: 2026-05-20. 2019. URL: https://developer.nvidia.com/blog/optimizing-gpu-performance-tensor-cores/.

A Appendix: Pseudocode

This section provides the complete algorithmic formulation for the gradual unfreezing scheduling strategy.

Algorithm 1: Gradual unfreezing FT schedule

Input: Pre-trained params θ, Total epochs T, Unfreeze interval τ, Target stage depth k
Output: Adapted parameters θ^*

Initialize $\theta^{(0)}_{\text{trainable}} \leftarrow \theta_{\text{head}}$
Initialize current unfreeze depth $d \leftarrow 1$
for $t = 1$ **to** T **do**
  if $t \bmod \tau == 0$ ***and*** $d \leq k$ **then**
    $\theta^{(t)}_{\text{trainable}} \leftarrow \theta^{(t-1)}_{\text{trainable}} \cup \theta_{S_{5-d}}$ `/* Unfreeze next deeper stage */`
    $d \leftarrow d + 1$
  Compute cost J on current batch
  $\theta^{(t)}_{\text{trainable}} \leftarrow \theta^{(t)}_{\text{trainable}} - \eta_t \nabla J$ `/* Update unfrozen weights */`