DTL: Parameter- and Memory-Efficient Disentangled Vision Learning

Minghao Fu, Ke Zhu, Zonghao Ding and Jianxin Wu, *Member, IEEE*

Abstract—The cost of finetuning a pretrained model on downstream tasks steadily increases as they grow larger. Parameter-efficient transfer learning (PETL) is proposed to reduce this cost by changing only a tiny subset of trainable parameters. But, the GPU memory footprint during training is *not* effectively reduced in PETL. This issue happens because trainable parameters from these methods are generally tightly entangled with the backbone, such that a lot of intermediate states have to be stored for back propagation. To alleviate this issue, we introduce Disentangled Transfer Learning (DTL), which disentangles the trainable parameters from the backbone using a lightweight Compact Side Network (CSN). By progressively extracting task-specific information with a few low-rank linear mappings and appropriately adding the information back to the backbone, CSN effectively realizes knowledge transfer in various downstream recognition tasks. We further extend DTL to more difficult tasks such as object detection and semantic segmentation by employing a more sparse architectural design. Extensive experiments validate the effectiveness of DTL, which not only reduces a large amount of GPU memory usage and trainable parameters, but also outperforms existing PETL methods by a significant margin in accuracy.

Index Terms—Transfer Learning, Efficient Training, Image Recognition, Object Detection.

1 Introduction

Large-scale pretraining followed by finetuning has been popularized in various domains [1], [2]. But traditional finetuning can be intractable due to GPU memory or time budget [2], since parameters of the entire large model have to be updated. Recently, parameter-efficient transfer learning (PETL) is proposed to update only a tiny subset of trainable parameters [3]. Because of its efficacy, numerous variants [4], [5], [6], [7], [8] of PETL successively emerged.

Nevertheless, a huge decrease in trainable parameters does *not* necessarily translate into an equivalent reduction in GPU memory usage. As shown in Fig. 1, existing PETL methods reduce trainable parameters by over 90%, yet the actual GPU memory savings remain below 30% on average. Large pretrained models can still run out of memory during finetuning. Hence, it is critical to develop methods that address both parameter and memory efficiency.

One reason for this gap is that all PETL baselines entangle their trainable modules tightly with the frozen backbone. During back-propagation, gradients flow through every frozen layer, which forces the framework to cache intermediate activations and their derivatives (i.e., gradients) for almost every layer. As a result, even a *lightweight* trainable module incurs substantial activation-cache overhead, limiting the overall GPU memory reduction.

We formalize this with a chain-rule derivation in Sec. 3, showing that activation storage, rather than the number of updated weights, dominates the GPU memory footprint. The empirical breakdown in Fig. 2 confirms that optimizer and gradient tensors shrink under PETL, whereas activation-related memory remains the bottleneck.

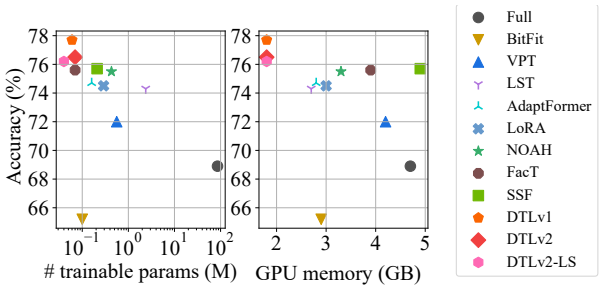


Figure 1: Top-1 accuracy on VTAB-1K [9] vs. different numbers of trainable parameters and GPU memory footprint. Our DTL achieves the highest accuracy with the least trainable parameters and smallest GPU memory usage.

To address this issue, we propose Disentangled Transfer Learning (DTL), which disentangles update of learnable weights from the backbone network by proposing a lightweight Compact Side Network (CSN). DTL not only greatly reduces GPU memory footprint, but also achieves high accuracy in knowledge transfer (cf. Fig. 1).

As shown in Fig. 3, our CSN composes of several low-rank linear mapping matrices to extract task-specific information, which is completely disentangled from the backbone. By injecting this information back to a few *late* backbone blocks using an additional global depthwise separable convolution (DWConv) [10], features generated by the pretrained model are adaptively calibrated to be more discriminative for downstream tasks. DTL is very simple and compatible with various backbone architectures.

The output of early blocks in the backbone (covered by the gray region in Fig. 3) is kept constant during finetuning, making it possible to reuse backbone features across *multiple downstream tasks* when the same input is provided.

- *All authors are with the National Key Laboratory for Novel Software Technology, Nanjing University, China and the School of Artificial Intelligence, Nanjing University, China. J. Wu is the corresponding author. E-mail: {fumh, zhuk, dingzh}@lamda.nju.edu.cn, wujx2001@gmail.com*
- *This research was partly supported by the National Natural Science Foundation of China under Grant #62276123 and #61921006.*

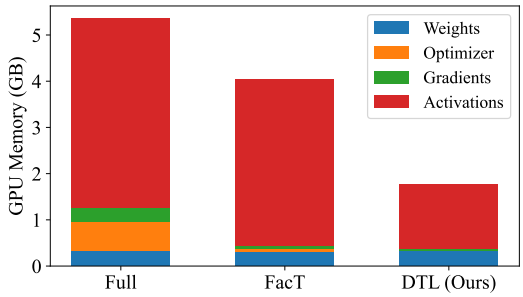


Figure 2: Per-iteration training GPU memory breakdown. Full finetuning incurs substantial overhead. PETL variants such as FacT [8] reduce optimizer and gradient memory demands but remain activation-intensive, which limits the overall GPU memory savings. In contrast, our DTL drastically reduce the requirements for activation caching thanks to its disentangled nature.

We further explore our DTL design for sophisticated tasks such as object detection and semantic segmentation, in which the simplest version of DTL (DTLv1) exhibits inferior performance. We then develop a more effective disentangled architecture (DTLv2), which successfully bridges the performance gap.

Our contributions can be summarized as follows:

- We analyze limitations of existing PETL methods from the perspective of GPU memory usage, which has a critical influence on the viability of finetuning.
- Motivated by our analysis, we propose DTL, a disentangled and simple framework for efficiently finetuning large-scale pretrained models with significantly less trainable parameters and GPU memory usage.
- Through empirical observations and analyses, we successfully extend DTL to dense prediction tasks.

We conducted extensive experiments to verify the effectiveness of DTL, which achieved superior accuracy with significantly less trainable parameters and GPU memory during finetuning compared to its traditional PETL counterparts.

A preliminary version describing DTLv1 was published as [11]. The code for both DTLv1 and DTLv2 are publicly available at https://github.com/heekhero/DTL.

2 Related Work

Classical methods adapt a pretrained backbone by freezing the majority of its parameters or applying model compression [12], [13], [14]. Early work in feature extraction freezed the backbone and trained only the final classifier, a strategy commonly employed in transfer learning [12]. Knowledge distillation [15], [16] transfers knowledge from a larger teacher model to a smaller student model by matching soft output distributions or intermediate feature representations. While both approaches alleviate computational and memory burdens, they are not parameter-efficient. We then focus on discussing parameter-efficient methods.

2.1 Additive PETL

Additive parameter-efficient transfer learning methods introduce small, lightweight modules into pretrained backbones

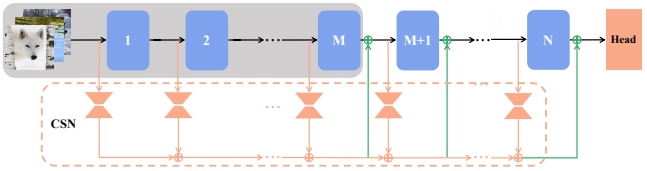


Figure 3: DTLv1 architecture. A Compact Side Network (CSN) with low-rank linear mappings is inserted parallel to backbone blocks. Only starting from the M-th block, side information is added back to adapt features.

for efficient model adaptation. Adapter [3], [17] architectures inserted bottleneck modules around attention and feed-forward layers. Recent ones extended this concept across modalities, including text-to-image [18], [19], image-to-video [20], and adaptive image/video generation [21].

Vision-language adapters [22], [23] aligned modalities with minimal overhead, while LLaMA-Adapter [24], [25] employed zero-initialized attention modules for rapid adaptation. Further refinements include convolutional adapters [26], side adapters for open-vocabulary segmentation [27], sparse adapters [28], and token-dependent adapters [29]. Moreover, additive adapters have been adapted to dense prediction tasks [30], [31], [32]. They tailored low-rank modules or leveraged sparse adapters on detection and segmentation.

2.2 Reparameterization PETL

Reparameterization PETL methods finetune models by compactly decomposing existing parameters. After finetuning, they merge additional learned parameters back into the model so that no extra inference overhead is introduced. LoRA [5] employed low-rank decompositions

$$X' = XW + XAB, \tag{1}$$

where $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$ are low-rank ($r \ll d$) to achieve parameter-efficiency. Variants like AdaLoRA [33], ARD-LoRA [34], Delta-LoRA [35], and SuperLoRA [36] further optimized rank allocation and model flexibility.

In computer vision, RepAdapter [37] and Consolidator [38] utilized structural and grouped reparameterizations, while ALoRE [39] aggregated low-rank expert adapters. MetaTT [40] applied global tensor-train structures, and CoPA-Merging [41] efficiently merged multimodal parameters.

Additional methods include bias-tuning via BitFit [42], tensor-factorization with FacT [8], and scaling-shifting features (SSF) [7]. Overall, these methods effectively reduced tuning complexity while retaining accuracy.

2.3 Prompt PETL

Prompt-based PETL methods adapt a pretrained model by inserting a small set of learnable prompt embeddings into the input context. During finetuning, only these prompt tokens are optimized, preserving parameter efficiency. VPT [4] prepended prompts to image patches within intermediate backbone layers. E2VPT [43] and VQT [44] improved efficiency through compact parameterization and intermediate feature querying. SRP [45] maintained foundational knowledge via self-distillation. In vision-language tasks,

CPL [46], DAPT [47], and KgCoOp [48] generated context-aware embeddings guided by class cues, data distributions, or external knowledge.

3 Limitations of Current PETL Methods

But, despite the significant reduction in trainable parameters achieved by PETL methods, the decrease in GPU memory usage is not proportional, rendering the finetuning of large-scale pretrained models still challenging.

Suppose there is an N layer feed-forward network $y = f_N(f_{N-1}(...f_1(x)))$, where layer i is composed of a weight matrix W_i and a bias term b_i. We denote o_{i+1}, z_{i+1} as the output and pre-activation of layer i, respectively. Then, $o_{i+1} = \sigma(z_{i+1}) = \sigma(W_i o_i + b_i)$, where σ is the activation function. [49] shows that the gradients back propagated from the loss L to W_i and b_i are

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial o_{i+1}} \sigma_i' o_i \,, \qquad \frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial o_{i+1}} \sigma_i' \,, \tag{2}$$

where σ_i' is the abbreviation of $\partial o_{i+1}/\partial z_{i+1}$. Furthermore, the term $\partial L/\partial o_{i+1}$ can be recursively expressed as

$$\frac{\partial L}{\partial o_{i+1}} = \frac{\partial L}{\partial o_{i+2}} \frac{\partial o_{i+2}}{\partial z_{i+2}} \frac{\partial z_{i+2}}{\partial o_{i+1}} = \frac{\partial L}{\partial o_{i+2}} \sigma_{i+1}' W_{i+1} \,. \tag{3}$$

To calculate the gradients, except for parameters from the model (in this case, W_i and b_i), *all corresponding $\{\sigma_i'\}$ in the chain rule have to be cached during finetuning*, which dominates the GPU memory usage.

Despite the small number of trainable parameters, *all* existing PETL methods remain tightly entangled with the backbone and thus fail to reduce GPU memory footprint. For instance, LoRA [5] injects low-rank updates $\Delta W = AB$ directly into each Transformer [50] block's query and value projections. Adapter [3] inserts bottleneck modules after both multi-head self-attention and feed-forward layers. SSF [7] applies per-channel affine scale and shift at every normalization layer. In each case, these additional modules operate on the same activations used by the backbone, *requiring storage of the corresponding σ_i' terms in nearly all backbone activations for back-propagation.* As a result, the activation-caching cost remains essentially the same as full finetuning, even though only a fraction of parameters is updated (see Fig. 2).

4 DTLv1: Starting from a Simple Design

To solve this difficulty, we propose Disentangled Transfer Learning (DTL). The core idea of DTL is to *disentangle* the weight updating of the small extra modules from the backbone network (cf. Fig. 3). Therefore, the σ_i' stored for back propagation can be drastically reduced (cf. Eq. 2 and 3). DTL is then not only parameter-efficient but also significantly reduces GPU memory requirements in finetuning large-scale pretrained models.

We begin by presenting the simplest instantiation of our DTL framework, denoted as DTLv1 [11]. As illustrated in Fig. 3, DTLv1 integrates a Compact Side Network (CSN) in parallel with backbone blocks. At each block CSN applies a low-rank linear mapping [5] to the input features to extract task-specific side information (orange arrows). Starting from block M, this side information is added back to the block outputs (green arrows) to adapt the backbone representations for downstream tasks. During finetuning, only the parameters from CSN and the task-specific classification head are updated (shown in orange).

Specifically, given a ViT backbone with N blocks, the forward computation can be formulated as $z = b_N(b_{N-1}(...b_1(x)))$, where b_i is the i-th block, $x \in \mathbb{R}^{(n+1)\times d}$ is the input tokens (patch tokens plus one *cls* token) and $z \in \mathbb{R}^{(n+1)\times d}$ is the output tokens, respectively. Denote z_{i+1} as the output of b_i, hence $z_{i+1} = b_i(z_i)$ and $z_1 = x$. Our CSN composes of N low-rank linear transformation matrices [5], with each being plugged into one block to extract task-specific information. Denote $w_i = a_i c_i \in \mathbb{R}^{d\times d}$ as the weight matrix accounting for the i-th block, with $a_i \in \mathbb{R}^{d\times d'}$, $c_i \in \mathbb{R}^{d'\times d}$ and $d' \ll d$, CSN progressively gathers information from each block as

$$h_{i+1} = h_i + z_i w_i \,, \tag{4}$$

where h_{i+1} is the output of the i-th layer of CSN ($h_1 = 0$).

After that, starting from the M-th block, the aggregated task-specific information h_{i+1} is used to adapt z_{i+1} to downstream tasks by adding it back to z_{i+1}. Specifically,

$$z'_{i+1} = z_{i+1} + g(\theta(h_{i+1})) \text{ when } i \geq M \,, \tag{5}$$

where z'_{i+1} is the adapted output of b_i, θ is the Swish activation [52] with $\theta(x) = \frac{x}{1+e^{-\beta x}}$, and g is an additional global depthwise separable convolution (DWConv) layer [10]. To prevent z'_{i+1} from drastically shifting away from z_{i+1} at the beginning of finetuning, a_i is initialized following a uniform distribution and c_i is zero-initialized. To sum up, the output from the i-th block is

$$z'_{i+1} = \begin{cases} z_{i+1} + g(\theta(h_{i+1})) & \text{if } i \geq M \\ z_{i+1} & \text{otherwise} \,. \end{cases} \tag{6}$$

Image recognition is inherently less complex than dense prediction tasks and therefore requires fewer adaptation parameters. Empirically, we find that a small d' (2 or 4) performs very well for image classification (see Sec. 7.6 for more detailed ablation on d'), which suggests high redundancy in the backbone features. Therefore, in addition to keep d' small, we use a large β (100) in Swish (i.e., θ) to further reduce the redundancy. Consequently, roughly half of $\theta(h_{i+1})$ are close to zero. The stride of g is set to 1 and zero-padding is used to ensure that g does not change feature size. By incorporating g, spatial dependencies are effectively captured and processed within the CSN, thereby enhancing the model's ability to generalize to and recognize novel categories.

The low-rank design of the CSN's linear mappings ensures that only a minimal number of parameters are updated during finetuning. To further constrain the parameter size, we share the weights of g across all CSN layers, making its overhead negligible relative to the core side network.

Aside from ViT [53], many modern architectures are organized into stages in which the feature dimensionality is constant within a stage but differs across stages (e.g., Swin [51] and ResNet [54]). By resetting the CSN hidden state h_i to zero at the start of each stage, our method naturally extends to backbones with changing feature dimensions.

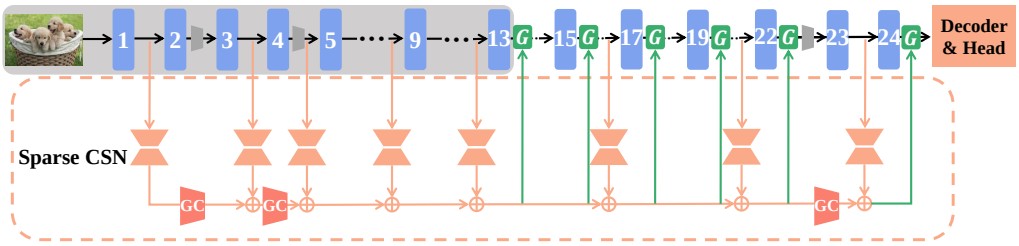


Figure 4: DTLv2 network for Swin-B/L [51]. Compared to DTLv1 we employ sparsification to remove some CSN layers and use an enhanced fusion module G to balance parameter efficiency and localization precision in dense prediction tasks.

5 DTLv2: Extending to Dense Prediction

Different from image classification, the goal of visual dense prediction tasks, such as object detection and semantic segmentation, is to recognize and localize multiple instances of interest, which naturally requires the model to capture and preserve more discriminative local information during progressive feature extraction. DTLv1 performs poorly if directly applied to these tasks (cf., Table 8)—due to its overly simple structure, the capacity of CSN is too small to capture subtle local features.

We thus design DTLv2 to enhance the ability to recognize local information *on top of pretrained features*. We primarily focus on Swin-B/L [51], a hierarchical backbone consisting of 24 blocks. As shown in Fig. 4, DTLv2 employs a sparsification strategy by applying low-rank transformations to a small subset of backbone blocks. The aggregated features are reintegrated into the backbone through the enhanced fusion module G. This approach reduces the number of trainable parameters and improves the parameter efficiency. DTLv2 is compatible with a variety of backbone architectures. In the following, we first present the enhanced fusion module G and then the sparsification of the CSN layers.

5.1 Enhance the Fusion Module

CSN involves two steps. First, it aggregates features from the backbone with low-rank transformation w_i. Then, this aggregated information is reintegrated into the backbone in the last few blocks. In DTLv1, this fusion consists of a non-linear activation function θ and a *shared* depthwise separable convolution layer g (cf., Eq. 6). This simple fusion approach obviously lacks ability to capture information from multiple instances in dense prediction tasks.

We need a more effective module to resolve this problem. Suppose h_{i+1} is the output of the i-th layer of CSN, our new feature fusion module (abbreviated as G) is

$$G(h_{i+1}) = \theta(g(h_{i+1}))w'_i + h_{i+1}\,, \tag{7}$$

where g, θ, and h_{i+1} are defined as in DTLv1 (Eq. 5), and w'_i is another new low-rank linear transformation. Overall, the output from the i-th block in the backbone within DTLv2 is

$$z'_{i+1} = \begin{cases} z_{i+1} + G(h_{i+1}) & \text{if } i \geq M \\ z_{i+1} & \text{otherwise}\,, \end{cases} \tag{8}$$

which is shown in Fig. 4 as the green box with a G inside it.

Comparing Eq. 6 and equations 8 and 7, the task-specific information is changed from $g(\theta(h_{i+1}))$ in DTLv1 to $\theta(g(h_{i+1}))w'_i + h_{i+1}$ in DTLv2. There are two differences: the additional low-rank linear transform w'_i and the extra residual connection ("$+h_{i+1}$").

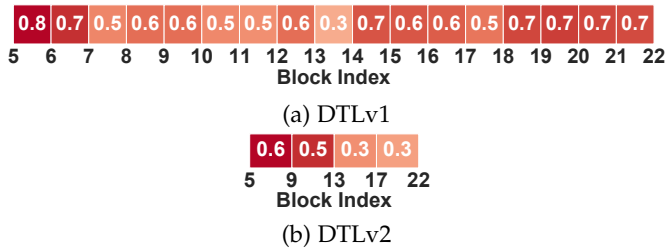
The additional low-rank linear transformation w'_i in G effectively aggregates information when multiple instances of interest occur in a single input image. Moreover, thanks to the depthwise convolution and low-rank transformation, the number of trainable parameters in w'_i is small, which makes G highly parameter-efficient. Furthermore, although in DTLv1 g is shared across different CSN layers, we also study the possibility of assigning different G to different CSN layers, which will be discussed soon.

5.2 Sparsify CSN for Parameter Efficiency

When the weights of feature fusion module G are not shared across different CSN layers, the method loses parameter efficiency and the number of trainable parameters increases substantially (cf. Table 11). This drawback can be overcame by sparsify CSN.

Sparsify low-rank linear transformation w. We first explore whether CSN's low-rank transforms $\{w_i\}$ exhibit substantial redundancy: if for any output z_i, z_j from different blocks in backbone, the corresponding outputs from w_i and w_j in CSN satisfy $z_i w_i \approx z_j w_j$, then their combined effect can be approximated as $z_i w_i + z_j w_j \approx 2 z_i w_i$, implying that CSN could reproduce both contributions by scaling w_i rather than maintaining separate parameters for w_j. To empirically validate this observation, we process the entire MS-COCO [55] training set through our trained DTL network with a Swin-B [51] backbone. For each transform w_i, we collect its output feature vectors from all images and average them to obtain a single mean feature vector. We then measure the cosine similarity scores between the mean vectors of adjacent w. As shown in Fig. 5a, these scores exhibit high values, with an average above 0.6 and even reaching 0.7-0.8 in early or late layers. This confirms the high redundancy among adjacent w and motivates dropping some of them.

Based on this observation, we drop some CSN modules to form a sparser CSN architecture (cf. Fig. 4). Specifically, we insert a low-rank transformation w approximately every m blocks. Given that a hierarchical backbone generates features with varying dimensionality across different stages, we apply this insertion rule within each stage to ensure that w is evenly distributed. By default, we keep $m = 4$, which yields the architecture in Fig. 4, where the input features of block 2, 4, 5, 9, 13, 17, 22, 24 are sampled as the input to CSN. We also check the cosine similarity of DTLv2



(a) DTLv1

(b) DTLv2

Figure 5: Cosine similarity of output features across adjacent w. The block indices are from 5 to 22 to ensure that the feature dimensionalities from different blocks are compatible.

after w is sparsified. According to Fig. 5b, compared to Fig. 5a, the cosine similarity of features (or redundancy) from adjacent low-rank transformations is significantly decreased, especially for late blocks.

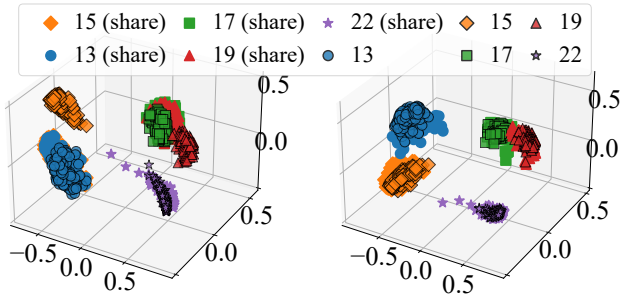
Sparsify enhanced feature fusion G. Following the same strategy, we propose a similar sparsification to the enhanced feature fusion module G. Starting from the middle block of the backbone (i.e., 13), the aggregated features from CSN are added back through G approximately every n blocks. In this case, we set $n = 2$, resulting in the output features of blocks 13, 15, 17, 19, 22 and 24 being adapted for downstream tasks, as shown in Fig. 4.

Linear adaptation for shared G. After sparsification, DTLv2 already uses very few trainable parameters. We now ask whether we can push parameter efficiency further by using a single fusion module G for multiple CSN layers.

Under weight sharing, the feature-fusion capacity of G is inherently constrained. To quantify this effect, we apply PCA to the output features of G across different backbone blocks under two configurations: one in which each CSN layer uses its own independent G (non-shared baseline), and another in which G's parameters are tied across layers (shared-weight variants). Figure 6 shows the resulting projections for two sharing strategies: (1) *Direct Share*, where G reuses a single set of parameters across all target backbone blocks; and (2) *L-Share*, where the shared core is augmented with per-layer scale γ_i and shift β_i parameters as defined in Eq. 9. For any block index k, the label "k" indicates that the visualized features are produced by a fusion module G for block k, with each block employing its own independent G (no weight sharing). Conversely, the label "k (share)" indicates that a single fusion module G with shared parameters is reused across all targeted blocks.

When G is directly shared, the only variables that can cause differences in the output of G are the inputs themselves. However, G for some blocks share the same input, i.e., 13 and 15, 17 and 19 (cf., Fig. 4), so the visualized features labeled "13 (share)" and "15 (share)" as well as "17 (share)" and "19 (share)" coincide completely in the left part of Fig. 6. We expect the outputs for each block before and after weight sharing to align closely, for example the features "13" and "13 (share)". The left part of Fig. 6 confirms that even when G is directly shared, the output features of G for some blocks (i.e., 13, 17, 22) are well aligned between the shared and non-shared scenarios, but there is still a significant shift for blocks 15 and 19.

We attribute this misalignment to the rigid weight tying of



(a) Direct Share (b) L-Share

Figure 6: PCA visualization of output features from G for different blocks. The plots compare two different sharing strategies: when G is directly shared (left), or our L-Share (right). L-share better aligns features between the shared and non-shared scenarios. Best viewed in color and zoomed-in.

direct sharing, which deprives G of the flexibility to relocate its outputs to the correct regions of feature space for each block. To overcome this drawback, we propose Linear Share (L-Share), which adaptively inserts scale and bias terms for the shared G when G share the same input from CSN. The computation of L-Share is

$$G'(h_{i+1}) = \begin{cases} G(h_{i+1}), & i = \min \mathcal{B}(i), \\ G(h_{i+1}) \odot \gamma_i + \beta_i, & \text{otherwise}, \end{cases} \tag{9}$$

where $\odot$ denotes element-wise multiplication, $\gamma_i, \beta_i \in \mathbb{R}^d$ are learnable scale and bias, and $\mathcal{B}(i)$ denotes the set of indices of blocks that share the same CSN input as block i (ordered by the forward pass), with $\min \mathcal{B}(i)$ its first index. This design ensures that, within any group where G shares the same input (e.g., 13 and 15), the scale and bias terms are introduced only for the subsequent blocks, not for the first.

We find that this technique makes the fused features added back to the backbone more discriminative and better aligned with the non-shared version. As illustrated in the right part of Fig. 6, our L-Share successfully resolves the shift problem for block 15 and 19. Experimental results in Table. 8 also demonstrate the superiority of L-Share (termed as DTLv2-LS[1]). Compared to DTLv2, the L-Share strategy reduces the number of trainable parameters by approximately 26% without compromising the AP.

5.3 Unify CSN with Different Dimensionalities

In DTLv1, for backbone structures comprising multiple stages with different feature dimensionalities (e.g., Swin Transformer [51]), direct feature aggregation in CSN via simple addition is not possible. To circumvent this issue, we reinitialize h_i to 0 at the beginning of each stage, thus avoiding dimensionality mismatch. But, this approach renders the inability to aggregate information across different stages, thereby creates a few separate CSNs.

We need to integrate separate CSNs into a unified one in a parameter- and memory-efficient manner. In the backbone, each downsampling operation within a stage reduces the spatial dimensions of the features by half and doubles

1. We refer to our method as DTL and denote specific variants as DTLv1, DTLv2, or DTLv2-LS.

the number of feature channels. Therefore, the connection between different CSNs must follow this rule.

The most intuitive approach is to reuse the downsampling module directly from the backbone. However, due to the differences between CSN features and backbone features, this approach can impede the CSN feature aggregation process and degrade performance. An alternative approach is to use a 2×2 convolution as a bridge. While effective, this method introduces a significant number of trainable parameters. Another option is to employ a depthwise separable convolution, which is more parameter-efficient. However, our experiments indicate that this method performs suboptimally. We propose to use a group convolution (GC) as a compromise. Our experiments demonstrate that this approach performs well even with a large number of groups (up to 1/4 of the input channels), cf. Table 15.

6 Advantages of Disentangled Learning

The proposed approach DTL has some significant advantages, which we discuss explicitly.

Disentangled. As shown in Figures 3 and 4, CSN is a plugin module that is mostly detached from the backbone. This characteristic makes our method easy to implement, and is *compatible with almost all backbone networks*. Modern networks are composed of repeated blocks with identical structure. For example, ViT [53] and Swin [51] use blocks built from Attention and MLP layers. ResNet uses groups of convolutional layers. When successive blocks produce feature maps of the same spatial dimensions, our CSN addition can be applied and DTL can finetune these backbones (for mismatched dimensions, see Sec. 5.3).

As to GPU memory, although the number of trainable parameters is small in previous PETL methods, they still require a lot of GPU memory to cache many $\{\sigma_i'\}$. Our DTL alleviates this issue by separating the forward pass of the backbone from CSN, and by only entangling them at late stages ($i \geq M$). *No gradients are back propagated to the first M blocks in the backbone* (the gray region in Fig. 3 and 4). Hence, the number of cached $\{\sigma_i'\}$ is highly reduced in CSN, which results in large GPU memory reduction.

Finally, there is another advantage of our disentangled architecture: possible feature reuse. Consider a scenario where we need to perform different tasks on one input image. We have several finetuned models, one for each task. In previous PETL methods, the intermediate features z_{i+1} in these finetuned models are *different to each other*, and there is no way to share computation in the backbone across different tasks. Therefore, the standard process is learning many task-specific parameters and conducting each task individually. Conversely, in our DTL the intermediate features before block M remain the same after finetuning, such that we can share part of the backbone computation between different tasks. Please refer to Sec. 7.6 for more details on feature reuse.

Simple. Since the CSN is disentangled from the backbone network, our method naturally shows higher simplicity compared to previous methods. Since PETL methods add various types of structural units as extra trainable parameters, we compare the number of such minimal structural units in Table 1.

Table 1: Number of minimal structural units in different methods. "Source" and "#unit" denote the types and number of minimal structural units. The backbone is ViT-B/16.

Method	Source	#unit
LoRA	low-rank matrices in W_q, W_v	24
NOAH	low-rank matrices, bottlenecks, prompts	36
FacT	decomposed tensors	144
SSF	pairs of γ, β	148
DTL	low-rank matrices, DWConv, (w')	13-14

The phrase *minimal structural unit* means the atomic modules inserted into the backbone network. For example, in LoRA [5], one minimal structural unit comprises of a pair of A and B matrices to constitute ΔW (cf. Eq. 1). Since it inserts ΔW into both W_q and W_v for MHSA in every Transformer block, the total number of these units is 24. It is similarly defined for other methods as well, which include: pairs of γ and β in SSF, the modules to be searched in supernet and maintained in subnet in NOAH, decomposed tensors in FacT, pairs of matrices a_i, c_i with the DWConv layer in DTLv1, as well as additional low-rank mapping w' in enhanced fusion module G of DTLv2 variants. As shown in Table 1, the proposed method requires much fewer minimal structural units compared to existing methods.

7 Experiments on Image Recognition

We conducted a comprehensive evaluation of our proposed method. First, we report the performance on large-scale recognition benchmark, ImageNet-1K [56], leveraging architectures from three backbone families: Vision Transformer [53], Swin Transformer [51], and convolutional neural networks. This is followed by an assessment of the method's adaptability using the FGVC [4] and VTAB-1K [9] benchmarks, which are widely used for evaluating PETL methods. Additionally, we demonstrate the generalization capabilities of our approach in the context of few-shot learning and domain generalization. Ablation studies are also presented.

7.1 Evaluation Settings

Datasets. We evaluate on ImageNet-1K [56], FGVC [4], VTAB-1K [9], five fine-grained few-shot [8] and domain generalization [6] benchmarks. Full dataset descriptions and protocols are provided in the appendix.

Implementation Details. We use AdamW [58] with a cosine learning rate schedule. Unless otherwise specified, the backbone is pretrained on ImageNet-21K [56]. For the low-rank linear mappings in the CSN, the rank d' is set to 2 for ViT-B/16 and 4 for other backbones. We set M (cf. Eq. 6 and 8) for all DTL variants to 7 in the ViT-B/16 backbone. It is similarly defined for other backbones with roughly half of the layers adapted accordingly. As the backbone becomes larger, the number of adapted blocks can be further reduced. Please refer to the appendix for more implementation details.

Baseline methods. First, two traditional finetuning techniques are included in all experiments. One is 'Full', which finetunes the entire pretrained model. The other is 'Fixed', which only finetunes task-specific classification head. Second, we choose Bias [42], VPT [4], LST [49], Adapter [3], AdaptFormer [17], LoRA [5], NOHA [6], FacT [8], SSF [7], ARC [57]

Table 2: Results on ImageNet-1K. "#p": number of trainable parameters (M), '#m': GPU memory usage during finetuning with batch size 256 (GB). The best results are in boldface.

Backbone	Method	#p	#m	Top-1
ViT-B/16	Full	85.8	34.3	83.6
	Fixed	0	3.1	82.0
	VPT	0.46	32.1	82.5
	Adapter	0.17	23.6	82.7
	Bias	0.10	23.2	82.7
	SSF	0.15	37.3	83.1
	DTLv1	0.06	13.2	82.2
	DTLv2	0.07	13.2	83.7
	DTLv2-LS	**0.04**	**13.2**	**83.7**
Swin-B	Full	86.7	47.6	85.2
	Fixed	0	5.6	83.3
	VPT	0.60	73.6	83.4
	Adapter	0.22	32.3	83.8
	Bias	0.20	31.6	83.9
	SSF	0.28	57.7	84.4
	DTLv1	0.13	**10.6**	84.4
	DTLv2	0.13	14.0	84.4
	DTLv2-LS	**0.10**	14.2	**84.4**
ConvNeXt-B	Full	87.6	38.8	85.8
	Fixed	0	4.4	84.1
	Adapter	0.31	21.5	84.5
	Bias	0.13	20.8	84.6
	SSF	0.26	49.5	84.9
	DTLv1	0.17	**12.0**	84.9
	DTLv2	0.18	13.3	84.9
	DTLv2-LS	**0.11**	13.6	**84.9**

Table 3: Results on 5 FGVC datasets. #m is measured with a batch size of 64. An asterisk (*) indicates results from [57], where AugReg is removed for a fair comparison.

Method	#p	#m	CUB	NABirds	Flowers	Dogs	Cars	Avg.
Full	85.8	9.7	87.3	82.7	98.8	89.4	84.5	88.5
Fixed	0	1.3	85.3	75.9	97.9	86.2	51.3	79.3
Adapter	0.17	5.8	87.1	84.3	98.5	89.8	68.6	85.7
SSF*	0.15	10.2	82.7	85.9	98.5	87.7	82.6	87.5
Bias	0.10	5.7	88.4	84.2	98.8	91.2	79.4	88.4
VPT	0.67	8.5	88.5	84.2	99.0	90.2	83.6	89.1
LoRA	0.29	5.9	88.3	85.6	99.2	91.0	83.2	89.5
ARC	0.10	6.1	88.5	85.3	99.3	91.9	85.7	90.1
RLRR	0.29	9.4	89.3	84.7	99.5	92.0	**87.0**	90.4
DTLv1	0.06	3.7	90.1	86.0	99.6	92.3	86.0	90.8
DTLv2	0.07	3.7	**90.1**	**86.2**	**99.6**	**92.7**	85.3	90.8
DTLv2-LS	**0.04**	**3.7**	90.0	86.1	99.6	92.6	86.1	**90.9**

and RLRR [59] as PETL baselines. We follow the setting in [6], [7] to report the results for a fair comparison.

7.2 Results on ImageNet-1K

As shown in Table 2, for the ViT-B/16 backbone, DTLv1 introduces only 0.06M trainable parameters, which is 40% less compared to the Bias method. Although DTLv2 introduces an enhanced feature fusion module G, our sparsification strategy ensures only a marginal increase in trainable parameters (+0.01M). With our L-Share, DTLv2-LS reduces the trainable parameters further, down to just 0.04M, even less than DTLv1. Additionally, all three DTL variants consume only 13.2 GB of GPU memory during finetuning, substantially less than other PETL baselines. Compared to full finetuning, GPU memory saving is about 62% on average. For other backbones, we observe a similar trend where our three DTL variants keep the least trainable parameters and GPU memory footprint. Compared to full finetuning, DTLv1 drastically saves GPU memory usage by 74% on average.

Our DTLv2 and DTLv2-LS achieve a 0.6% improvement in top-1 accuracy compared to the previous state-of-the-art method SSF on ViT-B/16 backbone. For Swin-B and ConvNeXt-B, all three DTL variants perform on par with SSF, demonstrating that in large-scale finetuning scenarios, our DTL approach not only maintains high parameter and memory efficiency but also delivers equally strong recognition accuracy.

7.3 Results on FGVC and VTAB-1K

The results of ViT-B/16 on FGVC benchmark are presented in Table 3. DTLv1 and DTLv2 show 0.4% gain on average accuracy compared to previous state-of-the-art method RLRR. With the incorporation of L-Share, DTLv2-LS further improves the average accuracy by 0.1%. Notably, DTLv2 shows strong effectiveness and generalization, achieving the highest accuracy on 4 out of the 5 datasets.

Results of ViT-B/16 on VTAB-1K benchmark are shown in Table 4. DTLv1 shows a 2.0% gain on average accuracy compared to the previous state-of-the-art method SSF. Specifically, DTLv1 reaches the best top-1 accuracy on 10 out of 19 datasets, where the improvements compared to SSF range from 0.3% to 19.9%. Even if the dataset 'dSpr-Loc' with the most significant gain of 19.9% is removed, DTLv1 is still far ahead on average accuracy of the remaining 18 datasets and outperforms SSF by 1.3%.

Although DTLv2 and DTLv2-LS perform slightly worse than DTLv1, they still outperform SSF with an accuracy gain of 0.8% and 0.5%, respectively. We observe that the most significant accuracy drop in DTLv2 compared to DTLv1 occurs in the 'Structured' group with a large domain gap. This pattern is also evident in Table 5 with different backbones (Swin-B and ConvNeXt-B). We hypothesize that since DTLv2 has a larger capacity than DTLv1, it is more prone to overfitting when dealing with large domain gaps.

As shown in Table 5, when switching to different backbone families, the performance of the three DTL variants on Swin-B and ConvNeXt-B follows a similar trend. DTLv1 surpasses previous best methods by significant margins of 1% and 1.9% on average accuracy, respectively. DTL also maintains the lowest number of trainable parameters and the smallest GPU memory footprint. Compared to full finetuning, DTL reduces GPU memory usage by 72% on average.

Moreover, we also analyzed the impact of changing the backbone size. When the backbone scales up to ViT-g/14 pretrained using DINOv2 [60] with 1136M trainable parameters, DTL still consistently outperforms previous methods with high parameter- and memory-efficiency. With DTL, we can even finetune the large-scale backbone, ViT-g/14, on a *single* RTX 3090 GPU, whereas traditional PETL methods fail in this scenario due to out-of-GPU-memory errors. Another interesting observation is that as the backbone size increases, the accuracy gap between DTLv1, DTLv2, and DTLv2-LS narrows. Specifically, for ViT-g/14, the accuracy gap between the three DTL variants shrinks to only less than 0.2%, indicating that DTLv2 and DTLv2-LS are especially useful for large-scale models.

Table 4: Results on the VTAB-1K benchmark with the ViT-B/16 backbone. "#m" specifies the peak GPU memory footprint when finetuning with batch size 32. "Average" is the group-wise average accuracy over three groups.

	#p	#m	Natural							Specialized				Structured								Average
			Cifar100	Caltech101	DTD	Flower102	Pets	SVHN	Sun397	Camelyon	EuroSAT	Resisc45	Retinopathy	Clevr-Count	Clevr-Dist	DMLab	KITTI-Dist	dSpr-Loc	dSpr-Ori	sNORB-Azim	sNORB-Ele	
Traditional finetuning																						
Full	85.8	4.7	68.9	87.7	64.3	97.2	86.9	87.4	38.8	79.7	95.7	84.2	73.9	56.3	58.6	41.7	65.5	57.5	46.7	25.7	29.1	68.9
Fixed	0	0.6	64.4	85.0	63.2	97.0	86.3	36.6	51.0	78.5	87.5	68.5	74.0	34.3	30.6	33.2	55.4	12.5	20.0	9.6	19.2	57.6
PETL methods																						
Bias	0.10	2.9	72.8	87.0	59.2	97.5	85.3	59.9	51.4	78.7	91.6	72.9	69.8	61.5	55.6	32.4	55.9	66.6	40.0	15.7	25.1	65.2
VPT	0.56	4.2	**78.8**	90.8	65.8	98.0	88.3	78.1	49.6	81.8	96.1	83.4	68.4	68.5	60.0	46.5	72.8	73.6	47.9	32.9	37.8	72.0
LST	2.38	2.7	59.5	91.5	69.0	99.2	89.9	79.5	54.6	86.9	95.9	85.3	74.1	81.8	61.8	52.2	81.0	71.7	49.5	33.7	45.2	74.3
LoRA	0.29	3.0	67.1	91.4	69.4	98.8	90.4	85.3	54.0	84.9	95.3	84.4	73.6	**82.9**	**69.2**	49.8	78.5	75.7	47.1	31.0	44.0	74.5
AdaptFormer	0.16	2.8	70.8	91.2	70.5	99.1	90.9	86.6	54.8	83.0	95.8	84.4	**76.3**	81.9	64.3	49.3	80.3	76.3	45.7	31.7	41.1	74.7
NOAH	0.43	3.3	69.6	92.7	70.2	99.1	90.4	86.1	53.7	84.4	95.4	83.9	75.8	82.8	68.9	49.9	81.7	81.8	48.3	32.8	44.2	75.5
FacT	0.07	3.9	70.6	90.6	70.8	99.1	90.7	88.6	54.1	84.8	96.2	84.5	75.7	82.6	68.2	49.8	80.7	80.8	47.4	33.2	43.0	75.6
SSF	0.21	4.9	69.0	92.6	**75.1**	99.4	91.8	**90.2**	52.9	87.4	95.9	**87.4**	75.5	75.9	62.3	**53.3**	80.6	77.3	54.9	29.5	37.9	75.7
DTLv1	0.06	1.8	70.4	**95.1**	71.5	**99.4**	**91.8**	87.5	56.8	**87.7**	**96.6**	86.9	74.7	81.6	65.1	51.3	**82.3**	**97.2**	**54.9**	**36.0**	**49.3**	**77.7**
DTLv2	0.07	1.8	71.6	93.9	71.1	99.4	91.6	82.8	**57.5**	87.0	95.7	86.0	74.8	82.0	63.3	48.8	80.9	96.3	51.9	30.7	45.5	76.5
DTLv2-LS	**0.04**	**1.8**	71.7	94.0	71.5	99.4	91.7	81.6	57.4	86.7	95.5	84.0	74.6	81.1	64.4	48.9	81.6	95.3	53.0	30.7	44.9	76.2

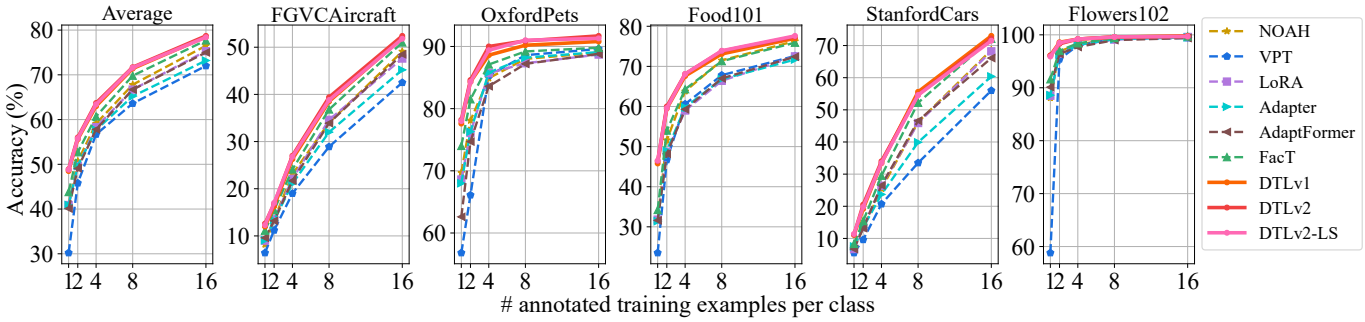


Figure 7: Top-1 accuracy on fine-grained few-shot benchmark. Best viewed in color. Note that our approach with less trainable parameters and GPU memory footprint outperforms all baseline methods.

7.4 Results on Few-shot Learning

As illustrated in Fig. 7, the proposed DTL method outperforms all baseline PETL methods in all cases. Furthermore, we observe that the average improvements of DTLv1 compared to previous state-of-the-art FacT across different shots are gradually decreased from 4.7% in 1-shot to 0.8% in 16-shot, which reveals that our method is consistently effective, especially in low-data regimes.

7.5 Results on Domain Generalization

The results of domain generalization experiments are shown in Table 6. We observe that compared to previous state-of-the-art method NOAH, our DTL achieves impressive gains in evaluation accuracy, especially on ImageNet, ImageNet-Sketch and ImageNet-R, where the average improvement is up to about 8%. These comparisons show robustness of DTL for alleviating the domain shift problem.

7.6 Ablation Studies

Sensitivity to hyper-parameters. In DTL, there are two key hyper-parameters. The first one is d' in the low-rank linear transformation. The second one is M, the beginning index of blocks in the backbone to add back the output of CSN for feature adaptation. In Fig. 8, we plot the accuracy curve by varying d' using ViT-B/16 and Swin-B as examples. We observe a small d' is sufficient to extract task-specific information for downstream recognition in ViT-B/16, although it tends to slightly overfit as d' increases. For Swin-B, a larger

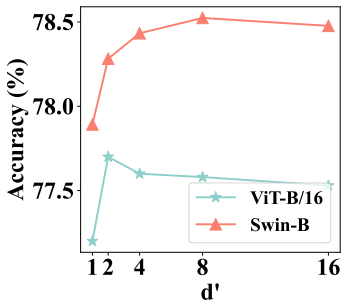


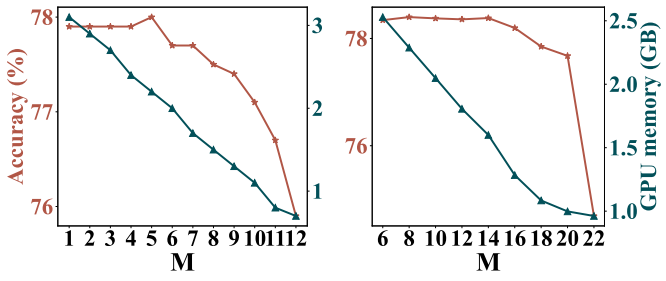
Figure 8: Top-1 accuracy on VTAB-1K under various d'. A small d' is enough to extract task-specific information in recognition tasks.

Table 5: Results on VTAB-1K benchmark with other backbones. Nat./Spe./Str./Avg. specify the results in three VTAB groups and their group-wise average. We simulate the scenario where only a single RTX 3090 GPU is available for conducting the experiments. Those marked with '-' means out-of-memory errors.

Backbone	Method	#p	#m	Nat.	Spe.	Str.	Avg.
Swin-B	Full	86.7	6.1	79.2	86.2	59.7	75.0
	Fixed	0	0.9	73.5	80.8	33.5	62.6
	Bias	0.20	3.7	74.2	80.1	42.4	65.6
	VPT	0.16	4.6	76.8	84.5	53.4	71.6
	FacT	0.14	5.6	**83.1**	86.9	62.1	77.4
	DTLv1	0.13	**1.6**	82.4	86.8	**66.0**	**78.4**
	DTLv2	0.13	2.0	82.5	**87.6**	64.2	78.1
	DTLv2-LS	**0.10**	2.1	82.5	87.5	63.6	77.8
ConvNeXt-B	Full	87.6	5.6	78.0	83.7	60.4	74.0
	Fixed	0	0.7	74.5	81.5	34.8	63.6
	LoRA	0.72	3.0	82.2	84.7	64.1	77.0
	Adapter	0.31	3.0	83.1	84.9	64.6	77.5
	DTLv1	0.17	**1.7**	**83.7**	86.7	**67.7**	**79.4**
	DTLv2	0.18	2.0	83.4	86.6	66.5	78.9
	DTLv2-LS	**0.11**	2.0	83.2	**86.7**	66.6	78.9
ViT-g/14	Full	1136	-	-	-	-	-
	Fixed	0	4.8	77.4	85.2	41.2	67.9
	LoRA	-	-	-	-	-	-
	VPT	-	-	-	-	-	-
	LST	30.2	19.9	82.6	86.5	64.2	77.7
	DTLv1	0.53	12.6	85.4	87.8	**67.3**	80.2
	DTLv2	0.27	10.7	**86.0**	**88.2**	67.1	**80.4**
	DTLv2-LS	**0.18**	**10.7**	85.8	88.1	66.6	80.2

Table 6: Domain generalization top-1 accuracy with ViT-B/16 backbone.

Method	Source	Target			
	ImageNet	-Sketch	-V2	-A	-R
Adapter	70.5	16.4	59.1	5.5	22.1
VPT	70.5	18.3	58.0	4.6	23.2
LoRA	70.8	20.0	59.3	6.9	23.3
NOAH	71.5	24.8	66.1	11.9	28.5
DTLv1	78.7	35.7	67.8	14.2	34.4
DTLv2	**78.7**	**36.7**	68.3	**15.0**	**35.5**
DTLv2-LS	78.6	36.3	**68.3**	14.8	35.2



(a) ViT-B/16 (b) Swin-B

Figure 9: Top-1 accuracy on VTAB-1K and peak GPU memory footprint under various M in Eq. 6. Our method is consistently effective across different M.

Table 7: Ablation results on VTAB-1K with ViT-B/16 backbone by varying different architectural choices, where the last line denotes DTLv1.

CSN	Swish (θ)	DWConv (g)	Avg.
			57.6
Skip			58.8
✓			76.0
✓	✓		76.7
	✓	✓	62.7
✓	✓	✓	77.7

d' (4 or 8) is preferred, but it is not highly sensitive to the exact value, and $d' = 2$ already performs well.

In Fig. 9, we plot the accuracy and GPU memory footprint curves by varying M using two different backbones. There is a clear trend that, regardless of the backbone used, GPU memory usage decreases almost linearly as M increases. For ViT-B/16, reducing M from 12 to 11 significantly boosts recognition accuracy from 75.9% to 76.7%, with improvements gradually saturating when $M < 6$. This suggests the feasibility and effectiveness of feature sharing as described in Sec 6. The accuracy differs by up to 2.1%, indicating that recognition accuracy is not highly sensitive to the exact value of M. This trend is also evident when using Swin-B. Therefore, by default, we set M to about half of the total number of blocks in the backbone to achieve the best tradeoff between effectiveness and efficiency–$M = 7$ for ViT-B and $M = 13$ for Swin-B. Please refer to the appendix for the specific values of M used with each backbone architecture.

Modular ablation. We now extend our ablation further to verify the contribution of each component in DTLv1. Results are in Table 7. The first row is the baseline which only learns a linear classifier. The second row, where the "CSN" item is marked by "Skip", means only the skip connection is kept (dropping CSN low-rank transformations, θ and DWConv, cf. Fig. 3), marginally improves the recognition accuracy. When CSN is integrated, the accuracy is increased to 76.0%, with a huge improvement of 17.2%. Moreover, compared to DTLv1 (last row), when CSN is removed (second row from bottom), the accuracy is drastically decreased from 77.7% to 62.7%. These two observations demonstrate that the information extracted by CSN is critical for downstream recognition. As Swish (θ) and DWConv (g) are gradually integrated, the accuracy improves steadily. Since DTLv2 is primarily designed for dense prediction tasks, we will defer the modular analysis of DTLv2 to the next section.

Potentials for feature reuse. We take the 19 datasets in VTAB-1K [9] to illustrate feature reuse in DTLv1. We finetune to obtain 19 models, and assume that we need to get all 19 classification results for one input image using all these 19 models. The goal is to check how much speedup can be achieved during inference. Firstly, we feed the same input image into 19 different models after finetuning with LoRA [5], which acts as the baseline. Then we implement our method to simultaneously conduct 19 tasks but with the backbone feature shared in the first 6 blocks (by setting $M = 7$). Experimental results show that approximately *45% inference latency is saved*.

Table 8: Results on MS-COCO. GPU memory usage during finetuning (#m) recorded with a local batch size of 2 on 4 RTX GPUs.

Swin-B (87M)	#p	#m	MS-COCO (Cascade Mask R-CNN)					
			AP_{Box}	AP^{50}_{Box}	AP^{75}_{Box}	AP_{Mask}	AP^{50}_{Mask}	AP^{75}_{Mask}
Full	86.8	69.1	52.4	71.0	56.9	45.3	68.3	49.4
Fixed	0	17.2	48.3	69.0	52.9	42.0	65.9	45.1
Adapter	3.1	54.8	52.1	71.6	56.9	45.0	68.8	49.0
AdaptFormer	3.1	53.4	52.0	71.6	56.8	45.1	68.6	48.8
LoRA	3.1	55.0	50.4	70.5	55.1	43.9	67.7	47.5
LoRand	**2.4**	73.8	50.5	70.4	55.1	44.0	67.5	47.7
DTLv1	5.8	**31.7**	52.0	71.2	56.7	45.1	68.6	48.7
DTLv2	3.9	35.0	**52.7**	72.0	**57.3**	45.6	69.2	49.6
DTLv2-LS	2.9	35.2	52.6	**72.0**	57.2	**45.7**	**69.4**	**49.7**

Table 9: Results on Pascal VOC. GPU memory footprint (#m) measured with a batch size of 2 on one GPU.

Swin-L (195M)	#p	#m	RetinaNet AP_{Box}
Full	195.0	11.6	83.6
Fixed	0	2.2	83.6
Adapter	4.7	7.4	86.6
AdaptFormer	4.6	7.0	86.6
LoRA	4.6	7.4	85.6
LoRand	3.6	10.5	86.6
DTLv1	4.4	**3.9**	86.2
DTLv2	3.0	4.2	**87.1**
DTLv2-LS	**2.2**	4.2	87.0

Table 10: Results on the ADE20K dataset using Swin-L as the pretrained backbone. The table details the GPU memory footprint during finetuning (#m), measured with a local batch size of 2 on 8 GPUs (in total 16).

Swin-L (195M)	#p	#m	UperNet mIoU
Full	195.0	87.8	50.4
Fixed	0	33.0	46.4
Adapter	4.7	56.1	50.2
AdaptFormer	4.6	52.5	50.3
LoRA	4.6	55.7	49.7
LoRand	3.6	72.1	50.0
DTLv1	6.5	**41.6**	50.3
DTLv2	5.7	43.1	**50.7**
DTLv2-LS	**3.6**	43.2	50.6

8 Experiments on Dense Prediction

Besides recognition, we also conducted extensive experiments on dense prediction tasks, focusing on DTLv2 and DTLv2-LS. In this section, we will first detail the experimental settings, followed by a presentation of the results along with our analysis. Finally, we conduct ablation studies to verify the contribution of each module in our method.

8.1 Evaluation Settings

Datasets. We conducted experiments on MS-COCO [55], Pascal VOC [61] and ADE20K [62]. Detailed dataset statistics and evaluation protocols are in the appendix.

Implementation Details. We use backbones pretrained on the ImageNet-21K [56] dataset. Our model is implemented using the OpenMMLab [63] codebase. Unlike the configuration used in image classification, we set d' in CSN to 256 for experiments on the MS-COCO dataset, 192 for ADE20K, and 128 for Pascal VOC across all DTL variants. The Swin-B and Swin-L models each contain 24 blocks. As in recognition, we set M 13. Additional implementation details can be found in the appendix. We use group convolution to connect CSN layers across different backbone stages, with the number of groups set to one-fourth of the input channels by default.

Baseline methods. The baseline methods are Full, Fixed, Adapter [3], AdaptFormer [17], LoRA [5] and LoRand [30]. Note that LoRand [30] is implemented by us because the authors of [30] have not released the official code.

8.2 Main Results

The results of our DTL method on dense prediction tasks are presented in Tables 8, 9, and 10. These tables demonstrate that the proposed DTL methods require significantly less GPU memory for finetuning the pretrained model when adapting to downstream dense prediction tasks, too. In MS-COCO (Table 8), DTL achieves an average reduction of 51% in GPU memory usage compared to full finetuning, making the finetuning process more efficient and accessible for real-world applications. The simplest version, DTLv1, requires only 31.7 GB of GPU memory, which is a substantial memory saving compared to AdaptFormer [17] (approximately 41% reduction), while still yielding comparable AP.

When turning to DTLv2 specially designed for dense prediction tasks, it significantly enhances AP across all IoU thresholds, with an average improvement of approximately 0.7%, even *surpassing the full finetuning method* by 0.5%, while also reducing the number of trainable parameters from 5.8M to 3.9M. Additionally, with the integration of the L-Share technique, DTLv2-LS further compresses the trainable parameters from 3.9M to 2.9M, while maintaining or even surpassing the AP performance of DTLv2 in some cases.

It is worth noting that some traditional PETL methods, such as LoRand, although requiring minimal trainable parameters, significantly increase GPU memory usage during finetuning, making them impractical when GPU resources are limited. In contrast, the trainable parameters and GPU memory usage of DTLv2-LS are generally much lower than them, and at the same time achieving high detection AP. This makes DTLv2-LS an optimal tradeoff between effectiveness and efficiency.

A similar trend is observed in Table 9 for the Pascal VOC dataset, where our DTL method consistently requires the least GPU memory for finetuning the pretrained model,

Table 11: Ablation of DTLv2 for Swin-B on MS-COCO.

Method	#p	#m	AP_{Box}	AP_{Mask}
DTLv1	5.8	31.7	52.0	45.1
+ enhanced G	6.6	37.1	52.3	45.2
+ enhanced G (not share)	9.5	37.6	52.5	45.4
+ sparsify w	5.1	33.7	52.5	45.4
+ sparsify G	3.7	**31.1**	52.5	45.4
+ unified CSN = DTLv2	3.9	35.0	**52.7**	45.6
+ L-Share G = DTLv2-LS	**2.9**	35.2	52.6	**45.7**

Table 12: Ablation on different architectures of fusion module on VTAB-1K. The third line in each group marked with "+ $g \rightarrow G$" is DTLv2-LS.

Backbone	Method	#p	#m	Nat.	Spe.	Str.	Avg.
ViT-B/16	DTLv1	0.06	1.8	81.8	86.5	64.7	77.7
	+ sparsify w & g	0.03	1.6	81.8	86.4	64.2	77.5
	+ $g \rightarrow G$	0.04	1.8	81.0	85.2	62.5	76.2
Swin-B	DTLv1	0.13	1.6	82.4	86.8	66.0	78.4
	+ sparsify w & g	0.07	1.5	82.3	87.0	65.2	78.2
	+ $g \rightarrow G$	0.10	2.1	82.5	87.5	63.6	77.8
ConvNeXt-B	DTLv1	0.17	1.7	83.7	86.7	67.7	79.4
	+ sparsify w & g	0.08	1.5	83.6	86.7	67.5	79.3
	+ $g \rightarrow G$	0.11	2.0	83.2	86.7	66.6	78.9

achieving a reduction of approximately 65% compared to full finetuning. Additionally, the DTLv2-LS variant exhibits the lowest number of trainable parameters, representing a 39% reduction compared to LoRand [30]. Furthermore, the AP_{Box} achieved by both DTLv2 and DTLv2-LS significantly outperforms traditional PETL methods, with a notable margin of 0.5% and 0.4%, respectively.

The results for the ADE20K semantic segmentation task are presented in Table 10. In this scenario, the model needs to capture fine-grained local texture details to effectively recognize the concept of each input pixel. DTLv2-LS requires only 43.2 GB of GPU memory for finetuning, representing a 51% reduction compared to full finetuning. Moreover, the mIoU achieved by DTLv2-LS not only surpasses traditional PETL methods but also exceeds the full finetuning method.

These results demonstrate that our DTLv2 and DTLv2-LS, specifically designed for dense prediction, excel on common object detection and semantic segmentation benchmarks. They exhibit both parameter and GPU memory efficiency, and deliver superior detection and segmentation accuracy.

8.3 Ablation Studies

Modular ablation of DTLv2 design. We now study the impact of each enhancement in changing DTLv1 to DTLv2-LS. As shown in Table 11, the enhanced feature fusion module G effectively captures more discriminative local features, leading to significant improvements in both AP_{Box} and AP_{Mask}, with particularly notable gains in AP_{Box}. When G is not shared among different CSN layers, the AP metrics are further improved by approximately 0.2% on average. But, this increases the number of trainable parameters from 6.6M to 9.5M, and compromises parameter efficiency.

To address this issue, the sparsification strategies introduced in Sec 5.2 for w and G reduce the trainable parameters from 9.5M to 3.7M, resulting in a 61% reduction

Table 13: Ablation of fusion operators in CSN. We compare average pooling, full convolution and depth-wise convolution on image recognition (VTAB-1K with ViT-B/16) and dense prediction (MS-COCO with Swin-B). Depth-wise convolution consistently achieves the highest performance across all metrics.

Method	#p	#m	Metrics			
Image Recognition			Nat.	Spe.	Str.	Avg.
DTLv1 (AvgPool)	0.04	1.7	81.4	85.9	63.4	76.9
DTLv1 (FullConv)	5.35	1.9	80.9	85.4	63.2	76.5
DTLv1 (DWConv)	0.06	1.8	81.8	86.5	64.7	77.7
DTLv2-LS (AvgPool)	0.03	1.7	80.9	85.4	61.5	75.9
DTLv2-LS (FullConv)	5.34	1.8	80.7	85.3	60.9	75.6
DTLv2-LS (DWConv)	0.04	1.8	81.0	85.2	62.5	76.2
Dense Prediction			AP_{Box}		AP_{Mask}	
DTLv2-LS (AvgPool)	2.9	35.0	52.0		45.2	
DTLv2-LS (FullConv)	14.7	35.5	52.7		45.7	
DTLv2-LS (DWConv)	2.9	35.2	52.6		45.7	

Table 14: Effect of different G sharing strategies for Swin-B on MS-COCO.

Method	#p	#m	AP_{Box}	AP_{Mask}
L-Share G	**2.9**	**35.2**	**52.6**	**45.7**
Direct Share G	2.9	35.2	52.4	45.4

while maintaining the same level of AP. Additionally, this sparsification strategy reduces the GPU memory footprint from 37.6 GB to 31.1 GB (17% saving). Furthermore, the unified CSN technique (Sec 5.3) consolidates previously separated CSN modules into a unified structure, and further improves AP by approximately 0.2%. Finally, the L-Share strategy reduces the number of trainable parameters to just 2.9M, while maintaining the AP metrics and even slightly improves AP_{Mask}. However, as shown in Table 14, when the L-Share strategy is removed and direct sharing is used, both AP metrics decline.

Ablation of CSN concatenation strategy. In Table 15, we analyze strategies for concatenating CSN layers from different backbone stages into a unified structure. The baseline is directly using the frozen downsampling module from the backbone. However, this method performs the worst, as backbone features and CSN features exhibit different patterns. To verify this hypothesis, we unfreeze these downsampling layers and finetune them together with CSN, i.e., 'Backbone (Learned)'. This approach significantly boosts AP, but leads to prohibitively high GPU memory usage.

We employ a group convolutional layer to achieve our objective. The notation 'Group Conv (x)' refers to a group convolution layer with the number of groups being one-xth of the number of input channels. Table 15 show that 'Group Conv (4)' strikes a good balance between computational efficiency and AP performance.

Why does DTLv2-LS underperform DTLv1 on certain classification benchmarks? Although DTLv2-LS consistently matches or exceeds DTLv1 on ImageNet-1K, FGVC, and domain-generalization tasks, we observe a modest drop on VTAB-1K, particularly in the *Structured* group. To gain deeper insight into this phenomenon and the intrinsic mechanisms of our DTL design, we conducted additional ablation

studies (Table 12). First, we endowed DTLv1 with the same sparsification strategies employed in DTLv2—namely, removing a subset of low-rank transformations (w) and fusion connections (g), followed by the linear-sharing scheme (cf. Sec. 5.2). By comparing the first and second rows within each backbone group, we observe that the average accuracy on VTAB-1K remains effectively unchanged (drop $< 0.2\%$), demonstrating the general effectiveness of our sparsification strategy across both dense prediction and image classification tasks.

In contrast, substituting DTLv1's simple DWConv fusion g with the more powerful module G results in pronounced declines, especially on Structured datasets. Specifically, For ViT-B/16 nearly every VTAB-1K domain shows some reduction, whereas for Swin-B and ConvNeXt-B the drop is largely confined to the Structured group, with Natural and Specialized accuracies remaining stable. These findings suggest that, while sparsification preserves model capacity under limited data, the increased complexity of G induced overfitting under severe domain shift.

We attribute the overfitting to the increased capacity of G. In DTLv1, the simple DWConv fusion g provides just enough flexibility while remaining regularized, whereas the more complex module G introduces additional low-rank transformations and residual connections. Although these augmentations improve expressivity when ample or in-domain data are available, they also risk fitting to spurious patterns under the pronounced domain shifts and limited samples of VTAB's *Structured* benchmarks, thereby degrading generalization. In contrast, on tasks with smaller domain gaps or sufficient training examples (e.g. ImageNet-1K or MS-COCO), this overfitting does not occur, and DTLv2-LS matches or surpasses DTLv1.

It is worth noting that in the domain generalization experiments finetuning is performed exclusively on ImageNet data, so the domain shift arises only at evaluation and the overfitting mechanism described above does not apply.

Effect of depth-wise convolution in CSN. To clarify the motivation for integrating depthwise convolution into the CSN fusion module (g or G) and its benefits, we compare three fusion operators: average pooling, full convolution and depthwise convolution. Depthwise convolution expands the spatial receptive field while maintaining minimal parameter count and memory overhead. It preserves spatial locality and supports channel-wise feature adaptation with only a small increase in trainable parameters. As shown in Table 13, depthwise convolution achieves the highest accuracy on VTAB-1K and the highest average precision on MS-COCO, demonstrating its effectiveness in capturing discriminative local patterns. These properties support its adoption in our unified CSN design.

Inference efficiency. We further evaluate the efficiency of our method during inference by comparing its throughput with several baselines. As shown in Table 16, we first simulate the image classification scenario for ViT-B/16 with a small $d' = 2$. Due to its simplicity, the empirical throughput of DTLv1 is consistently higher than previous PETL counterparts. When the inference batch size is small (≤ 4), DTLv2 and DTLv2-LS are also faster than existing PETL methods.

We then shift to dense prediction, where Swin-B is used and d' is 256. The input resolution is 1344×896. The results show that all three DTL variants achieve higher throughput compared to existing PETL methods.

Table 15: Choices for CSN concatenation (Swin-B on MS-COCO). 'Group Conv (4)' is the default choice in DTLv2.

Method	#p	#m	AP_{Box}	AP_{Mask}
Backbone (Frozen)	3.9	**34.8**	52.4	45.4
Backbone (Learned)	6.7	50.8	52.7	45.6
Full Conv	6.7	35.7	52.6	45.6
Group Conv (8)	4.0	35.1	52.5	45.5
Group Conv (4)	**3.9**	35.0	**52.7**	**45.6**
Group Conv (2)	3.9	35.0	52.4	45.4
Group Conv (1)	3.9	35.0	52.3	45.4

Table 16: Throughput (number of images processed per second) measured on one GPU with mixed precision inference.

Backbone & Input Size	Method	Throughput (imgs/sec)		
		bs=1	bs=4	bs=16
ViT-B/16 $d' = 2$ (224×224)	Full	161	636	952
	LST	71	279	729
	NOAH	79	306	798
	AdaptFormer	108	436	876
	DTLv1	120	469	**877**
	DTLv2	127	495	850
	DTLv2-LS	**130**	**504**	854
Swin-B $d' = 256$ (1344×896)	Full	31	33	34
	Adapter	27	30	32
	AdaptFormer	28	30	32
	DTLv1	**29**	**31**	**33**
	DTLv2	29	31	33
	DTLv2-LS	29	31	32

9 Conclusions and Limitations

In this paper, we proposed Disentangled Transfer Learning (DTL) for finetuning large pretrained vision models, with three variants, DTLv1, DTLv2, and DTLv2-LS. The central idea is to disentangle updates of trainable parameters from a frozen backbone, which substantially reduces GPU memory footprint and the number of trainable parameters. On recognition benchmarks, DTL matched or surpassed state-of-the-art PETL baselines, and on dense prediction tasks, where PETL often lags behind full finetuning, DTL attained higher AP.

Despite these gains, the interaction between CSN and the backbone remains coarse. CSN interacts only at predefined insertion points, typically at the block level, which limits fine-grained adjustments across layers. As a result, tasks that require subtle adaptation of intermediate representations do not fully benefit from the disentangled design.

To mitigate this limitation while preserving memory savings, we are developing finer interactions between CSN and the backbone. This includes inserting CSN modules within backbone blocks at carefully selected linear layers and exploring multi-scale connections with dynamic gating to balance disentanglement with richer feature exchange. Taken together, these directions position DTL as a practical foundation for parameter- and memory-efficient adaptation in resource-constrained deployments, enabling broader extensions across diverse vision tasks.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional Transformers for language understanding," *arXiv:1810.04805*, 2018. 1

[2] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 979–15 988. 1

[3] N. Houlsby, A. Giurgiu, S. Jastrzebski *et al.*, "Parameter-efficient transfer learning for NLP," in *International Conference on Machine Learning*, 2019, pp. 2790–2799. 1, 2, 3, 6, 10

[4] M. Jia, L. Tang, B.-C. Chen *et al.*, "Visual prompt tuning," in *European Conference on Computer Vision*, ser. LNCS, vol. 13693, 2022, pp. 709–727. 1, 2, 6

[5] E. J. Hu, Y. Shen, P. Wallis *et al.*, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022, pp. 1–13. 1, 2, 3, 6, 9, 10

[6] Y. Zhang, K. Zhou, and Z. Liu, "Neural prompt search," *arXiv:2206.04673*, 2022. 1, 6, 7

[7] D. Lian, D. Zhou, J. Feng, and X. Wang, "Scaling & shifting your features: A new baseline for efficient model tuning," in *Advances in Neural Information Processing Systems*, 2022, pp. 109–123. 1, 2, 3, 6, 7

[8] S. Jie and Z.-H. Deng, "FacT: Factor-Tuning for lightweight adaptation on Vision Transformer," in *AAAI Conference on Artificial Intelligence*, 2023, pp. 1060–1068. 1, 2, 6

[9] X. Zhai, J. Puigcerver, A. Kolesnikov *et al.*, "A large-scale study of representation learning with the Visual Task Adaptation Benchmark," *arXiv:1910.04867*, 2019. 1, 6, 9

[10] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1800–1807. 1, 3

[11] M. Fu, K. Zhu, and J. Wu, "DTL: Disentangled transfer learning for visual recognition," in *AAAI Conference on Artificial Intelligence*, 2024, pp. 12 082–12 090. 2, 3

[12] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. 2

[13] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Advances in Neural Information Processing Systems*, 2015, p. 1135–1143. 2

[14] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations*, 2017, pp. 1–12. 2

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015. 2

[16] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *International Conference on Learning Representations*, 2015, pp. 1–10. 2

[17] S. Chen, C. GE, Z. Tong *et al.*, "AdaptFormer: Adapting Vision Transformers for scalable visual recognition," in *Advances in Neural Information Processing Systems*, 2022, pp. 16 664–16 678. 2, 6, 10

[18] C. Mou, X. Wang, L. Xie, Y. Wu, J. Zhang, Z. Qi, and Y. Shan, "T2I-Adapter: learning adapters to dig out more controllable ability for text-to-image diffusion models," in *AAAI Conference on Artificial Intelligence*, 2024, pp. 4296–4304. 2

[19] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *IEEE/CVF International Conference on Computer Vision*, 2023. 2

[20] X. Guo, M. Zheng, L. Hou *et al.*, "I2V-Adapter: A general image-to-video adapter for diffusion models," in *ACM SIGGRAPH 2024 Conference*, 2024, pp. 1–12. 2

[21] B. Peng, J. Wang, Y. Zhang, W. Li, M.-C. Yang, and J. Jia, "ControlNeXt: Powerful and efficient control for image and video generation," *arXiv:2408.06070*, 2024. 2

[22] P. Gao, S. Geng, R. Zhang *et al.*, "CLIP-Adapter: Better vision-language models with feature adapters," *International Journal of Computer Vision*, vol. 132, no. 2, p. 581–595, 2023. 2

[23] R. Zhang, W. Zhang, R. Fang *et al.*, "Tip-Adapter: Training-free adaption of CLIP for few-shot classification," in *European Conference on Computer Vision*, ser. LNCS, vol. 13695, 2022, pp. 493–510. 2

[24] R. Zhang, J. Han, C. Liu *et al.*, "LLaMA-Adapter: Efficient fine-tuning of language models with zero-init attention," in *International Conference on Learning Representations*, 2024, pp. 1–16. 2

[25] P. Gao, J. Han, R. Zhang *et al.*, "LLaMA-Adapter v2: Parameter-efficient visual instruction model," *arXiv:2304.15010*, 2023. 2

[26] S. Jie, Z.-H. Deng, S. Chen, and Z. Jin, "Convolutional bypasses are better Vision Transformer adapters," in *European Conference on Artificial Intelligence*, 2024, pp. 202–209. 2

[27] M. Xu, Z. Zhang, F. Wei, H. Hu, and X. Bai, "SAN: Side adapter network for open-vocabulary semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 12, pp. 15 546–15 561, 2023. 2

[28] S. He, L. Ding, D. Dong, J. Zhang, and D. Tao, "SparseAdapter: An easy approach for improving the parameter-efficiency of Adapters," in *Empirical Methods in Natural Language Processing*, 2022, pp. 2184–2190. 2

[29] C.-L. Fu, Z.-C. Chen, Y.-R. Lee, and H.-y. Lee, "AdapterBias: Parameter-efficient token-dependent representation shift for adapters in NLP tasks," in *North American Chapter of the Association for Computational Linguistics*, 2022, pp. 2608–2621. 2

[30] D. Yin, Y. Yang, Z. Wang, H. Yu, K. Wei, and X. Sun, "1% vs 100%: Parameter-efficient low rank Adapter for dense predictions," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 20 116–20 126. 2, 10, 11

[31] D. Yin, L. Hu, B. Li, Y. Zhang, and X. Yang, "5% >100%: Breaking performance shackles of full fine-tuning on visual recognition tasks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025, pp. 20 071–20 081. 2

[32] D. Yin, X. Han, B. Li, H. Feng, and J. Bai, "Parameter-efficient is not sufficient: Exploring parameter, memory, and time efficient adapter tuning for dense predictions," in *ACM International Conference on Multimedia*, 2024, p. 1398–1406. 2

[33] Q. Zhang, M. Chen, A. Bukharin *et al.*, "AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning," in *International Conference on Learning Representations*, 2023, pp. 1–12. 2

[34] H. U. K. Shinwari and M. Usama, "ARD-LoRA: Dynamic rank allocation for parameter-efficient fine-tuning of foundation models with heterogeneous adaptation needs," *arXiv:2506.18267*, 2025. 2

[35] B. Zi, X. Qi, L. Wang, J. Wang, K.-F. Wong, and L. Zhang, "Delta-LoRA: Fine-tuning high-rank parameters with the delta of low-rank matrices," *arXiv:2309.02411*, 2023. 2

[36] X. Chen, J. Liu, Y. Wang, P. P. Wang, M. Brand, G. Wang, and T. Koike-Akino, "SuperLoRA: Parameter-efficient unified adaptation for large vision models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2024, pp. 1–6. 2

[37] G. Luo, M. Huang, Y. Zhou, X. Sun, G. Jiang, Z. Wang, and R. Ji, "Towards efficient visual adaption via structural re-parameterization," *arXiv:2302.08106*, 2023. 2

[38] T. Hao, H. Chen, Y. Guo, and G. Ding, "Consolidator: Mergable Adapter with group connections for visual adaptation," in *International Conference on Learning Representations*, 2023, pp. 1–13. 2

[39] S. Du, G. Zhang, K. Wang *et al.*, "ALoRE: Efficient visual adaptation via aggregating low rank experts," *arXiv:2412.08341*, 2024. 2

[40] J. Lopez-Piqueres, P. Deshpande, A. Ray, M. J. Villani, M. Pistoia, and N. Kumar, "MetaTT: A global tensor-train Adapter for parameter-efficient fine-tuning," *arXiv:2506.09105*, 2025. 2

[41] F. Zeng, H. Guo, F. Zhu, L. Shen, and H. Tang, "Parameter efficient merging for multimodal large language models with complementary parameter adaptation," *arXiv:2502.17159*, 2025. 2

[42] E. Ben Zaken, Y. Goldberg, and S. Ravfogel, "BitFit: Simple parameter-efficient fine-tuning for Transformer-based masked language-models," in *60th Annual Meeting of the Association for Computational Linguistics*, 2022, pp. 1–9. 2, 6

[43] C. Han, Q. Wang, Y. Cui, Z. Cao, W. Wang, S. Qi, and D. Liu, "E2VPT: An effective and efficient approach for visual prompt tuning," in *IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 445–17 456. 2

[44] C.-H. Tu, Z. Mai, and W.-L. Chao, "Visual query tuning: Towards effective usage of intermediate representations for parameter and memory efficient transfer learning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7725–7735. 2

[45] M. U. Khattak, S. T. Wasim, M. Naseer, S. Khan, M.-H. Yang, and F. S. Khan, "Self-regulating prompts: Foundational model adaptation without forgetting," in *IEEE/CVF International Conference on Computer Vision*, 2023, pp. 15 144–15 154. 2

[46] K. Zhou, J. Yang, C. C. Loy, and Z. Liu, "Conditional prompt learning for vision-language models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 795–16 804. 3

[47] E. Cho, J. Kim, and H. J. Kim, "Distribution-aware prompt tuning for vision-language models," in *IEEE/CVF International Conference on Computer Vision*, 2023, pp. 21 947–21 956. 3

[48] H. Yao, R. Zhang, and C. Xu, "Visual-language prompt tuning with knowledge-guided context optimization," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 6757–6767. 3
[49] Y.-L. Sung, J. Cho, and M. Bansal, "LST: Ladder Side-Tuning for parameter and memory efficient transfer learning," in *Advances in Neural Information Processing Systems*, 2022, pp. 12 991–13 005. 3, 6
[50] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 6000–6010. 3
[51] Z. Liu, Y. Lin, Y. Cao *et al.*, "Swin Transformer: Hierarchical Vision Transformer using shifted windows," in *IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9992–10 002. 3, 4, 5, 6
[52] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv:1710.05941*, 2017. 3
[53] A. Dosovitskiy, L. Beyer, A. Kolesnikov *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021, pp. 1–21. 3, 6
[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778. 3
[55] T.-Y. Lin, M. Maire, S. Belongie *et al.*, "Microsoft COCO: Common objects in context," in *European Conference on Computer Vision*, ser. LNCS, vol. 8693, 2014, pp. 740–755. 4, 10
[56] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. 6, 10
[57] W. Dong, D. Yan, Z. Lin, and P. Wang, "Efficient adaptation of large Vision Transformer via Adapter re-composing," in *Advances in Neural Information Processing Systems*, 2023, pp. 52 548–52 567. 6, 7
[58] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019, pp. 1–18. 6
[59] W. Dong, X. Zhang, B. Chen *et al.*, "Low-rank rescaled Vision Transformer fine-tuning: A residual design approach," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 16 101–16 110. 7
[60] M. Oquab, T. Darcet, T. Moutakanni *et al.*, "DINOv2: Learning robust visual features without supervision," *arXiv:2304.07193*, 2023. 7
[61] M. Everingham, S. M. Eslami, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015. 10
[62] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ADE20K dataset," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5122–5130. 10
[63] K. Chen, J. Wang, J. Pang *et al.*, "MMDetection: Open MMLab detection toolbox and benchmark," *arXiv:1906.07155*, 2019. 10

Minghao Fu received his BS degree in Computer Science and Technology from the University of Electronic Science and Technology of China, and is currently a Ph.D. student at the School of Artificial Intelligence, Nanjing University, China. His research interests include computer vision and machine learning.

Ke Zhu received his BS degree in Automation Science and Technology from the Department of Electronics and Information, Xi'an Jiaotong University in 2020. He is now a PhD student in the School of Artificial Intelligence in Nanjing University, China. His research interests are long-tailed learning, self-supervised learning and object detection.

Zonghao Ding received his BS degree in Automation from Northeastern University, China. Currently, he is a graduate student at the School of Artificial Intelligence, Nanjing University, China. His research interests are computer vision and machine learning.

Jianxin Wu received his BS and MS degrees from Nanjing University, and PhD degree from the Georgia Institute of Technology, all in computer science. He is a professor in the School of Artificial Intelligence at Nanjing University and the National Key Laboratory for Novel Software Technology, China. He has served as a program chair for CVPR'24, (senior) area chair for CVPR, ICCV, ECCV, AAAI and IJCAI, and as an associate editor for IEEE T-PAMI. His research interests are computer vision and machine learning.