Appendix

We provide details that are omitted from the main paper.

- Appendix A: Experiment and dataset details
- Appendix B: Detailed descriptions of ViT and compared methods.
- Appendix C: Additional results not presented in main paper
- Appendix D: Discussion about further impacts of this work

A. Experiment and Dataset Details

A.1. Experiment Details

VTAB-1K We employ AdamW optimizer [61] with a batch size of 64 and utilize the cosine decay learning rate scheduler. We train all methods with 100 epochs. The learning rate is tuned from [1e-3, 1e-2] and weight decay from [1e-4, 1e-3]. The method-specific hyperparameter searching grip is shown in Table 3, along with the tunable parameter ranges (in millions). Since most method-specific hyperparameters affect the number of tunable parameters in the PEFT methods, we set a cap on the tunable parameters for each PEFT method to be less than or equal to **1.5%** of the total parameters in ViT-B/16, which is approximately equal to the number of parameters in the Query, Key, and Value matrices of a single MSA block. Consistent with the original VTAB-1k paper [104], most PEFT studies [42–44, 55, 63, 63, 107] don't apply data augmentation as it's challenging to identify a set of augmentations that uniformly benefits all 19 datasets[3]. To ensure that our results are directly comparable and that any performance differences are attributable to the methods themselves rather than data augmentation, we don't apply data augmentation.

Many-shot We also employ AdamW optimizer with a batch size of 64 and a cosine decay learning rate scheduler. The learning rate is tuned from [5e-4, 1e-3] and weight decay keeps the same range of [1e-4, 1e-3]. We apply horizontal flipping for CIFAR100, horizontal and vertical flipping for Resisc, and no augmentation for Clevr. We train all methods with 40 epochs.

Robustness Model CLIP models are trained on image-caption pairs collected from the web. Given a dataset of such pairs $\{(x_1, s_1), \ldots, (x_B, s_B)\}$, these models learn an image encoder g and a text encoder h that aim to maximize the similarity $\langle g(x_i), h(s_i)\rangle$ between matching image and caption embeddings while minimizing it for non-matching pairs. For zero-shot classification, the models predict the class of an input image x from a set of k class names $C = \{c_1, \ldots, c_k\}$ by matching x with captions derived from these class names. Specifically, for each class c_j, a caption is formulated as $s_j =$ "a photo of a c_j". The predicted class is then determined by selecting the one whose caption embedding has the highest similarity with the image embedding: $\hat{y} = \arg\max_j \langle g(x), h(s_j)\rangle$. Alternatively, a weight matrix $W_{\text{zero-shot}} \in \mathbb{R}^{d\times k}$ can be constructed, where each column is the embedding $h(s_j)$ corresponding to class c_j. The model's output scores for each class are then computed as $f(x) = g(x)^\top W_{\text{zero-shot}}$. To generate $W_{\text{zero-shot}}$, we ensemble the 80 prompts provided by CLIP at https://github.com/openai/CLIP.

Robustness Setup In section 3 and section 5, the fine-tuning train set and test set are from the same data distribution. In section 7, we fine-tune the CLIP model using ImageNet-1K training data (100 shots) and subsequently evaluate the fine-tuned model not only on the test set of ImageNet-1K but also on four additional datasets with distribution shifts: ImageNet-V2, ImageNet-R, ImageNet-S, and ImageNet-A, as shown in Figure 7. Following [96], we set a small learning rate as 3e-5 and weight decay as 5e-3. We use a strong data augmentation following [107].

Computation We used a workstation with eight NVIDIA RTX 6000 Ada GPUs, two AMD EPYC 9554 64-Core Processors, and 800GB of RAM.

What do we not investigate? There are many aspects that one can ask about PEFT. Our study focuses more on their learning and prediction behaviors, not the computation-specific properties like memory usage and FLOPS.

A.2. Dataset Details

VTAB-1K The processed VTAB-1K can be downloaded from our official code base to ensure reproducibility.

Many-shot Datasets We perform 90/10 train-val split for CIFAR-100, RESISC and Clevr-Distance. The split details are provided in our code base for reproducibility. We apply horizontal flipping for CIFAR100, horizontal and vertical flipping for Resisc, and no augmentation for Clevr. All data are normalized by ImageNet mean and standard deviation.

B. Background

B.1. Vision Transformer

Overview of ViT. Inspired by the recent success of Transformer-based models [90] in NLP [95], Vision Transformer (ViT) [19] has become widely used in computer vision. To handle 2D images, ViT divides an image

[3]To demonstrate it, we apply simple data augmentations (RandomResizedCrop, RandomVerticalFlip and RandomHorizontalFlip) on three datasets in each group, as shown in Table 4.

(a) ImageNet [15] (b) ImageNetV2 [78] (c) ImageNet-R [34]

(d) ImageNet Sketch [93] (e) ImageNet-A [35]

Figure 7. Samples of the class lemon, from the fine-tuned dataset ImageNet and distribution shifts datasets (ImageNet-V2, ImageNet-R, ImageNet-S, and ImageNet-A). The CLIP model is fine-tuned with PEFT on ImageNet and evaluated on distribution shifts datasets to measure the robustness of fine-tuned models. The figures are modified based on [96].

Method	Hyperparamters	#Params (M)
VPT-Shallow	Prompt Number: [5, 10, 50, 100, 200]	0.0003 ∼ 0.153
VPT-Deep	Prompt Number: [5, 10, 50, 100]	0.046 ∼ 0.921
BitFit	N/A	0.102
DiffFit	N/A	0.140
LayerNorm	N/A	0.038
SSF	N/A	0.205
Pfeif. Adapter	Adapter Scale Factor: [0.01, 0.1, 1, 10] Adapter Bottleneck: [4, 8, 16, 32]	0.082 ∼0.599
Houl. Adapter	Adapter Scale Factor: [0.01, 0.1, 1, 10] Adapter Bottleneck: [4, 8, 16, 32]	0.165 ∼1.198
AdaptFormer	Adapter Scale Factor: [0.05, 0.1, 0.2] Adapter Bottleneck: [4, 16, 32]	0.082 ∼0.599
RepAdapter	RepAdapter Scale Factor: [0.1, 0.5, 1, 5, 10] RepAdapter Bottleneck: [8, 16, 32]	0.239 ∼0.903
Convpass	Convpass Scale Factor: [0.01, 0.1, 1, 10, 100] Convpass Bottleneck: [8, 16] Convpass Xavier Init: [True, False]	0.327 ∼0.664
LoRA	LoRA Bottleneck: [1, 8, 16, 32]	0.036 ∼1.179
FacT_TT	FacT Scale Factor: [0.01, 0.1, 1, 10, 100] FacT Bottleneck: [8, 16, 32]	0.021 ∼0.196
FacT_TK	FacT Bottleneck: [16, 32, 64] FacT Scale Factor: [0.01, 0.1, 1, 10, 100]	0.030 ∼0.369

Table 3. Methods-specific hyperparameter searching grip for VTAB-1K experiment.

$\boldsymbol{I} \in \mathbb{R}^{H\times W\times C}$ into N non-overlapping patches $\{\boldsymbol{I}^{(n)} \in \mathbb{R}^{P^2\times C}\}_{n=1}^{N}$, where (H, W) is the resolution of the input image, C is the number of channels, $N = HW/P^2$ and (P, P) is the resolution of each patch. Each patch $\boldsymbol{I}^{(n)}$ is flattened and embedded into a D-dimensional vector $\boldsymbol{x}_0^{(n)}$ with a trainable linear projection. Incorporating the BERT design approach [46], a "Class" token $\boldsymbol{x}_0^{(\text{Class})}$ is prepended to the sequence of embedded patches, whose output state at the last Transformer layer is utilized as the image representation. Finally, position embeddings $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{D\times(1+N)}$ are added to preserve positional information and form the input $\mathbf{Z}_0 \in \mathbb{R}^{D\times(1+N)}$ to the ViT, which can be formulated by:

$$\boldsymbol{Z}_0 = \left[\boldsymbol{x}_0^{(\text{Class})}, \boldsymbol{x}_0^{(1)}, \boldsymbol{x}_0^{(2)}, \cdots, \boldsymbol{x}_0^{(N)}\right] + \mathbf{E}_{\text{pos}} \qquad (5)$$

			Linear	Full	VPT-Shallow	VPT-Deep	BitFit	DiffFit	LayerNorm	SSF	Pfeif. Adapter	Houl. Adapter	Adapt-Former	Rep-Adapter	Convpass	LoRA	FacT_TT	Fact_TK
	Caltech101	Simple DA	84.4	76.8	84.9	84.8	83.8	85.7	85.8	86.1	87.4	86.0	84.9	86.4	85.2	86.8	85.5	86.0
		Default	86.6	89.9	88.7	91.5	90.5	90.2	89.7	89.8	91.5	92.1	91.8	92.5	92.1	92.6	91.8	92.5
		Δ	-2.2	-13.1	-3.8	-6.7	-6.7	-4.5	-3.9	-3.7	-4.1	-6.1	-6.9	-6.1	-6.9	-5.8	-6.3	-6.5
Natural	DTD	Simple DA	67.5	57.8	69.0	71.1	70.7	73.7	73.5	68.4	72.7	72.6	70.6	71.5	71.8	73.0	72.2	71.9
		Default	65.7	61.9	67.9	69.4	70.3	71.2	72.2	68.8	72.1	72.3	70.5	69.1	72.0	69.8	71.5	71.8
		Δ	1.8	-4.1	1.1	1.7	0.4	2.5	1.3	-0.4	0.6	0.3	0.1	2.4	-0.2	3.2	0.7	0.1
	Flower102	Simple DA	98.1	92.2	98.2	98.6	98.0	98.8	98.8	98.8	98.4	97.5	98.7	98.0	98.9	98.7	98.7	98.7
		Default	98.9	97.4	99.1	99.1	98.9	99.2	99.1	99.1	99.2	98.0	99.2	99.1	99.3	99.1	99.3	99.1
		Δ	-0.8	-5.2	-0.9	-0.5	-0.9	-0.4	-0.3	-0.3	-0.8	-0.5	-0.5	-1.1	-0.4	-0.4	-0.6	-0.4
	EuroSAT	Simple DA	87.3	91.0	88.4	92.0	92.0	91.7	91.9	92.5	92.1	92.5	92.3	93.1	92.7	92.8	93.3	92.8
		Default	90.0	88.1	90.3	94.9	95.0	94.1	93.8	94.5	95.5	95.3	95.0	95.3	95.8	94.9	94.9	95.5
		Δ	-2.7	2.9	-1.9	-2.9	-3.0	-2.4	-1.9	-2.0	-3.4	-2.8	-2.7	-2.2	-3.1	-2.1	-1.6	-2.7
Specialized	Resisc45	Simple DA	74.3	75.0	74.4	80.1	81.0	78.5	80.7	80.6	80.6	81.6	82.2	81.5	81.5	82.2	80.7	82.9
		Default	74.9	81.6	77.2	84.2	85.3	80.9	83.0	83.2	85.3	86.5	86.5	86.0	85.9	85.9	85.0	86.0
		Δ	-0.6	-6.6	-2.8	-4.1	-4.3	-2.4	-2.3	-2.6	-4.7	-4.9	-4.3	-4.5	-4.4	-3.7	-4.3	-3.1
	Retinopathy	Simple DA	74.5	73.6	74.7	76.3	76.3	76.7	76.4	76.4	77.3	75.6	77.0	77.1	76.8	76.2	75.3	77.0
		Default	74.6	73.6	74.4	73.9	75.5	75.2	75.2	74.8	76.2	75.2	76.3	75.4	75.9	75.7	75.6	75.7
		Δ	-0.1	0.0	0.3	2.4	0.8	1.5	1.2	1.6	1.1	0.4	0.7	1.7	0.9	0.5	-0.3	1.3
	dSpr-Ori	Simple DA	22.6	29.9	24.3	29.6	28.7	29.0	29.0	27.9	28.9	22.9	30.9	31.4	30.5	30.3	32.5	28.4
		Default	29.4	46.6	43.1	56.4	53.9	52.8	52.1	52.1	56.6	54.3	53.0	52.1	55.3	47.2	53.1	53.1
		Δ	-6.8	-16.7	-18.8	-26.8	-25.2	-23.8	-23.1	-24.2	-27.7	-31.4	-22.1	-20.7	-24.8	-16.9	-20.6	-24.7
Structured	KITTI	Simple DA	49.8	48.7	49.4	52.7	52.6	54.7	52.7	50.6	53.9	53.6	53.2	52.2	51.3	52.9	52.0	53.3
		Default	64.6	77.9	66.5	77.9	79.2	81.0	78.1	81.4	80.2	79.6	80.0	80.2	78.1	79.9	79.3	78.9
		Δ	-14.8	-29.2	-17.1	-25.2	-26.6	-26.3	-25.4	-30.8	-26.3	-26.0	-26.8	-28.0	-26.8	-27.0	-27.3	-25.6
	sNORB-Azim	Simple DA	15.0	24.2	12.8	21.2	21.9	23.4	18.8	24.5	25.1	26.4	24.8	26.9	25.7	24.5	23.4	18.2
		Default	17.3	31.0	15.2	33.2	30.1	30.7	24.3	31.9	33.8	34.2	33.0	35.7	38.6	33.4	32.8	27.8
		Δ	-2.3	-6.8	-2.4	-12.0	-8.2	-7.3	-5.5	-7.4	-8.7	-7.8	-8.2	-8.8	-12.9	-8.9	-9.4	-9.6

Table 4. We apply simple data augmentations (DA) (RandomResizedCrop, RandomVerticalFlip and RandomHorizontalFlip) on three datasets in each group. Data augmentation does not benefit most of VTAB-1K datasets and thus, most recent PEFT papers [42–44, 55, 63, 63, 107] skip it. Figure 8 shows examples of how some data augmentation transforms are harmful for a specific task. Therefore, to ensure that our results are directly comparable to existing papers, we don't apply data augmentation.

Symbol (Abbreviation)	Definition
(H, W)	Resolution of input images
C	Number of channels (input images)
P	Resolution of patches
N	Number of patches (tokens)
N_h	Number of head in each Transformer layer
D	Embedding dimension
D_h	Embedding dimension for single-head attention
L_m	m-th Transformer layer
M	Number of Transformer layer
Z_{m-1}	Input of m-th Transformer layer
ViT	Vision Transformer
LN	Layer Normalization
MSA	Multi-head Self-Attention
MLP	Multi-Layer Perceptron
FC	Fully-connected layer

Table 5. Definitions of symbols and abbreviation used in Appendix B

As shown in the left part of Figure 9, a ViT typically consists of M layers, denoted by $\{L_m\}_{m=1}^{M}$. The input $\boldsymbol{Z}_0$ mentioned above is fed into the first layer L_1, producing the output $\boldsymbol{Z}_1 = L_1(\boldsymbol{Z}_0) = [\boldsymbol{x}_1^{(\text{Class})}, \boldsymbol{x}_1^{(1)}, \cdots, \boldsymbol{x}_1^{(N)}] \in \mathbb{R}^{D\times(1+N)}$, which maintains the same size as $\boldsymbol{Z}_0$. Namely, $\boldsymbol{Z}_1$ comprises $1+N$ feature tokens, and each corresponds to the same column in $\boldsymbol{Z}_0$. Similarly, for $m = 2, \cdots, M$, each layer L_m takes the output of the previous layer as input and generates the output, $\boldsymbol{Z}_m = L_m(\boldsymbol{Z}_{m-1})$. Finally, the "Class" vector $\boldsymbol{x}_M^{(\text{Class})}$ in $\boldsymbol{Z}_M$ serves as the image feature for prediction. When dealing with classification tasks, the predicted label $\hat{y} = \mathsf{Head}(\boldsymbol{x}_M^{(\text{Class})})$ is generated through a linear head (*i.e.*, a fully-connected layer).

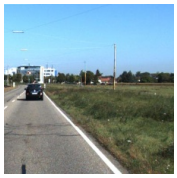
Details of each Transformer layer. As shown in the right part of Figure 9, each Transformer layer consists of a Multi-head Self-Attention (MSA) block, a Multi-Layer Perceptron (MLP) block, and two Layer Normalization (LN) layers [4]. Formally, a Transformer layer L_m can be defined as

$$\begin{aligned} \boldsymbol{Z}'_m &= \text{MSA}\left(\text{LN}\left(\boldsymbol{Z}_{m-1}\right)\right) + \boldsymbol{Z}_{m-1} \\ \boldsymbol{Z}_m &= \text{MLP}\left(\text{LN}\left(\boldsymbol{Z}'_m\right)\right) + \boldsymbol{Z}'_m \end{aligned} \tag{6}$$

where $\boldsymbol{Z}_{m-1} = [\boldsymbol{x}_{m-1}^{(\text{Class})}, \boldsymbol{x}_{m-1}^{(1)}, \cdots, \boldsymbol{x}_{m-1}^{(N)}] \in \mathbb{R}^{D\times(1+N)}$ is the output of the preceding $(m-1)$-th Transformer layer. The MLP is applied to each column vector of $\boldsymbol{Z}'_m$ indepen-

(a) Original

(b) Vertical Flipped



(c) Cropped

(d) Resized

Figure 8. Example of augmented images from KITTI: (a) Original, (b) RandomVerticalFlip, (c) RandomResizedCrop, (d) Resize only. The KITTI task needs to predict the depth to the nearest vehicle (car, van, or truck) in the image. RandomResizedCrop may crop out the nearest vehicle. RandomVerticalFlip may make the task more difficult.

dently.

In order to encapsulate multiple complex relationships amongst different elements in the sequence, the MSA block comprises N_h single-head self-attention blocks. For the i^{th} single-head self-attention block, an generic input $\boldsymbol{Z}$ is first projected into three matrices, namely Query $\boldsymbol{Q}^{(i)}$, Key $\boldsymbol{K}^{(i)}$, and Value $\boldsymbol{V}^{(i)}$

$$\boldsymbol{Q}^{(i)} = \boldsymbol{W}_Q^{(i)}\boldsymbol{Z}, \quad \boldsymbol{K}^{(i)} = \boldsymbol{W}_K^{(i)}\boldsymbol{Z}, \quad \boldsymbol{V}^{(i)} = \boldsymbol{W}_V^{(i)}\boldsymbol{Z}, \tag{7}$$

where $\boldsymbol{W}_{Q/K/V}^{(i)} \in \mathbb{R}^{D_h \times D}$ [4] where D_h is the embedding dimension for a single head self-attention block and typically set to D/N_h. The i^{th} self-attention head in MSA is formulated as

$$\text{Attn}^{(i)}(\boldsymbol{Z}) = \boldsymbol{V}^{(i)} \times \text{Softmax}\left(\frac{\boldsymbol{K}^{(i)\top}\boldsymbol{Q}^{(i)}}{\sqrt{D_h}}\right) \in \mathbb{R}^{D_h \times (1+N)} \tag{8}$$

[4] For brevity, we ignore the layer index m for the projection matrices $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$, but each layer has its own projection matrices.

The outputs of all heads are concatenated and linearly projected by a fully connected layer (FC_{attn}) with weight $\boldsymbol{W}_O \in \mathbb{R}^{D \times (D_h \cdot N_h)}$ as the output of the MSA block.

$$\text{MSA}(\boldsymbol{Z}) = \boldsymbol{W}_O \left[\text{Attn}^0(\boldsymbol{Z}), \ldots, \text{Attn}^{N_h}(\boldsymbol{Z})\right] \tag{9}$$

The MLP block can be defined as

$$\text{MLP}(\boldsymbol{Z}) = GELU\left(\boldsymbol{Z}\boldsymbol{W}_1 + b_1\right)\boldsymbol{W}_2 + b_2 \tag{10}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{D \times 4D}$ [5], $\boldsymbol{W}_2 \in \mathbb{R}^{4D \times D}$, $b_1 \in \mathbb{R}^{4D}$, $b_2 \in \mathbb{R}^D$ are weights and biases for two FC layers (FC_1 and FC_2) respectively.

Since PEFT methods often entail incorporating additional components to modify the intermediate features within or between Transformer layers, we adopt the notation $\{h_1, \ldots, h_{10}\}$ to denote the intermediate features in the unravelled view of a Transformer layer (as depicted in Figure 9) to facilitate a clearer illustration of the PEFT methods discussed in the subsequent section.

B.2. Evaluated Methods

In this section, we dive into the details of 12 state-of-the-art PEFT approaches, categorized into three groups: Prompt-based, Adapter-based, and Selective Parameter Tuning. We will describe the distinctions and tradeoffs between them. A consolidated overview of these approaches is summarized in Table 6.

B.2.1. Prompt-based Methods

Prompt-based learning emerged in NLP as an effective approach to adapt pre-trained models for downstream tasks [54, 57]. The core concept involves augmenting the model input with task-specific hints (prompts), which aid the pre-trained model in addressing novel tasks with its existing knowledge. Hard prompts are human-interpretable natural language hints, encompassing task instructions, in-context examples, or supporting information. Alternatively, soft prompts are continuous vector hints that are incorporated into the input embeddings of the input layers or hidden states of other layers. Soft prompts are updated during the fine-tuning process using gradient-based methods, guided by the downstream task-specific loss functions, while the pre-trained model itself remains fixed. The splendent success of prompts in NLP has sparked a growing interest in adopting it in computer vision [88, 102] and multi-modal domains [27].

In this paper, we investigate a prominent and strong prompt-based method called **Visual Prompt Tuning (VPT)** [42], which represents one of the early endeavours in introducing prompts to computer vision. Specifically, VPT-Shallow adds l prompts $\boldsymbol{P}_0 \in \mathbb{R}^{l \times D}$ to the input of the

[5] 4 is the MLP ratio in ViT-B

Table 6. PEFT Methods Summary: Prompt-based and adapter-based methods incorporate additional parameters to modify features while keeping the original backbone intact. However, these added parameters introduce additional inference overhead. In contrast, selective tuning methods modify the backbone by updating selective parameters, thereby incurring no additional inference overhead.

Method	What	Tunable Parameters	Hyper Parameters	Modified Type	Inference Efficient
VPT-Deep	$\boldsymbol{h_1} = [\boldsymbol{h_1}, \boldsymbol{P}]$	$\boldsymbol{P} \in \mathbb{R}^{l \times D}$	l: Number of prompts	Feature	✗
AdaptFormer	$\boldsymbol{h_9} = \boldsymbol{h_9} + \text{Adapter}(\boldsymbol{h_7})$	$\boldsymbol{W}_{\text{down /up}} \in \mathbb{R}^{r \times D/D \times r}$ in Adapter	s: Scale factor in Adapter r: Bottleneck dimension	Feature	✗
Pfeif. Adapter	$\boldsymbol{h_9} = \text{Adapter}(\boldsymbol{h_9})$	$\boldsymbol{W}_{\text{down /up}} \in \mathbb{R}^{r \times D/D \times r}$ in Adapter	s: Scale factor in Adapter r: Bottleneck dimension	Feature	✗
Houl. Adapter	$\boldsymbol{h_5} = \text{Adapter}_1(\boldsymbol{h_5})$ $\boldsymbol{h_9} = \text{Adapter}_2(\boldsymbol{h_9})$	$\boldsymbol{W}^1_{\text{down /up}} \in \mathbb{R}^{r \times D/D \times r}$ in Adapter_1 $\boldsymbol{W}^2_{\text{down /up}} \in \mathbb{R}^{r \times D/D \times r}$ in Adapter_2	s: Scale factor in Adapter r: Bottleneck dimension	Feature	✗
Convpass	$\boldsymbol{h_5} = \text{Convpass}_1(\boldsymbol{h_2}) + \boldsymbol{h_5}$ $\boldsymbol{h_9} = \text{Convpass}_2(\boldsymbol{h_7}) + \boldsymbol{h_9}$	$\boldsymbol{W}^1_{conv2d} \in \mathbb{R}^{r \times r \times k \times k}$ $\boldsymbol{W}^1_{\text{down /up}} \in \mathbb{R}^{r \times D/D \times r}$ in Convpass_1 $\boldsymbol{W}^2_{conv2d} \in \mathbb{R}^{r \times r \times k \times k}$ $\boldsymbol{W}^2_{\text{down /up}} \in \mathbb{R}^{r \times D/D \times r}$ in Convpass_2	s: Scale factor in Convpass r: Bottleneck dimension k: Kernel size of $conv2d$	Feature	✗
RepAdpater	$\boldsymbol{h_2} = \text{RepAdapter}_1(\boldsymbol{h_2})$ $\boldsymbol{h_7} = \text{RepAdapter}_2(\boldsymbol{h_7})$	$\boldsymbol{W}^1_{conv1d} \in \mathbb{R}^{r \times D}$ $\boldsymbol{b}^1 \in \mathbb{R}^r$ in RepAdapter_1 $\boldsymbol{W}^2_{conv1d} \in \mathbb{R}^{D \times \frac{r}{G}}$ $\boldsymbol{b}^2 \in \mathbb{R}^D$ in RepAdapter_2	s: Scale factor in RepAdapter r: Bottleneck dimension G: Number of groups	Feature	✗
LayerNorm	$\boldsymbol{h_2} = \text{LayerNorm}_1(\boldsymbol{h_1})$ $\boldsymbol{h_7} = \text{LayerNorm}_2(\boldsymbol{h_6})$	$\boldsymbol{W}^{1(2)}, \boldsymbol{b}^{1(2)} \in \mathbb{R}^D$ in $\text{LayerNorm}_{1(2)}$	N/A	Backbone	✓
BitFit	Fine-tune all bias terms in the network	$\boldsymbol{b}^{1(2)} \in \mathbb{R}^D$ in $\text{LayerNorm}_{1(2)}$ $\boldsymbol{b}^{Q/K/V} \in \mathbb{R}^D$ in Q/K/V $\boldsymbol{b}^{FC_{attn}} \in \mathbb{R}^D$ in FC_{attn} $\boldsymbol{b}^1 \in \mathbb{R}^{4D}$, in FC_1, $\boldsymbol{b}^2 \in \mathbb{R}^D$ in FC_2	N/A	Backbone	✓
DiffFit	• LayerNorm + BitFit • $\boldsymbol{h_5} = \gamma_1 \cdot \boldsymbol{h_5}$ $\boldsymbol{h_9} = \gamma_2 \cdot \boldsymbol{h_9}$	• All tunable parameters in LayerNorm & BitFit • $\gamma_1, \gamma_2 \in \mathbb{R}^D$	N/A	Backbone	✓
SSF	$\boldsymbol{h_2} = \text{SSF}_2(\boldsymbol{h_2}),\ \boldsymbol{h_3} = \text{SSF}_3(\boldsymbol{h_3})$ $\boldsymbol{h_5} = \text{SSF}_5(\boldsymbol{h_5}),\ \boldsymbol{h_7} = \text{SSF}_7(\boldsymbol{h_7})$ $\boldsymbol{h_8} = \text{SSF}_7(\boldsymbol{h_8}),\ \boldsymbol{h_9} = \text{SSF}_9(\boldsymbol{h_9})$	$\boldsymbol{W}^{2,5,7,9} \in \mathbb{R}^D, \boldsymbol{b}^{2,5,7,9} \in \mathbb{R}^D$ $\boldsymbol{W}^3 \in \mathbb{R}^{3D}, \boldsymbol{b}^3 \in \mathbb{R}^{3D}$ $\boldsymbol{W}^8 \in \mathbb{R}^{4D}, \boldsymbol{b}^8 \in \mathbb{R}^{4D}$	N/A	Backbone	✓
LoRA	$\boldsymbol{h_3} = \text{LoRA}(\boldsymbol{h_2}) + \boldsymbol{h_3}$	$\boldsymbol{W}^{Q/K/V}_{\text{down/up}} \in \mathbb{R}^{r \times D/D \times r}$ in LoRA	r: Bottleneck dimension	Backbone	✓
$\text{FacT}_{\text{TT(TK)}}$	$\boldsymbol{h_3} = \text{FacT}_{\text{TT(TK)}}(\boldsymbol{h_2}) + \boldsymbol{h_3}$ $\boldsymbol{h_5} = \text{FacT}_{\text{TT(TK)}}(\boldsymbol{h_4}) + \boldsymbol{h_5}$ $\boldsymbol{h_8} = \text{FacT}_{\text{TT(TK)}}(\boldsymbol{h_7}) + \boldsymbol{h_8}$ $\boldsymbol{h_9} = \text{FacT}_{\text{TT(TK)}}(\boldsymbol{h_8}) + \boldsymbol{h_9}$	$\boldsymbol{U} \in \mathbb{R}^{D \times r}, \boldsymbol{V} \in \mathbb{R}^{D \times r}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{12L \times r \times r}$ in FacT_{TT} --- $\boldsymbol{U} \in \mathbb{R}^{D \times r}, \boldsymbol{V} \in \mathbb{R}^{D \times r}$, $\boldsymbol{A} \in \mathbb{R}^{12L \times r}, \boldsymbol{B} \in \mathbb{R}^{r \times r \times r}$ in FacT_{TK}	s: Scale factor in $\text{FacT}_{\text{TT(TK)}}$ r: Bottleneck dimension	Backbone	✓

first Transformer layer $\boldsymbol{Z}_0$ and the output $\tilde{\boldsymbol{P}}_0$ of $\boldsymbol{P}_0$ serves as the input for the next layer as depicted in Equation 11. VPT-Shallow can be perceived as the addition of learnable pixels to the original images. On the other hand, VPT-Deep inserts l prompts $\{\boldsymbol{P}_m \in \mathbb{R}^{l \times D}\}_{m=0}^{M}$ to the input of every Transformer layer $\boldsymbol{Z}_m$ but their outputs are discarded at the end of the layer as illustrated in Equation 12.

$$
\begin{aligned}
[\tilde{\boldsymbol{P}}_1, \boldsymbol{Z}_1] &= L_m([\boldsymbol{P}_0, \boldsymbol{Z}_0]) \\
[\tilde{\boldsymbol{P}}_m, \boldsymbol{Z}_m] &= L_m([\tilde{\boldsymbol{P}}_{m-1}, \boldsymbol{Z}_{m-1}]) \quad m = 2, 3, \ldots, M
\end{aligned} \tag{11}
$$

$$
[\_, \boldsymbol{Z}_m] = L_m([\boldsymbol{P}_{m-1}, \boldsymbol{Z}_{m-1}]) \quad m = 1, 2, 3, \ldots, M \tag{12}
$$

Throughout the adaptation process, the pre-trained model

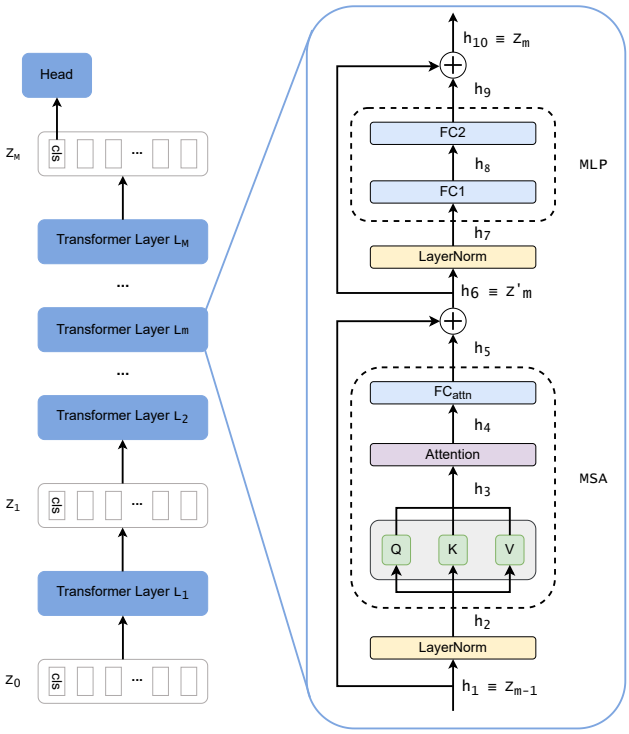


Figure 9. An overview of a Transformer block in ViT. We adopt the notation $\{h_1, \ldots, h_{10}\}$ to denote the intermediate features within a Transformer block to facilitate a clearer illustration of the PEFT methods discussed in subsection B.2.

is frozen and no additional weights are introduced to the model, thereby preserving the model's original behaviour. During the forward pass, the output $\boldsymbol{Z}_m$ of layer m is changed because of the interaction between $\boldsymbol{Z}_{m-1}$ and $\boldsymbol{P}_{m-1}$ (or $\tilde{\boldsymbol{P}}_{m-1}$) in the MSA block. Thus, the output feature is adapted to the downstream tasks by iteratively tuning the prompts through gradient descent.

B.2.2. Adapter-based Methods

Adapter-based methods typically introduce additional trainable parameters into a frozen pre-trained model to facilitate learning of downstream tasks [54]. Initially developed for multi-domain adaptation [76, 77] and continual learning [66, 80], the idea of Adapters is subsequently embraced by Houlsby *et al*. [36] in the NLP domain to adapt Transformer-based networks for downstream tasks, and it also has garnered increasing interest in the computer vision field [102]. In this comparative analysis, we concentrate on*five* popular Adapter-based methods, encompassing the original Adapter, along with variants focusing on adjusting the positions of Adapters [11, 74], introducing visual inductive biases [43], as well as employing re-parameterization to reduce the number of trainable parameters and inference latency [63].

Houl. Adapter [36] inserts two lightweight bottleneck-structured modules into each Transformer layer: one after the MSA block and the other after the MLP block. As depicted in Figure 10a, the Adapter is composed of a down-projection layer with $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{r \times D}$, a nonlinear activation function σ, an up-projection layer with $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{D \times r}$, a scaling factor s and a skip-connection. To limit the number of trainable parameters, the bottleneck dimension is much smaller than the feature dimension $r \ll D$. Formally, Houl. Adapter can be defined as:

$$h_5 = \text{Adapter}_1(h_5) \quad h_9 = \text{Adapter}_2(h_9) \tag{13}$$

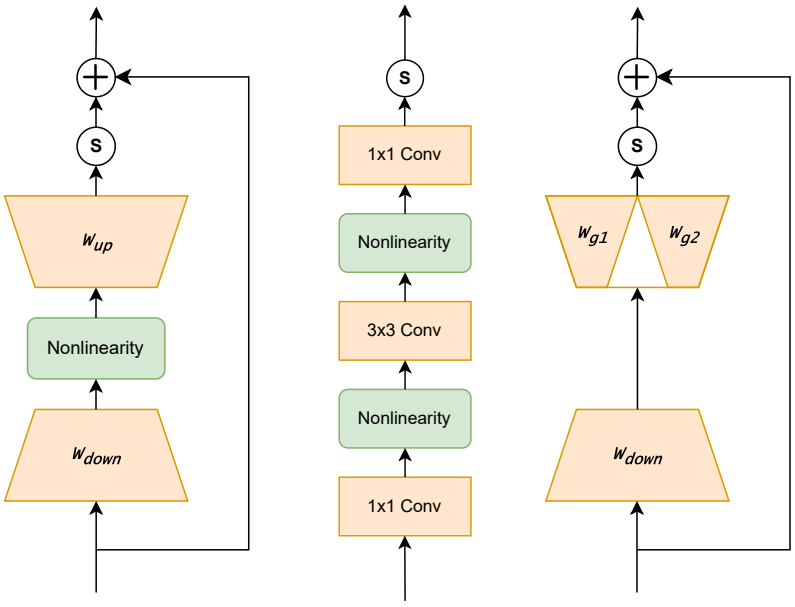
$$\text{Adapter}(h) = s \cdot \boldsymbol{W}_{\text{up}} \sigma(\boldsymbol{W}_{\text{down}}\, h) + h \tag{14}$$

Pfeif. Adapter [74] is a more efficient variant that introduces the Adapter solely after the MLP block, a strategy that has demonstrated effectiveness in recent studies [37]. Pfeif. Adapter can be defined formally as $h_9 = \text{Adapter}(h_9)$ where Adapter follows Equation 14.

AdaptFormer [11] proposed to insert the Adapter in parallel with the MLP block, which differs from the sequential design of Houl. and Pfeif. Adapter. The rationale behind this parallel design lies in the belief that the domain-specific features generated by the Adapter can complement the domain-agnostic features derived from the original MLP block, leading to an improved feature ensemble [85]. Formally, AdaptFormer can be defined as $\boldsymbol{h_9} = \boldsymbol{h_9} + \text{Adapter}(\boldsymbol{h7})$ where Adapter follows Equation 14.

ConvPass (Convolutional By-Passes) [43] addresses the concern that many existing Adapters lack visual inductive bias, potentially limiting their performance for downstream vision tasks with limited data. To this end, the authors introduce a convolutional bottleneck module, running in parallel with the MSA or(and) MLP block. This module encompasses a 1×1 convolution reducing the channel with $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{r \times D}$, a 3×3 convolution with the same input and output channel, a 1×1 convolution expanding the channel $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{D \times r}$, two nonlinear functions σ and a scaling factor s, as shown in Figure 10b. The authors argue that Convpass is more efficient at capturing visual information in low-data scenarios due to its hard-coded locality of convolutional layers. The formal definition of Convpass is shown in Equation 15.

$$h_5 = \text{Convpass}_1(h_2) + h_5 \quad h_9 = \text{Convpass}_2(h_7) + h_9$$
$$\text{Convpass}(h) = s \cdot \boldsymbol{W}_{\text{up}} \sigma(\text{Conv2d}(\sigma(\boldsymbol{W}_{\text{down}}\, h))) \tag{15}$$



(a) Adapter (b) Convpass (c) RepAdapter

Figure 10. Comparison of three Adapter structures.

RepAdapter [63] found that the removal of the non-linear function in the Adapter does not result in performance degradation for vision tasks. In light of this finding, the authors propose a linear Adapter with group-wise transformation [62] and sequentially added two of these linear Adapters to both MSA and MLP blocks. Owing to the sequential placement of the RepAdapter and its inherent linearity, the additional parameters can be re-parameterized to the original MSA or MLP block after training, thereby incurring zero additional costs during inference. RepAdapter is illustrated in Figure 10c and formally defined in Equation 16.

$$\begin{aligned} h_5 = \text{RepAdapter}_1(h_2) \quad h_7 = \text{RepAdapter}_2(h_7) \\ \text{RepAdapter}(h) = s \cdot \phi_{\text{up}}(\phi_{\text{down}}\,(h)) + h \\ \tilde{h} = \phi_{\text{down}}\,(h) = \boldsymbol{W}_{\text{down}}\, h \\ \phi_{\text{up}}(\tilde{h}) = [\boldsymbol{W}_{g1}\tilde{h}_{g1}, \ldots, \boldsymbol{W}_{gG}\tilde{h}_{gG}] \end{aligned} \tag{16}$$

where $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{r\times D}$, $\tilde{h}_{g(1,\ldots,G)} \in \mathbb{R}^{\frac{r}{G}\times(N+1)}$ is the features splitted from $\tilde{h} \in \mathbb{R}^{r\times(N+1)}$ and G is the number of groups in group-wise transformation [62]. $\boldsymbol{W}_{g(1,\ldots,G)} \in \mathbb{R}^{\frac{D}{G}\times\frac{r}{G}}$ is the projection weight matrix.

B.2.3. Selective Parameter Tuning Methods

The methods falling within this category aim to selectively update the parameters of a pre-trained model for downstream tasks. Within transfer learning, two prominent strategies, namely *full fine-tuning* and *linear probing* [48, 111], represent the two extremes of this category. *Full fine-tuning* updates all the model parameters end-to-end based on the new dataset while *linear probing* treats the pre-trained model as a feature extractor and only updates the prediction heads while keeping the backbone frozen. Although *full fine-tuning* generally exhibits superior performance compared to *linear probing* [104], it possesses certain limitations that may hinder its practicality in real-world production settings. Firstly, it requires running gradient descent for all parameters and necessitates storing a separate fine-tuned model for each task, incurring significant computational, memory, and storage overhead. These challenges become more salient with Transformer-based models whose parameters grow exponentially. Secondly, *full fine-tuning* may distort pre-trained features and underperform *linear probing* in out-of-distribution (OOD) scenarios [50].

To cope with the above issues, a cohort of PEFT methods has emerged under this category. In addition to the two common approaches mentioned above, our investigation encompasses seven methods that can be further categorized into two groups: direct selective tuning [7, 97, 103], which involves the direct modification of selective weights, and efficient selective tuning [37, 44, 55], which approximates the weight updates with low-rank factors.

Notably, an extra advantage of methods in this category is that they introduce **no additional inference latency**, making them particularly favourable when inference efficiency is a priority. Methods within the direct selective tuning group abstain from introducing any new modules, thus inherently avoiding extra inference latency. Meanwhile, for methods in

the efficient selective tuning group, the added modules can often be seamlessly integrated into weights of the pre-trained models through the re-parameterization techniques [18, 41], thereby ensuring the absence of increased inference latency as well.

Direct Selective Tuning

BitFit [103] is a simple yet effective method that only tunes the bias parts of the pre-trained model. For each Transformer layer in ViT, BitFit updates the bias terms in the QKV projections and the FC layer in the MSA block, two FC layers in the MLP block and two LN blocks. It also updates the bias in the projection for patch embedding. The original authors underscore BitFit's capability to achieve performance comparable to full fine-tuning or even surpass it under low and medium-data scenarios in BERT models [46].

LayerNorm [7] represents another simple but strong baseline that solely tunes the two LN blocks in each Transformer layer - one before the MSA block and another before the MLP block. Given that each LN block contains merely two trainable parameters $\{\boldsymbol{W}_{LN}, \boldsymbol{b}_{LN}\} \in \mathbb{R}^D$, LN-tune stands out as an exceedingly light-weight approach compared to other PEFT methods. For instance, ViT-B/16 ($\sim$86M parameters) has only $\sim$38K LN parameters, accounting for $\sim$0.04% of the total parameters.

DiffFit [97] is a recently proposed PEFT strategy designed for adapting large pre-trained diffusion models to the new domains. DiffFit exclusively fine-tunes the bias terms and the LN blocks within the network. Furthermore, it inserts learnable scale factors γ to shift the features after the MSA and the MLP blocks, as shown in Equation 17. Consequently, DiffFit can be regarded as a combination of the BitFit and Ln-Tune, incorporating additional feature shift factors.

$$\begin{aligned} h_5 &= \gamma_1 \cdot h_5 \\ h_9 &= \gamma_2 \cdot h_9 \end{aligned} \tag{17}$$

Efficient Selective Tuning

LoRA (Low-Rank Adaptation) [37] drew inspiration from recent investigations demonstrating that the learned over-parametrized models in fact reside on a low intrinsic dimension [1, 52]. Building upon this insight, the authors hypothesize that the change in weights during model adaptation also exhibits a low intrinsic rank and injects trainable low-rank decomposition matrices to approximate the weight updates. The LoRA update methodology is strategically applied to the Query/Value projection weights $\boldsymbol{W}_{Q/V} \in \mathbb{R}^{D\times D}$ within the MSA block. Concretely, the weight updates are approximated as $\boldsymbol{W}_{Q/V} + \Delta\boldsymbol{W}_{Q/V} = \boldsymbol{W}_{Q/V} + \boldsymbol{W}^{Q/V}_{\text{down}}\boldsymbol{W}^{Q/V}_{\text{up}}$ where $\boldsymbol{W}^{Q/V}_{\text{down/up}} \in \mathbb{R}^{D\times r/r\times D}$ and rank $r \ll D$. The authors use a random Gaussian initialization for $\boldsymbol{W}^{Q/V}_{\text{up}}$ and zero for $\boldsymbol{W}^{Q/V}_{\text{down}}$ so that $\Delta\boldsymbol{W}_{Q/V} = \boldsymbol{W}^{Q/V}_{\text{down}}\boldsymbol{W}^{Q/V}_{\text{up}}$ is zero at the beginning of training. The formal definition of LoRA is articulated in Equation 18, utilizing the notations delineated in Figure 9.

$$\begin{aligned} h_3 &= \text{LoRA}(h_2) + h_3 \\ h_3 &= [\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}] \\ \text{LoRA}(h_2) &= [\boldsymbol{W}^{Q}_{\text{down}}\boldsymbol{W}^{Q}_{\text{up}}h_2, 0, \boldsymbol{W}^{V}_{\text{down}}\boldsymbol{W}^{V}_{\text{up}}h_2] \end{aligned} \tag{18}$$

FacT (Factor Tuning) [44] is inspired by the recent advances in Transformer compression [92, 106]and exploited the low-rank update paradigm (e.g., LoRA) to the extreme. While LoRA posits that the update for an individual weight matrix manifests a low-rank characteristic during fine-tuning, FacT advances the proposition that the weight updates spanning different matrices can also be effectively approximated using low-rank decomposition matrices. Specifically, FacT encapsulates the four weight matrices $\boldsymbol{W}_{Q/K/V/O} \in \mathbb{R}^{D\times D}$ in the MSA block and the two weight matrices $\boldsymbol{W}_1 \in \mathbb{R}^{D\times 4D}$, $\boldsymbol{W}_2 \in \mathbb{R}^{4D\times D}$ in the MLP block into a single $\boldsymbol{W}_{FacT} \in \mathbb{R}^{12M\times D\times D}$ tensor where M is the number of Transformer layer. The update of $\boldsymbol{W}_{FacT}$, $\Delta\boldsymbol{W}_{FacT}$, can be decomposed into several factors to promote parameter efficiency. To this end, the authors leverage the well-established Tensor-Train (TT) [72]and the Tucker (TK) [14] format to decompose $\Delta\boldsymbol{W}_{FacT}$. FacT$_{\text{TT}}$ and FacT$_{\text{TK}}$ are used to denote different decomposition formats for FacT and their formal definitions can be found in Equation 19.

$$\text{FacT}_{\text{TT}} : \Delta\boldsymbol{W}_{FacT} = s \cdot \boldsymbol{\Sigma} \times_2 \boldsymbol{U}^\top \times_3 \boldsymbol{V}^\top \tag{19}$$

$$\text{FacT}_{\text{TK}} : \Delta\boldsymbol{W}_{FacT} = s \cdot \boldsymbol{A} \times_1 \boldsymbol{B}^\top \times_2 \boldsymbol{U}^\top \times_3 \boldsymbol{V}^\top \tag{20}$$

where $\boldsymbol{U} \in \mathbb{R}^{D\times r}, \boldsymbol{V} \in \mathbb{R}^{D\times r}, \boldsymbol{\Sigma} \in \mathbb{R}^{12L\times r\times r}, \boldsymbol{B} \in \mathbb{R}^{12L\times r}\boldsymbol{A} \in \mathbb{R}^{r\times r\times r}$ and the $\times_j$ denotes mode-j product and s is the scaling factor.

Since $\Delta\boldsymbol{W}_{FacT}$ contains the updates for $\boldsymbol{W}_{Q/K/V/O}, \boldsymbol{W}_{1/2}$, the modified forward pass inherently influences h_3, h_5, h_8, h_9. Let's consider h_5 for elucidation. Once the weight update $\Delta\boldsymbol{W}_{FacT}$ is calculated with FacT$_{\text{TT(TK)}}$ in Equation 19, the corresponding update for $\boldsymbol{W}_O$, $\Delta\boldsymbol{W}_O$, is extracted from $\Delta\boldsymbol{W}_{FacT}$. Similar to the modified forward pass of LoRA, $h_5 = h_4\Delta\boldsymbol{W}_O + h_5$.

SSF (Scale & Shift deep Features) [55] employs linear transformations to adapt the intermediate features extracted

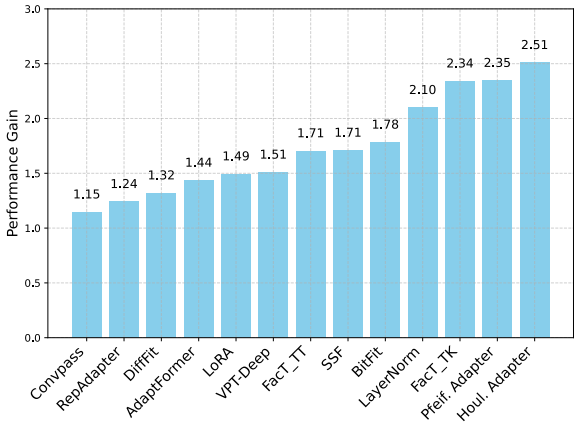


Figure 11. Performance gain for PEFT methods by turning drop-path-rate on.

by a pre-trained model. Motivated by the feature modulation methods [39, 73], SSF is designed to accommodate the distribution difference between the upstream and downstream datasets. Specifically, SSF modulates the features residing at $h_2, h_3, h_5, h_7, h_8, h_9$ by incorporating scale and shift factors. To demonstrate the mechanism of SSF, let's consider $h_5 \in \mathbb{R}^{(N+1)\times D}$ as an illustrative example and other features can similarly undergo the same transformative process. Formally, the modulated h_5 is formulated as follows.

$$h_5 = \mathrm{SSF}_5(h_5) = \boldsymbol{w}^5 \odot h_5 + \boldsymbol{b}^5 \tag{21}$$

where $\boldsymbol{w}^5 \in \mathbb{R}^D$, $\boldsymbol{b}^5 \in \mathbb{R}^D$ are the scale and shift factors affiliated with the SSF module attributed to h_5, and $\odot$ is the dot product. It is noteworthy that each modulated feature has its own SSF module with corresponding scale and shift factors. The modification details for other features are summarized in Table 6.

C. More Detailed Results

Drop-path-rate. Learning with low-shot data is prone to over-fitting. We find that if the drop path rate — which stochastically drops a transformer block per sample [38] — is set not as default (*i.e.*, nonzero), all the methods can benefit from such a regularization. Figure 11 shows the performance gain by tuning the drop-path-rate on compared with the default 0.

More results on prediction similarity analysis. Figure 13 shows the prediction analysis discussed in section 4 for all the datasets in VTAB-1K. It is expected that their predictions are similar for datasets with very high accuracy, such as Flowers102 (avg 99.1%) and Caltech101 (avg 91.4%). Beyond them, we find that most PEFT methods show diverse predictions in other datasets in VTAB-1K.

Prediction similarity within the same PEFT group. To verify if methods within the same PEFT group share more prediction similarity, we plotted the prediction overlap for adapter-based methods, selective-tuning methods, and methods from different groups. As shown in Figure 12, methods within the same group share slightly more prediction similarity than those from different groups, but they still exhibit distinct predictions. Figure 3a in the main paper also supports this observation. Methods are grouped based on the categories defined in subsection 2.2. If methods within the same group had very high similarities, we would see bright squares, which are only slightly evident around BitFit, DiffFit, LayerNorm, and SSF.

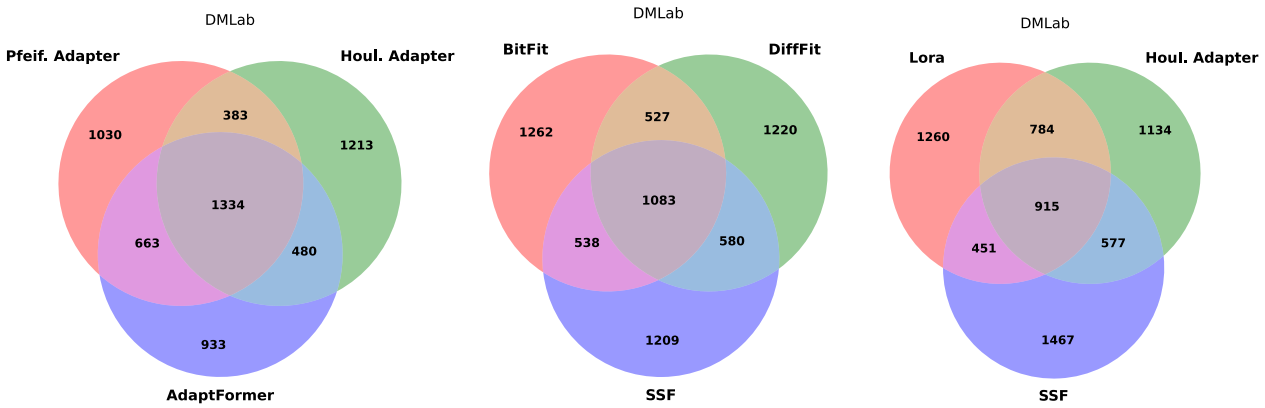
WiSE PEFT results for all distribution shift datasets. We provide detailed WiSE PEFT performance for each distribution shift dataset in Figure 14. WiSE improves both the robustness and the in-distribution performance of PEFT methods. Interestingly, even though full fine-tuning is generally less robust than PEFT methods, applying WiSE allows it to achieve better performance in both target distribution and distribution shift data.

Performance comparison between DINOv2 and IN21k. To compare the performance of DINOv2 and IN21k, we selected several PEFT methods and datasets from VTAB-1K. The results presented in Table 7 reveal several interesting findings:

(1) **Improved Linear Probing Performance**: Linear probing generally shows improved results with DINOv2, indicating that its extracted features are more robust and discriminative than those from IN21k.

(2) **Deteriorated Full Fine-Tuning Performance**: Conversely, full fine-tuning performance significantly worsens with DINOv2, suggesting that models fully fine-tuned on DINOv2 are more susceptible to overfitting.

(3) **Adapter-Based Methods Performance**: Among the three adapter-based methods evaluated—Houl. Adapter, AdaptFormer, and Convpass—we observe performance enhancements in most datasets for AdaptFormer and Convpass. In contrast, Houl. Adapter exhibits significant degradation across all datasets. This disparity may be attributed to architectural differences: AdaptFormer and Convpass insert their adapter modules in parallel with existing modules such as Multi-Head Self-Attention (MSA) and/or Multi-Layer Perceptron (MLP), whereas Houl. Adapter inserts its adapter sequentially after the MSA and MLP layers. We hypothesize that the sequential design of Houl. Adapter leads to more substantial alterations of intermediate features compared to the parallel design, potentially explaining the observed decrease in performance.



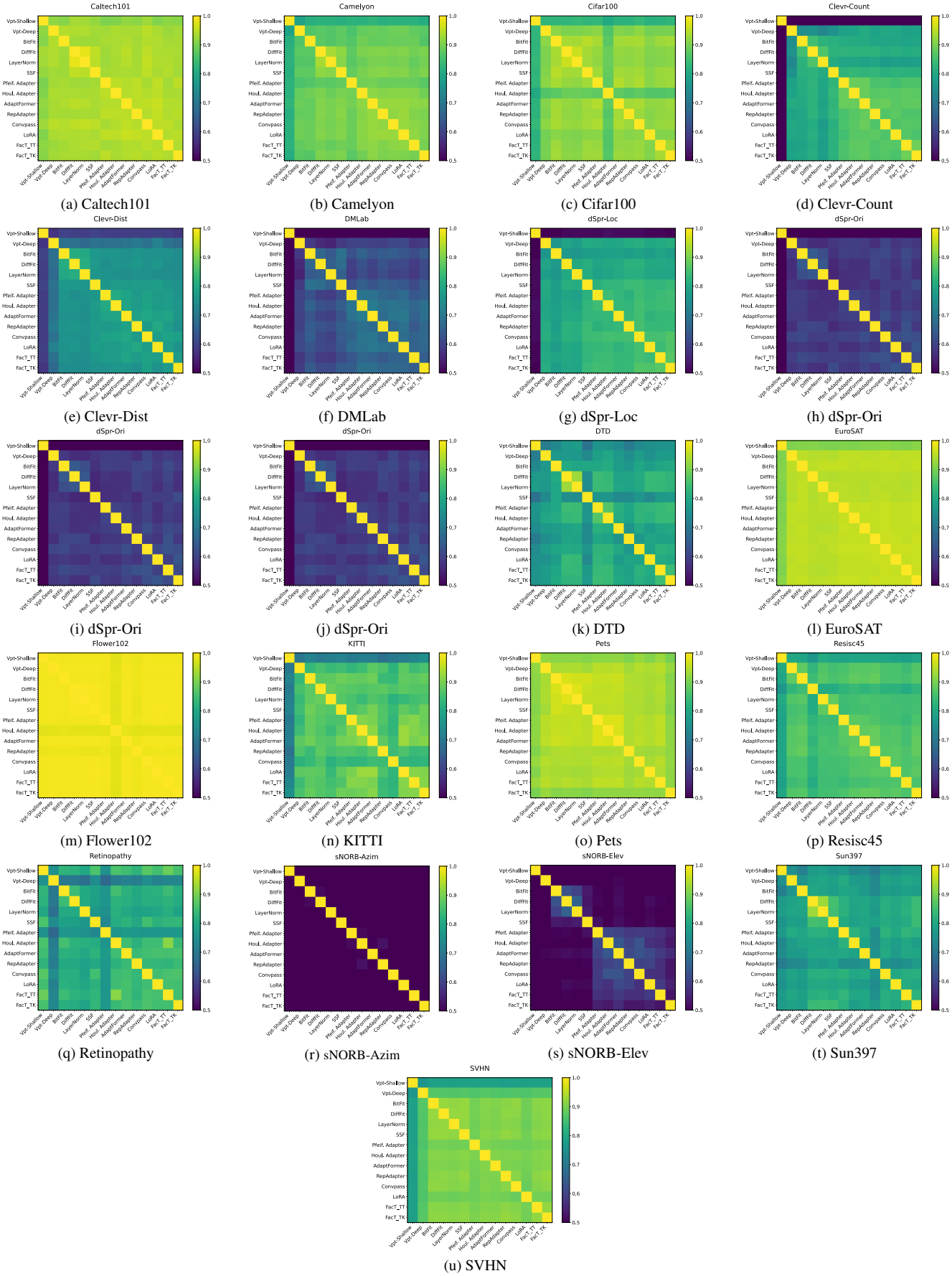
(a) Adapter-based methods (b) Selective tuning methods (c) Methods from different groups

Figure 12. Prediction overlap for the 5K most confident samples. Although methods from the same group share slightly more prediction overlap than methods from other groups, they still have quite different predictions

			Linear	Full	SSF	Houl. Adapter	Adapt-Former	Convpass	LoRA
Natural	Caltech101	Dinov2	89.8	83.2	90.3	22.1	92.5	91.7	92.3
		IN21k	86.6	89.9	89.8	92.1	91.8	92.1	92.6
		Δ	3.2	-6.7	0.5	-70.0	0.7	-0.4	-0.3
	DTD	Dinov2	74.9	45.2	77.0	14.6	78.8	77.0	78.4
		IN21k	65.7	61.9	68.8	72.3	70.5	72.0	69.8
		Δ	9.2	-16.7	8.2	-57.7	8.3	5.0	8.6
	Pets	Dinov2	93.4	68.7	92.6	7.1	94.0	92.3	94.1
		IN21k	89.3	85.8	91.4	91.7	91.8	91.3	90.5
		Δ	4.1	-17.1	1.2	-84.6	2.2	1.0	3.6
	Sun397	Dinov2	55.1	23.9	52.5	2.6	56.5	56.1	55.7
		IN21k	53.2	36.8	56.5	55.4	56.7	55.9	55.5
		Δ	1.9	-12.9	-4.0	-52.8	-0.2	0.2	0.2
Specialized	Camelyon	Dinov2	83.2	77.9	83.9	77.1	84.4	86.3	85.4
		IN21k	83.1	81.6	86.1	88.7	86.8	87.7	87.5
		Δ	0.1	-3.7	-2.2	-11.6	-2.4	-1.4	-2.1
	EuroSAT	Dinov2	89.2	66.9	93.9	60.2	93.8	93.8	94.2
		IN21k	90.0	88.1	94.5	95.3	95.0	95.8	94.9
		Δ	-0.8	-21.2	-0.6	-35.1	-1.2	-2.0	-0.7
	Resisc45	Dinov2	78.8	25.9	82.0	24.5	88.6	87.6	84.2
		IN21k	74.9	81.6	83.2	86.5	86.5	85.9	85.9
		Δ	3.9	-55.7	-1.2	-62.0	2.1	1.7	-1.7
	Retinopathy	Dinov2	75.3	73.6	76.0	73.6	76.0	76.0	75.5
		IN21k	74.6	73.6	74.8	75.2	76.3	75.9	75.7
		Δ	0.7	0.0	1.2	-1.6	-0.3	0.1	-0.2
Structured	Clevr-Count	Dinov2	47.5	27.3	71.2	38.1	91.2	87.8	89.8
		IN21k	37.5	56.2	80.1	82.9	82.9	82.3	82.9
		Δ	10.0	-28.9	-8.9	-44.8	8.3	5.5	6.9
	DMLab	Dinov2	44.1	30.7	51.8	39.1	51.8	53.1	54.5
		IN21k	36.5	48.2	53.0	53.8	52.8	53.8	51.8
		Δ	7.6	-17.5	-1.2	-14.7	-1.0	-0.7	2.7
	KITTI	Dinov2	60.3	47.1	81.0	46.8	83.4	82.6	83.8
		IN21k	64.6	77.9	81.4	79.6	80.0	78.1	79.9
		Δ	-4.3	-30.8	-0.4	-32.8	3.4	4.5	3.9
	dSpr-Ori	Dinov2	47.2	17.5	56.1	10.0	57.9	55.6	57.2
		IN21k	29.4	46.6	52.1	54.3	53.0	55.3	47.2
		Δ	17.8	-29.1	4.0	-44.3	4.9	0.3	10.0

Table 7. Performance comparison between DINOv2 and IN21k.

Figure 1a details. This figure illustrates the relative performance compared to linear probing (×) on VTAB-1K. The range between the highest and lowest accuracy across 14 PEFT methods is represented by ●-●, while (■) denotes the performance of full fine-tuning.



(a) Caltech101 (b) Camelyon (c) Cifar100 (d) Clevr-Count

(e) Clevr-Dist (f) DMLab (g) dSpr-Loc (h) dSpr-Ori

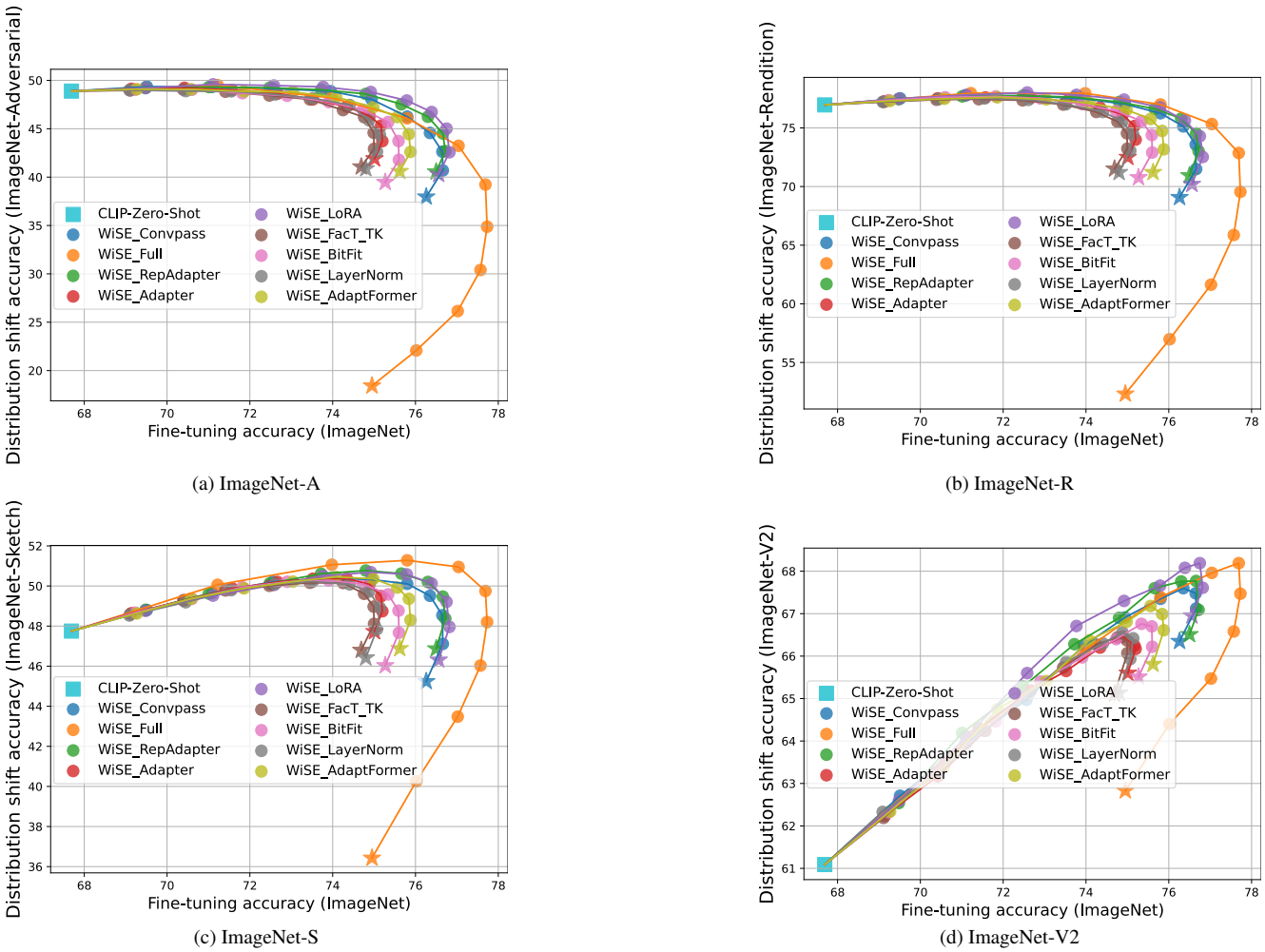
(i) dSpr-Ori (j) dSpr-Ori (k) DTD (l) EuroSAT

(m) Flower102 (n) KITTI (o) Pets (p) Resisc45

(q) Retinopathy (r) sNORB-Azim (s) sNORB-Elev (t) Sun397

(u) SVHN

Figure 13. Prediction similarity analysis on other datasets.



(a) ImageNet-A

(b) ImageNet-R

(c) ImageNet-S

(d) ImageNet-V2

Figure 14. WiSE PEFT performance on all distribution shift datasets. Target distribution vs. distribution shifts

Figure 1c details. The X-axis represents the accuracy on ImageNet-1K, while the Y-axis shows the distribution shift accuracy (averaged across ImageNet-V2, ImageNet-S, ImageNet-R, and ImageNet-A). The cyan squares (■) represent the zero-shot performance of the CLIP model, and stars ($\star$) denote the performance of fine-tuned models. Each curve corresponds to the WiSE+PEFT method, with dots • indicating different mixing coefficients α as described in section 7.

Figure 2 details. For each dataset in VTAB-1K, 15 methods (14 PEFT methods plus linear probing) are ranked by accuracy. Within each dataset group (*e.g.*, Natural), the element (i, j) in the ranking frequency matrix indicates how often method i ranks j^{th}. For instance, in the Natural group matrix, the entry (1, 3) equals 2, meaning DiffFit ranked 3rd in two datasets within this group. The row sums correspond to the total number of datasets in each group (*e.g.*, 7 datasets for the Natural group). Methods are sorted by their average rank (shown in brackets), and the parameters column indicates the number of trainable parameters in millions.

Figure 3a details. Each entry (i, j) in the prediction similarity matrix represents the percentage of test samples for which methods i and j made the same prediction. The diagonal entries are always 1, indicating perfect agreement with themselves. To compute (i, j), predictions from models fine-tuned by methods i and j are compared, with (i, j) equaling the number of matching predictions divided by the total test samples.

Figure 3b details. The Venn diagrams are generated by fine-tuning a pre-trained model on CIFAR100 (VTAB-1K) using LoRA, SSF, and Adapter methods. For Figure 3b(a), we selected the correct predictions from the top 5K most confident samples for each method and visualized the overlap

among the three methods. For Figure 3b(b), we did the same for the wrong predictions, selecting from the 5K least confident samples.

Figure 4 details. For each VTAB-1K dataset, the worst-performing PEFT method serves as the baseline (✳). Each ● represents the relative performance of other PEFT methods compared to this baseline. An ensemble prediction (▲) is generated based on the average logits of all PEFT methods for each test sample.

Figure 5 details. Different colors represent various PEFT methods. Each • along a curve (corresponding to a single PEFT method) indicates the accuracy at a specific tunable parameter size, allowing us to observe how the size of tunable parameters impacts accuracy.

Figure 6 details. Each sub-figure displays accuracy on the Y-axis, with columns representing linear probing (left), the best PEFT methods (middle), and full fine-tuning (right). Sub-figures (a) and (b) correspond to VTAB-1K (low-shot), while (c) and (d) correspond to many-shot settings. Different colors represent distinct datasets.

D. Broader Impacts

Our study provides a unifying study of PEFT in visual recognition. We expect it to serve as a valuable practical user guide to benefit society. Specifically, fine-tuning large models needs significant computation. A unifying study of PEFT will ease end-users to apply more parameter-efficient and computation-efficient ways for fine-tuning. To our knowledge, our paper does not introduce any additional negative societal impacts compared to existing papers on PEFT.