FacT: Factor-Tuning for Lightweight Adaptation on Vision Transformer

Shibo Jie, Zhi-Hong Deng*

School of Intelligence Science and Technology, Peking University
{parsley, zhdeng}@pku.edu.cn

Abstract

Recent work has explored the potential to adapt a pre-trained *vision transformer* (ViT) by updating only a few parameters so as to improve storage efficiency, called *parameter-efficient transfer learning* (PETL). Current PETL methods have shown that by tuning only 0.5% of the parameters, ViT can be adapted to downstream tasks with even better performance than full fine-tuning. In this paper, we aim to further promote the efficiency of PETL to meet the extreme storage constraint in real-world applications. To this end, we propose a tensorization-decomposition framework to store the weight increments, in which the weights of each ViT are tensorized into a single 3D tensor, and their increments are then decomposed into lightweight factors. In the fine-tuning process, only the factors need to be updated and stored, called *Factor-Tuning* (**FacT**). On VTAB-1K benchmark, our method performs on par with NOAH, the state-of-the-art PETL method, while the parameters of ours are only 19% of that of NOAH. We also present a tiny version which only uses 8K (0.01% of ViT's parameters) trainable parameters but outperforms full fine-tuning and many other PETL methods such as VPT and BitFit. In few-shot settings, **FacT** also beats all PETL baselines using the fewest parameters, demonstrating its strong capability in the low-data regime.

1 Introduction

The de-facto paradigm for achieving state-of-the-art performance on visual tasks involves pre-training on large datasets like ImageNet (Deng et al. 2009), and then fully fine-tuning on downstream tasks. However, this paradigm is not storage-efficient because each downstream task necessitates the storage of an entire model, which becomes prohibitive in some scenarios (*e.g.*, storing customized models for each user) as the size of vision models grows exponentially.

To promote storage efficiency, recent work on *parameter-efficient transfer learning* (PETL) attempts to fine-tune only a small part of the parameters to adapt the large pre-trained models to downstream tasks. These methods either insert additional trainable structures (*e.g.*, adapters (Houlsby et al. 2019) or prompt tokens (Jia et al. 2022)) to a frozen *vision transformer* (ViT) (Dosovitskiy et al. 2021) backbone or selectively fine-tune some of the ViT's own parameters (*e.g.*, all bias parameters (Zaken, Goldberg, and Ravfogel 2022)). Recently, LoRA (Hu et al. 2022) also shows that optimizing the low-rank decomposition matrices of *weight increments*[1] of the dense layers is a promising way to adapt large models.

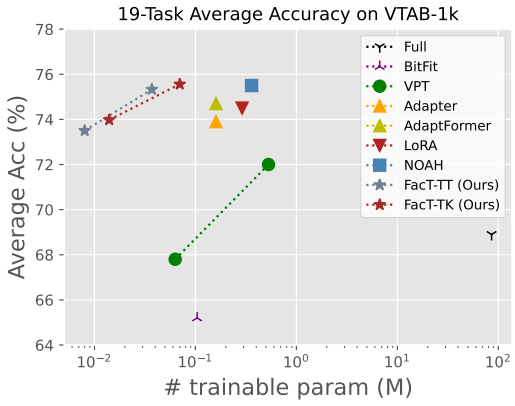

Figure 1: **Average accuracy *vs.* number of trainable parameters (log axis) on VTAB-1K benchmark.** Our **FacT** significantly reduces the number of trainable parameters.

However, though LoRA's matrix decomposition significantly reduces the storage overhead of fine-tuned dense layers, it is far from exploiting the low-rank properties of neural networks to the extreme. Inspired by recent work on compression of the transformer-based *pre-trained language models* (PLMs) (Wang et al. 2022), we infer that during fine-tuning, the weight increments of pre-trained ViT are also redundant in terms of *intra-weight rank* and *inter-weight rank*. The former is reflected in that the dense increments matrix can be low-rank as in LoRA, while the latter is implied by the fact that cross-layer weight-sharing has already been adopted in some lightweight ViT structures (Zhang et al. 2022), which is not taken into consideration by LoRA. To fully develop the potential of PETL, we propose a *tensorization-decomposition* framework. Instead of decomposing the weight increment matrices individually like LoRA, we represent the whole ViT as a single

*Corresponding Author.
Preprint

[1] We refer to the weight's change during fine-tuning as its *increment*, which is a matrix or tensor of the same shape as the weight.

tensor, in which all weight increment matrices are decomposed together. This means that the different matrices could share their factors to reduce inter-weight rank redundancy as well. The factors are then fine-tuned with the ViT backbone frozen, called *Factor-Tuning* (**FacT**). After fine-tuning, only the classification head and these factors need to be stored, and thus the storage efficiency is significantly promoted.

In this paper, we explore the feasibility of applying various tensor decomposition formats to the tensorization-decomposition framework, including *Tensor-Train* format (Oseledets 2011) (denoted as **FacT-TT**) and *Tucker* format (Lathauwer, Moor, and Vandewalle 2000) (denoted as **FacT-TK**). We also show that LoRA can be regarded as a special case of **FacT** when using *Matrix-Batch* format. As the experimental results on VTAB-1K benchmark shown in Fig 1, both **FacT-TT** and **FacT-TK** significantly reduce the number of trainable parameters while maintaining competitive performance. **FacT-TK** achieves performance on-par with that of current state-of-the-art (SOTA) method NOAH (Zhang, Zhou, and Liu 2022) with only 19% of NOAH's parameters. In a more extreme case, **FacT-TT** can adapt the ViT with 85.8M parameters by only tuning the factors with 8K parameters, while still outperforming full fine-tuning and some other PETL methods such as VPT (Jia et al. 2022) and BitFit (Zaken, Goldberg, and Ravfogel 2022). In few-shot learning, **FacT-TT** beats all other PETL baselines, demonstrating its superiority in the low-data regime.

The contributions of our work are as follows:

- Inspired by the assumption that weight increments of pre-trained ViT during fine-tuning are rank-redundant, we propose **FacT**, a tensorization-decomposition framework to adapt ViT by tuning the factors of increments.
- Under this framework, we propose **FacT-TT** and **FacT-TK**, decomposing the tensorized ViT with Tensor-Train and Tucker format, respectively.
- Experimental results show that our methods are much more lightweight than other PETL methods yet maintain competitive performances on VTAB-1K benchmark and SOTA results on few-shot learning datasets, demonstrating the potential of PETL with an extremely small storage budget.

2 Related Work

2.1 Parameter-Efficient Transfer Learning

PETL has been investigated in the field of both *natural language processing* (NLP) and *computer vision*, which aims to fine-tune a small number of trainable parameters to transfer large pre-trained models to downstream tasks. We here introduce some PETL methods either proposed for ViT or ported from PLMs.

Adapter (Houlsby et al. 2019; Mahabadi, Henderson, and Ruder 2021; Rebuffi, Bilen, and Vedaldi 2017) is typically a bottleneck block composed of two fully connected layers, whose weights are $\boldsymbol{W}_{down} \in \mathbb{R}^{d\times h}$ and $\boldsymbol{W}_{up} \in \mathbb{R}^{h\times d}$, where $h << d$. There are two common ways to insert adapters. The original one is sequential way (Houlsby et al. 2019; Pfeiffer et al. 2021), formulated as

$$\boldsymbol{X}' \leftarrow \boldsymbol{X} + \phi(\boldsymbol{X}\boldsymbol{W}_{down})\boldsymbol{W}_{up}$$

where $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ is the output of Feed-Forward Network (FFN) blocks and ϕ is a nonlinear function. The other is parallel way (He et al. 2022; Chen et al. 2022), formulated as

$$\boldsymbol{X}' \leftarrow \boldsymbol{X} + FFN(\boldsymbol{X}) + s \cdot \phi(\boldsymbol{X}\boldsymbol{W}_{down})\boldsymbol{W}_{up}$$

where s is a hyper-parameter, $\boldsymbol{X}$ is the input of FFN blocks. The parallel adapter design is also referred to as **AdaptFormer** by Chen et al. (2022). Concurrently, Jie and Deng (2022) suggest using convolutional adapters for ViT. Lian et al. (2022b) propose shifting and scaling the intermediate features of the models which can also be regarded as a simplified linear adapter.

LoRA (Hu et al. 2022) decomposes the increments of query transformation $\boldsymbol{W}_q$ and value transformation $\boldsymbol{W}_v$ into low-rank $\boldsymbol{A}_{q/v} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{B}_{q/v} \in \mathbb{R}^{r\times d}$ where $r << d$. The query and value are then computed as

$$\boldsymbol{Q/V} \leftarrow \boldsymbol{X}\boldsymbol{W}_{q/v} + s \cdot \boldsymbol{X}\boldsymbol{A}_{q/v}\boldsymbol{B}_{q/v}$$

in which s is a hyper-parameter.

VPT (Jia et al. 2022) concatenates the input $\boldsymbol{X}$ with several trainable prompts $\boldsymbol{P} \in \mathbb{R}^{l\times d}$ before feeding it into transformer layers. This extended sequence is formulated as

$$\boldsymbol{X}' \leftarrow [\boldsymbol{X}, \boldsymbol{P}]$$

In VPT-Deep, these prompts are concatenated before every layer and then discarded at the end of the layer. While in VPT-Shallow, the prompts are only inserted before the first layer and will be maintained until the last layer.

NOAH (Zhang, Zhou, and Liu 2022) focuses on combining existing PETL methods without manual design. It trains a large supernet at first and then performs neural architecture search on hidden dimension h of Adapter, rank r of LoRA, and prompt length l of VPT.

In all the aforementioned methods, the parameters of pre-trained ViT are frozen. There are also methods that only fine-tune a few of the pre-trained parameters without introducing new parameters, such as **BitFit** (Zaken, Goldberg, and Ravfogel 2022), which fine-tunes the bias parameters only.

2.2 Tensor Decomposition for Network Compression

Tensor decomposition is an important research area aiming to approximate a tensor with a set of low-rank factors that has been studied for many years. Previous work on deep learning has investigated the use of tensor decomposition to compress neural networks so as to reduce the size of models, including ConvNets (Denton et al. 2014; Lebedev et al. 2015), RNNs (Winata et al. 2019; Ye et al. 2018; Yang, Krompass, and Tresp 2017), and Transformers (Noach and Goldberg 2020; Lan et al. 2020).

Note that model compression and PETL have a similar purpose: to reduce the storage overhead. But the difference lies in that model compression aims at reducing the size of the whole model, while PETL only considers reducing the trainable parameters on a pre-trained model since the pre-trained weights are always required when fine-tuning on subsequent new tasks. Therefore, in this paper, we take inspiration from model compression but compress the increments of the weights instead of the pre-trained weights themselves.

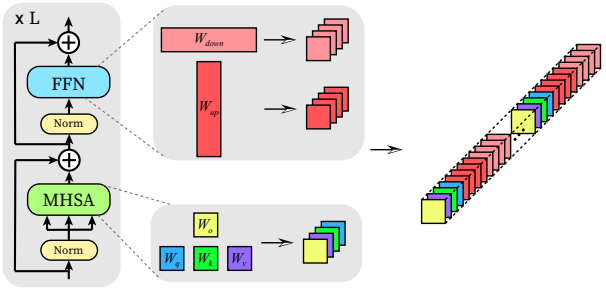

Figure 2: **Illustration of tensorizing ViT.** The ViT is tensorized into a single $12L \times d \times d$ tensor.

3 Method

3.1 Tensorizing Vision Transformer

Tensorizing a neural network means representing its parameters using a single tensor. Previous vision models such as ResNet (He et al. 2016) usually use weights of different sizes in different layers, *e.g.*, different kernel sizes and input/output channels. This property limits their capability of tensorization. However, due to the consistency of Transformer layers in ViT, we can tensorize ViT in a much simpler way.

Besides the patch embedding and classification head, a ViT is composed of two types of blocks: Multi-Head Self-Attention (MHSA) and Feed-Forward Network (FFN, also referred to as Multi-Layer Perceptron). In MHSA, the query, key, value, and output transformations are parametrized by $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v, \boldsymbol{W}_o \in \mathbb{R}^{d\times d}$, respectively. These transformations are further divided into N_h heads: $\{\boldsymbol{W}_q^{(i)}\}_{i=1}^{N_h}, \{\boldsymbol{W}_k^{(i)}\}_{i=1}^{N_h}, \{\boldsymbol{W}_v^{(i)}\}_{i=1}^{N_h}, \{\boldsymbol{W}_o^{(i)}\}_{i=1}^{N_h}$. Then, the MHSA is formulated as[2]

$$MHSA(\boldsymbol{X}) = \sum_{i=1}^{N_h} softmax\left(\frac{\boldsymbol{X}\boldsymbol{W}_q^{(i)}\boldsymbol{W}_k^{(i)\intercal}\boldsymbol{X}^\intercal}{\sqrt{d}}\right)\boldsymbol{X}\boldsymbol{W}_v^{(i)}\boldsymbol{W}_o^{(i)\intercal} \quad (1)$$

An FFN block consists of two fully-connected (FC) layers. Ignoring the bias parameters for simplicity, the FFN is formulated as

$$FFN(\boldsymbol{X}) = GELU(\boldsymbol{X}\boldsymbol{W}_{up})\boldsymbol{W}_{down} \quad (2)$$

where $\boldsymbol{W}_{up} \in \mathbb{R}^{d\times 4d}$ and $\boldsymbol{W}_{down} \in \mathbb{R}^{4d\times d}$ are the weights of the FC layers.

The FFN can also be regarded as a multi-head block. We divide $\boldsymbol{W}_{up}$ and $\boldsymbol{W}_{down}$ into four matrices of size $d\times d$, *i.e.*, $\{\boldsymbol{W}_{up}^{(i)}\}_{i=1}^4$ and $\{\boldsymbol{W}_{down}^{(i)}\}_{i=1}^4$, respectively. The FFN can be rewritten as

$$FFN(\boldsymbol{X}) = \sum_{i=1}^{4} GELU\left(\boldsymbol{X}\boldsymbol{W}_{up}^{(i)}\right)\boldsymbol{W}_{down}^{(i)} \quad (3)$$

In each layer, there are four $d \times d$ matrices in the MHSA block and eight $d \times d$ matrices in the FFN block. Supposing the number of layers in a ViT is L, we can stack all weights of the Transformer layers together as a single $12L \times d \times d$ tensor[3]

$$\boldsymbol{\mathcal{W}} = \Big\{\{\boldsymbol{W}_q^j, \boldsymbol{W}_k^j, \boldsymbol{W}_v^j, \boldsymbol{W}_o^j\} \cup \{\boldsymbol{W}_{up}^{j,(i)}\}_{i=1}^4 \cup \{\boldsymbol{W}_{down}^{j,(i)}\}_{i=1}^4\Big\}_{j=1}^{L} \in \mathbb{R}^{12L\times d\times d} \quad (4)$$

as shown in Fig 2.

Note that the classification head, patch embedding, normalization, and all bias parameters are not taken into account in this tensorized format since they are scarce and irregular. For simplicity, we suppose the classification head is not tensorized, and others (patch embedding, normalization, and bias parameters) are frozen during fine-tuning.

3.2 Factor-Tuning: a Unified Perspective

Let $\boldsymbol{\mathcal{W}}_0$ denote a tensorized pre-trained ViT. During fine-tuning, the ViT is updated to $\boldsymbol{\mathcal{W}}_{ft}$, and we use $\Delta\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{W}}_{ft} - \boldsymbol{\mathcal{W}}_0$ to denote the increment of the ViT weight tensor. When fine-tuning, the gradient is calculated as[4]

$$g_{\boldsymbol{\mathcal{W}}} = \frac{\partial\mathcal{L}(\mathcal{D};\boldsymbol{\mathcal{W}})}{\partial\boldsymbol{\mathcal{W}}} \quad (5)$$

where $\mathcal{D}$ is training dataset and $\boldsymbol{\mathcal{W}}$ is initialized as $\boldsymbol{\mathcal{W}}_0$. We can rewrite this equation in another equivalent form

$$g_{\boldsymbol{\mathcal{W}}} = g_{\Delta\boldsymbol{\mathcal{W}}} = \frac{\partial\mathcal{L}(\mathcal{D};\boldsymbol{\mathcal{W}}_0 + \Delta\boldsymbol{\mathcal{W}})}{\partial\Delta\boldsymbol{\mathcal{W}}} \quad (6)$$

where $\Delta\boldsymbol{\mathcal{W}}$ is initialized as a zero tensor.

Traditional fine-tuning updates all parameters in a ViT, which means that we need to store at least a dense $\Delta\boldsymbol{\mathcal{W}}$ (for Eq (6)) or $\boldsymbol{\mathcal{W}}_{ft}$ (for Eq (5)) for each downstream task, resulting in a storage overhead of $\mathcal{O}(Ld^2)$ per task.

In the two modules of ViT, all the weight matrices multiply the inputs in a fully-connected way, implying the existence of redundancy. In the field of NLP, many studies have already found that the weight matrices in the transformer-based PLMs are redundant in rank (Noach and Goldberg 2020; Ma et al. 2019; Lan et al. 2020; Wang et al. 2022). The rank redundancies include *intra-weight* redundancy, when each of the dense weight matrices can be decomposed (*e.g.*, SVD) into and approximated by low-rank factors; and *inter-weight* redundancy, when the model could work well with cross-layer shared weights (*e.g.*, ALBERT (Lan et al. 2020)). Since the blocks of ViT and of PLMs are highly similar, we infer that the weights of pretrained ViT could be redundant in rank as well, suggesting that the rank of weight increments $\Delta\boldsymbol{\mathcal{W}}$ could also be redundant.

Due to the redundancies of $\Delta\boldsymbol{\mathcal{W}}$, we can decompose $\Delta\boldsymbol{\mathcal{W}}$ into some factors to promote storage efficiency. We consider several well-known formats to decompose $\Delta\boldsymbol{\mathcal{W}} \in \mathbb{R}^{12L\times d\times d}$: Matrix-Batch format, Tensor-Train format (Oseledets 2011), and Tucker format (Lathauwer, Moor, and Vandewalle 2000), as illustrated in Fig 3.

[2] We use superscript (i) to denote the i-th head.

[3] We use superscript j to represent that the matrix is in j-th layer.

[4] Tensor flattening notations are omitted for simplicity.

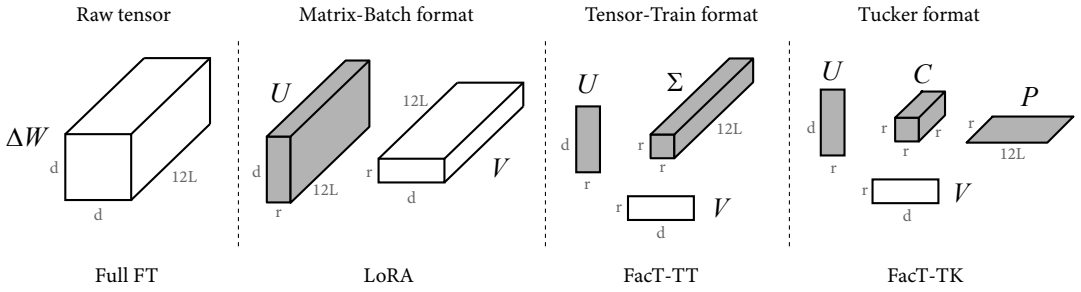

Figure 3: **Factor-Tuning with different tensor decomposition methods.** Full fine-tuning optimizes the raw tensor. LoRA can be regarded as optimizing factors of Matrix-Batch format. **FacT-TT** and **FacT-TK** optimize Tensor-Train and Tucker factors, respectively. Grey and white tensors/matrices indicate random and zero initialization, respectively.

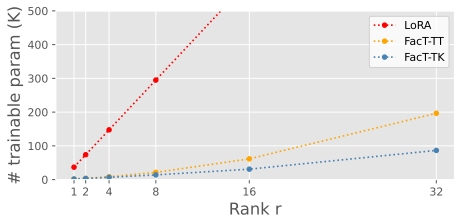

Figure 4: **Number of trainable parameters *vs.* rank r.** With the same rank, **FacT-TK** uses fewer parameters than **FacT-TT**, and LoRA is significantly larger than **FacT-TT** and **FacT-TK**.

Interpreting LoRA in our framework: Matrix-Batch Matrix-Batch format regards the first dimension of $\Delta\mathcal{W}$ as the batch dimension and decomposes all the matrices of $d\times d$ in the batch individually. Formally, $\Delta\mathcal{W}$ is decomposed into $\boldsymbol{U}\in\mathbb{R}^{12L\times d\times r}$ and $\boldsymbol{V}\in\mathbb{R}^{12L\times r\times d}$, where

$$\Delta\mathcal{W}_{i,:,:}=s\cdot\boldsymbol{U}_{i,:,:}\boldsymbol{V}_{i,:,:}\qquad\forall i\in\{1,2,...,12L\}\tag{7}$$

in which s is a hyper-parameter for scaling. If we only consider $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$ in tensorization, *i.e.*, let $\mathcal{W}=\left\{\boldsymbol{W}_q^j,\boldsymbol{W}_v^j\right\}_{j=1}^{L}\in\mathbb{R}^{2L\times d\times d}$ instead, **FacT** with Matrix-Batch format will become exactly LoRA. The size of LoRA's factors is $4Ldr\sim\mathcal{O}(Ldr)$, where $r<<d$.

However, since the weight matrices are discomposed individually in Matrix-Batch format, LoRA only reduces the intra-weight redundancy. We now introduce two other formats that take inter-weight redundancy into consideration.

Proposed Format I: Tensor-Train $\Delta\mathcal{W}$ is decomposed into $\boldsymbol{U}\in\mathbb{R}^{d\times r_1}$, $\boldsymbol{V}\in\mathbb{R}^{d\times r_2}$, and $\boldsymbol{\Sigma}\in\mathbb{R}^{12L\times r_1\times r_2}$, where

$$\Delta\mathcal{W}=s\cdot\boldsymbol{\Sigma}\times_2\boldsymbol{U}^\intercal\times_3\boldsymbol{V}^\intercal\tag{8}$$

in which $\times_i$ is mode-i product, *i.e.*,

$$\Delta\mathcal{W}_{i,j,k}=s\cdot\sum_{t_1=1}^{r_1}\sum_{t_2=1}^{r_2}\boldsymbol{\Sigma}_{i,t_1,t_2}\boldsymbol{U}_{j,t_1}\boldsymbol{V}_{k,t_2}\qquad\forall i\in\{1,2,...,12L\},\forall j,k\in\{1,2,...,d\}\tag{9}$$

For simplicity, we set $r=r_1=r_2<<d$. The size of factors is $2dr+12Lr^2\sim\mathcal{O}(dr+Lr^2)$.

Proposed Format II: Tucker $\Delta\mathcal{W}$ is decomposed into $\boldsymbol{U}\in\mathbb{R}^{d\times r_2}$, $\boldsymbol{V}\in\mathbb{R}^{d\times r_3}$, $\boldsymbol{P}\in\mathbb{R}^{12L\times r_1}$, and $\boldsymbol{C}\in\mathbb{R}^{r_1\times r_2\times r_3}$, where

$$\Delta\mathcal{W}=s\cdot\boldsymbol{C}\times_1\boldsymbol{P}^\intercal\times_2\boldsymbol{U}^\intercal\times_3\boldsymbol{V}^\intercal\tag{10}$$

i.e.,

$$\Delta\mathcal{W}_{i,j,k}=s\cdot\sum_{t_1=1}^{r_1}\sum_{t_2=1}^{r_2}\sum_{t_3=1}^{r_3}\boldsymbol{C}_{t_1,t_2,t_3}\boldsymbol{P}_{i,t_1}\boldsymbol{U}_{j,t_2}\boldsymbol{V}_{k,t_3}\qquad\forall i\in\{1,2,...,12L\},\forall j,k\in\{1,2,...,d\}\tag{11}$$

For simplicity, we set $r=r_1=r_2=r_3<<d$. The size of factors is $2dr+12Lr+r^3\sim\mathcal{O}(dr+Lr+r^3)$.

Inspired by Hu et al. (2022), we adopt a decompose-then-train paradigm, *i.e.*, decompose the $\Delta\mathcal{W}$ before fine-tuning, and then update the factors end-to-end during fine-tuning. This paradigm benefits fine-tuning process in several ways. First, since $\Delta\mathcal{W}$ is initially a zero tensor, we can use a predefined rule to initialize the factors directly instead of running the expensive decomposition algorithm. Second, decomposing a tensor means an uncontrollable loss of information. Because the factors are optimized via gradient descent, we can expect the most useful information to be retained when minimizing the training loss.

In each of the two formats, the factor $\boldsymbol{V}$ is zero-initialized and other factors are randomly initialized, so that the $\Delta\mathcal{W}$ is initially a zero tensor. After decomposition, we fine-tune the factors end-to-end. Taking the Tensor-Train format as an example, the gradient w.r.t. $\boldsymbol{U}$ is calculated as

$$g_{\boldsymbol{U}}=\frac{\partial\mathcal{L}(\mathcal{D};\mathcal{W}_0+\Delta\mathcal{W})}{\partial\boldsymbol{U}}=s\cdot g_{\mathcal{W}}\frac{\partial(\boldsymbol{\Sigma}\times_2\boldsymbol{U}^\intercal\times_3\boldsymbol{V}^\intercal)}{\partial\boldsymbol{U}}\tag{12}$$

and the same for $\boldsymbol{V}$ and $\boldsymbol{\Sigma}$. Note that the role of hyper-parameter s in Eq (12) is to adjust the learning rate of factors.

After fine-tuning, we only need to store the lightweight factors for each task. We use **FacT-TT** and **FacT-TK** to denote **FacT** using Tensor-Train and Tucker formats, respectively. **FacT-TT** and **FacT-TK** reduce both intra- and inter-weight redundancies, so they can use fewer parameters to store task-specific information and are more storage-efficient, as shown in Fig 4. The factors can be absorbed into $\mathcal{W}_{ft}$ before inference, so **FacT** adds no extra computational cost or latency during the inference phase.

	# param (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	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
Linear	0	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																					
BitFit	0.103	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-Shallow	**0.063**	77.7	86.9	62.6	97.5	87.3	74.5	51.2	78.2	92.0	75.6	72.9	50.5	58.6	40.5	67.1	68.7	36.1	20.2	34.1	67.8
VPT-Deep	0.531	**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
Adapter	0.157	69.2	90.1	68.0	98.8	89.9	82.8	54.3	84.0	94.9	81.9	75.5	80.9	65.3	48.6	78.3	74.8	48.5	29.9	41.6	73.9
AdaptFormer	0.157	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
LoRA	0.295	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
NOAH	0.361	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-TT$_4$	**0.008**	69.4	88.5	70.6	98.8	90.0	83.3	53.7	83.9	95.1	81.5	75.4	78.2	**69.0**	47.7	79.0	75.2	42.7	27.2	38.7	73.5
FacT-TT$_{\leq 16}$	0.037	**71.3**	89.6	70.7	98.9	**91.0**	87.8	**54.6**	**85.2**	95.5	83.4	**75.7**	82.0	**69.0**	**49.8**	80.0	79.2	**48.4**	**34.2**	41.4	75.3
FacT-TK$_8$	0.014	70.3	88.7	69.8	99.0	90.4	84.2	53.5	82.8	95.6	82.8	**75.7**	81.1	68.0	48.0	80.5	74.6	44.0	29.2	41.1	74.0
FacT-TK$_{\leq 32}$	0.069	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**

Table 1: **Full results on the VTAB-1K benchmark**. "# params" specifies the number of trainable parameters in backbones. Both average accuracy and # params are averaged over group-wise average values. Our **FacT-TK**$_{\leq 32}$ outperforms all previous PETL methods while using significantly fewer parameters.

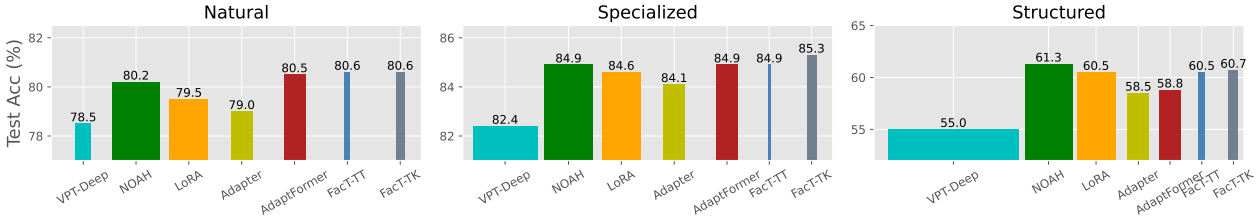

Figure 5: **Group-wise results on VTAB-1K.** The width of the bars is proportional to the number of trainable parameters.

4 Experiments

4.1 Transfer Learning on VTAB-1K Benchmark

First of all, we evaluate our method on the basic transfer learning scenario – fine-tuning the pre-trained models on various downstream tasks.

Datasets We use VTAB-1K benchmark (Zhai et al. 2019) to evaluate the performance of our methods in terms of PETL. VTAB-1K consists of 19 different visual classification datasets, which can be divided into three groups: **Natural**, **Specialized**, and **Structured**. Each dataset only contains 1,000 training samples. We report top-1 accuracy on test set in all experiments. These datasets cover a large range of the possible domains where downstream tasks come from, and thus the effectiveness of PETL methods can be measured comprehensively.

Compared Methods We compare our methods to various competitive baselines, including **BitFit** (Zaken, Goldberg, and Ravfogel 2022), **VPT-Shallow** (Jia et al. 2022), **VPT-Deep** (Jia et al. 2022), **Adapter** (Pfeiffer et al. 2021; Mahabadi, Henderson, and Ruder 2021), **AdaptFormer** (Chen et al. 2022), **LoRA** (Hu et al. 2022), and current SOTA method **NOAH** (Zhang, Zhou, and Liu 2022). Following Zhang, Zhou, and Liu (2022), the hidden dimension h of Adapter and AdaptFormer, as well as the rank r of LoRA, are all set to 8. The prompt length l of VPT follows the recipes in the original paper. We also report the results of two traditional transfer learning methods: **Full** fine-tuning, which updates all parameters on downstream tasks, and **Linear** probing, which learns a linear classification head on the pre-trained backbone.

For our methods, we report four settings: **FacT-TT**$_4$ with $r = 4$; **FacT-TT**$_{\leq 16}$ with r searched from $\{1, 2, 4, 8, 16\}$ for each tasks; **FacT-TK**$_8$ with $r = 8$; and **FacT-TK**$_{\leq 32}$ with r searched from $\{2, 4, 8, 16, 32\}$. Following Zhang, Zhou, and Liu (2022), we use AdamW optimizer with a learning rate of 1e-3 and batch size of 64 to train for 100 epochs. The hyper-parameter s is roughly swept from $\{0.01, 0.1, 1, 10, 100\}$.

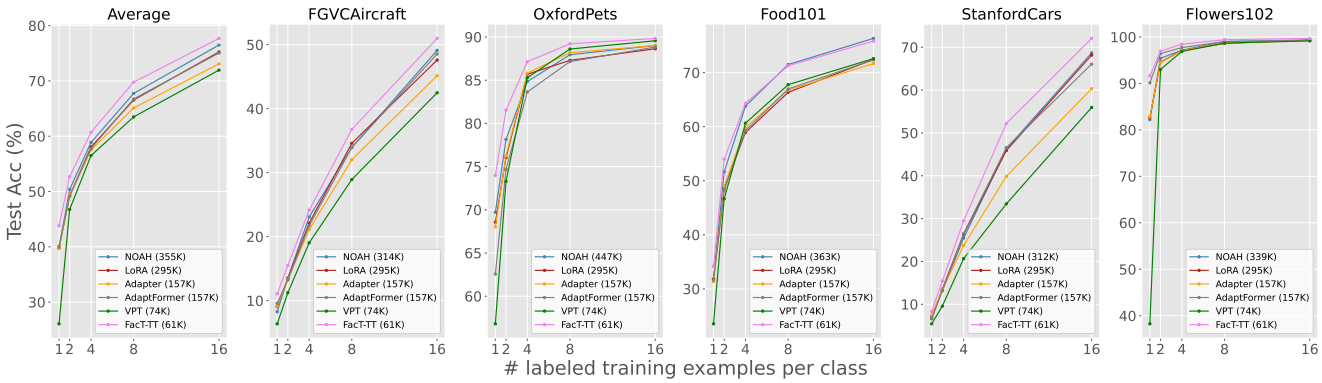

Figure 6: **Top-1 accuracy on fine-grained few-shot datasets.** The average numbers of trainable parameters in backbones are shown in parentheses. Our **FacT-TT** outperforms other baselines using the fewest trainable parameters.

Pre-trained Backbone For all methods, we use a ViT-B/16 (Dosovitskiy et al. 2021) pre-trained on supervised ImageNet-21K (Deng et al. 2009) as backbone.

Results Experimental results are shown in Table 1, from which we can see that:

(1) **FacT-TT** and **FacT-TK** have competitive results with respect to previous SOTA PETL methods while using much fewer trainable parameters. **FacT-TT**$_{\leq 16}$ and **FacT-TK**$_{\leq 32}$ introduce only 37K and 70K trainable parameters, respectively. However, they outperform NOAH, the previous SOTA methods with 361K trainable parameters, on 11 out of 19 tasks. Moreover, **FacT-TT**$_{\leq 16}$ and **FacT-TK**$_{\leq 32}$ also achieve new SOTA results on 7 out of 19 tasks. It's worth noting that NOAH trains an additional large supernet for 500 epochs used for architecture search, and thus **FacT** is also superior to NOAH in terms of training efficiency.

(2) For our two methods, neither **FacT-TT** nor **FacT-TK** shows an advantage over the other one. Note that Tucker format can be regarded as further decomposing the $\boldsymbol{\Sigma}$ in Tensor-Train format into $\boldsymbol{P}$ and $\boldsymbol{C}$. Although Tucker format has higher compression ratio, **FacT-TK** does not clearly outperforms **FacT-TT**, implying that **FacT-TT** is sufficiently compact for adaptation and thus further compression does not lead to obvious improvement.

(3) Though sub-optimal to many baselines in terms of absolute performance, **FacT-TT**$_4$ and **FacT-TK**$_8$ still provide non-trivial improvement under extreme storage constraint. **FacT-TT**$_4$ and **FacT-TK**$_8$ use only 8K and 14K parameters (0.01% and 0.02% of the 85.8M ViT-B) to adapt the ViT backbones. In contrast, the parameters of the classification head will be even more than 8K as long as the task contains more than 10 classes. In other words, **FacT-TT**$_4$ and **FacT-TK**$_8$ use trainable parameters of the same magnitude as linear probing, while achieving performance better than full fine-tuning and VPT.

In Fig 5, we also report the group-wise results on VTAB-1K. **FacT** achieves SOTA results on Natural and Specialized, but underperforms NOAH on Structured. We still emphasize that our methods are much more lightweight, and thus more efficient than other baselines.

4.2 Fine-Grained Few-Shot Learning

Few-shot learning is a common scenario when the data of downstream tasks are hard to obtain, and there are only few training samples for each task that can be utilized.

Datasets To evaluate the capability of our method in the low-data regime, we conduct experiments on five fine-grained datasets in few-shot setting. The five datasets include: **FGVC-Aircraft** (Maji et al. 2013a), **Oxford-Pets** (Parkhi et al. 2012), **Food-101** (Bossard, Guillaumin, and Gool 2014), **Stanford Cars** (Krause et al. 2013), and **Oxford-Flowers102** (Nilsback and Zisserman 2006), which contains fine-grained classes from five categories: aircraft, pets, food, cars, and flowers. Following previous work (Zhang, Zhou, and Liu 2022), we evaluate on {1, 2, 4, 8, 16}-shot settings.

Compared Methods We compare our method with five baselines that perform the best on VTAB-1K: VPT-Deep, Adapter, AdaptFormer, LoRA, and NOAH. The hyper-parameter h, r, and l of the baselines are all set to 8. As for our method, we report the results of **FacT-TT**$_{16}$, *i.e.*, **FacT-TT** with a fixed $r = 16$. Other settings are the same as on VTAB-1K. All results are averaged over three runs with different random seeds.

Results As the results shown in Fig 6, we can find that:

(1) Though using the fewest trainable parameters, **FacT-TT** still achieves SOTA results on average. Note that all these datasets can be categorized into the Natural group, so this observation is in line with what we have found on VTAB-1K that **FacT-TT** is better at Natural tasks.

(2) **FacT-TT** performs the best across all settings on four out of five datasets except for Food-101, where **FacT-TT** slightly underperforms NOAH in 8-shot and 16-shot settings.

These observations confirm the capability and efficiency of our method in low-data regime. Again, it verifies the effectiveness of reducing intra- and inter-rank redundancies of weight increments for PETL.

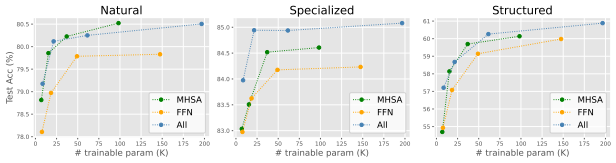

Figure 7: **Results on VTAB-1K across different rank r and tensorization strategies.** All: the original tensorization method that tensorizes both MHSA and FFN; MHSA/FFN: tensorize MHSA/FFN blocks only. We report results with $r \in \{4, 8, 16, 32\}$ for each setting.

4.3 FacT for Hierarchical Transformers

After the original ViT was proposed, it was found that ViT lacks visual inductive bias which limits its performance, especially on the dense prediction tasks. Subsequently, a series of studies improved ViT by introducing hierarchical structures (Liu et al. 2021; Wang et al. 2021; Wu et al. 2021), among which Swin Transformer (Liu et al. 2021) is a widely used and representative design. Therefore, we also extend **FacT** to Swin Transformer.

A challenge when applying tensorization to such hierarchical structures is that the hidden dimension d is different across layers. However, these models usually partition the layers into several stages, where the hidden dimension is consistent within each stage. Therefore, we propose a *partitioned tensorization* strategy for these models, which individually tensorizes each stage into a single tensor.

Taking **Swin-B** as an instance, its four stages consist $\{2, 2, 18, 2\}$ layers with hidden dimensions of $\{128, 256, 512, 1024\}$, respectively. Therefore, we can tensorize them into four tensors of the size $\{24\times128\times128, 24\times256\times256, 216\times512\times512, 24\times1024\times1024\}$ following the steps described in Section 3.1. These tensors are then decomposed individually.

We report the results of **FacT-TT**$_{16}$ on VTAB-1K in Table 2, using Swin-B pre-trained on supervised ImageNet-21K as backbone. We compare our method with baselines that can also be employed on Swin: Full, Linear, BitFit, and VPT. We can see that our **FacT-TT** outperforms other PETL methods by a large margin. Although partitioned tensorization weakens the efficiency of **FacT** to some extent since we have to store a set of factors for each stage, **FacT-TT** still uses fewer parameters than VPT-Deep and BitFit. These results demonstrate that **FacT** can also be applied to Swin Transformer while keeping its advantages. Since **FacT** is an architecture-agnostic framework, it can be extended to various models (*e.g.*, ConvNets (Liu et al. 2022) and MLPs (Tolstikhin et al. 2021; Lian et al. 2022a)) and tasks (*e.g.*, NLP and multimodal tasks), as long as the models can be tensorized in an appropriate way.

4.4 Ablation Analyses

We ablate the tensorization-decomposition framework on VTAB-1K benchmark to show the effect of tensorization strategy and decomposition rank.

The default tensorization method tensorizes both MHSA and FFN blocks, resulting in a $12L \times d \times d$ tensor. We here consider two ablated strategies: tensorizing only MHSA or FFN blocks, *i.e.*,

$$\mathcal{W} = \left\{\boldsymbol{W}_q^j, \boldsymbol{W}_k^j, \boldsymbol{W}_v^j, \boldsymbol{W}_o^j\right\}_{j=1}^{L} \in \mathbb{R}^{4L\times d\times d} \tag{13}$$

or

$$\mathcal{W} = \left\{\{\boldsymbol{W}_{up}^{j,(i)}\}_{i=1}^{4} \cup \{\boldsymbol{W}_{down}^{j,(i)}\}_{i=1}^{4}\right\}_{j=1}^{L} \in \mathbb{R}^{8L\times d\times d} \tag{14}$$

respectively. The blocks that are not tensorized keep frozen during fine-tuning. As for the decomposition part, we use Tensor-Train format with rank $r \in \{4, 8, 16, 32\}$. The results are shown in Fig 7.

On all three groups, tensorizing MHSA is better than tensorizing FFN, suggesting that MHSA blocks play a more important role than FFN in downstream transfer tasks. Tensorizing all blocks is better than the other two strategies on Specialized and Structured. However, we find that only tensorizing MHSA is more efficient than tensorizing all on Natural when $r \geq 16$, indicating that the potential of **FacT** can be further developed by searching the tensorization strategy for different datasets in VTAB-1K.

Besides, we also find that as rank r increases, the performance is always improved across the three tensorization settings. Since the size of factor $\boldsymbol{\Sigma}$ increases with the square of r, **FacT-TT** is no longer efficient when r becomes too large. But this limitation will not have a significant negative impact since the performance is saturated when $r \geq 16$ and thus a large r is not necessary.

Method	# param (M)	Avg.	Nat.	Spe.	Str.
Full	86.7	75.0	79.2	86.2	59.7
Linear	0	62.6	73.5	80.8	33.5
BitFit	0.201	65.6	74.2	80.1	42.4
VPT-Shallow	0.003	66.7	79.9	82.5	37.8
VPT-Deep	0.162	71.6	76.8	84.5	53.4
FacT-TT$_{16}$	0.135	**77.4**	**83.1**	**86.9**	**62.1**

Table 2: **Results on VTAB-1K with Swin-B as backbone.** Avg./Nat./Spe./Str.: Average/Natural/Specialized/Structured results. # params: # of trainable parameters in backbones.

5 Conclusion

In this paper, we propose Factor-Tuning, a tensorization-decomposition framework for PETL on ViT. Under this framework, we present an approach to making ViT tensorized, and employ two tensor decomposition methods to factorize its increment. By updating and storing the factors only, our methods reduce both intra- and inter-weight redundancy of weight increment and thus are much more efficient. **FacT** achieves competitive results on VTAB-1K benchmark with a significantly reduced number of parameters, and outperforms all PETL baselines on few-shot learning. Our work demonstrates that the storage efficiency of PETL has not been fully exploited yet, and **FacT** provides a promising framework for future work.

References

Beattie, C.; Leibo, J. Z.; Teplyashin, D.; Ward, T.; Wainwright, M.; Küttler, H.; Lefrancq, A.; Green, S.; Valdés, V.; Sadik, A.; et al. 2016. Deepmind lab. *arXiv preprint*, abs/1612.03801.

Bossard, L.; Guillaumin, M.; and Gool, L. V. 2014. Food-101–mining discriminative components with random forests. In *ECCV*.

Chen, S.; Ge, C.; Tong, Z.; Wang, J.; Song, Y.; Wang, J.; and Luo, P. 2022. AdaptFormer: Adapting Vision Transformers for Scalable Visual Recognition. In *NeurIPS*.

Cheng, G.; Han, J.; and Lu, X. 2017. Remote Sensing Image Scene Classification: Benchmark and State of the Art. *Proc. IEEE*.

Cimpoi, M.; Maji, S.; Kokkinos, I.; Mohamed, S.; ; and Vedaldi, A. 2014. Describing Textures in the Wild. In *CVPR*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR*.

Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In *NIPS*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.

Fei-Fei, L.; Fergus, R.; and Perona, P. 2004. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR workshops*.

Geiger, A.; Lenz, P.; Stiller, C.; and Urtasun, R. 2013. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*.

He, J.; Zhou, C.; Ma, X.; Berg-Kirkpatrick, T.; and Neubig, G. 2022. Towards a Unified View of Parameter-Efficient Transfer Learning. In *ICLR*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.

Helber, P.; Bischke, B.; Dengel, A.; and Borth, D. 2019. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*.

Higgins, I.; Matthey, L.; Pal, A.; Burgess, C. P.; Glorot, X.; Botvinick, M. M.; Mohamed, S.; and Lerchner, A. 2017. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *ICLR*.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; de Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-Efficient Transfer Learning for NLP. In *ICML*.

Hu, E. J.; yelong shen; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR*.

Jia, M.; Tang, L.; Chen, B.; Cardie, C.; Belongie, S. J.; Hariharan, B.; and Lim, S. 2022. Visual Prompt Tuning. In *ECCV*.

Jie, S.; and Deng, Z. 2022. Convolutional Bypasses Are Better Vision Transformer Adapters. *arXiv preprint*, abs/2207.07039.

Johnson, J.; Hariharan, B.; Van Der Maaten, L.; Fei-Fei, L.; Lawrence Zitnick, C.; and Girshick, R. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*.

Kaggle; and EyePacs. 2015. Kaggle diabetic retinopathy detection.

Krause, J.; Stark, M.; Deng, J.; and Fei-Fei, L. 2013. 3d object representations for fine-grained categorization. In *CVPR workshops*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *ICLR*.

Lathauwer, L. D.; Moor, B. D.; and Vandewalle, J. 2000. A Multilinear Singular Value Decomposition. *SIAM J. Matrix Anal. Appl.*

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I. V.; and Lempitsky, V. S. 2015. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. In *ICLR*.

LeCun, Y.; Huang, F. J.; and Bottou, L. 2004. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*.

Lian, D.; Yu, Z.; Sun, X.; and Gao, S. 2022a. AS-MLP: An Axial Shifted MLP Architecture for Vision. In *ICLR*.

Lian, D.; Zhou, D.; Feng, J.; and Wang, X. 2022b. Scaling & Shifting Your Features: A New Baseline for Efficient Model Tuning. In *NeurIPS*.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *ICCV*.

Liu, Z.; Mao, H.; Wu, C.; Feichtenhofer, C.; Darrell, T.; and Xie, S. 2022. A ConvNet for the 2020s. In *CVPR*.

Ma, X.; Zhang, P.; Zhang, S.; Duan, N.; Hou, Y.; Zhou, M.; and Song, D. 2019. A Tensorized Transformer for Language Modeling. In *NeurIPS*.

Mahabadi, R. K.; Henderson, J.; and Ruder, S. 2021. Compacter: Efficient Low-Rank Hypercomplex Adapter Layers. In *NeurIPS*.

Maji, S.; Rahtu, E.; Kannala, J.; Blaschko, M.; and Vedaldi, A. 2013a. Fine-grained visual classification of aircraft. *arXiv preprint*, abs/1306.5151.

Maji, S.; Rahtu, E.; Kannala, J.; Blaschko, M. B.; and Vedaldi, A. 2013b. Fine-Grained Visual Classification of Aircraft. *arXiv preprint*, abs/1306.5151.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshops*.

Nilsback, M.-E.; and Zisserman, A. 2006. A visual vocabulary for flower classification. In *CVPR*.

Noach, M. B.; and Goldberg, Y. 2020. Compressing Pre-trained Language Models by Matrix Decomposition. In *AACL/IJCNLP*.

Oseledets, I. V. 2011. Tensor-Train Decomposition. *SIAM J. Sci. Comput.*

Parkhi, O. M.; Vedaldi, A.; Zisserman, A.; and Jawahar, C. 2012. Cats and dogs. In *CVPR*.

Pfeiffer, J.; Kamath, A.; Rücklé, A.; Cho, K.; and Gurevych, I. 2021. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. In *EACL*.

Rebuffi, S.-A.; Bilen, H.; and Vedaldi, A. 2017. Learning multiple visual domains with residual adapters. In *NIPS*.

Tolstikhin, I. O.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; Lucic, M.; and Dosovitskiy, A. 2021. MLP-Mixer: An all-MLP Architecture for Vision. In *NeurIPS*.

Veeling, B. S.; Linmans, J.; Winkens, J.; Cohen, T.; and Welling, M. 2018. Rotation Equivariant CNNs for Digital Pathology. *arXiv preprint*, abs/1806.03962.

Wang, B.; Ren, Y.; Shang, L.; Jiang, X.; and Liu, Q. 2022. Exploring extreme parameter compression for pre-trained language models. In *ICLR*.

Wang, W.; Xie, E.; Li, X.; Fan, D.; Song, K.; Liang, D.; Lu, T.; Luo, P.; and Shao, L. 2021. Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. In *ICCV*.

Winata, G. I.; Madotto, A.; Shin, J.; Barezi, E. J.; and Fung, P. 2019. On the Effectiveness of Low-Rank Matrix Factorization for LSTM Model Compression. *arXiv preprint*, abs/1908.09982.

Wu, H.; Xiao, B.; Codella, N.; Liu, M.; Dai, X.; Yuan, L.; and Zhang, L. 2021. CvT: Introducing Convolutions to Vision Transformers. In *ICCV*.

Xiao, J.; Hays, J.; Ehinger, K. A.; Oliva, A.; and Torralba, A. 2010. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*.

Yang, Y.; Krompass, D.; and Tresp, V. 2017. Tensor-Train Recurrent Neural Networks for Video Classification. In *ICML*.

Ye, J.; Wang, L.; Li, G.; Chen, D.; Zhe, S.; Chu, X.; and Xu, Z. 2018. Learning Compact Recurrent Neural Networks With Block-Term Tensor Decomposition. In *CVPR*.

Zaken, E. B.; Goldberg, Y.; and Ravfogel, S. 2022. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. In *ACL*.

Zhai, X.; Puigcerver, J.; Kolesnikov, A.; Ruyssen, P.; Riquelme, C.; Lucic, M.; Djolonga, J.; Pinto, A. S.; Neumann, M.; Dosovitskiy, A.; Beyer, L.; Bachem, O.; Tschannen, M.; Michalski, M.; Bousquet, O.; Gelly, S.; and Houlsby, N. 2019. The Visual Task Adaptation Benchmark. *arXiv preprint*, abs/1910.04867.

Zhang, J.; Peng, H.; Wu, K.; Liu, M.; Xiao, B.; Fu, J.; and Yuan, L. 2022. MiniViT: Compressing Vision Transformers With Weight Multiplexing. In *CVPR*.

Zhang, Y.; Zhou, K.; and Liu, Z. 2022. Neural Prompt Search. *arXiv preprint*, abs/2206.04673.

Appendix

A Datasets

See Table 3. The 1,000 training samples of VTAB-1K are further divided into a training set (800 samples) and a validation set (200 samples) in hyper-parameter tuning. The final results are produced by training on all 1,000 training samples and testing on the test set.

	Dataset	# Classes	Train	Val	Test
	VTAB-1K (Zhai et al. 2019)				
Natural	CIFAR100 (Krizhevsky, Hinton et al. 2009)	100	800/1,000	200	10,000
	Caltech101 (Fei-Fei, Fergus, and Perona 2004)	102			6,084
	DTD (Cimpoi et al. 2014)	47			1,880
	Oxford-Flowers102 (Nilsback and Zisserman 2006)	102			6,149
	Oxford-Pets (Parkhi et al. 2012)	37			3,669
	SVHN (Netzer et al. 2011)	10			26,032
	Sun397 (Xiao et al. 2010)	397			21,750
Specialized	Patch Camelyon (Veeling et al. 2018)	2	800/1,000	200	32,768
	EuroSAT (Helber et al. 2019)	10			5,400
	Resisc45 (Cheng, Han, and Lu 2017)	45			6,300
	Retinopathy (Kaggle and EyePacs 2015)	5			42,670
Structured	Clevr/count (Johnson et al. 2017)	8	800/1,000	200	15,000
	Clevr/distance (Johnson et al. 2017)	6			15,000
	DMLab (Beattie et al. 2016)	6			22,735
	KITTI-Dist (Geiger et al. 2013)	4			711
	dSprites/location (Higgins et al. 2017)	16			73,728
	dSprites/orientation (Higgins et al. 2017)	16			73,728
	SmallNORB/azimuth (LeCun, Huang, and Bottou 2004)	18			12,150
	SmallNORB/elevation (LeCun, Huang, and Bottou 2004)	18			12,150
	Few-shot learning				
	Food-101 (Bossard, Guillaumin, and Gool 2014)	101	1/2/4/8/16 per class	20,200	30,300
	Stanford Cars (Krause et al. 2013)	196		1,635	8,041
	Oxford-Flowers102 (Nilsback and Zisserman 2006)	102		1,633	2,463
	FGVC-Aircraft (Maji et al. 2013b)	100		3,333	3,333
	Oxford-Pets (Parkhi et al. 2012)	37		736	3,669

Table 3: **Statistics of used datasets.**

B Experimental Details

B.1 Pretrained Backbones

See Table 4.

Model	Pretraining Dataset	Size (M)	Pretrained Weights
ViT-B/16 (Dosovitskiy et al. 2021)	ImageNet-21K	85.8	[a]checkpoint
Swin-B (Liu et al. 2021)	ImageNet-21K	86.7	[b]checkpoint

[a]https://storage.googleapis.com/vit_models/imagenet21K/ViT-B_16.npz
[b]https://github.com/SwinTransformer/storage/releases/download/v1.0.0/swin_base_patch4_window7_224_22k.pth

Table 4: **Pretrained backbones.**

B.2 Code Implementation

We use [5]*PyTorch* and [6]*timm* to implement all experiments on NVIDIA RTX 3090 GPUs.

[5]https://pytorch.org/
[6]https://rwightman.github.io/pytorch-image-models/

B.3 Data Augmentation

VTAB-1K Following Zhang, Zhou, and Liu (2022), we resize the images to 224×224, and then **normalize them with ImageNet's mean and standard deviation**.

We notice that some methods like VPT (Jia et al. 2022) and concurrent SSF (Lian et al. 2022b) does not normalize the input images of VTAB-1K. Such setting could benefit downstream tasks by closing the gap between the inputs of pre-training and fine-tuning, since the inputs are also not normalized in the pre-training process of ViT-B. We find that the average accuracy on VTAB-1K of PETL methods (including ours) could gain about 1-2% by using non-normalized inputs. However, since we reuse some results reported by NOAH, we still adopt the sub-optimal pipeline (*i.e.*, normalized inputs as in NOAH) for a fair comparison with the state-of-the-art.

Few-shot learning Following Zhang, Zhou, and Liu (2022), for training samples, we use color-jitter and RandAugmentation; for validation/test samples, we resize them to 256×256, crop them to 224×224 at the center, and then normalize them with ImageNet's mean and standard deviation.

B.4 Hyper-parameters

s is searched from $\{0.01, 0.1, 1, 10, 100\}$. See Table 5 for other hyper-parameters. We basically follow the hyper-parameters used by Zhang, Zhou, and Liu (2022).

	optimizer	batch size	learning rate	weight decay	# epochs	lr decay	# warm-up epochs
VTAB-1K	AdamW	64	1e-3	1e-4	100	cosine	10
Few-shot learning	AdamW	64	5e-3	1e-4	100	cosine	10

Table 5: **Hyper-parameters.**

B.5 Reproducibility of Baselines

The results of baselines can be reproduced by the official codebases of VPT (Jia et al. 2022) and NOAH (Zhang, Zhou, and Liu 2022).

We notice that it is reported by Jia et al. (2022) that Adapter significantly underperforms VPT. This is because their implementation uses zero-initialization for weights of both the two FC layers in Adapter, which blocks the back-propagation of gradient. Therefore, we follow Zhang, Zhou, and Liu (2022) using Xavier-initialization for weights and zero-initialization for biases, and report results that Adapter outperforms VPT. Similar observation is also reported in NLP literature (Houlsby et al. 2019; He et al. 2022).