Adapters Strike Back

Jan-Martin O. Steitz[1] Stefan Roth[1,2]
[1]Department of Computer Science, TU Darmstadt [2] hessian.AI

Abstract

Adapters provide an efficient and lightweight mechanism for adapting trained transformer models to a variety of different tasks. However, they have often been found to be outperformed by other adaptation mechanisms including low-rank adaptation. In this paper, we provide an in-depth study of adapters, their internal structure, as well as various implementation choices. We uncover pitfalls for using adapters and suggest a concrete, improved adapter architecture, called Adapter+*, that not only outperforms previous adapter implementations but surpasses a number of other, more complex adaptation mechanisms in several challenging settings. Despite this, our suggested adapter is highly robust and, unlike previous work, requires little to no manual intervention when addressing a novel scenario. Adapter+ reaches state-of-the-art average accuracy on the VTAB benchmark, even without a per-task hyperparameter optimization.*[†]

1. Introduction

Transfer learning from an off-the-shelf model, pre-trained on a large dataset like ImageNet [53] to a downstream task by fully fine-tuning the model's parameters is a common paradigm. A typical CNN architecture, like a ResNet [23], has several tens of millions of parameters. However, since the introduction of transformers [56] into the realm of computer vision [4, 5, 12, 49, 50, 60], model sizes have grown exponentially from around a hundred million parameters for a vision transformer (ViT) [12] to more than a billion parameters [9, 45]. This leads to huge storage requirements when fine-tuning on multiple downstream tasks because a complete set of the model's parameters needs to be saved per task. Additionally, large models require correspondingly large datasets [*e.g.*, 54] to be trained to their full potential, yet tend to overfit easily if the target dataset in transfer learning is too small. One solution is linear probing [11], where only the linear classifier is trained, but this usually yields inferior results compared to full fine-tuning.

As a consequence, there is a growing interest in parameter-efficient tuning methods. The main idea is to freeze the parameters of the pre-trained model and add a comparatively small amount of parameters to the model, which are then tuned together with the classifier to adapt the model to the downstream task at hand. Representative methods with different underlying concepts include VPT [30], which prepends the sequence of image tokens in the attention with trainable tokens to learn a prompt tuning, LoRA [28], where the attention weights are updated with learnable low-rank decomposition matrices, and Adapters [27], which are small bottleneck modules that are added to every transformer layer of the network. Adapters were first proposed for CNNs by Rebuffi et al. [51] and various formulations [21, 27, 47] exist for the now common ViT architecture.

Recent work on parameter-efficient transfer learning [*e.g.*, 19, 20, 30, 31, 38, 63] presents adapters as a baseline method for the adaptation to downstream tasks in computer vision. However, we identified various common issues in their implementations, which we find to have a negative influence on the

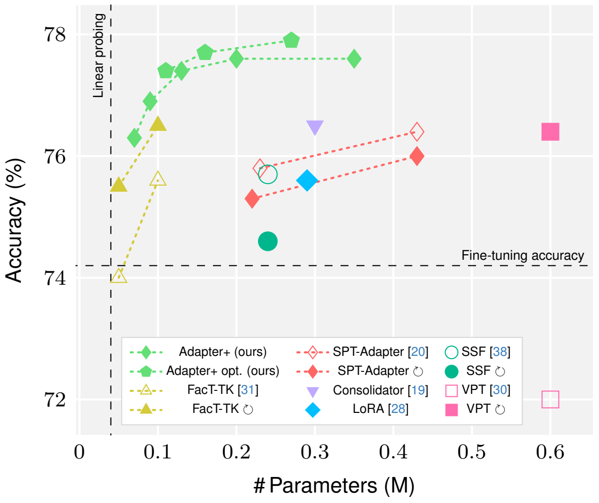


Figure 1. **Parameter-accuracy characteristics of adaptation methods on the VTAB [62]** ***test sets.*** We report original results and re-evaluations (↻) after a complete training schedule with suitable data normalization. Our Adapter+ has clearly the best parameter-accuracy trade-off. The vertical, dashed line shows the possible minimal number of tunable parameters when only the classifiers are trained, *i.e.*, using linear probing (61% accuracy).

[†]Code is available at https://github.com/visinf/adapter_plus.

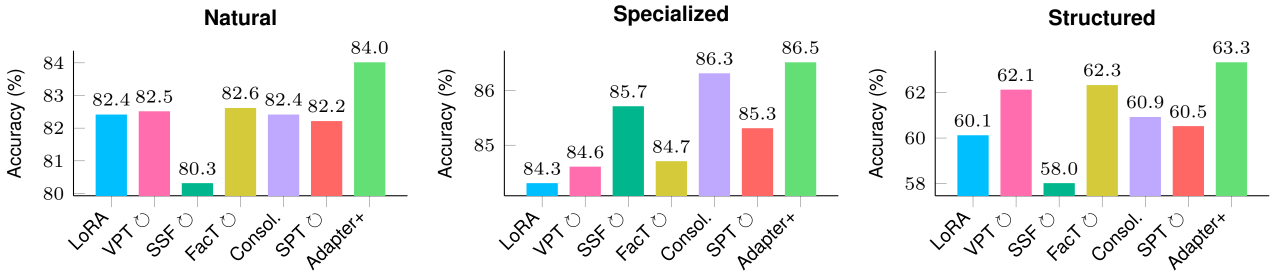


Figure 2. **Average accuracy for VTAB subgroups on the *test sets*.** For methods marked with ↻, we report results of our re-evaluation after a complete training schedule with suitable data normalization to ensure a fair comparison. Adapter+ is evaluated with rank $r \in [1..32]$.

adaptation performance. For further details, refer to the supplemental material. Additionally, while adapters have been well studied in natural language processing (NLP), there is no study that broadly examines the different adapter configurations for vision transformers. As a result, adapters have seemed to underperform in comparison to recent parameter-efficient adaptation methods, *e.g.*, reported accuracies of adapters on VTAB of 73.9% in [63] and 60.8% in [30].

In this work, we therefore revisit the idea of adapters and investigate how they can perform at their best in connection with ViTs. Our contribution hereby is threefold: *(1)* We show the *first in-depth and systematic study* on the effects of the adapter position in the transformer and of the adapter's inner structure with ViTs, as well as evaluate different variants of parameter initialization. *(2)* We further propose a *learnable, channel-wise scaling* as extension to plain adapters, which proves to be beneficial for computer vision tasks. *(3)* Finally, we present Adapter+, an adapter configuration with an *excellent parameter-accuracy trade-off* compared to other work, as shown in Fig. 1. Adapter+ reaches a state-of-the-art average accuracy of 77.6% on VTAB [62] *without any hyperparameter optimization per task* and 3.7 percentage points (pp) over previous adapter baselines. We also reach a state-of-the-art accuracy of 90.7% on FGVC [30] with the lowest number of parameters compared to other methods. Finally, Adapter+ shows the best robustness in terms of accuracy across the VTAB subgroups, see Fig. 2.

2. Related work

One possibility to adapt a pre-trained network to a novel task, apart from full fine-tuning, is to only selectively tune some of the parameters, *e.g.*, only training the classifier [11]. Cai et al. [3] proposed to tune only the biases of an otherwise frozen network to adapt it to a downstream task. BitFit [61] then showed the efficacy of this method for NLP transformers.

Modular adaptation. The concept of adding small, trainable modules with only a few parameters to an otherwise frozen network was first proposed for adapting CNNs by Rebuffi et al. [51] and called adapters. Other approaches replaced all convolutions in the network with depth-wise separable convolutions and only tuned their spatial parts [18], learned binary masks to prune a pre-trained network per target task [40], or created a student network by augmenting the original network with adapter-like modules and skip connections, which then mimicked a teacher network by disabling parts of its pre-trained and added modules [42].

Following the rise of transformers in NLP [10, 48, 56], Houlsby et al. [27] proposed adapter modules in the form of bottlenecks for transformer layers. Pfeiffer et al. [47] conducted an architecture search on NLP tasks to find a more parameter-efficient configuration of adapter modules that only acts on the transformer's feed-forward network (FFN), thus saving roughly half of the parameters over [27].

Prompt tuning. Inspired by changing the output of a network for NLP with hand-crafted textual prompts, which modifies the attention over the original input tokens, Lester et al. [36] proposed prompt tuning: A set of learnable tokens is added to the input sequence and trained with backpropagation to prompt a frozen language model to perform downstream tasks. Li and Liang [37] extended on prompt tuning by adding learnable tokens at every transformer layer of the model, which they termed prefix tuning. Jia et al. [30] applied prompt tuning to vision transformers, then called visual prompt tuning (VPT), by preprending the sequence of image patch embeddings with such trainable tokens (VPT-Shallow). They also showed a variant resembling prefix tuning with stronger adaptation capabilities that adds tokens at every layer of the network (VPT-Deep).

Low-rank approaches. Also focusing on the attention part of the transformer layers, Hu et al. [28] proposed low-rank adaptation (LoRA) where the attention weights are updated with low-rank decomposition matrices. The matrices can be merged with the attention weights for inference. The structure of LoRA is very similar to an adapter, which can be seen as a superset of LoRA acting on the transformer's FFN. He et al. [21] proposed a formalism to unify LoRA, adapters, and prefix tuning [37]. It allowed them to combine the beneficial aspects of all three methods into a scaled parallel adapter (Scaled PA) for NLP tasks. AdaptFormer [6] then applied the concept of Scaled PA to vision transformers.

Other related work. Newer approaches for vision transformers proposed different techniques to further enhance the parameter-accuracy trade-off in adaptation. NOAH [63] performs an architecture search for a combination of adapters, LoRA, and VPT for each task. SSF [38] scales and shifts the features in the network after every operation, *i.e.*, attention, FFN, layer normalization, with task-specific, trainable modules. Jie and Deng [31] aggregate the weights of a ViT into a single 3D tensor. Task-specific weight updates of this tensor are learned as a matrix decomposed into parameter-efficient factors, hence they termed their method factor-tuning (FacT). SPT [20] measures the importance of the weights of a pre-trained network for a downstream task. Based on a desired parameter budget, the most important parameters are chosen for tuning and adapters or LoRA are used for weight matrices that contain enough parameters of importance. Consolidator [19] adapts weights in multiple orderings of channel-wise groups. The updates for all groups are merged for efficient storage and inference.

Despite these new developments, we show that the simple concept of *adapters exhibits an even better parameter-accuracy trade-off* in combination with vision transformers – if done right and with the addition of a channel-wise scaling.

3. Adapters for vision transformers

3.1. Vision transformer basics

In this work, we concentrate on the parameter-efficient adaptation of vision transformers (ViT) [12]. The ViT is closely modeled after the transformer model for natural language processing (NLP) proposed by Vaswani et al. [56]. A learned linear projection embeds non-overlapping and flattened patches of the input image into a sequence of n tokens $\boldsymbol{x} \in \mathbb{R}^{n \times d}$, where d is called the hidden dimension of the transformer. A positional encoding is added to the embeddings and the sequence is prepended with a trainable `[CLS]` token. The sequence length and the dimension of the tokens stay fixed throughout the architecture. The sequence is sent through consecutive transformer layers that each consist of a multi-head self-attention and a feed-forward network (FFN). For the self-attention, the tokens are projected to queries, keys, and values ($\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$) and the output of each of the M attention heads is calculated as

$$\text{Attention}(\boldsymbol{x}) = \text{Softmax}\left(\frac{\boldsymbol{Q}(\boldsymbol{x})\boldsymbol{K}(\boldsymbol{x})^\mathsf{T}}{\sqrt{d'}}\right)\boldsymbol{V}(\boldsymbol{x}), \qquad (1)$$

with $d' = {}^{d}/_{M}$ being the inner dimension of the head. The FFN consists of a multilayer perceptron with two linear layers (with weights $\boldsymbol{W}_i$ and biases $\boldsymbol{b}_i$) and a GELU [25] non-linearity as activation in between:

$$\text{FFN}(\boldsymbol{x}) = \text{GELU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2. \qquad (2)$$

Both attention and FFN are employed with a preceding layer normalization (LN) [1] and a skip connection and, therefore, transform an input sequence $\boldsymbol{x}$ sequentially as

$$\boldsymbol{x} \mapsto \text{Attention}(\text{LN}(\boldsymbol{x})) + \boldsymbol{x} \qquad (3a)$$

$$\boldsymbol{x} \mapsto \text{FFN}(\text{LN}(\boldsymbol{x})) + \boldsymbol{x}\,. \qquad (3b)$$

To keep the notation concise, we will omit the LNs of attention and FFN in the following; each attention and FFN is assumed to be always preceded by an LN.

3.2. Adapters and their inner structure

Adapters [27] are small modules that are added to the transformer layers. They allow to tailor a network to a new task or domain, where instead of tuning the parameters of the whole network, only the adapter parameters and the classifier are trained. Adapters take the form of bottlenecks with an inner dimension of $r \ll d$. We call r the rank of the adapter. In detail, a down-projection to dimension r with weights $\boldsymbol{W}_\text{down} \in \mathbb{R}^{d \times r}$ and biases $\boldsymbol{b}_\text{down} \in \mathbb{R}^r$ is followed by a non-linear activation function $\sigma(\cdot)$, typically a GELU [25] as used throughout the ViT, and an up-projection with weights $\boldsymbol{W}_\text{up} \in \mathbb{R}^{r \times d}$ and biases $\boldsymbol{b}_\text{up} \in \mathbb{R}^d$ back to the hidden dimension d of the transformer layer. This yields a base adapter module

$$\text{Adapter}_\text{base}(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{W}_\text{down} + \boldsymbol{b}_\text{down})\boldsymbol{W}_\text{up} + \boldsymbol{b}_\text{up}\,. \qquad (4)$$

The base adapter module can be further enhanced with a normalization layer, *e.g.*, a layer normalization (LN) [1]. Additionally, the output of the bottleneck can be scaled by s as

$$\text{Adapter}(\boldsymbol{x}) = s \cdot \text{Adapter}_\text{base}\big(\text{LN}(\boldsymbol{x})\big)\,. \qquad (5)$$

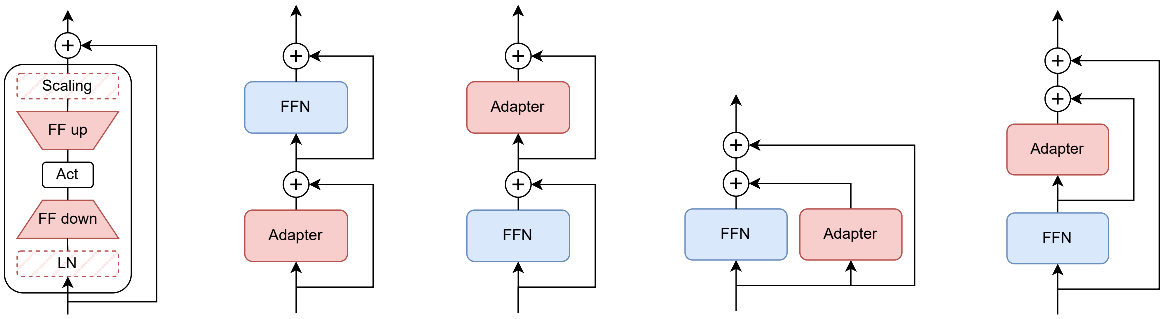
For layer-wise scaling, the factor s is taken to be a scalar, *i.e.* $s \in \mathbb{R}$, and can be either fixed as a hyperparameter or learned during training. Layer-wise scaling was proposed by He et al. [21] and Hu et al. [28] but deemed not effective compared to a fixed scaling for tasks in NLP. Here, we additionally propose to use a *channel-wise, learned scaling* where $\boldsymbol{s} \in \mathbb{R}^d$. We investigate its capabilities in Sec. 4.3. In most cases, the adapter is used with a skip connection, hence the complete feature transformation becomes

$$\boldsymbol{x} \mapsto \text{Adapter}(\boldsymbol{x}) + \boldsymbol{x}\,. \qquad (6)$$

The complete inner structure of an adapter including its skip connection is visualized in Fig. 3a.

3.3. Adapter positions

Although the architecture of bottleneck adapters for transformers is rather simple, there are various ways to plug them into the transformer layer. Previous work has not yet investigated what the optimum position is for the use with a ViT [12]. Here, we evaluate four possible adapter positions, shown in Figs. 3b to 3e. We postulate that it is easier for an adapter to learn to modify features previously transformed



(a) Inner structure (b) Pre (c) Post (d) Parallel (e) Intermediate

Figure 3. Illustrations of (a) the **inner structure of an adapter** with feed-forward layers (FF), activation layer (Act), and optional layer normalization (LN) and scaling, (b)–(d) different possible **adapter positions** to connect the adapter to the FFN section of the transformer layer. Modules with trainable parameters are shown in *red* and frozen modules in *blue*.

by a frozen module in the network rather than to anticipate what changes to the features are needed in adapting for a frozen module that follows the adapter. Putting it differently, we argue that the adapter should follow a frozen module.

Pre-Adapter. The first adapter position we analyze applies the adapter to the output $\boldsymbol{x}$ of the attention section of the transformer layer before it is passed into the FFN, but with the skip connection of the attention already added (Fig. 3b). The feature transformation of the FFN section with the adapter attached, therefore, becomes

$$\boldsymbol{x} \mapsto \mathrm{FFN}\big(\mathrm{Adapter}(\boldsymbol{x}) + \boldsymbol{x}\big) + \big(\mathrm{Adapter}(\boldsymbol{x}) + \boldsymbol{x}\big)\,. \tag{7}$$

Note that the two occurrences of $\mathrm{Adapter}(\boldsymbol{x})$ in Eq. (7) refer to the same instantiation. In this configuration, the adapter has the full information from the feature transformation happening in the attention but needs to estimate the transformation that will be happening in the FFN that follows. As a result, especially the last FFN before the linear classifier will be hard to adapt. To the best of our knowledge, this adapter position has not been considered in the literature.

Post-Adapter. In this case, the adapter is positioned at the very end of the transformer layer on the output of the FFN with its skip connection added as

$$\boldsymbol{x} \mapsto \mathrm{Adapter}\big(\mathrm{FFN}(\boldsymbol{x}) + \boldsymbol{x}\big) + \big(\mathrm{FFN}(\boldsymbol{x}) + \boldsymbol{x}\big)\,, \tag{8}$$

where the FFNs refer to the same intantiation (Fig. 3c). That way, the adapter has access to the feature transformation happening in the FFN and the unmodified features via the skip connection. This position has been proposed by Pfeiffer et al. [47] as the result of an architecture search, but only for adapting transformers for NLP tasks and not for a ViT.

Parallel-Adapter. Next, we consider a parallel setting as proposed by [21], where the adapter is located parallel to the FFN and both share a skip connection (Fig. 3d):

$$\boldsymbol{x} \mapsto \mathrm{FFN}(\boldsymbol{x}) + \mathrm{Adapter}(\boldsymbol{x}) + \boldsymbol{x}\,. \tag{9}$$

Therefore, both adapter and FFN work on the output of the attention section of the transformer layer and the adapter needs to learn the necessary residual transformation to the one produced by the frozen FFN.

Intermediate-Adapter. Finally, we consider the original adapter position as proposed by Houlsby et al. [27]. The adapter is plugged behind the FFN but before the skip connection of the FFN is added (Fig. 3e). The adapter additionally possesses its own skip connection:

$$\boldsymbol{x} \mapsto \mathrm{Adapter}\big(\mathrm{FFN}(\boldsymbol{x})\big) + \mathrm{FFN}(\boldsymbol{x}) + \boldsymbol{x}\,. \tag{10}$$

Note that the two occurrences of $\mathrm{FFN}(\boldsymbol{x})$ in Eq. (10) refer to the same instantiation. The adapter sees the transformed features coming from the FFN but cannot access the features added later on by the skip connection of the FFN.

3.4. Initialization of adapter parameters

Since training a deep learning model is a non-convex optimization problem, the initialization of parameters is important. In this work, we evaluate three different variants of parameter initializations for adapters proposed in the literature. All of them have the goal to initialize the adapters in a way that minimizes the initial influence of the adapters at the start of their training. This is a sensible goal since adapters extend an already pre-trained frozen network.

Houlsby initialization. Houlsby et al. [27] propose to draw the weights of the projection matrices from a zero-centered Gaussian distribution with a standard deviation of $\sigma = 0.01$, truncated at 2σ, and use zero for their biases.

BERT initialization. For the BERT model [10], the initialization works similar to [27] but the Gaussian distribution has a standard deviation of $\sigma = 0.02$ and is not truncated. This form of initialization is used by Pfeiffer et al. [47].

LoRA initialization. LoRA [28] initializes the weights and biases of the down-projection with a uniform Kaiming He initialization [22]; the weights and biases of the up-projection are initialized to zero. Therefore, the output of the adapter at the beginning of training equals zero and the adapter initially does not contribute.

3.5. Data normalization in pre-processing

Data normalization is common practice during image pre-processing. It is typically done by shifting and scaling of each input pixel x_{ij} for each channel c as

$$\hat{x}_{ijc} = (x_{ijc} - \mu_c)/\sigma_c \,. \tag{11}$$

Most widely used are the mean $\boldsymbol{\mu} = (0.485, 0.456, 0.406)^\mathsf{T}$ and standard deviation $\boldsymbol{\sigma} = (0.229, 0.224, 0.225)^\mathsf{T}$ of the ImageNet dataset [53], commonly referred to as ImageNet normalization. Another option is using 0.5 for every element of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, which is commonly referred to as Inception normalization because it is used for the Inception family of CNN architectures, starting with Inception-v3 [55]. The ImageNet normalization aims to center the input data around 0 with a standard deviation of 1. The Inception normalization, on the other hand, transforms the input values such they are strictly in range $[-1, 1]$.

Because we try to adapt to a target domain on a very low parameter budget, it is important to use the data normalization the network saw during its pre-training. Otherwise, the parameter-efficient transfer method of choice needs to first compensate for the shift in input data statistics and loses parts of its capacity to adapt to the target domain.

4. Experiments

4.1. Datasets

In order to carry out a detailed study of the utility of adapters in the context of ViT models, we experiment with two standard benchmarks for task adaptation.

VTAB. The Visual Task Adaptation Benchmark (VTAB) [62] consists of 19 tasks, which are further grouped into three categories: Natural, Specialized, and Structured. The *Natural* group contains natural images captured using standard photographic equipment. The *Specialized* group is built from datasets of images captured with specialized equipment, from remote sensing and medical domains. Lastly, the *Structured* group is for evaluating the understanding of the scene structure. Here, the majority of the datasets are compiled from synthetic images with scenes that are easy to assess for humans but have a large domain gap to natural image datasets. Each task of VTAB consists of 800 training and 200 validation images. The test sets have the same number of images as the test sets in the original datasets.

FGVC. Following Jia et al. [30], we compile five datasets for fine-grained visual classification (FGVC): CUB-200-2011 [58], NABirds [26], Oxford Flowers [44], Stanford Dogs [33], and Stanford Cars [16]. Because VTAB benchmarks task adaptation in a low-data regime in terms of the number of available training images, we use FGVC to evaluate adaptation methods in settings where training data is abundant. Where validation sets are not available in FGVC, we follow Jia et al. [30] to create the validation splits.

Table 1. **Adapter position.** We report the average accuracy in % (± std. dev.) on the VTAB *val sets* for different adapter positions. Adapter$_\text{base}$ with Houlsby initialization and rank $r = 8$ is used in all experiments.

Position	Natural	Specialized	Structured	Average
Pre	82.4 ± 0.4	**86.2** ± 0.8	57.5 ± 0.5	75.3 ± 0.3
Intermediate	**83.0** ± 0.4	85.0 ± 0.8	57.2 ± 0.5	75.1 ± 0.3
Parallel	**83.0** ± 0.3	**86.2** ± 0.6	57.7 ± 0.6	75.6 ± 0.3
Post	**83.0** ± 0.3	85.7 ± 0.4	**59.1** ± 0.3	**76.0** ± 0.2

For further details regarding the dataset properties of VTAB and FGVC, see supplemental material.

4.2. Experimental settings

For all our experiments, we use a ViT-B/16 network [12] that was pre-trained on ImageNet-21k [53]. We follow its pre-training settings, in particular, regarding input data normalization. We train all models with an AdamW [39] optimizer with a learning rate of 10^{-3}, a weight decay of 10^{-4}, and a batch size of 64, following [63]. For full fine-tuning, we use a learning rate of 10^{-4}, which we found leads to better results. We use a cosine learning rate schedule with a linear warm-up over the first 10 epochs and train for 100 epochs in total. We use stochastic depth with linearly increasing drop rates as a function of network depth from 0 to 0.1 for the frozen network and with a drop rate of 0.1 for the adapters during training. Apart from data normalization (*cf*. Sec. 3.4), we resize input images to 224×224 px for VTAB and use a randomly resize crop to 224×224 px and horizontal flipping for FGVC. For the ablations and to determine hyperparameters, we evaluate on the validation splits. We include the validation sets in the training data for producing final results.

4.3. Exploring adapter configurations

Adapter position. We first evaluate the four possible positions to connect an adapter to the FFN section of the transformer layer, as described in Sec. 3.3. In our ablation, we use Adapter$_\text{base}$ (*cf*. Eq. (4)) with rank $r = 8$ and use the Houlsby initialization. In this experiment, the adapters neither have a layer normalization nor use scaling.

The results on the VTAB validation set for all four adapter positions are presented in Tab. 1. The *Post-Adapter yields the best result* with 76.0% average accuracy over all VTAB subgroups. It confirms our hypothesis that the adapter should follow the frozen FFN module because it can then post-hoc modify the features flowing through the network. The parallel configuration comes in second with 75.6% average accuracy, receiving the same input as the FFN but having to

Table 2. **Inner adapter structure.** We evaluate the different components of the adapter structure, *e.g.*, normalization layer (*Norm*), *layer*-wise and *channel*-wise learnable scaling on the VTAB *val sets*. The difference to Adapter$_{\text{base}}$ (*first row*) is shown in Δ_{base}.

Bias	Norm	Scaling	Initialization	Accuracy (%)	Δ_{base}
✓			Houlsby	76.0 ± 0.2	0.0
			Houlsby	75.6 ± 0.4	–0.4
✓			LoRA	75.5 ± 0.3	–0.5
✓			BERT	75.8 ± 0.3	–0.2
✓	✓		Houlsby	75.9 ± 0.3	–0.1
✓	✓	layer	Houlsby	75.9 ± 0.3	–0.1
✓		layer	Houlsby	$\underline{76.2}$ ± 0.3	$\underline{+0.2}$
✓	✓	channel	Houlsby	75.8 ± 0.3	–0.2
✓		channel	Houlsby	**76.5** ± 0.2	**+0.5**

learn a residual modification to the FFN instead of a subsequent one. Pre-Adapter and Intermediate-Adapter are subpar compared to the other positions. They either do not have access to the feature transformation happening afterwards in the FFN or to the features of the skip connection containing the output of the attention.

Inner structure. Next, we investigate the impact of the inner structure of adapters including their initialization. Tab. 2 shows our findings with average accuracies calculated over the three VTAB subgroups. Removing the biases from the linear layers leads to a decrease in accuracy of 0.4 percentage points (pp). We find that the *Houlsby initialization of the adapter parameters is best* while BERT and LoRA initializations reduce the accuracy by 0.2 pp and 0.5 pp. Adding layer normalization (LN) to the adapter is slightly detrimental for all settings, both with scaling and without, while additionally adding $2d$ parameters per layer. We find that *a learned scaling is in general beneficial* for image-classification tasks. Adding layer-wise scaling leads to a gain of 0.2 pp. The inclusion of a learned, channel-wise scaling, as proposed here, gives the strongest improvement of 0.5 pp, reaching an accuracy of 76.5% on the VTAB validation set while only adding half of the parameters compared to LN.

What makes a great adapter? From our systematic exploration of possible adapter configurations, we conclude that adapter modules in the **Post-Adapter** position with a learnable, **channel-wise scaling** and **Houlsby initialization** work best for computer vision tasks. We call our proposed adapter configuration **Adapter+**. The addition of layer normalization, as suggested by Pfeiffer et al. [47], is not necessary and even leads to detrimental effects in our setting.

Configurations from previous work. Different configurations of adapters have been established in previous work. We compare their configurations to our systematic approach with rank $r = 8$ on the VTAB validation sets. Using our own implementations already leads to better results than reported in literature but enables us to compare on equal footing. Houlsby et al. [27] use an Intermediate-Adapter with their proposed initialization both at the FFN section as well at the attention part of the transformer layer. Additionally, they adapt the LN parameters of the backbone. We, therefore, compare their setting additionally with $r = 4$ to compare on roughly the same parameter budget. Pfeiffer et al. [47] suggest a Post-Adapter like us but with a BERT initialization and they employ a layer normalization inside the adapter. AdaptFormer [6] has the same configuration as a scaled parallel adapter (Scaled PA) [21], which was proposed for NLP tasks, the only difference being the layer-wise scaling s. Scaled PA uses a fixed scaling of $s = 4$ for the adapters whereas AdaptFormer suggests to use $s = 0.1$ for vision tasks. Optimizing s for VTAB may lead to better results. Our results are presented in Tab. 3. We see a clear advantage of our Adapter+ configuration, gaining at least 0.7 pp over all previous adapter realizations considered despite having the second lowest number of trainable parameters.

Table 3. **Comparison of Adapter+ with adapter configurations from previous work.** We report the average accuracy in % (± std. dev.) of each subgroup and across all groups on the VTAB *val sets*.

Configuration	# Param (M)	Natural	Specialized	Structured	Average
Houlsby [27], $r = 8$	0.39	$\underline{82.9}$ ± 0.2	85.5 ± 0.3	$\underline{58.9}$ ± 0.8	$\underline{75.8}$ ± 0.3
Houlsby [27], $r = 4$	0.24	$\underline{82.9}$ ± 0.4	84.9 ± 0.3	58.3 ± 0.6	75.4 ± 0.3
Pfeiffer [47]	0.21	$\underline{82.9}$ ± 0.3	$\underline{86.1}$ ± 0.9	58.4 ± 0.7	$\underline{75.8}$ ± 0.4
AdaptFormer [6]	**0.19**	**83.0** ± 0.4	85.0 ± 0.2	57.4 ± 0.5	75.2 ± 0.2
Adapter+	$\underline{0.20}$	**83.0** ± 0.2	**86.8** ± 0.6	**59.7** ± 0.4	**76.5** ± 0.2

4.4. Main results

VTAB. We evaluate Adapter+ on the VTAB test sets and compare to other methods in Tab. 4. We provide results for *full* fine-tuning and tuning only the *linear* classifier while freezing the rest of the backbone [11] as a baseline of classical fine-tuning methods. As competing parameter-efficient tuning methods, we include LoRA [28], VPT [30], NOAH [63], SSF [38], FacT [31], Consolidator [19], and SPT [20].

Wherever possible, we re-evaluate the other methods with a suitable data normalization for the pre-trained backbone and after the full training schedule to enable a fair comparison. For LoRA, we use our own implementation because the original work does not cover VTAB. For VPT, we adopt the number of tokens per task from their hyperparameter optimization but find that we do not need to tune learning rate and weight decay per task. Additionally, deviating from the original implementation, we optimize with AdamW [39] instead of SGD [52] and change to an appropriate data normalization. We present the original results from [30] on VTAB together with our re-evaluation. Our improved implementation of VPT increases the average accuracy by 4.4 pp from 72.0% to 76.4%. SSF, FacT, and SPT released code to evaluate on VTAB. For FacT and SPT, we change the data normalization to match the backbone; SSF already uses the correct one. We re-run the provided code and present the

Table 4. **Detailed results on the VTAB *test sets*.** We report original results and re-evaluations (↻) in % after a complete training schedule with suitable data normalization. Grayed out numbers are not included in the ranking for **best** and second best results. †: Early-stopping based on the *test set*, ●: unsuitable data normalization, ↯: per-task hyperparameter optimization. [1]Average across the average accuracies of the VTAB groups, following previous work. [2]No complete code release for Consolidator, hence training and evaluation details are unknown.

		Natural								Specialized					Structured									
	# Param (M)	Cifar100 [34]	Caltech101 [14]	DTD [8]	Flower102 [44]	Pets [46]	SVHN [43]	Sun397 [59]	Average	Camelyon [57]	EuroSAT [24]	Resisc45 [7]	Retinopathy [13]	Average	Clevr-Count [32]	Clevr-Dist. [32]	DMLab [2]	KITTI-Dist. [17]	dSpr-Loc. [41]	dSpr-Ori [41]	sNORB-Azi. [35]	sNORB-Ele. [35]	Average	Global Average[1]
Full	85.8	73.2	92.6	70.4	97.9	86.2	90.6	39.6	78.6	87.1	96.6	87.5	74.0	86.3	66.6	61.0	49.8	79.7	82.6	51.9	33.5	37.0	57.8	74.2
Linear	0.04	78.1	88.1	69.0	99.1	90.0	36.0	56.9	73.9	79.8	90.7	73.7	73.7	79.5	32.4	30.5	35.9	61.9	11.2	26.2	14.3	24.5	29.6	61.0
LoRA [28]	0.29	83.0	91.7	71.6	99.2	90.9	83.8	56.7	82.4	86.2	95.7	83.5	71.9	84.3	77.7	62.3	49.0	80.2	82.2	51.7	31.0	47.0	60.1	75.6
VPT-Deep ↯● [30]	0.60	78.8	90.8	65.8	98.0	88.3	78.1	49.6	78.5	81.8	96.1	83.4	68.4	82.4	68.5	60.0	46.5	72.8	73.6	47.9	32.9	37.8	55.0	72.0
VPT-Deep ↯↻	0.60	83.0	93.0	71.2	99.0	91.3	84.1	56.0	82.5	84.9	96.6	82.5	74.5	84.6	77.5	58.7	49.7	79.6	86.2	56.1	**37.9**	**50.7**	62.1	76.4
NOAH ↯†◒ [63]	0.43	69.6	92.7	70.2	99.1	90.4	86.1	53.7	80.2	84.4	95.4	83.9	75.8	84.9	82.8	**68.9**	49.9	81.7	81.8	48.3	32.8	44.2	61.3	75.5
SSF ↯† [38]	0.24	69.0	92.6	75.1	99.4	91.8	90.2	52.9	81.6	87.4	95.9	87.4	75.5	86.6	75.9	62.3	53.3	80.6	77.3	54.9	29.5	37.9	59.0	75.7
SSF ↯↻	0.24	61.9	92.3	73.4	**99.4**	**92.0**	90.8	52.0	80.3	86.5	95.8	87.5	72.8	85.7	77.4	57.6	53.4	77.0	78.2	54.3	30.3	36.1	58.0	74.6
FacT-TK$_8$ ↯†● [31]	0.05	70.3	88.7	69.8	99.0	90.4	84.2	53.5	79.4	82.8	95.6	82.8	75.7	84.2	81.1	68.0	48.0	80.5	74.6	44.0	29.2	41.1	58.3	74.0
FacT-TK$_8$ ↯↻	**0.05**	74.9	92.7	73.7	99.1	91.3	85.5	57.7	82.1	86.8	94.9	84.1	70.9	84.2	81.9	64.1	49.2	77.2	83.8	53.1	28.2	44.7	60.3	75.5
FacT-TK$_{\leq 32}$ ↯†● [31]	0.10	70.6	90.6	70.8	99.1	90.7	88.6	54.1	80.6	84.8	96.2	84.5	75.7	85.3	82.6	68.2	49.8	80.7	80.8	47.4	33.2	43.0	60.7	75.6
FacT-TK$_{\leq 32}$ ↯↻	0.10	74.6	93.7	73.6	99.3	90.6	88.7	57.5	82.6	87.6	95.4	85.5	70.4	84.7	**84.3**	62.6	51.9	79.2	85.5	52.0	36.4	46.6	62.3	76.5
Consolidator [2] [19]	0.30	74.2	90.9	**73.9**	**99.4**	91.6	**91.5**	55.5	82.4	86.9	95.7	86.6	**75.9**	86.3	81.2	68.2	51.6	**83.5**	79.8	52.3	31.9	38.5	60.9	76.5
SPT-Adapter †● [20]	0.23	72.9	93.2	72.5	99.3	91.4	84.6	55.2	81.3	85.3	96.0	84.3	75.5	85.3	82.2	68.0	49.3	80.0	82.4	51.9	31.7	41.2	60.8	75.8
SPT-Adapter ↻	0.22	74.7	94.1	73.0	99.1	91.2	84.5	57.5	82.0	85.7	94.9	85.7	70.2	84.1	81.3	63.2	49.1	80.7	83.5	52.0	26.4	41.5	59.7	75.3
SPT-Adapter †● [20]	0.43	72.9	93.2	72.5	99.3	91.4	88.8	55.8	82.0	86.2	96.1	85.5	75.5	85.8	83.0	68.0	51.9	81.2	82.4	51.9	31.7	41.2	61.4	76.4
SPT-Adapter ↻	0.43	74.9	93.2	71.6	99.2	91.1	87.9	57.2	82.2	87.0	95.4	86.5	72.4	85.3	81.1	63.2	50.3	80.2	84.4	51.4	31.5	42.2	60.5	76.0
Adapter+, $r=1$	0.07	**85.4**	92.4	73.1	99.1	91.3	83.1	**58.1**	83.2	87.2	96.6	85.3	72.6	85.5	80.7	60.6	50.9	79.9	83.3	55.6	27.1	43.0	60.1	76.3
Adapter+, $r=2$	0.09	**85.4**	93.0	72.7	99.2	90.6	85.3	58.0	83.5	**87.9**	96.8	85.5	71.4	85.4	83.2	61.0	51.6	80.1	86.1	**56.3**	30.7	46.5	61.9	76.9
Adapter+, $r=4$	0.13	84.8	93.8	72.7	99.2	90.6	86.5	57.4	83.6	87.5	96.9	85.9	71.5	85.4	83.4	61.6	53.6	81.4	87.3	55.3	34.4	48.1	63.1	77.4
Adapter+, $r=8$	0.20	84.6	**94.2**	72.3	99.3	90.7	87.6	56.7	83.6	87.7	**97.0**	86.7	72.3	85.9	83.2	60.9	**53.8**	80.3	88.1	55.6	35.7	47.7	63.1	77.6
Adapter+, $r=16$	0.35	83.7	**94.2**	71.5	99.3	90.6	88.2	55.8	83.3	87.5	**97.0**	87.4	72.9	86.2	82.9	60.9	53.7	80.8	**88.4**	55.2	37.3	46.9	**63.3**	77.6
Adapter+, $r \in [1..4]$ ↯	0.11	**85.4**	93.8	72.7	99.1	90.6	86.5	**58.1**	83.7	87.5	96.8	85.9	71.4	85.4	83.4	61.0	53.6	81.4	87.3	55.3	34.4	48.1	63.1	77.4
Adapter+, $r \in [1..8]$ ↯	0.16	**85.4**	93.8	72.7	99.1	90.7	87.6	**58.1**	83.9	87.7	96.8	86.7	72.3	85.9	83.4	60.9	**53.8**	80.3	88.1	55.3	35.7	47.7	63.1	77.7
Adapter+, $r \in [1..32]$ ↯	0.27	**85.4**	93.8	72.7	99.1	90.7	88.2	**58.1**	**84.0**	87.5	96.8	**87.8**	73.9	**86.5**	83.4	60.9	**53.8**	80.3	87.2	55.3	**37.9**	47.7	**63.3**	**77.9**

results after a full training schedule. For completeness, we also report the results from the original publications. However, we found that the code releases of [20, 31, 38] use early stopping based on the best result on the *test set*. We argue that tuning hyperparameters such as the number of training epochs on the test set goes against established practices in machine learning; rather the validation set should be used for early stopping. Yet, due to the limited size of the training and validation sets in VTAB, it is not feasible to report test results without also training on the validation data. Hence, we chose to complete a full training schedule of 100 epochs instead of using early stopping. Training SSF for the full schedule leads to a decrease in average accuracy of 1.1 pp over the original publication and re-evaluating SPT leads to a decrease of up to 0.5 pp, even with a corrected data normalization. FacT on the other hand benefits from our re-revaluation, since the accuracy decrease from training a complete schedule is offset by improvements from applying the appropriate data normalization. There was no complete code release with configurations to train Consolidator on VTAB at the time of writing, hence we report results as-is.

Adapter+ shows the best parameter-accuracy trade-off among all methods evaluated. This can also be clearly seen in Fig. 1. Additionally, Adapter+ sets a new state of the art with an average accuracy of up to 77.6% over all VTAB subgroups *even without any per-task hyperparameter optimization*. If we determine the optimal rank r per task on the validation set, we can further improve the accuracy to 77.9%. Optimizing the rank leads to a better parameter-accuracy trade-off than using a fixed rank across all tasks.

In Fig. 2, we compare the average accuracy on the subgroups of VTAB. Wherever possible, we present the results of re-evaluating methods after the last training epoch and matching the data normalization to the backbone. The average accuracies of Adapter+ with $r \in [1..32]$ are consistently higher than those of the competing methods. Note that the accuracies of other methods except SPT differ drastically across the different VTAB subgroups. Adapter+, on the other hand, shows a high degree of robustness to the domain shifts between groups.

FGVC. Next, we present our results on the FGVC benchmark in Tab. 5. From the contenders, only SSF [38] has released code and hyperparameter configurations for training on FGVC at the time of writing. As we know from the code releases for VTAB, the reported numbers show the accuracy for early stopping based on the *test set*. Therefore, we expect a similar evaluation for FGVC. While we do not endorse early stopping based on the test set, we ad-

Table 5. **Detailed results on the FGVC *test sets*.** We report original results and re-evaluations (↻) in % after a complete training schedule with suitable data normalization. Grayed out numbers are not included in the ranking for **best** and second best results.

	# Param (M)	CUB200 [58]	NABirds [26]	Oxford Flowers [44]	Stanford Dogs [33]	Stanford Cars [16]	Average
Full	86.0	88.0	81.5	99.2	85.6	90.6	89.0
Linear	0.18	88.9	81.8	99.5	92.6	52.8	83.1
VPT-Deep [30]	0.85	88.5	84.2	99.0	90.2	83.6	89.1
VPT-Deep ↻	0.85	**90.1**	83.3	**99.6**	90.3	85.0	89.7
SSF [38]	0.39	89.5	85.7	99.6	89.6	89.2	90.7
SSF ↻	0.39	88.9	**85.0**	**99.6**	88.9	88.9	90.3
SPT-Adapter [20]	0.40	89.1	83.3	99.2	91.1	86.2	89.8
SPT-LoRA [20]	0.52	88.6	83.4	99.5	91.4	87.3	90.1
Adapter+, $r \in [1..32]$	**0.34**	90.0	83.2	**99.6**	91.6	89.1	**90.7**
Adapter+ (best epoch)	0.34	90.4	85.0	99.7	92.6	89.1	91.4

ditionally provide numbers for that setting in Tab. 5 for the sake of comparability. Even when training for a complete schedule, Adapter+ shows the best average accuracy with 90.7% over all five datasets in FGVC, 0.4 pp over the second best method under similar evaluation. When early stopping with the test set, Adapter+ reaches 91.4% average accuracy, 0.7 pp over the second best method and 2.4 pp better than full fine-tuning. This demonstrates that Adapter+ also yields state-of-the-art results for task adaptation when training data is abundant while having the best parameter efficiency.

4.5. Ablations

Data normalization. We showcase the effect of using an unsuitable data normalization for the chosen ViT in Tab. 6. The gap between ImageNet and Inception normalization (see Sec. 3.5) is largest for VPT [30], with a 3.4 pp difference in average accuracy, which explains around two-thirds of the gain for our re-evaluation as shown in Fig. 1. We suspect that VPT has less of an ability to scale and shift the data because the learnable tokens only act on the attention mechanism. LoRA [28], FacT [31], and adapters all employ linear layers that can directly scale and shift the features of the frozen backbone and thus compensate better for improper data normalization. It is worth mentioning that our Adapter+ is the most robust to improper normalization out of the methods evaluated, with a gap of only 2.6 pp average accuracy.

Training regularization. We investigate the importance of training regularization methods like stochastic depth [29] and dropout [15] for training adapters on a frozen ViT backbone and evaluate on the VTAB validation sets. We use linearly increasing drop rates as a function of network depth from 0 to 0.1 for the frozen layers of the ViT model, and a drop rate of 0.1 when using dropout or stochastic depth for the adapter modules. The results in Tab. 7 show a clear benefit for using stochastic regularization for the frozen layers as well as the adapters during training. Using dropout in the adapters is only slightly better than no regularization for adapters, with a gain of only 0.1 pp. With an increase in accuracy of 0.7 pp, *stochastic depth is the preferred regularization method for adapters*. However, our results show that the more important part is the *stochastic depth regularization for the frozen modules of the ViT backbone*. Disabling it in training leads to a loss of 1.5 pp accuracy compared to a training where stochastic depth is used throughout the model.

Table 6. **Effects of ImageNet *vs*. Inception data normalization.** All methods are evaluated on the VTAB *val sets*. In column Δ_{Average} we report the increase in accuracy in pp across all VTAB subgroups.

	ImageNet norm				Inception norm				
	Natural	Specialized	Structured	Average	Natural	Specialized	Structured	Average	Δ_{Average}
VPT	79.2	83.0	53.8	72.0	82.2	86.2	57.9	75.4	3.4
LoRA	78.4	84.1	53.2	71.9	82.0	85.8	56.4	74.7	2.8
FacT-TK	78.0	83.3	56.1	72.4	81.6	85.6	58.1	75.1	2.7
Adapter+	**80.5**	**85.0**	**56.0**	**73.9**	**83.0**	**86.8**	**59.7**	**76.5**	**2.6**

Table 7. **Influence of training regularization.** We evaluate accuracy in % with Adapter$_{\text{base}}$ with rank $r = 8$ on the VTAB *val sets*.

		Adapter		
		Stochastic Depth	Dropout	None
ViT	Stochastic Depth	**76.0**	75.4	75.3
	None	74.5	74.3	73.7

5. Conclusion

Applied at the right position and with an optimal inner structure, the simple concept of adapters produces state-of-the-art results for task adaptation. To understand how adapters can "strike back", we conducted the first systematic and in-depth study on how to best construct adapters and integrate them with vision transformers. This allowed us to determine the optimal connection point for the adapter in the transformer layer. Further, we proposed to use a learnable, channel-wise scaling and showed its benefit for computer vision tasks. Our insights led us to the creation of Adapter+ that yields the highest accuracy and the best parameter-accuracy trade-off on VTAB (77.6%, 0.2M) without any per-task hyperparameter optimization and on FGVC (90.7%, 0.34M), showing its superiority over more complicated methods.

Acknowledgements. This work has been funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center.

References

[1] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv:1607.06450 [stat.ML]*, 2016.

[2] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. DeepMind Lab. *arXiv:1612.03801 [cs.AI]*, 2016.

[3] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. TinyTL: Reduce memory, not parameters for efficient on-device learning. In *NeurIPS*2020*.

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, pages 213–229, 2020.

[5] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, pages 9630–9640, 2021.

[6] Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. AdaptFormer: Adapting vision transformers for scalable visual recognition. In *NeurIPS*2022*.

[7] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proc. IEEE*, 105(10):1865–1883, 2017.

[8] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *CVPR*, pages 3606–3613, 2014.

[9] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters. In *ICML*, pages 7480–7512, 2023.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[11] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[13] Emma Dugas, Jorge Jared, and Will Cukierski. Diabetic retinopathy detection. Kaggle, 2015.

[14] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE T. Pattern Anal. Mach. Intell.*, 28(4):594–611, 2006.

[15] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, pages 1050–1059, 2016.

[16] Timnit Gebru, Jonathan Krause, Yilun Wang, Duyun Chen, Jia Deng, and Li Fei-Fei. Fine-grained car detection for visual census estimation. In *AAAI*, pages 4502–4508, 2017.

[17] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *Int. J. Robotics Res.*, 32(11):1231–1237, 2013.

[18] Yunhui Guo, Yandong Li, Liqiang Wang, and Tajana Rosing. Depthwise convolution is all you need for learning multiple visual domains. In *AAAI*, pages 8368–8375, 2019.

[19] Tianxiang Hao, Hui Chen, Yuchen Guo, and Guiguang Ding. Consolidator: Mergable adapter with group connections for visual adaptation. In *ICLR*, 2023.

[20] Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Sensitivity-aware visual parameter-efficient tuning. In *ICCV*, 2023.

[21] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *ICLR*, 2022.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[24] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.*, 12(7):2217–2226, 2019.

[25] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv:1606.08415 [cs.LG]*, 2023.

[26] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge J. Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *CVPR*, pages 595–604, 2015.

[27] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *ICML*, pages 2790–2799, 2019.

[28] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

[29] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, pages 646–661, 2016.

[30] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge J. Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *ECCV*, pages 709–727, 2022.

[31] Shibo Jie and Zhi-Hong Deng. FacT: Factor-tuning for lightweight adaptation on vision transformer. In *AAAI*, pages 1060–1068, 2023.

[32] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, pages 1988–1997, 2017.

[33] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *CVPR Workshop on Fine-grained Visual Classification*, 2011.

[34] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Canadian Institute for Advanced Research, 2009.

[35] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*, pages 97–104, 2004.

[36] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, pages 3045–3059, 2021.

[37] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL/IJCNLP*, pages 4582–4597, 2021.

[38] Dongze Lian, Daquan Zhou, Jiashi Feng, and Xinchao Wang. Scaling & shifting your features: A new baseline for efficient model tuning. In *NeurIPS*2022*.

[39] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[40] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, pages 72–88, 2018.

[41] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dSprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset, 2017.

[42] Pedro Morgado and Nuno Vasconcelos. NetTailor: Tuning the architecture, not just the weights. In *CVPR*, pages 3044–3054, 2019.

[43] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[44] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *ICVGIP*, pages 722–729, 2008.

[45] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael G. Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *arXiv:2304.07193 [cs.CV]*, 2023.

[46] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *CVPR*, pages 3498–3505, 2012.

[47] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *EACL*, pages 487–503, 2021.

[48] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

[49] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763, ICML.

[50] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *ICCV*, pages 12159–12168, 2021.

[51] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*2017*, pages 506–516.

[52] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65(6):386–408, 1958.

[53] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(13):211–252, 2015.

[54] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5B: An open large-scale dataset for training next generation image-text models. In *NeurIPS*2022*.

[55] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*2017*, pages 5998–6008.

[57] Bastiaan S. Veeling, Jasper Linmans, Jim Winkens, Taco Cohen, and Max Welling. Rotation equivariant CNNs for digital pathology. In *MICCAI*, pages 210–218, 2018.

[58] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 dataset. Technical report, California Institute of Technology, 2011.

[59] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*, pages 3485–3492, 2010.

[60] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, José M. Álvarez, and Ping Luo. SegFormer: Simple and

efficient design for semantic segmentation with transformers. In *NeurIPS*2021*, pages 12077–12090.

[61] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *ACL*, 2022.

[62] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, Lucas Beyer, Olivier Bachem, Michael Tschannen, Marcin Michalski, Olivier Bousquet, Sylvain Gelly, and Neil Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv:1910.04867 [cs.CV]*, 2020.

[63] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. Neural prompt search. *arXiv:2206.04673 [cs.CV]*, 2022.