Time-, Memory- and Parameter-Efficient Visual Adaptation

Otniel-Bogdan Mercea[1, 2*]   Alexey Gritsenko[1]   Cordelia Schmid[1]   Anurag Arnab[1†]

[1]Google   [2]University of Tübingen

Abstract

As foundation models become more popular, there is a growing need to efficiently finetune them for downstream tasks. Although numerous adaptation methods have been proposed, they are designed to be efficient only in terms of how many parameters are trained. They, however, typically still require backpropagating gradients throughout the model, meaning that their training-time and -memory cost does not reduce as significantly.

We propose an adaptation method which does not backpropagate gradients through the backbone. We achieve this by designing a lightweight network in parallel that operates on features from the frozen, pretrained backbone. As a result, our method is efficient not only in terms of parameters, but also in training-time and memory usage. Our approach achieves state-of-the-art accuracy-parameter tradeoffs on the popular VTAB benchmark, and we further show how we outperform prior works with respect to training-time and -memory usage too. We further demonstrate the training efficiency and scalability of our method by adapting a vision transformer backbone of 4 billion parameters for the computationally demanding task of video classification, without any intricate model parallelism. Here, we outperform a prior adaptor-based method which could only scale to a 1 billion parameter backbone, or fully-finetuning a smaller backbone, with the same GPU and less training time. To facilitate further research, we release code at https://github.com/google-research/scenic.

1. Introduction

Foundation models [3, 6, 9, 35, 42] are becoming the *de facto* tools of modern vision systems: Large models, trained on massive datasets, have diverse abilities across a range of applications. Such foundation models are typically generalists that perform well in zero- or few-shot settings across a range of tasks [1, 3]. However, they typically achieve their best results on individual tasks when finetuned specifically for it, particularly when there is a large domain gap

---

*Work done during an internship at Google.
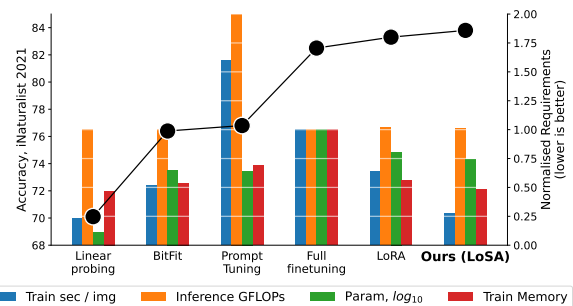
†Correspondence to aarnab@google.com.

Figure 1. Parameter-efficient adaptation methods proposed in the literature are not necessarily efficient in terms of other efficiency metrics. Prompt-tuning [25], for example, learns only a few learnable prompt tokens, but is in fact slower than fully finetuning the network due to the added tokens. LoRA [21] and Bit-Fit [52], do not substantially reduce training time as they still need to backpropagate through the whole network. Our method, Low-Rank Side Adaptation (LoSA), in contrast, achieves improvements across multiple efficiency metrics and tasks. Experiments conducted by adapting ViT-g with 1 billion parameters.

to the web-sourced pretraining data [1, 5, 35]. As these models continue growing, it is crucial to be able to efficiently adapt, or partially finetune, large pretrained models for downstream tasks. And particulary so for safety-critical tasks where the highest possible accuracy is critical.

Numerous efficient adaptation methods for large, pretrained models have been proposed in the literature, including LoRA [21], adapters [20, 48] and prompt-tuning [25, 34, 36] among others. These methods are typically designed to be parameter-efficient, or in other words, learn the smallest number of parameters to achieve high task accuracy. However, there are numerous other relevant efficiency metrics such as training time and memory usage that are crucial, as they determine whether the adaptation method is feasible or not for larger models or low computational resources. As shown in Fig. 1, although many prior methods dramatically reduce the number of learned parameters, they typically do not improve other efficiency metrics like training-time and memory as significantly. This fact is because even though they only train a small number of parameters, they still require backpropagating gradients throughout the entire backbone of the pre-trained model. Prompt-tuning [25, 34, 36] in particular, also adds additional tokens to the network, and therefore increases the training- and inference-time signifi-

cantly as well, to the point that it may be even slower than fully finetuning the network as shown in Fig. 1.

More broadly, efficiency for machine learning models is a multifaced topic, as it can be measured in terms of numerous cost metrics which ultimately depend on the use case [8, 30]. Nevertheless, most existing work on adapting foundation models focus only on parameter-efficiency. In this work, we propose a method, Low-Rank Side Adaptation (LoSA), that is efficient not only in terms of the number of learned parameters, but also in the time- and accelerator-memory required during training. This is motivated by the fact that these are key considerations for efficiently adapting large-scale models in practice, as the training costs determine whether the adaptation method is feasible in the first place. Our work does not focus on the inference costs of such models though, which is bounded primarily by the size of the pretrained model that we are adapting.

Our method does not require backpropagating gradients through the backbone, as we keep the entire backbone frozen, and instead learn a parallel network which refines these frozen backbone features for the target task. We find that the architecture of our parallel network is important to achieve good accuracy-efficiency trade-offs. In particular, low-rank MLP projections which alternate between the channel, spatial and temporal (if present) dimensions achieve state-of-the-art trade-offs between accuracy, learned parameters, training-time and memory (Fig. 1). And notably, unlike prior work, we thoroughly evaluate both our method and baselines on multiple efficiency indicators, not just the number of learned parameters.

Our approach achieves state-of-the-art accuracy-parameter trade-offs on the VTAB benchmark studied by most prior work in visual adaptation [4, 25, 27, 38], using the same setting of adapting ViT-Base [12] with 86 million parameters. However, as we believe that the goal of efficient adaptation is ultimately to adapt large-scale, pretrained foundation models, we demonstrate how our method does indeed scale to large model sizes and datasets where efficiency is paramount. Specifically, we show how we can efficiently adapt ViT-e [6], a vision transformer with 4 billion parameters, on the demanding video classification task on a V100 GPU without employing any intricate model parallelism. Moreover, we outperform prior works based on fully-finetuning smaller backbones, and a previous adaptation approach [45] for video which only scales to a 1 billion parameter backbone with the same computational budget, all whilst also training significantly faster.

In summary, we propose Low-Rank Side Adaptation (LoSA), an adaptation method which is not only parameter-efficient, but also training-time and -memory efficient. We can scale our method to larger backbones than previously possible (*i.e*. ViViT-e with 4 billion parameters for video classification). We benchmark our and prior methods on

multiple efficiency indicators beyond parameters, showing that we achieve superior accuracy-efficiency trade-offs across learned parameters, training-time and -memory. To further research in this area, we will release code for our method, and related baselines, upon acceptance.

2. Related Work

As large, pretrained models have become more prevalent, there has been a growing literature in efficient methods to adapt them to downstream tasks [11, 37]. The vast majority of these approaches are parameter-efficient finetuning methods which train only a small fraction of parameters in the pretrained model. Parameter-efficient finetuning (PEFT) methods can broadly be categorised into "additive" methods, which add a few, new parameters to pretrained model where the original weights are frozen, and "selective" methods which finetune a small subset of the original network's weights [37]. Additive methods broadly consist of adapters [4, 20, 31, 38, 45, 47, 48], which insert new learnable layers and parameters into an existing network, and prompt-tuning [25, 34, 36], which adds learnable prompt tokens to the inputs [34] or within multiple layers [25, 36] of a transformer network. Selective methods on the other hand, finetune specific parameters of the original network. For example, BitFit [52] trains only the bias terms of the network, whereas other works finetune only parameters of the attention or MLP-layers [16, 51]. Another common theme is to reparameterise learned parameters, for example as the product of low-rank matrices [21] or as Kronecker products [13] to reduce the number of learned parameters. Numerous approaches also combine the above ideas: For example, MAM Adapters [16] combine both adapters and prompt tuning, whilst UniPelt [44] makes use of low-rank weight paramterisation too.

Although the aforementioned approaches are designed with parameter-efficiency in mind, they are not necessarily computationally cheap to train. As these methods either insert or train parameters within the original pretrained model, in order to train them, we still need to backpropagate gradients through the original network (Fig. 2). This incurs substantial floating point operations (FLOPs) in the backward pass. Moreover, to efficiently compute gradients in the forward pass, we typically need to cache activations during the forward pass [15, 29]. Prompt-tuning [25, 36] on the other hand adds even further computational overhead during both training and inference, as it inserts new tokens and thus increases the sequence length of the transformers. Our proposed method, in contrast, keeps the entire backbone frozen and learns a parallel network that operates on activations from this backbone. As a result, we do not need to backpropagate through the backbone, and therefore save significant computation during training.

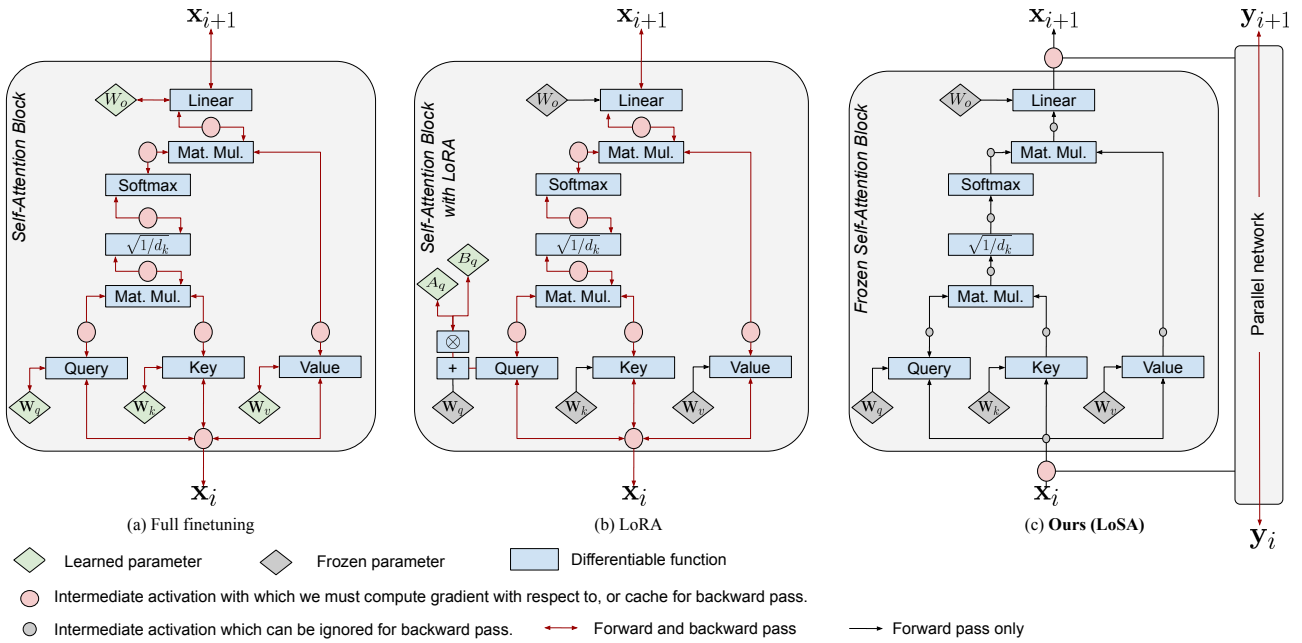We note that parameter-efficient methods that backprop-

Figure 2. Computational graphs of Self-Attention (SA) for the backward pass in the backbone network when performing (a) Full finetuning which requires caching or recomputing gradients with respect to large activation tensors (red ovals), which is memory- and compute-intensive (b) LoRA [21] and (c) our proposed LoSA. Although LoRA (b) trains only a small number of parameters per SA block – namely A_q and B_q, it still requires backpropagating gradients throughout the entire backbone. Thus the computational graph is quite similar to full finetuning ($\otimes$ denotes the multiplication of two-low rank matrices). Our method (c), in contrast, freezes the backbone completely and does not need to backpropagate through it at all, which results in significant reductions in training time and memory.

agate gradients through the backbone, do still save memory over full-finetuning, as only a fraction of the optimiser state needs to be maintained. For example, Adam [33] maintain two momentum terms per parameter, and these do not need to be kept for any of the frozen parameters in the backbone during adaptation. As our method is still parameter-efficient, we also benefit from this property, and achieve further memory reduction during training as we need to cache fewer activations from the backbone for the backward pass.

We note that some prior works have also developed training time- and memory-efficient adaptation algorithms based on training parallel networks on top of frozen backbone features in natural language [14, 43, 49]. $\mathcal{Y}$-Tuning [43] and LeTS [14] obtained substantial time and memory reductions using this technique, However, their methods were developed for natural language tasks, and do not apply directly to the vision problems studied in this paper, as [43] is based on adapting the textual label space, and [14] relies on bidirectional LSTMs [22] which are not typically used in vision. LST [49] is applicable to vision tasks. However, their results were not competitive in terms of accuracy-to-parameters trade-offs compared to existing PEFT methods. We carefully develop our network architecture such that we outperform prior approaches not only in terms of training-time and memory but also in terms of learned parameters.

Finally, we note that evaluating the efficiency of machine learning models is a complex subject [8, 19, 30, 46]. There are numerous efficiency indicators (*e.g.* the number of parameters, FLOPs, runtime, memory consumption), and

there is often poor correlation between metrics among different techniques (for example prompt-tuning is parameter-efficient, but not efficient in terms of the other aforementioned metrics). As selective reporting of metrics can obscure the true efficiency of an approach [7, 8, 30], we strive to be as thorough as possible when measuring the efficiency of our approach and baselines, and thus go beyond reporting only parameter counts as done by prior works in this field.

3. Proposed Approach

3.1. Background

Consider a neural network with L layers, where the output of the ith layer, $\mathbf{x}_i$, is a function of the previous output, $\mathbf{x}_{i-1}$, and learned parameters, θ_i:

$$\mathbf{x}_i = f_i(\mathbf{x}_{i-1}, \theta_i). \tag{1}$$

The input to the network is $\mathbf{x}_0$. The standard way to learn the network parameters, $\theta = \{\theta_1, \ldots, \theta_L\}$ is to minimise a scalar-valued loss, $\mathcal{L}$, applied to the network's output via optimisers based on stochastic gradient descent. Concretely, the gradient for parameter, θ_i is computed via the chain rule during backpropagation as follows:

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \underbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}} \cdots \frac{\partial \mathbf{x}_{i+1}}{\partial \mathbf{x}_i}}_{\text{gradient w.r.t. previous activations}} \frac{\partial \mathbf{x}_i}{\partial \theta_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_i} \frac{\partial \mathbf{x}_i}{\partial \theta_i}. \tag{2}$$

By applying the chain rule, we observe that the gradient with respect to a parameter θ_i also depends on the gradi-

ent with respect to the activations from subsequent layers (note that this analysis can easily be extended to arbitrary direct acyclic graphs [15, 29]). In order to compute these gradients, we need to perform a "backward pass" which has similar computational complexity as the "forward pass" when computing the loss, $\mathcal{L}$, and typically requires caching the outputs of each operation, or network activations, $\mathbf{x}_i$, for efficient calculation [15, 29]. Note that in practice, a neural network layer, such as self-attention in fact contains multiple operations within its computational graph. For example, as shown in Fig. 2a, a self-attention layer requires computing 8 intermediate gradient terms with respect to activations, $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_i}$, in order to compute gradients with respect to its query and value parameters.

Why parameter efficient is not always sufficient When training, there is substantial computational overhead in terms of the floating point operations (FLOPs) required to compute the gradient terms, $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_i}$, as well as memory overhead from the activations that typically need to be cached during the forward pass in order to compute them. Parameter-efficient tuning method such as adaptors [20], prompt-tuning [25] and LoRA [21] learn only a few parameters, meaning that computation can be saved by not having to compute $\frac{\partial \mathcal{L}}{\partial \theta_i}$ for the $\theta_i \in \theta$ which are frozen. However, the majority of the computation in the backward pass is in fact from computing gradients with respect to activations, $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_i}$, which must still be computed as (Fig. 2).

For example, prompt-tuning [25], which adds learnable prompt tokens at the start of the network, still requires computing $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_i}$ with respect to each operation in the backbone, meaning that it still consumes significant amounts of FLOPs and memory during training. A similar analysis holds for applying LoRA or other adaptors [20] at each layer of the network. We can, however, substantially reduce the compute cost of these adaptors by only applying them to the last k layers of the network, since we do not need to cache activations, or perform backpropagation, before these layers.

Based on our observation that the FLOP requirements (and therefore training time) and memory requirements of most parameter-efficient finetuning methods is still quite substantial as they must backpropagate through the model being adapted, we instead propose an alternate approach (Figs. 2 and 3) based on learning a *parallel* network on frozen activations from the backbone model next.

3.2. Low-rank Side Adaptation

Motivated by our observations from the previous section, we design a *parallel* network that does not require back-propagating gradients through the backbone in order to train it. Our proposed method can be thought of as a lightweight, parallel network to the main backbone, B (typically a large pretrained transformer), that refines the backbone output to produce more accurate representations.
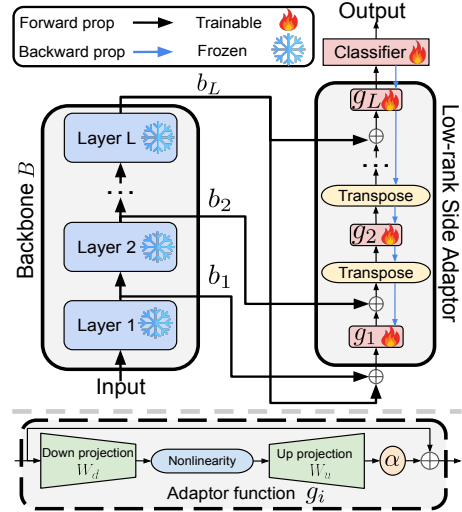

Figure 3. Overview of our approach. Top: We learn a parallel side network which iteratively refines the features obtained from a frozen backbone, B. Bottom: Our adaptation function consists of low-rank mixer modules, which allow for achieving high accuracy on a wide range of downstream tasks without sacrificing efficiency.

Concretely, given a neural network backbone, B, consisting of L layers, and therefore L intermediate outputs, $\mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_L$, each consisting of n tokens with a hidden dimensionality of d, $\mathbf{b}_i \in \mathbb{R}^{n \times d}$, we learn parallel adaptor functions, g, which operate on these intermediate outputs to refine them. We denote the activations of our parallel adaptor as $\mathbf{y}_i$ where i denotes the layer index, and

$$\mathbf{y}_i = g_i(\mathbf{b}_i + \mathbf{y}_{i-1}) + \mathbf{y}_{i-1}. \tag{3}$$

As shown in Fig. 3, we freeze the backbone, B, and do not backpropagate gradients to it. Moreover, the initial input to our adaptor, $\mathbf{y}_0$ is in fact the output of the backbone, $\mathbf{b}_L$. Together with the residual connection to the output of the previous layer, this ensures that when we train our model, our adaptor acts as an identity function with the original backbone output being retained. The rationale is that we view our parallel adaptor, g, as a function that refines the features representations from the original backbone, using the original backbone features to do so.

We design our adaptor function, $g : \mathbb{R}^d \to \mathbb{R}^d$, to be both accurate on the downstream task and efficient (in terms of parameters and floating point operations). To this end, g is a low-rank factorisation of a projection [21], with a down-projection, $\mathbf{W}_d : \mathbb{R}^d \to \mathbb{R}^r$, a GeLU non-linearity [18], and then another up-projection, $\mathbf{W}_u : \mathbb{R}^r \to \mathbb{R}^d$ where $r \ll d$. As in [17, 21, 41], we also learn a scaling term, α, meaning that our adaptor function can be expressed as

$$g(\mathbf{x}) = \alpha \mathbf{W}_u \text{GeLU}(\mathbf{W}_d \mathbf{x}). \tag{4}$$

Note that each parallel adaptor layers contains $2rd + 1$ parameters due to the low-rank decomposition of the weight

matrices [21]. Using additional biases in the up- and down-projections would still only add another $r + d$ parameters.

Modelling token interactions As our adaptor function, Eq. 4, only consists of projections and a point-wise non-linearity, it only operates on the hidden dimension of the backbone features, and does not model interactions along the spatial and temporal (in the case of videos) axes of the input data. Therefore, we take inspiration from MLP-Mixer [50], and alternate between applying our adaptor function along the channel- and token-dimensions respectively. We denote these operations as:

$$g_i(\mathbf{x}_i) = \begin{cases} g^{\text{token}}(\mathbf{x}_i) & \text{if} \quad i \mod 2 \neq 0, \\ g^{\text{channel}}(\mathbf{x}_i) & \text{otherwise.} \end{cases} \quad (5)$$

i is the backbone layer index, thus even layers correspond to "token mixing" and odd layers to "channel mixing" [50].

Extending to video When the input is a video, we assume that the backbone features have spatiotemporal dimensions, that is $\mathbf{b}_i \in \mathbb{R}^{n_t \cdot n_s \times d}$ where n_s and n_t denote the spatial- and temporal-dimensions. In this case, we factorise the token dimension further into separate spatial- and temporal-axes, that is $\mathbf{b}_i \in \mathbb{R}^{n_t \times n_s \times d}$, and apply our adaptor function separately along each dimension before summing them up. Therefore, our adaptor function for each even layer is

$$g^{\text{token}}(\mathbf{x}_i) = g^{\text{spatial}}(\mathbf{x}_i) + g^{\text{temporal}}(\mathbf{x}_i). \quad (6)$$

3.3. Discussion

The fact that we keep the entire original backbone, B, frozen, and train a parallel subnetwork, means that the storage requirements of our adapted models is small, as we only need to store the parameters of our side network for each task. And moreover, this makes deploying numerous backbones adapted from the same backbone, B, simple and efficient in terms of storage. Other adaptor-based methods [20, 21, 48] share this desirable property, although they require backpropagating through the backbone to train the model, and hence are not as efficient during training as our proposed method. Moreover, in contrast to these prior works, our method does not require changing the internal model architecture of the backbone at all, which makes its practical implementation straightforward.

We note that some prior works have trained lightweight networks in parallel to frozen backbones for efficient adaptation in natural language processing [14, 43, 49]. Ladder Side Tuning (LST) [49] is the most related to our approach. However, LST was not competitive in terms of accuracy-vs-parameter trade-offs to approaches such as LoRA [21]. We show experimentally in Sec. 4.2 and 4.4 that the architectural improvements that we make over LST substantially improve the accuracy and efficiency of our approach, such that it outperforms prior works on accuracy-vs-parameter

trade-offs, and substantially outperforms them in terms of training-time and -memory. Our improvements over LST include using our low-rank mixer, instead of a regular transformer as used by LST and using the output of the backbone as the input to the parallel network, g. The fact that we use our proposed low-rank mixer means that our parallel network g can efficiently operate at the same hidden dimension, d, as the backbone, and does not require down-projecting backbone activations to the latent space of the transformer as in LST [49], saving compute.

Finally, we note that concurrent work, HST [39], used a side network for dense prediction. However, they use transformer layers in the side network and also employ prompt tuning in the backbone, meaning that they have to back-propagate the gradients through both the side network and the backbone. Therefore, HST [39] does not provide benefits in terms of training-memory and speed like our method.

4. Experimental Evaluation

After describing our experimental setup, we present experiments on image classification in Sec. 4.2, video classification in Sec. 4.3, and then conclude by evaluating each of our modelling choices in Sec. 4.4.

4.1. Experimental Setup

Pretrained models We experimentally evaluate our method by adapting vision transformers of various backbone sizes for both image and video classification. We note that ViT-Base [12] (consisting of 86 million parameters) pretrained on ImageNet-21K [10] is the most commonly used backbone across prior works [4, 25, 27, 38]. However, as a motivation of efficient adaptation is to adapt large, pretrained models, we also consider much larger vision transformers, namely ViT-H (632 million parameters), ViT-g (1 billion), ViT-G (1.8 billion) and ViT-e (4 billion) [6, 54].

Efficiency metrics In addition to accuracy on the final task, as our efficiency metrics, we report the training time (measured in the number of images per second per core on a V100 GPU), training memory consumption in Gigabytes (GB), number of FLOPs during inference, and number of trainable parameters. We find that the inference time is largely determined by the size of the pretrained backbone, and not the adaptation method (except for prompt-tuning where it increases substantially). Therefore, we only include it in the supplement for completeness. When reporting the number of learnable parameters, we do not report the parameters in the final classification layer for clarity. The reason being that for datasets with a large number of classes, *e.g.* iNaturalist2021 has 10 000 classes, the number of learned parameters is dominated by the final layer, and not the architecture of the various adaptation methods.

Implementation details We include exhaustive details of our training hyperparameters in the supplement. We will

	Param↓ (10^6)	Natural							Specialized				Structured								Avg Natural	Avg Specialized	Avg 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[25, 27]	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	75.9	83.4	47.6	68.9
Linear[25, 27]	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	69.1	77.1	26.9	57.6
Efficient adaptation methods																								
BitFit[28, 52]	0.10	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	73.3	78.3	44.1	65.2
VPT[25, 27]	0.53	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	78.5	82.4	55.0	72.0
RS-Bypass. [26]	0.42	64.5	88.8	73.2	99.4	90.6	63.5	57.2	85.5	95.2	82.4	75.2	70.4	61.0	40.2	66.8	79.2	52.6	26.0	49.3	76.7	84.6	55.7	72.3
Adapter [20, 27]	0.16	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	79.0	84.1	58.5	73.9
LoRA[21, 27]	0.29	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	79.5	84.6	59.8	74.5
AdaptFormer[4, 27]	0.16	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	80.6	84.9	58.8	74.7
NOAH [27, 55]	0.36	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	80.3	84.9	61.3	75.5
FacT-TK [28]	0.06	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	80.6	85.3	60.7	75.6
RS [26]	0.55	75.2	92.7	71.9	99.3	91.9	86.7	58.5	86.7	95.6	85.0	74.6	80.2	63.6	50.6	80.2	85.4	55.7	31.9	42.0	82.3	85.5	61.2	76.3
Convpass [27]	0.33	72.3	91.2	72.2	99.2	90.9	91.3	54.9	84.2	96.1	85.3	75.6	82.3	67.9	51.3	80.0	85.9	53.1	36.4	44.4	81.7	85.3	62.7	76.6
HST [39]	0.78	76.7	94.1	74.8	99.6	91.1	91.2	52.3	87.1	96.3	88.6	76.5	85.4	63.7	52.9	81.7	87.2	56.8	35.8	52.1	82.8	87.1	64.5	78.1
LoSA, $r = 16$	0.19	82.5	92.8	76.1	99.7	90.5	82.0	55.8	86.6	97.1	87.0	76.7	81.5	62.3	48.6	82.1	94.2	61.7	47.9	45.6	82.8	86.9	65.5	78.4
LoSA, $r = 8$	0.10	82.2	92.7	76.7	99.7	90.7	81.0	55.4	86.9	97.1	87.4	76.5	79.9	61.8	48.6	82.4	92.3	61.1	48.7	47.3	82.6	87.0	65.3	78.3
LoSA, $r = 4$	0.05	82.7	93.0	76.2	99.7	89.8	80.0	56.1	86.3	96.7	86.7	76.3	78.8	61.4	48.0	82.6	91.7	58.4	46.9	47.6	82.5	86.5	64.4	77.8

Table 1. Comparison to state-of-the-art parameter-efficient finetuning methods on VTAB-1K [53]. Following standard practice, the final "Average" is the average of three preceding groupwise averages. Parameters denotes the number of learnable parameters excluding the final classification layer, as the number of parameters in this final layer depend on the number of classes, which varies between 2 and 397. Each variant of our model, which we obtain by varying the rank r of our Low-rank Mixer Block, achieves better accuracy-parameter trade-offs than previous approaches, when using the same ViT-B backbone. **Best results** are bolded, and second-best underlined.

also release code for our method, and the various baselines that we have implemented upon acceptance.

4.2. Experiments on Images

The most common setting for studying parameter-efficient finetuning is on the VTAB [53] benchmark, by adapting ViT-B pretrained on ImageNet-21K. We first compare our method to prior works in this scenario. However, as a key motivation for efficient adaptation is to use large-scale models, we perform experiments on more challenging, large-scale classification datasets including iNaturalist2018 [23] and 2021 [24] and Places365 [56] with ViT-g. Note that we do not consider ImageNet, as we found performance to be saturated and not representative for comparing different methods. We nevertheless include it in the supplement.

VTAB benchmark VTAB [53] is the most common benchmark in the vision community for evaluating parameter-efficient finetuning methods. It consists of 19 datasets spanning a range of image domains. The benchmark is designed to evaluate transfer-learning performance, and each dataset contains exactly 1000 training examples.

Prior works have evaluated on this benchmark, using a ViT-Base backbone and benchmarking efficiency through the number of trained parameters. We compare to these works, in the same setting in Tab. 1. We observe that we achieve state-of-the-art trade-offs between the accuracy and the number of trainable parameters. We can control the number of parameters in our model, by varying r, the size

of our hidden layer within our side network (Sec. 3.2). With $r = 16$, we obtain a parameter-efficient model which outperforms prior works on this benchmark using the same ViT-Base model pretrained on ImageNet-21K. In particular, we outperform concurrent work, HST [39], whilst using 4 times fewer parameters. And when we decrease r to $r = 4$ and $r = 8$, we continue to achieve superior parameter-accuracy trade-offs, with $r = 4$ having the lowest number of parameters and still achieving the second-highest accuracy.

Table 1 therefore shows that our proposed LoSA method outperforms the state-of-the-art with respect to accuracy-parameter trade-offs. Next, we analyse our method on additional efficiency indicators at a larger scale.

Large-scale image classification We consider more challenging, larger-scale image classification datasets in the form of iNaturalist 2018 [23] (438K training examples, 8142 classes), iNaturalist 2021 [24] (2.7M training examples, 10000 classes) and Places365 [56] (1.8M training examples, 365 classes). The iNaturalist datasets also have a long-tail, increasing their difficulty.

Figure 4 evaluates our method and baselines across a wide range of efficiency indicators. We compare to LoRA [21], BitFit [52], prompt tuning [25] and LST [49] as representative efficient adaptation algorithms. We also compare to full-finetuning, linear probing, finetuning the last one or two layers, and also finetuning only the attention or MLP layers [51]. Extensive implementation details of these baselines are included in the supplement. We visualise
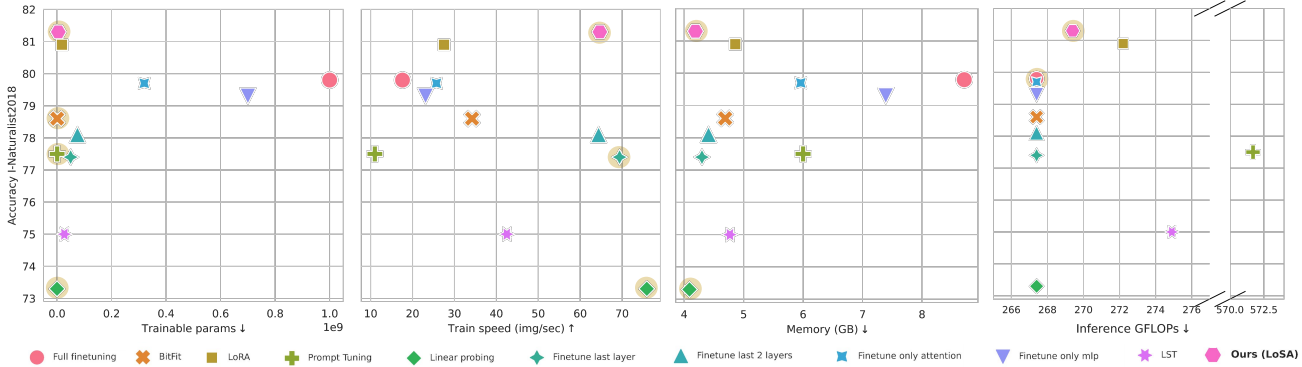
Figure 4. Comparison of trade-offs of accuracy with respect to learned parameters, training memory, inference GFLOPs and training speed. Our approach, LoSA, is consistently on the Pareto frontier (denoted by shaded yellow circles), as there is no method that is both more accurate and more efficient than it, across multiple efficiency metrics. Results are on the iNaturalist 2018 dataset, using a ViT-g backbone with 1 billion frozen parameters.

the Pareto frontier across each of these methods with yellow, shaded circles, to show the relative trade-offs in each efficiency metric.

Consistent with our experiments on VTAB (Tab. 1), we show superior trade-offs in terms of accuracy and parameters. We also outperform full-finetuning, and this suggests that large backbones are overparameterised, and can overfit when fully-finetuned. Therefore, efficiently adapting a subset of parameters may lead to higher performance. Numerous works across domains [4, 40, 45, 49] have also observed that efficient finetuning methods can outperform full-finetuning in certain tasks.

Figure 4 also shows that our method uses less training memory than other baselines except linear probing. As described in Sec. 3, these improvements stem from the fact that we do not need to backpropagate gradients through the backbone. However, we also improve upon the accuracy of linear probing by 8 points. We note similar observations when comparing the training speed to overall accuracy for the same reason that we do not need to backpropagate through the backbone. Notably, we are 2.35 times faster than LoRA whilst achieving 0.4 points higher accuracy.

Overall, our method is Pareto-optimal for learned parmeters, training speed and training memory consumption, emphasising the advantages of our approach. The inference GFLOPs is largely unchanged among the different approaches, as they are all dominated by the cost of the ViT-g backbone which is constant across them. Notably, prompt tuning uses substantially more GFLOPs and training time as in order to achieve good accuracy, we needed to use 16 prompts in the first 24 layers of the network.

We show full tables with the complete data for Fig. 4 in the supplement, as well as accuracies on iNaturalist 2021 (Fig. 1) and Places365 where we observe consistent trends.

4.3. Experiments on Video

We further demonstrate the scalabilty of our method by performing video classification experiments on Kinetics

Method	Backbone	Params ↓ (10^6)	Memory ↓ (GB)	Train ↑ (clip/second)	Accuracy ↑
Full finetuning [2]	ViViT-H	635	14.34	0.61	84.9
	ViViT-g	1039	–	–	OOM
	ViViT-G	1879	–	–	OOM
	ViViT-e	3879	–	–	OOM
ST Adapter [45]	ViViT-H	127	9.17	0.66	80.9
	ViViT-g	174	11.47	0.50	83.9
	ViViT-G	247	–	–	OOM
	ViViT-e	310	–	–	OOM
Linear Probing	ViViT-H	0	**2.88**	**2.47**	75.0
	ViViT-g	0	3.62	1.94	80.7
	ViViT-G	0	5.24	1.37	81.1
	ViViT-e	0	9.05	0.92	83.1
Ours, LoSA	ViViT-H	3.61	3.41	2.33	80.0
	ViViT-g	4.78	4.92	1.73	84.0
	ViViT-G	6.53	7.72	1.22	84.4
	ViViT-e	8.03	12.54	0.83	**86.2**

Table 2. Video classification results on Kinetics 400, using Nvidia V100 GPUs with 16 GB of RAM. Our proposed method can scale to a frozen ViViT-e backbone, as it does not backpropagate gradients through the backbone. And thanks to the larger backbone size, can outperform fully-finetuning a smaller ViViT-H model.

400 [32]. We use an unfactorised ViViT model [2], which allows us to use the same pretrained image checkpoints from our previous experiments.

Video classification requires processing substantially more tokens: each model in Tab. 2 is trained with 32 frames, a frame resolution of 224, and a tubelet size of $14 \times 14 \times 2$, which corresponds to 4096 tokens. As shown in Tab. 2 it is only possible to fully finetune a ViViT-H model with 636 million parameters. We also compare to ST-Adapter [45], an existing work for adapting video models, based on the authors' public code as detailed in the suppement, and observe that it does not scale beyond ViViT-g with 1 billion parameters. However, our approach, as it does not backpropagate gradients through the backbone unlike the previous two approaches, can scale up to ViViT-e with 4 billion frozen parameters, using Nvidia V100 GPUs with 16GB of RAM. Moreover, we achieve higher accuracy than either ST-Adapter or full finetuning (86.2 vs 84.9) whilst also training faster and using less memory. We note, however, that although our model is faster to train than these two baselines, our final ViViT-e model is still slower at infer-

Adaptor function, g	Learned ↓ Params (M)	Train Memory ↓ (GB)	Train ↑ (img/sec)	GFLOPs ↓	Acc ↑
Transformer	27	4.77	42.5	274.9	75.0
Low-rank MLP	7.7	4.21	**65.7**	269.4	80.7
Low-rank Mixer (LoSA)	**4.8**	**4.19**	64.8	269.4	**81.3**

Table 3. Ablation of our adaptor function, g. LST [49] uses a regular transformer which we find to perform the worst, whilst our Low-rank Mixer achieves the best accuracy-efficiency trade-off.

Backbone activation	Learned ↓ Params (M)	Train Memory↓ (GB)	Train ↑ (img/sec)	GFLOPs ↓	Acc ↑
MLP	4.8	4.19	64.8	269.4	77.7
Multihead attention	4.8	4.19	64.8	269.4	79.1
MLP and Multihead attention	9.1	4.29	51.8	271.2	81.0
Encoder block (LoSA)	4.8	4.19	64.8	269.4	**81.3**

Table 4. The effect of different activations extracted from the backbone. The output of the transformer encoder block performs the best as it is the sum of the MLP- and self-attention-block outputs.

ence time due to its larger backbone.

Linear probing is the only baseline that allows us to scale to the ViViT-e backbone. However, its performance is lower as well, and consistent with our previous experiments on images. Note that whilst it is possible in theory to use model parallelism to split a large model across multiple accelerators, we have not implemented this for any approach in Tab. 2 as it is not common practice.

4.4. Ablation Study

Finally, we analyse the effect of our various modelling choices. Unless otherwise stated, we perform experiments on iNaturalist 2018 [23] using a ViT-g backbone.

Adaptor function g Table 3 considers different alternatives for our adaptor function, g. Our first baseline is to use a regular transformer, with hidden dimension $r = 48$, as also done in LST [49]. However, as shown in Tab. 3, this performs the worst, both in terms of accuracy and efficiency. Although we could use smaller hidden dimensions to improve efficiency, we assume fewer parameters would not increase accuracy, leaving a clear gap to a Low-rank MLP (Eq. 4). Alternating between "channel mixing" and "token mixing" [50] with our proposed Low-rank Mixer (Eq. 5) achieves the best accuracy. Moreover, this also improves efficiency metrics, because for images, the number of tokens (256) is less than the hidden size ($d = 1408$) for our backbone. Therefore, every second mixer block has fewer parameters and GFLOPs compared to the Low-rank MLP.

Backbone activation   Table 4 analyses the various activations that we can extract from our backbone to pass to the adaptor function, g. We find that the final output of the transformer encoder block, performs better than the alternatives: output of the MLP, output of the Multihead Self-Attention (MHSA) block, or in fact, the combination of both. When we use outputs of both MHSA- and MLP-layers, our side network effectively has double the depth as we apply g to each output in sequence. Using the final encoder block likely performs the best, as via the use

Layers in side network	Learned ↓ Params (M)	Train Memory ↓ (GB)	Train ↑ (img/sec)	GFLOPs ↓	Acc ↑
Linear Probing (0)	0	4.09	75.7	267.4	73.3
1	0.184	4.09	75.2	267.5	77.8
2	0.217	4.10	74.8	267.5	78.0
4	0.431	4.10	74.2	267.6	79.3
8	0.861	4.11	73.2	267.8	80.0
40 (All)	4.298	4.19	64.8	269.3	80.6

Table 5. Effect of the number of layers in our side network. As we use a ViT-g backbone, there are 40 layers in the backbone, and we uniformly sample backbone activations to pass to our side network. No layers corresponds to linear probing, which performs substantially worse than using just a single side adaptor layer.

Side network input, $\mathbf{y}_0$	Learned ↓ Param (M)	Train Memory ↓ (GB)	Train ↑ (img/sec)	GFLOPs ↓	Acc ↑
Backbone input	4.8	4.06	64.8	269.4	60.9
Backbone output (LoSA)	4.8	4.06	64.8	269.4	**61.3**

Table 6. Ablation of inputs to our side network on Places365. Using the backbone outputs provides a small improvement compared to using the backbone inputs, as done in LST [49].

of residual connections, it is the sum of both MHSA- and MLP-blocks [12], and thus contains information from both.

Number of layers in side network   Another method of controlling the accuracy-efficiency trade-off of our proposed method is to vary the number of layers in our side network. Instead of operating on activations from each layer of the backbone, we select k evenly-spaced layers instead in Tab. 5. Using no layers from our side network results in linear probing, and we observe that adding just a single side adaptor layer to the end of the backbone improves accuracy significantly by 4.5 points. This suggests that our side network is able to learn complementary information to the backbone. Adding further layers gives small and consistent improvements. In our previous experiments, we used all layers from the backbone to obtain higher accuracy.

Side network inputs   Finally, Tab. 6 considers the effect of using either the output of the backbone, $\mathbf{b}_L$, as the input to our side network, $\mathbf{y}_0$, or rather using the input to the backbone itself (*i.e.* tokenised image patches, $\mathbf{b}_0$). The interpretation of $\mathbf{b}_L$ is that the side network refines the outputs of the backbone, given further intermediate activations from the backbone. The interpretation of $\mathbf{b}_0$ is that the side network iteratively refines backbone features of the ith layer given activations from layers before it, and was also used in LST [49]. We find that the former approach improves results slightly, and thus used it for all experiments.

5. Conclusion and Future Work

We have introduced Low-rank Side Adaptation (LoSA), an adaptation method that is efficient not only in terms of parameters, but also training-time and memory, since we do not have to backpropagate gradients through the backbone. We achieve state-of-the-art results on VTAB, and show that we can scale our method, unlike previous works, to ViT-e (4 billion parameters) without model parallelism. Future work is to extend our method to more complex vision tasks.

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. In *arXiv preprint arXiv:2204.14198*, 2022. 1

[2] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A Video Vision Transformer. In *ICCV*, 2021. 7

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, et al. Language models are few-shot learners. In *NeurIPS*, 2020. 1

[4] 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. 2, 5, 6, 7

[5] Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, et al. Pali-x: On scaling up a multilingual vision and language model. In *arXiv preprint arXiv:2305.18565*, 2023. 1

[6] Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, et al. Pali: A jointly-scaled multilingual language-image model. In *ICLR*, 2023. 1, 2, 5

[7] George E Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, et al. Benchmarking neural network training algorithms. In *arXiv preprint arXiv:2306.07179*, 2023. 3

[8] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. In *ICLR*, 2022. 2, 3

[9] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *ICML*, 2023. 1

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

[11] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. In *arXiv preprint arXiv:2203.06904*, 2022. 2

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 2, 5, 8

[13] Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. Krona: Parameter efficient tuning with kronecker adapter. In *arXiv preprint arXiv:2212.10650*, 2022. 2

[14] Cheng Fu, Hanxian Huang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. Learn-to-share: A hardware-friendly transfer learning framework exploiting computation and parameter sharing. In *ICML*, 2021. 3, 5

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016. 2, 4

[16] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *ICLR*, 2022. 2

[17] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *ICLR*, 2022. 4

[18] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). In *arXiv preprint arXiv:1606.08415*, 2016. 4

[19] Sara Hooker. The hardware lottery. *Communications of the ACM*, 64(12), 2021. 3

[20] 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*, 2019. 1, 2, 4, 5, 6

[21] 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. 1, 2, 3, 4, 5, 6

[22] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. In *arXiv preprint arXiv:1508.01991*, 2015. 3

[23] iNaturalist 2018 competition dataset. iNaturalist 2018 competition dataset. https://github.com/visipedia/inat_comp/tree/master/2018, 2018. 6, 8

[24] iNaturalist 2021 competition dataset. iNaturalist 2021 competition dataset. https://github.com/visipedia/inat_comp/tree/master/2021, 2021. 6

[25] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *ECCV*, 2022. 1, 2, 4, 5, 6

[26] Zeyinzi Jiang, Chaojie Mao, Ziyuan Huang, Ao Ma, Yiliang Lv, Yujun Shen, Deli Zhao, and Jingren Zhou. Res-tuning: A flexible and efficient tuning paradigm via unbinding tuner from backbone. In *NeurIPS*, 2023. 6

[27] Shibo Jie and Zhi-Hong Deng. Convolutional bypasses are better vision transformer adapters. In *arXiv preprint arXiv:2207.07039*, 2022. 2, 5, 6

[28] Shibo Jie and Zhi-Hong Deng. Fact: Factor-tuning for lightweight adaptation on vision transformer. In *AAAI*, 2023. 6

[29] Matthew James Johnson. *Bayesian Time Series Models and Scalable Inference*. PhD thesis, Massachusetts Institute of Technology, 2014. 2, 4

[30] Jean Kaddour, Oscar Key, Piotr Nawrot, Pasquale Minervini, and Matt J Kusner. No train no gain: Revisiting efficient training algorithms for transformer-based language models. In *arXiv preprint arXiv:2307.06440*, 2023. 2, 3

[31] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. In *NeurIPS*, 2021. 2

[32] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. In *arXiv preprint arXiv:1705.06950*, 2017. 7

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 3

[34] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021. 1, 2

[35] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023. 1

[36] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL*, 2021. 1, 2

[37] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. In *arXiv preprint arXiv:2303.15647*, 2023. 2

[38] Dongze Lian, Daquan Zhou, Jiashi Feng, and Xinchao Wang. Scaling & shifting your features: A new baseline for efficient model tuning. In *NeurIPS*, 2022. 2, 5

[39] Weifeng Lin, Ziheng Wu, Jiayu Chen, Wentao Yang, Mingxin Huang, Jun Huang, and Lianwen Jin. Hierarchical side-tuning for vision transformers. In *arXiv preprint arXiv:2310.05393*, 2023. 5, 6

[40] Ziyi Lin, Shijie Geng, Renrui Zhang, Peng Gao, Gerard de Melo, Xiaogang Wang, Jifeng Dai, Yu Qiao, and Hongsheng Li. Frozen clip models are efficient video learners. In *ECCV*, 2022. 7

[41] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *NeurIPS*, 2022. 4

[42] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023. 1

[43] Yitao Liu, Chen An, and Xipeng Qiu. Y-tuning: An efficient tuning paradigm for large-scale pre-trained models via label representation learning. In *arXiv preprint arXiv:2202.09817*, 2022. 3, 5

[44] Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Scott Yih, and Madian Khabsa. Unipelt: A unified framework for parameter-efficient language model tuning. In *ACL*, 2022. 2

[45] Junting Pan, Ziyi Lin, Xiatian Zhu, Jing Shao, and Hongsheng Li. St-adapter: Parameter-efficient image-to-video transfer learning. In *NeurIPS*, 2022. 2, 7

[46] Hao Peng, Qingqing Cao, Jesse Dodge, Matthew E Peters, Jared Fernandez, Tom Sherborne, Kyle Lo, Sam Skjonsberg, Emma Strubell, Darrell Plessas, et al. Efficiency pentathlon: A standardized arena for efficiency evaluation. In *arXiv preprint arXiv:2307.09701*, 2023. 3

[47] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. In *arXiv preprint arXiv:2005.00247*, 2020. 2

[48] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NeurIPS*, 2017. 1, 2, 5

[49] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. In *NeurIPS*, 2022. 3, 5, 6, 7, 8

[50] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, 2021. 5, 8

[51] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Jakob Verbeek, and Hervé Jégou. Three things everyone should know about vision transformers. In *ECCV*, 2022. 2, 6

[52] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *arXiv preprint arXiv:2106.10199*, 2021. 1, 2, 6

[53] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. In *arXiv preprint arXiv:1910.04867*, 2019. 6

[54] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *CVPR*, 2022. 5

[55] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. Neural prompt search. In *arXiv preprint arXiv:2206.04673*, 2022. 6

[56] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. In *PAMI*, 2017. 6