Parameter-efficient Fine-tuning for Vision Transformers

Xuehai He[1] **Chunyuan Li**[2] **Pengchuan Zhang**[2] **Jianwei Yang**[2] **Xin Eric Wang**[1]
[1]UC Santa Cruz, [2]Microsoft Research
{xhe89,xwang366}@ucsc.edu, {chunyl,penzhan,jianwyan}@microsoft.com

Abstract

In computer vision, it has achieved great success in adapting large-scale pretrained vision models (e.g., Vision Transformer) to downstream tasks via fine-tuning. Common approaches for fine-tuning either update all model parameters or leverage linear probes. In this paper, we aim to study parameter-efficient fine-tuning strategies for Vision Transformers on vision tasks. We formulate efficient fine-tuning as a subspace training problem and perform a comprehensive benchmarking over different efficient fine-tuning methods. We conduct an empirical study on each efficient fine-tuning method focusing on its performance alongside parameter cost. Furthermore, we also propose a parameter-efficient fine-tuning framework, which first selects submodules by measuring local intrinsic dimensions and then projects them into subspace for further decomposition via a novel Kronecker Adaptation method. We analyze and compare our method with a diverse set of baseline fine-tuning methods (including state-of-the-art methods for pretrained language models). Our method performs the best in terms of the tradeoff between accuracy and parameter efficiency across three commonly used image classification datasets.

1 Introduction

In the last few years, large-scale vision models and language models pretrained on ultra large-scale data have seen a great surge of interest with promising performance [1–4]. Meanwhile, aided by the rapid gains in hardware, their sizes keep growing rapidly. Currently, Vision Transformer (ViT) models with billions of parameters such as *ViT-Large* [5] have been released. It is expected that pretrained vision models with even larger orders of magnitude will emerge in the foreseeable future.

These large-scale pretrained models are powerful when transferred to downstream vision tasks. However, deploying many independent instances of fine-tuned models can also cause substantial storage and deployment costs and hinder the applicability of large-scale Vision Transformers to real-world problems. Motivated by this and the importance of parameter-efficient learning [6–10], we aim to study the parameter-efficient fine-tuning strategy for vision transformers. Conventional wisdom for transfer learning in our computer vision community is fine-tuning all model parameters or leveraging linear probes. However, performing full-model fine-tuning of pretrained Vision Transformers may incur both financial and environmental costs [11], and requires a high computational budget and becomes increasingly infeasible as the model size continuously grows. Another go-to strategy is performing linear probing by stacking an additional trainable multi-layer perceptron (MLP) layer in the end. It is parameter-efficient yet suboptimal in terms of performance. Ideally, we hope to design fine-tuning strategies that can achieve the best tradeoff between efficiency and effectiveness (see Fig. 1)—optimizing fine-tuning parameter-efficiency while allowing for the model to maintain the effectiveness of transfer learning on downstream vision tasks, especially the image classification.

To this end, an essential question to ask is, *what are the general guidelines one should adopt while fine-tuning large-scale pretrained vision models on the downstream datasets?* This work aims to

Preprint. Under review.

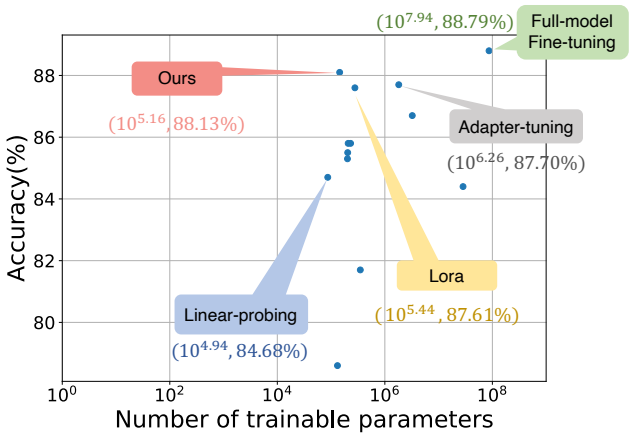

Figure 1: The tradeoff between accuracy and parameter efficiency of various fine-tuning methods. The accuracy and parameter efficiency are measured using the Vision Transformer (ViT-B-224/16) via supervised pretraining across the average of three image classification datasets. Our method places in the topleft corner and achieves the best tradeoff between accuracy and parameter efficiency.

answer the question by building a benchmark for fine-tuning Vision Transformers and proposing a better and more parameter-efficient fine-tuning method. We choose Vision Transformers as the pretrained vision models for efficient fine-tuning, which are representative mainstream state-of-the-art (SOTA) models on a wide range of downstream vision tasks. Specifically, we experiment with two types of Vision Transformers in the remainder of this paper: the one via Contrastive Language-Image Pretraining (also known as CLIP) [12], and the one via supervised pretraining (we refer to as Supervised ViT) [13]. In addition to Full-model Fine-tuning and linear probing, we re-implement several SOTA efficient fine-tuning methods [6, 14, 7, 8, 15] (originally proposed for pretrained language models) on vision tasks, and propose various baseline methods for comparison.

Aghajanyan *et al.* [16] show that pretrained language models have a low intrinsic dimension and can still learn efficiently despite a low-dimensional reparameterization. Motivated by this observation, we reformulate the task of efficient fine-tuning as a subspace training problem. Within this framework, we measure the *local intrinsic dimension* of each module in Vision Transformers, which empirically shows that the attention module dominates the training progress. Moreover, we propose a novel parameter-efficient fine-tuning strategy named *Kronecker Adaptation*, where during fine-tuning, pretrained weights are frozen, and only the updates to weights receive gradients. The weight updates are decomposed to a set of Kronecker products, with the *slow weights* [17] shared across layers and *fast weights* [17] further decomposed into low-rank matrices product to improve parameter efficiency. We apply Kronecker Adaptation to attention weights, and it achieves the best average accuracy among efficient fine-tuning methods while containing much less trainable parameters, e.g., 0.17% of all the model parameters in Supervised ViT.

The contributions of this paper are summarized below:

- We build a benchmark[1] for parameter-efficient fine-tuning of Vision Transformers on image classification tasks by introducing our new baseline methods and several state-of-the-art efficient fine-tuning strategies inspired from the NLP community. To our best knowledge, this is the first empirical study of efficient fine-tuning of Transformers to date that considers vision tasks.
- We formulate efficient fine-tuning as a subspace training problem and propose a novel framework to solve it. We first choose submodules by measuring the local intrinsic dimension. Then we employ the proposed Kronecker Adaptation method to decompose the weight updates of submodules for trainable parameter deduction.
- We experiment our approach on three commonly used image classification datasets: CIFAR-10 [18], CIFAR-100 [18], and SUN-397 [19]. The results demonstrate the effectiveness of

[1]We will release implementations of all the methods studied in this work.

our method, which achieves the best tradeoff between accuracy and parameter efficiency, as shown in Fig. 1.

2 Related Work

2.1 Transformers

Transformer [13] is a sequence-to-sequence architecture that makes heavy use of self-attention mechanisms to replace the recurrence and convolution operations. It is initially used for machine translation [20] and has shown outstanding performance in a wide range of natural language processing (NLP) tasks, including reading comprehension [21], question answering [22], vision-and-language tasks [23], etc. Since Radford *et al*. [1] first applied a stack of Transformer decoders to autoregressive language modeling, Transformer-based language models have dominated NLP, achieving the state-of-the-art in many tasks. A trend surfaced that using larger pretrained Transformers generally results in better performance. Pretraining huge Transformer models like GPT-3 [24], which is an autoregressive language model with 175 billion parameters, remains an active research direction.

At the same time, various types of Vision Transformers have gained attraction for computer vision tasks. Dosovitskiy *et al*. [5] demonstrated state-of-the-art performance on image classification datasets by large-scale pretraining and fine-tuning of a vanilla Vision Transformer. End-to-end models based on Vision Transformers have also shown prominence on object detection [25]. Recently, there are also other variants, including hierarchical Vision Transformers with varying resolutions and spatial embeddings [26, 27] been proposed. Beyond doubt, the recent progress of large Transformer-alike models posts great demands for developing efficient fine-tuning strategies.

2.2 Pretraining

In computer vision, fine-tuning pretrained models, e.g. [28, 29], has come to the forefront of deep learning techniques due to its success in downstream tasks that can substantially outperform tuning models with random initialization. For downstream tasks, such as image classification, image retrieval [30, 31] and object detection [32], the resulting models after pretraining can extract feature representations for input images, which will be further used in following task-specific models. Pretraining in NLP [4, 2], has also achieved state-of-the-art performances in many downstream tasks [33, 34]. Other pretrained models such as VLBERT [35, 36] demonstrate their effectiveness on downstream vision-and-language tasks, e.g., recent works on vision-and-language pretraining such as OSCAR [23] perform cross-modal alignment in their pretraining models. It is widely deemed that fine-tuning on downstream datasets after pretraining on general domain data [2, 1] could provide a substantial performance gain compared to training on task-specific data directly. In this paper, we will mainly focus on the parameter-efficient fine-tuning of pretrained ViT.

2.3 Efficient Fine-tuning in NLP

In the natural language processing domain, existing fine-tuning methods proposed by NLP researchers can be divided mainly into two categories depending on whether new trainable parameters are introduced. Specifically, one is to train a subset of the model parameters, where the most common approach is to use a linear probe on top of pretrained features [12]. The other alternative method surfaces by including new parameters in between the network [15, 14, 6, 7, 37, 38]. Nevertheless, two problems arise when adopting these methods for fine-tuning Vision Transformers. First, these methodologies have not been investigated in the computer vision scenario, and it remains unclear which one is preferred when adapted to vision tasks. It is furthermore uncertain if findings from NLP tasks (e.g., question answering [39], natural language understanding [40], etc.) will transfer to downstream vision applications. Second, considering these NLP fine-tuning methodologies are not architecture-agnostic, there remain many open questions pertaining to the behavior of these methods when they are applied to Vision Transformers. All these mentioned above comprise the purpose of our work.

3 Efficient Fine-tuning with Subspace Training

Given a large pretrained Vision Transformer model $\mathcal{M}$ with size $|\mathcal{M}|$. Our goal is to develop a parameter-efficient fine-tuning technique with trainable parameters θ of size $d \ll |\mathcal{M}|$, that can attain comparable performance with fine-tuning the whole model. Our ultimate goal is that one could achieve satisfactory results in both efficacy and efficiency without the hassle of fine-tuning the full model.

3.1 Subspace Training

A typical neural network contains numerous dense layers that perform matrix multiplication. The weight matrices in these layers can be full-rank. When adapting to a specific task, however, Aghajanyan *et al*. [16] show that the pretrained language models have a low *intrinsic dimension* and can still learn efficiently despite a low-dimensional reparameterization.

Drawing inspiration from their observation and study, we hypothesize that the updates to weights of Vision Transformers during each step in fine-tuning also have a low intrinsic rank when adapting to downstream tasks and propose our method based upon this hypothesis. The intuition behind our method is to perform subspace training on weight updates. In the standard training paradigm of neural network models, the gradient is first computed, and then a step in the entire parameter space D is taken. While in subspace training, we instead build a random d-dimensional parameter subspace from $\mathcal{M}$, where generally $d \ll |\mathcal{M}|$, and optimize directly in this subspace.

In fact, most current parameter-efficient NLP fine-tuning strategies perform subspace training. First, given a large pretrained language model $\mathcal{M}$ with size $|\mathcal{M}|$, they either select a submodule from $\mathcal{M}$ or add an additional module to $\mathcal{M}$. Denote the parameter vector from this module as $\Theta \in \mathbb{R}^D$. Then, they learn a projection $\mathcal{P}$ mapping Θ into a random d-dimensional subspace and perform fine-tuning in that subspace to reduce computational cost. Therefore the efficient fine-tuning problem is decomposed into two subproblems via subspace training: *how to choose these submodules* and *how to make the subspace projection*.

3.2 The Proposed Kronecker Adaptation

To answer the two fundamental questions of efficient fine-tuning, *how to choose these submodules* and *how to make the subspace projection*, we propose a novel framework that consists of two corresponding strategies. First, we attempt to select submodules by measuring the local intrinsic dimension. Second, we propose a Kronecker Adaptation method to perform the subspace projection on the selected modules by exploiting parameterized hypercomplex multiplication layers (PHM) [41]. Zhang *et al*. [41] shows that the PHM layer can reduce learnable parameters to $1/n$ compared with the fully-connected layer counterpart and achieve comparable performance. n is the user-defined hyperparameter. Built upon the success of PHM, which uses a sum of Kronecker products that generalize the vector outer products to higher dimensions in real space, we learn the projection matrix P as a sum of Kronecker products.

3.2.1 Local Intrinsic Dimension

To measure the local intrinsic dimension, we follow the similar definition of [42] and define Θ in subspace in the following way

$$\Theta = \Theta_0 + P\theta, \tag{1}$$

where $\Theta_0 \in \mathbb{R}^D$ is the initial parameter vector of Θ when the training begins, $P \in \mathbb{R}^{D \times d}$ is the projection matrix generated by the Fastfood transform [43], and $\theta \in \mathbb{R}^d$ is the parameter vector in the subspace. Subspace training proceeds by computing gradients with respect to θ and taking steps in that subspace. By performing experiments with gradually larger values of d, we can find the subspace dimension d_t at which the performance of the model $\mathcal{M}$ starts to reach a satisfactory solution. We refer to d_t the local intrinsic dimension of the module.

The module with the lowest local intrinsic dimension is selected. After selecting the attention weight matrices as submodules, we project them into subspace for fine-tuning with the proposed Kronecker Adaptation method. It trains attention weight matrices indirectly by optimizing decomposition matrices of the update of attention weight matrices. We compute the decomposition as the sum of Kronecker products while keeping the original matrices frozen to reduce the parameter cost.

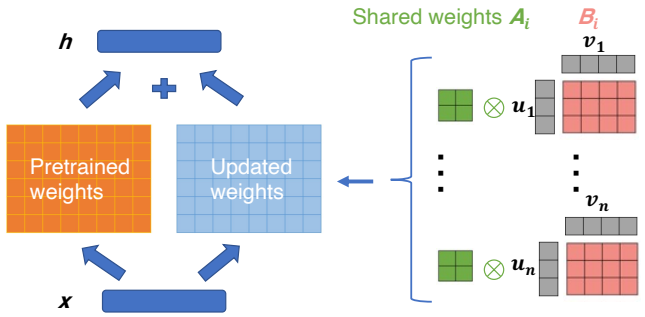

Figure 2: An illustration of Kronecker Adaptation. $\boldsymbol{A_i}$ denotes the shared weight matrix, with $i \in \{1, \ldots, n\}$. $\boldsymbol{B_i}$ is decomposed into two low-rank matrices $\boldsymbol{u_i}$ and $\boldsymbol{v_i}$. $\boldsymbol{h}$ is the output of the selected ViT submodule. $\boldsymbol{x}$ is the input to the ViT submodule. During fine-tuning process, only matrices $\boldsymbol{A_i}$, $\boldsymbol{u_i}$, and $\boldsymbol{v_i}$ receive gradients so that to improve parameter efficiency.

3.2.2 Kronecker Product

The Kronecker product between matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, denoted by $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{mp \times nq}$, is mathematically written in the following form:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{B} & \cdots & a_{mn}\boldsymbol{B}, \end{pmatrix} \tag{2}$$

where a_{ij} shows the element in the i-th row and j-th column of $\boldsymbol{A}$.

3.2.3 Kronecker Adaptation

Low-rank methods [16, 42] have demonstrated that strong performance can be achieved by optimizing a task in a low-rank subspace. Similarly, we hypothesize that for an update matrix $\Delta \boldsymbol{W} \in \mathbb{R}^{k \times d}$ in the ViT, it can also be effectively adapted by learning transformations in a low-rank subspace. We compute $\Delta \boldsymbol{W}$ as the sum of n Kronecker products as follows:

$$\Delta \boldsymbol{W} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \boldsymbol{B_i}, \tag{3}$$

where n is a hyperparameter representing the number of Kronecker products, $\boldsymbol{A_i} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{B_i} \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}$. The proposed representation of the weight updates is composed of a sum of Kronecker products between shared *slow weights* $\boldsymbol{A}_i$ and independent *fast weights* $\boldsymbol{B}_i$, with $i \in \{1, \ldots, n\}$. $\boldsymbol{B}_i$ is of low rank and it is further decomposed into two low-rank matrices. We propose to parameterize $\boldsymbol{B_i} \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}$ as the product of $\boldsymbol{u_i} \in \mathbb{R}^{\frac{k}{n} \times r}$ and $\boldsymbol{v_i} \in \mathbb{R}^{r \times \frac{d}{n}}$, where r is the rank of the matrix. Overall, the expression of $\Delta \boldsymbol{W}$ is:

$$\Delta \boldsymbol{W} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \boldsymbol{B_i} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \left(\boldsymbol{u_i} \boldsymbol{v_i}^{\top}\right). \tag{4}$$

The illustration of the Kronecker Adaptation method is shown in Fig. 2.

4 Analysis of Efficient Fine-tuning Methods

4.1 Discussion of SOTA methods

Adapter-tuning [6] is the first efficient fine-tuning work in the NLP community. It brings in an additional trainable set of modules by adding a trainable bottleneck layer after the feedforward network in each Transformer layer of the pretrained language models. A bottleneck layer consists of a down and up projection pair that shrinks and recovers the size of token hidden states.

Table 1: Parameter count in Adapter-tuning, LoRA, Compacter, and Kronecker Adaptation. L is the number of layers in Transformer. k is the size of the input dimension to the Adapter layer. d is the bottleneck dimension in the Adapter layer. d_{model} is the Transformer hidden size. r denotes the rank in the low-rank decomposition step. n is the number of Kronecker products. n is a hyperparameter and usually very small, e.g., the number is 4 in our experiments.

Method	# of parameters	Complexity
Adapter-tuning [6]	$4Lkd$	$\mathcal{O}\left(kd\right)$
LoRA [7]	$2Lrd_{model}$	$\mathcal{O}\left(rd_{model}\right)$
Compacter [9]	$4L(\frac{k}{n}+\frac{d}{n})+n^3$	$\mathcal{O}\left(\frac{k+d}{n}\right)$
Ours	$2L(\frac{d_{model}}{n}+\frac{r}{n})+n^3$	$\mathcal{O}\left(\frac{r+d_{model}}{n}\right)$

Similar to the Adapter-tuning method where they use the bottleneck structure in the additional layer, our method implements low-rank decomposition on the *fast* rank-one matrices [17]. The critical functional difference is that our learned weights can be merged with the main weights during inference, thus not introducing any latency, which is not the case for the Adapter layers.

LoRA[7] is another line of work for parameter-efficient language model tuning: it treats the model parameters after fine-tuning as an addition of the pretrained parameters $\Theta_{\text{pretrained}}$ and task-specific differences θ_{task} , where $\Theta_{\text{pretrained}}$ is fixed and a new subset of model parameters are added on top. Given a pretrained weight matrix $\boldsymbol{W}_0 \in \mathbb{R}^{d\times k}$, they constrain its update by performing low-rank decomposition on it: $\boldsymbol{W_0} + \Delta\boldsymbol{W} = \boldsymbol{W}_0 + \boldsymbol{BA}$, where $\boldsymbol{A} \in \mathbb{R}^{r\times k}$, $\boldsymbol{B} \in \mathbb{R}^{d\times r}$, and the rank $r \ll \min(d, k)$. By doing this, the weight matrices are split into two parts, where during training, $\boldsymbol{W}_0$ is frozen and does not receive gradient updates, while only $\boldsymbol{A}$ and $\boldsymbol{B}$ contain trainable parameters.

Our work differs from LoRA in that apart from performing low-rank decomposition on weight updates, we also decompose weight updates to a set of Kronecker product decomposition. The decomposed *slow weight* are shared across layers, further reducing the parameter cost.

Compacter [9] is a method for fine-tuning large-scale language models with an excellent tradeoff between the number of trainable parameters, memory footprint, task performance compared to existing methods. The Compacter method builds on ideas from Adapters [6] and it also brings in an additional trainable set of modules by modifying the standard Adapter layer. It inserts task-specific weight matrices into weights of pretrained models. Each Compacter weight matrix is computed as the sum of Kronecker products between shared *slow* weights and *fast* rank-one matrices defined per Compacter layer.

In a similar vein to Compacter, we also leverage the Kronecker product in our method to further reduce parameter cost. Yet, apart from application domains, our method differs from Compacter in that we do not bring additional trainable layers and introduce no latency. We first select submodules by measuring the local intrinsic dimension and perform decomposition over these submodules. In addition, all of our decomposition operations are implemented on the updates to weights rather than weights themselves.

4.2 Analysis of Parameter Efficiency

We compute the parameter-efficiency of those fine-tuning methodologies, including ours as below:

- *Adapter-tuning*: In the standard setting, two Adapters are added per layer of a Transformer model [44]. Each Adapter layer consists of $2 \times k \times d$ parameters for the down and up-projection matrices respectively, where k is the size of the input dimension and d is the Adapter's bottleneck dimension. In the standard setting, the total number of parameters for Adapters for a Transformer model with L layers of both an encoder and a decoder is, therefore, $|\Theta| = 2 \times L \times 2 \times k \times d$, which scales linearly with all three variables.
- *LoRA*: LoRA adds trainable pairs of rank decomposition matrices to existing weight matrices. The number of trainable parameters is determined by the rank r and the shape of the original weights: $|\Theta| = 2 \times L \times d_{\text{model}} \times r$, where d_{model} is Transformer hidden size.

- *Compacter*: Compacter shares the trained weight matrices $\{\boldsymbol{A_i}\}_{i=1}^{n}$ consisting of n^3 parameters across all layers, where n is the number of Kronecker products. Compacter also has two rank-one weights for each Adapter layer consisting of $\frac{k}{n} + \frac{d}{n}$ parameters, where the Adapter layers are of size $k \times d$, resulting in a total of $2 \times \left(\frac{k}{n} + \frac{d}{n}\right)$ parameters for down and up-projection weights. Therefore, the total number of parameters of Compacter is $4 \times L \times \left(\frac{k}{n} + \frac{d}{n}\right) + n^3$ for a Transformer with L layers in the encoder and decoder.

- *Our approach*: In our approach, similarly, we decompose the updates to weights into a sum of Kronecker products first and then further perform low-rank decomposition for the *fast weights*. Because it is conducted on the weight updates, the total number of parameters in this scenario will be: $2 \times L \times \left(\frac{r+d_{model}}{n}\right) + n^3$.

The overall comparison of parameter counts is shown in Table 1. Our method has a complexity of $\mathcal{O}\left(\frac{r+d_{model}}{n}\right)$ with r being a small integer. This is lower than the other three SOTA methods, and our approach greatly reduces the number of parameters. The exact numbers of fine-tuned parameters are present in Table 2.

4.3 Explanations with Local Intrinsic Dimension

Measuring the intrinsic dimension of an objective function was first proposed in [42]. It is extended in analyzing the quality of pretrained language models in Aghajanyan *et al*. [16]. They find that analyzing fine-tuning through the lens of intrinsic dimension can offer empirical and theoretical intuitions. Both of them study the intrinsic dimension of the entire model.

We propose to measure the intrinsic dimension of each module to show which components in ViT are more critical empirically. We define the intrinsic dimension of each module as *local intrinsic dimension*, to distinguish it from the intrinsic dimension of the whole model. We presume the local intrinsic dimension is indicative of the contribution of each module during fine-tuning. The conventional standard method of measuring the intrinsic dimensionality of an objective [42] asks for performing grid search over different subspace dimensions d, training using standard SGD [45] over the subspace reparameterization, and selecting the smallest d which can produce a satisfactory solution (e.g., 90% of the full training metric). Likewise, we measure the local intrinsic dimension via finding the smallest d for the measured module that can reach 90% of the full accuracy.

5 Experiments

Datasets We summarize the results by computing the average performance on CIFAR-10 [18], CIFAR-100 [18], and SUN-397 [19]. CIFAR-10 contains 10 classes, and CIFAR-100 contains 100 classes. Both these datasets contain 60K images, with each class having the same number of images. SUN-397 contains 130,519 images from 397 scene categories, including the abbey, balcony, cafeteria, etc. Each category has more than 100 images. We use the official split for each of these datasets. More results are given in the supplementary materials.

Implementation Details For benchmark experiments, we use the SGD [45] optimizer with the learning rate and weight decay being automatically searched for all methods so that these two hyperparameters will have the optimum combination. The batch size is set to 64. Training epochs are set via grid search. We use different learning rates during Full-model Fine-tuning: lower learning rate for the pretrained part (image encoder) than the randomly initialized part (linear head). We test two types of pretrained 12-layer Vision Transformers: the one via unsupervised pretraining (*CLIP*) and the one via supervised pretraining (*Supervised ViT*).

For intrinsic dimension experiments, we use the AdamW [46] as the optimizer, with the weight decay of 10^{-8}, learning rate of 10^{-5}, and batch size of 32. The Fastfood transform [43] is applied to the attention and multi-layer perceptron (MLP) module in the first layer of Supervised ViT, respectively. The dimension d is measured from $0 - 1000$ in both scenarios. Each model is fine-tuned for 300 epochs.

Table 2: Experimental result comparison on CIFAR-10 [18], CIFAR-100 [18], SUN-397 [19] datasets in terms of accuracy (%) and number of fine-tuning parameters (#params). Two Vision Transformers are evaluated, CLIP [12] and Supervised ViT [5]. We compare the proposed Kronecker Adaptation method with three baseline categories: (1) commonly-used fine-tuning methods for vision models (Full-model Fine-tuning and Linear-probing); (2) the re-implementation of SOTA methods for NLP models; (3) various baseline methods introduced in this work. Our method achieves the best tradeoff between accuracy and parameter efficiency: it obtains the best average accuracy among all efficient fine-tuning methods, which is also comparable to Full-model Fine-tuning, while updating only 0.07% of the model parameters in CLIP and 0.17% in Supervised ViT.

Method	CLIP					Supervised ViT				
	CIFAR-10	CIFAR-100	SUN-397	Average	#params	CIFAR-10	CIFAR-100	SUN-397	Average	#params
Commonly-used fine-tuning methods for vision models										
Full-model Fine-tuning	97.7	85.4	73.8	85.6	151,364,010	99.0	92.4	75.0	88.8	86,697,617
Linear-probing	94.8	80.1	72.4	82.4	86,697	96.3	87.7	70.1	84.7	86,697
SOTA methods for NLP models										
BitFit [8]	92.1	76.0	70.8	79.6	233,873	92.3	81.0	71.8	81.7	349,701
Adapter-tuning [6]	94.7	81.4	77.1	84.4	190,545	98.4	90.6	74.2	87.7	1,813,929
AdapterDrop [14]	93.3	78.3	71.4	81.0	95,351	96.8	88.4	72.3	85.8	230,633
LoRA [7]	95.1	78.1	80.8	84.7	185,001	98.7	90.6	73.6	87.6	277,417
Baseline methods developed in this work										
Transformer-probing	95.6	80.1	74.3	83.3	3,236,521	96.5	86.9	76.7	86.7	3,236,521
LoRA-Fix	92.5	77.1	60.0	76.5	135,849	96.2	88.3	72.0	85.5	203,689
LayerNorm Tuning	82.5	76.6	66.7	75.2	89,769	92.2	71.7	72.0	78.6	131,497
Attention Tuning	96.8	81.8	73.1	83.9	41,042,601	93.9	85.7	73.8	84.4	284,783,77
LePE Tuning	95.1	78.9	68.0	80.7	148,137	93.7	90.8	73.2	85.8	207,801
RPB Tuning	94.7	77.1	68.4	80.1	102,921	96.7	87.0	72.4	85.3	201,704
Kronecker Adaptation (Ours)	95.9	84.8	74.0	84.9	107,727	97.9	91.2	75.1	88.1	144,396

5.1 Baselines

Apart from our proposed method, for other researchers to benchmark with, we tested the following baselines using both the Supervised ViT and CLIP:

First are commonly-used fine-tuning methods for vision models.

- *Full-model Fine-tuning*: it fine-tunes all the parameters of the model.
- *Linear-probing*: all parameters are fixed except for the task-specific classification layer.

The second types are SOTA methods borrowed from the NLP community.

- *BitFit* [8]: BitFit freezes all ViT parameters except for the bias-terms and the task-specific classification layer.
- *Adapter*-tuning [6]: two Adapters are added in each Transformer layer.
- *AdapterDrop* [14]: Adapterdrop is an extension of Adapter-tuning methods where it drops Adapters from lower Transformer layers during training and inference. In our experiments, we dropped Adapters from all layers except for the last layer in ViT.
- *LoRA* [7]: LoRA adds trainable pairs of low rank decomposition matrices to pretrained weight matrices. During fine-tuning, only the low rank decomposition matrices and task-specific classification layers are updated. Similar to Hu *et al*. [7], we apply LoRA to $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$ matrices in the attention module.

The third types are the new baseline methods we developed.

- *Transformer-probing*: we stack an additional trainable Transformer block before the task-specific classification layer. During fine-tuning, we keep the original pretrained model parameters frozen and only tune the additional Transformer block and the task-specific classification layers in the ViT and fix all other blocks.
- *LoRA-Fix*: we fix the matrix $\boldsymbol{A}$ in LoRA and only the matrix $\boldsymbol{B}$ receives gradients. $\boldsymbol{B}$ will be updated along with the task-specific classification layer.

- *LayerNorm Tuning*: we only tune the layernorm layers and the task-specific classification layers in the ViT and fix all other blocks.
- *Attention Tuning*: we only tune the attention layers and the task-specific classification layers in the ViT and fix all other blocks.
- *LePE Tuning*: We refer to the work in Dong *et al*. [27] by adding an additional locally-enhanced positional encoding (LePE) in the pretrained Vision Transformer, and implement it with a depthwise convolution operator [47] applying on matrix $\boldsymbol{V}$ in the attention layer:

$$\mathrm{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{SoftMax}\left(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d}\right)\boldsymbol{V} + \mathrm{DWConv}(\boldsymbol{V}). \tag{5}$$

  During fine-tuning, we keep the original pretrained model parameters frozen and only tune the LePE and the task-specific classification layer in the ViT and fix all other blocks.
- *Relative Position Bias (RPB) Tuning*: resorting to the SWIN Transformer [26], where they include a relative position bias $\boldsymbol{B}$ in computing self-attention in the Vision Transformer:

$$\mathrm{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{SoftMax}\left(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d} + \boldsymbol{B}\right)\boldsymbol{V}. \tag{6}$$

  During fine-tuning, we keep the original pretrained model parameters frozen and only tune the relative position bias and the task-specific classification layers in the ViT and fix all other blocks.

LayerNorm Tuning, Attention Tuning, and BitFit shed light on which parameters in ViT matter more during fine-tuning. Among all modules in ViT, multi-layer perceptron (MLP) tuning is not considered a baseline because it is too parameter-expensive compared to others. Given that the special structure of ViT and its variants, e.g., depthwise convolution operator and relative position bias, are different from the NLP transformer, we actually made the first step towards parameter-efficient fine-tuning for ViT via LePE Tuning and Relative Position Bias Tuning.

5.2 Results and Analysis

The experimental results of measured average accuracy across the three datasets are shown in Table 2. In our analytical experiments, we first observe that Full-model Fine-tuning has the highest accuracy in both scenarios, serving as a performance upperbound. Second, different efficient fine-tuning methods exhibit diverse characteristics and perform differently on the same task. Third, the results from CLIP are mostly consistent with the results from Supervised ViT. This suggests that the pretraining strategy may not affect the selection of downstream fine-tuning strategy much. Fourth, previous methods such as Adapter-tuning [6] and LoRA [7] are still effective, and their accuracy is substantially higher than naive baselines, including BitFit and Attention-tuning regardless of the pretrained checkpoint. Fifth, among naive baselines where only submodules or task-specific classification heads are tuned, tuning the parameters of the attention layer turns out to be a surprisingly effective approach even compared to some SOTA methods, though its parameter cost is significantly higher. This further validates the effectiveness of our method by applying Kronecker Adaptation to attention weights. *Finally, our method outperforms all the SOTA methods borrowed from the NLP community as well as their variants in both scenarios.*

Furthermore, the average number of trainable parameters across three datasets is also shown in Table 2. As can be seen, our Kronecker Adaptation method contains the lowest parameter cost compared with other SOTA methods. This phenomenon is obviously noticeable when compared with Full-model Fine-tuning, where our method takes less than 0.17% trainable parameters of end-to-end Full-model Fine-tuning but it is capable of achieving comparable performance.

5.3 Intrinsic Dimension

Measuring intrinsic dimension [42] allows us to perform some comparison across the importance of fine-tuning each module in the Transformer block. In this section, we measure the intrinsic dimension of the two most fundamental modules in the Vision Transformer — the MLP module and the attention module. The checkpoint we evaluate is the *ViT-B-224/16* via supervised pretraining. The MLP module and the attention module we evaluated are both in the first layer of ViT. The results are representative and other layers follow the same trend. We use the remarkable Fastfood

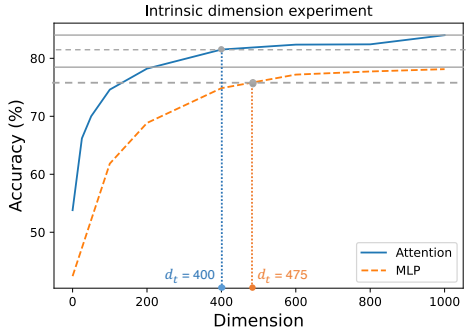

Figure 3: Validation Accuracy *vs.* Subspace Dimension d of MLP and the attention module for Supervised ViT on CIFAR-100. The local intrinsic dimension d_t of the attention module is lower than that of the MLP.

Table 3: Kronecker Adaptation and Adapter-tuning ablation experiments with Supervised ViT on CIFAR-10 [18], CIFAR-100 [18], and SUN-397 [19]. We report the average accuracy (%) across the three datasets.

Method	Average Accuracy
Kronecker Adaptation	88.1
Kronecker Adaptation to MLP modules	86.6
Adapters on attention layer	54.1
Standard Adapter-tuning	87.7

transform [43] to do the projection. The results are shown in Fig. 3. As a substantiating point to performing Kronecker Adapting on attention layers, we can see the attention module has a lower intrinsic dimension than the MLP module.

5.4 Ablation Studies

We ablate our method and Adapter-tuning using the settings in Table 2. Several intriguing properties are observed. We first test the variant of our method by applying Kronecker Adaptation to MLP modules. As can be seen, it performs worse than the original method where we apply Kronecker Adaptation to attention modules. This phenomenon is consistent with our findings from naive baseline experiments and intrinsic dimension experiments. Second, we test another variant of Adapter-tuning. Instead of inserting two Adapters after the attention and feedforward modules respectively following Houlsby *et al*. [6], we add Adapters in the attention layers. It can be observed that the standard Adapter-tuning outperforms this variance, which suggests the effectiveness of the vanilla Adapter-tuning when it is adapted to vision tasks.

6 Conclusion

In this paper, we conduct the first comprehensive comparison of efficient fine-tuning on the image classification tasks using Vision Transformers. We also propose a better parameter-efficient fine-tuning strategy in the principle of subspace training and parameterized hypercomplex multiplication, which achieves the best tradeoff between accuracy and parameter efficiency. Looking into the future, we plan to explore the generalization of our method to other tasks, especially in the vision-and-language domain. We will release a benchmark by providing the implementation of all the methods studied in this paper, which could be directly used in developing future efficient fine-tuning strategies and will hopefully facilitate research in this area.

References

[1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[5] 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. *arXiv preprint arXiv:2010.11929*, 2020.

[6] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. *arXiv:1902.00751 [cs, stat]*, June 2019.

[7] 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. *arXiv:2106.09685 [cs]*, October 2021.

[8] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. *arXiv:2106.10199 [cs]*, June 2021.

[9] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient Low-Rank Hypercomplex Adapter Layers. *arXiv:2106.04647 [cs]*, June 2021.

[10] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[11] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[12] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[14] Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. AdapterDrop: On the Efficiency of Adapters in Transformers. *arXiv:2010.11918 [cs]*, October 2021.

[15] Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. *arXiv:2101.00190 [cs]*, January 2021.

[16] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. *arXiv:2012.13255 [cs]*, December 2020.

[17] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.

[18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report, 2009.

[19] Jianxiong Xiao, Krista A Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 119 (1):3–22, 2016.

[20] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2014.

[21] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *arXiv preprint arXiv:2006.03236*, 2020.

[22] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. Knowledge guided text retrieval and reading for open domain question answering. *arXiv preprint arXiv:1911.03868*, 2019.

[23] Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *European Conference on Computer Vision*, pages 121–137. Springer, 2020.

[24] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[25] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[26] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *arXiv:2103.14030 [cs]*, August 2021.

[27] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. *arXiv preprint arXiv:2107.00652*, 2021.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[30] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[31] Kuang-Huei Lee, Xi Chen, Gang Hua, Houdong Hu, and Xiaodong He. Stacked cross attention for image-text matching. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 201–216, 2018.

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[33] Tan Thongtan and Tanasanee Phienthrakul. Sentiment classification using document embeddings trained with cosine similarity. In *ACL SRW*, 2019. URL `https://www.aclweb.org/anthology/P19-2057`.

[34] Aaron Steven White, Pushpendre Rastogi, Kevin Duh, and Benjamin Van Durme. Inference is everything: Recasting semantic resources into a unified evaluation framework. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 996–1005, 2017.

[35] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, pages 13–23, 2019.

[36] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7464–7473, 2019.

[37] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. *arXiv:2005.00247 [cs]*, January 2021.

[38] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks. *arXiv preprint arXiv:2112.06825*, 2021.

[39] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[40] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[41] Aston Zhang, Yi Tay, Shuai Zhang, Alvin Chan, Anh Tuan Luu, Siu Cheung Hui, and Jie Fu. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with $1/n$ parameters. *arXiv preprint arXiv:2102.08597*, 2021.

[42] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the Intrinsic Dimension of Objective Landscapes. *arXiv:1804.08838 [cs, stat]*, April 2018.

[43] Quoc Viet Le, Tamás Sarlós, and Alexander Johannes Smola. Fastfood: Approximate kernel expansions in loglinear time. *arXiv preprint arXiv:1408.3060*, 2014.

[44] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *ICLR*, 2019. URL `https://arxiv.org/abs/1809.10853`.

[45] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[46] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.