1% VS 100%: Parameter-Efficient Low Rank Adapter for Dense Predictions

Dongshuo Yin\textsuperscript{1,2,†}, Yiran Yang\textsuperscript{1,2,†}, Zhechao Wang\textsuperscript{1,2}, Hongfeng Yu\textsuperscript{1}, Kaiwen Wei\textsuperscript{1,2}, Xian Sun\textsuperscript{1,2,*}

\textsuperscript{1}Key Laboratory of Network Information System Technology, Aerospace Information Research Institute, Chinese Academy of Sciences
\textsuperscript{2}School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences
\{yindongshuo19, yangyiran19, wangzhechao21, weikaiwen19\}@mails.ucas.ac.cn
\{yuhf, sunxian\}@aircas.ac.cn

Abstract

Fine-tuning large-scale pre-trained vision models to downstream tasks is a standard technique for achieving state-of-the-art performance on computer vision benchmarks. However, fine-tuning the whole model with millions of parameters is inefficient as it requires storing a same-sized new model copy for each task. In this work, we propose LoRand, a method for fine-tuning large-scale vision models with a better trade-off between task performance and the number of trainable parameters. LoRand generates tiny adapter structures with low-rank synthesis while keeping the original backbone parameters fixed, resulting in high parameter sharing. To demonstrate LoRand’s effectiveness, we implement extensive experiments on object detection, semantic segmentation, and instance segmentation tasks. By only training a small percentage (1% to 3%) of the pre-trained backbone parameters, LoRand achieves comparable performance to standard fine-tuning on COCO and ADE20K and outperforms fine-tuning in low-resource PASCAL VOC dataset.

1. Introduction

With the rapid development of computer vision, parameters in deep models are surging. Giant models need to be trained with massive resources to achieve superior performance [3, 17, 47, 58], which is often unavailable to many academics and institutions. “Pretrain & Finetuning” paradigm is widely used to alleviate this dilemma. Teams with sufficient computation resources utilise enormous datasets [2, 9, 40, 50] to train superior backbones [4, 32, 40, 48] and optimise the models with ideal performances. Models pretrained in this way usually have a superior understanding of homogeneous data. After that, researchers with limited computational resources can transfer the understanding capabilities of the pre-trained models to downstream tasks with promising performances by fine-tuning [1, 26, 46, 53].

However, the fine-tuned model will produce a new set of parameters as large as the pre-trained model. New parameters are independent of the pre-trained models and unshareable, which are very hardware intensive for cloud service providers [23, 49]. Figure 1 compares the parameter quantities of some remarkable backbones and their performances on the COCO [28] dataset. Recent advances in natural language processing (NLP) [30, 38] show that large pre-trained models trained with rich data have strong gener-
Computer vision has been continuously inspired by NLP in recent years, including the visual transformer series [5,13,29,32] and self-supervised MAE series [15,19,60]. In fact, NLP is leading new training trends different from fine-tuning. Fine-tuning produces a new parameter set for each new task, which is parametrically inefficient for plenty of linguistic tasks [22,30]. To solve this problem, [30] and [22] have proposed “Prompt Tuning” and “Adapter Tuning” respectively, both of which fix all parameters of the backbone.

### 2. Related Work

#### 2.1. Training Paradigms in NLP

To find a parameter-efficient paradigm with promising performance in computer vision, we explore the potential of Adapter Tuning for visual dense predictions. We employ the advanced Swin Transformer [32] trained with ImageNet-22K [9] as the pre-trained model. After that, we add bottleneck adapter structures [22] behind each SwinBlock and freeze the original backbone parameters when training, but this approach cannot achieve comparable performance to fine-tuning as mentioned in [24]. In the experiments, we find that the models perform better with sparser adapter structures. To improve the performance of Adapter Tuning, we propose **Low-Rank Adapter (LoRand)** to reduce the adapter parameters, as shown in Figure 2. LoRand sparsely parameterizes the matrices in adapters by low-rank synthesis. Specifically, the projection matrix of the fully-connected layer (FC) in LoRand is a product of multiple low-rank matrices, which reduces FC parameters by more than 80%. We implement extensive experiments on object detection (PASCAL VOC [14]), semantic segmentation (ADE20K [62]), and instance segmentation (MS COCO [28]) to verify the capability of LoRand. Experimental results show that LoRand-Tuning is comparable to fine-tuning on multiple tasks with only 1.8% to 2.8% new backbone parameters, which suggests that the pre-trained backbone parameters can be fully shared. More interestingly, our method completely outperforms fine-tuning on the PASCAL VOC dataset, illustrating that LoRand-Tuning can reduce the impairment of fine-tuning on pre-trained models in low-resource configurations. Our method demonstrates that the LoRand-Tuning paradigm can substantially save storage resources and achieve competitive performances on most dense prediction tasks. In summary, our contributions are three-fold:

- We demonstrate that visual pre-trained models are highly generalisable and shareable. With our training methods, new tasks require only a few trainable parameters to achieve performances comparable to fine-tuning, which can save massive hardware resources.

- We propose the LoRand structure for sparser adapters based on low-rank synthesis. We demonstrate that the backbone parameters in fine-tuning are highly redundant, which can be replaced by 1.8% to 2.8% additional parameters in LoRand.

- Extensive experiments on object detection, semantic segmentation, and instance segmentation show that LoRand-Tuning can achieve remarkable performances and reduce massive new parameters in challenging dense prediction tasks.
and plug a few tiny trainable structures (less than 10% of the backbone) to adapt the pre-trained model to the new tasks. “Prompt tuning” adds learnable parameters (also known as prompts) to the input or intermediate layers to change the input space of the new tasks. “Prompts” can motivate the model to remember knowledge learned in the previous tasks. “Adapter tuning” adds learnable bottleneck structures after each block to connect the pre-trained model with new tasks. Adapter and prompt demonstrate the coexistence of parameter efficiency and high performances in NLP, stimulating studies in CV. [24] proposes Visual Prompt Tuning (VPT) for image classification and semantic segmentation, but the performance of VPT on semantic segmentation is still far from fine-tuning. This phenomenon motivates us to explore whether adapter tuning can bring a new paradigm in computer vision with fewer parameters and better performances. In this work, we try to explore parameter-efficient and high-performance adapter structures.

2.2. Adapter Tuning

Adapters have been widely studied in NLP. Houlsby et al. [22] first add a bottleneck adapter structure to the transformer blocks and fix the original backbone, which achieves comparable performances to fine-tuning. Figure 3 illustrates the differences between fine-tuning and adapter-tuning. [37, 44, 59] further reduce parameters in the adapter with closer performances to fine-tuning. [18, 34, 39] outperform fine-tuning on low-resource tasks, demonstrating that more parameters may not improve performance when fine-tuning pre-trained models [36]. In computer vision, [41] add convolutional adapters to the ResNet [20] and obtain competitive results in image classification. Adapter concept has also been applied in multimodal [33], vision-and-language [51], and domain adaptation [56], but these methods are only applicable under specific conditions. [7, 21, 25, 31] investigate the potential of adapter-tuning for visual classification. [8] apply the adapter structure to visual dense predictions without fixing any original parameters, which indeed trades more parameters for better performances.

2.3. Low-rank Approximation

The low-rank approximation uses multiple low-dimensional tensors to approximate a larger tensor with higher dimensions. Tensor dimensions and sizes in machine learning are very large, so low-rank approximations are widely used in face recognition [61], distributed training [54], transfer learning [11], and cross-domain [10]. A $b \times c$ matrix $M$ can be approximated with $N$ low-rank matrices $Q$ by the following equation:

$$ M_{b \times c} = \prod_{i=1}^{N} Q_{r_i \times s_i}, \quad (1) $$

where $N$ has different values depending on the approximation methods, we implement low-rank approximation of the adapter matrices by heuristic learning.

3. Method

In this section, we will elaborate on the proposed low-rank adapter (LoRand) in three parts: adapter tuning paradigm, LoRand, and parameter analysis.

3.1. Adapter Tuning Paradigm

For dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, fine-tuning calculates the loss between inference results and labels according to the formula:

$$ L(D, \theta) = \sum_{i=1}^{N} \text{loss}(f_{\theta}(x_i), y_i), \quad (2) $$

where $f_{\theta}$ denotes the network forward function and loss represents the loss function. After that, $\theta$ is optimized through

$$ \theta \leftarrow \arg \min_{\theta} L(D, \theta), \quad (3) $$

In adapter tuning paradigm, parameters consist of two parts, including parameters in adapter $\theta_A$ and parameters in the original architecture $\theta$. Here, $\theta$ is further divided into frozen part $\theta_F$ and trainable part $\theta_T$, noted as $\theta = \{\theta_F, \theta_T\}$. Let $\Omega$ be all the trainable parameters, then $\Omega = \{\theta_A, \theta_T\}$. The loss function and optimization formula in adapter can be written as:

$$ L(D, \theta_F, \Omega) = \sum_{i=1}^{N} \text{loss}(f_{\theta_F, \Omega}(x_i), y_i), \quad (4) $$

20118
Figure 4. **Left:** Multi-branch projection in LoRand. The down-projection $W^D_i$ and up-projection $W^U_i$ matrices are the summation of $\alpha$ branches $W^D_1 \ldots W^D_\alpha$ ($W^U_1 \ldots W^U_\alpha$). $K_i$ in $i$-th branch is shared between $W^D_i$ and $W^U_i$. All the $P$, $Q$, and $K$ are trainable, while all the $W$ matrices are calculated. **Right:** Comparisons of the same-sized projection matrices between LoRand and Adapter. $(m, n)$ in the table are typical values in SwinBlocks. LoRand has far fewer parameters than Adapter. With the same projection dimension, LoRand saves over 80% parameters of the Adapter in Swin Transformers. $(\alpha, \beta)$ here are $(2, 8)$, the same as the experiments.

\[ \Omega \leftarrow \arg \min_{\Omega} L(D, \theta_F, \Omega). \quad (5) \]

### 3.2. LoRand

Before introducing LoRand, we first review the existing adapter structure. Conventional adapters are bottleneck structures containing a down-projection, an up-projection, and a non-linear activation function. Besides, adapters ensure the robustness of the model by adding residual [20] structures. Adapter layer can be formulated as follows:

\[ A^l = U^l \left( \text{GeLU} \left( D^l(x) \right) \right) + x, \quad (6) \]

where $U^l$ and $D^l$ represent the up and down projections in the $l$-th adapter layer, and GeLU is the activation function. It is clear that the parameters in adapter come from the projections. The projection process can be written as:

\[ y = W x + b, \quad (7) \]

which means most adapter parameters are in $W$.

To reduce the adapter parameters, we propose a low-rank adapter (LoRand) structure to replace the $W$ in the projection structures. Figure 2 shows the simplified structure of LoRand. Here we approximate not a specific matrix $W$ but an ideal matrix $W_{\text{best}}$ that can transform the feature space of the pre-trained model into new tasks by heuristic learning. The approximation matrix $\hat{W}$ has the same size as $W$, but the low-rank design makes $\hat{W}$ have far fewer free degrees than a common $W$.

Specifically, we synthesize each $W$ by multiplying three low-rank matrices $P \in \mathbb{R}^{\beta \times m}, K \in \mathbb{R}^{\beta \times \beta}, Q \in \mathbb{R}^{\beta \times n}$, that is:

\[ W = P^T K Q, \quad (8) \]

where $\beta \ll \min(m, n)$ ensuring that $P$ and $Q$ are low-rank matrices. $K$ can be regarded as a kernel matrix that controls the parameter size of LoRand.

After that, we add multi-branch structures to LoRand to increase the robustness and stability of low-rank matrices, which is inspired by MoE [43] and adaboost [45, 52]. Every $W$ consists of $\alpha$ branches, that is:

\[ W = \sum_{i=1}^{\alpha} W_i = \sum_{i=1}^{\alpha} (P^U_i) K_i Q^U_i, \quad (9) \]

In addition, we share the kernel matrix $K_i$ of the two projection layers within each branch. We hope the sharing mechanism can promote the coherence of two projection layers during training process. Besides, the shared $K$ also slightly reduces the number of LoRand parameters. Up to now, the $W^U$ and $W^D$ in a complete LoRand structure can be represented as:

\[ W^U = \sum_{i=1}^{\alpha} W^U_i = \sum_{i=1}^{\alpha} (P^U_i)^T K_i Q^U_i, \quad (10) \]

\[ W^D = \sum_{i=1}^{\alpha} W^D_i = \sum_{i=1}^{\alpha} (P^D_i)^T K_i Q^D_i, \quad (11) \]

where $K_i$ is shared in $W^U$ and $W^D$. Figure 4 presents the detailed designs of the multi-branch projection.
3.3. Parameter Analysis

In this section, we will compare the parameters of LoRand and typical adapter [22] with the same size of projection matrix.

Adapter Let $m$ be the input dimension of the adapter and $n$ be the middle layer dimension after down projection. Then the number of parameters in each adapter is $2mn$ (ignoring the few biases). In general, adapter tuning places two adapter modules in each block, so the space complexity of all adapter parameters in $\gamma$ blocks can be written as:

$$O(4\gamma mn). \quad (12)$$

LoRand According to section 3.2, each $W$ contains $\alpha$ sets of $\{P, Q, K\}$, that is:

$$\alpha(m\beta + \beta^2 + n\beta). \quad (13)$$

Each LoRand consists of two $W$ and $\alpha$ shared $K$, so the parameter quantity of each LoRand is:

$$2\alpha (m\beta + \beta^2 + n\beta) - \alpha\beta^2 = 2\alpha\beta(m + n + \beta/2). \quad (14)$$

Each block has two LoRand structures, so the number of parameters in $\gamma$ blocks is:

$$4\alpha\beta\gamma (m + n) + 2\alpha\beta^2\gamma. \quad (15)$$

As $\alpha, \beta, \gamma \ll \min(m, n)$, the space complexity here can be written as:

$$O(4\alpha\beta\gamma (m + n)). \quad (16)$$

Comparison between Formulas 12 and 16 can be simplified as:

$$O(\alpha\beta (m + n)). \quad (17)$$

and

$$O(\alpha\beta (m + n)). \quad (18)$$

Given that $\alpha, \beta \ll \min(m, n)$, the space complexity of LoRand is far lower than the typical adapter. The table in Figure 4 illustrates that LoRand saves most Adapter parameters with the same projecting dimension.

4. Experiments

We evaluate LoRand on multiple dense prediction tasks, including object detection, semantic segmentation, and instance segmentation. We also evaluate LoRand under low-resource conditions. We first describe our experimental setup in Section 4.1, including pre-trained backbones, baselines, LoRand settings, and downstream tasks. Then we present the main results of three benchmarks in Section 4.2. We also implement ablation study in Section 4.3 to investigate the impact of structural settings in LoRand.

4.1. Experimental Setup

Pretrained Backbones We conduct experiments on the advanced Swin Transformer [32] architectures. All backbones in this section are pre-trained by ImageNet-22k [9]. Pre-trained models are provided by OpenMMLab [6].

Baselines We compare LoRand with three other common training methods:

(a) FULL: update all parameters in the architecture.
(b) FIXED: fix pre-trained parameters in Swin and train other parts of the architecture (neck, head).
(c) ADAPTER: add two trainable adapter structures in each SwinBlock following [22], and freeze other parts of the backbone. We evaluate two forms of adapter with different middle layer dimensions ($D_{ML}$):
- ADAPTER-B: $D_{ML}$ is a half of input dimension.
- ADAPTER-T: $D_{ML}$ is a quarter of input dimension.

LoRand Settings We conducted experiments on three LoRand variants, which have different branch numbers $\alpha$ and kernel matrix dimensions $\beta$.
- LoRand: $\alpha = 2, \beta = 8$ (Standard).
- LoRand+: $\alpha = 4, \beta = 8$.
- LoRand++: $\alpha = 4, \beta = 16$.

Downstream Tasks We conducted experiments on COCO [28], ADE20K [62], and PASCAL VOC [14] benchmarks to widely evaluate LoRand’s performance on main dense prediction tasks.

COCO 2017 [28] is the most commonly used dataset for object detection and instance segmentation, which contains 118K training and 5K validation images. We perform experiments on the validation set. For a fair comparison, all experiments performed on COCO employ Cascade MASK R-CNN [32] as the detector.

ADE20K [62] is the most widely used semantic segmentation dataset, which contains 20K training and 2K validation images. We also conduct experiments on the ADE20K validation set and utilise UperNet [57] as the framework.

PASCAL VOC 0712 [14] is also widely used in object detection, which contains about 16K training and 5K validation images. VOC 0712 is much smaller than the latest benchmarks, so we treat it as a low-resource case. We adopt Faster RCNN [42] as the detector for VOC 0712.

All our experiments are conducted with 8x NVIDIA Tesla V100 GPUs. The experiments on PASCAL VOC and
4.2. Main Results

We first compare the trainable backbone parameters and performance of these methods on three benchmarks in Tables 1 and 2. Table 1 shows the results of PASCAL VOC and ADE20K datasets based on Swin-L, and Table 2 shows the results of COCO based on Swin-B. From Tables 1 and 2, we can see that:

1) LoRand can effectively address the dilemma of fine-tuning in low-resource situations. Table 1 shows that FIXED outperforms FULL on the PASCAL VOC dataset, which implies that the powerful generalization ability of pre-trained model is severely weakened during fine-tuning. Fine-tuning with low-resource data reduces the feature understanding of pre-trained models, which leads to the poor performance on downstream tasks. LoRand avoids this disadvantage by fixing the original parameters. More importantly, LoRand can absorb features from the new data by its smaller trainable structures. Table 1 indicates that LoRand outperforms FULL and FIXED by 2.69% and 1.93% on the low-resource dataset with only 1.84% trainable backbone parameters. LoRand+ and LoRand++ also outperform FULL by 3.2% and 3.68% with 3.62% and 7.17% backbone parameters. In fact, there are many other common computer vision datasets with similar volumes to the PASCAL VOC, including CUB-200-2011 [55], Oxford 102 Flowers [35], Stanford Cars [27], and Caltech-256 [16]. The prevalence of “Pretrained & Finetuning” leads us to focus more on giant benchmarks, but Table 1 suggests we need a better training paradigm to cope with many low-resource situations in industrial applications. LoRand-Tuning proves to be a competitive candidate who brings promising performance and parameter-efficient approaches to low-resource cases.

2) LoRand effectively balances the number of trainable backbone parameters and downstream task performance. ADE20K are based on Swin-S, Swin-B, and Swin-L pre-trained models. Limited by GPU memory, the COCO experiments are based on Swin-T, Swin-S, and Swin-B.

Table 1. Results of baselines and our methods on Pascal VOC and ADE20K benchmarks. Swin-L is employed as the pre-trained model here. We present the numbers and percentages of trainable backbone parameters on the left and all the performances on the right. ∗ denotes the trainable parameters in backbones.

<table>
<thead>
<tr>
<th>Swin-L (198M)</th>
<th>Trained+ Params %</th>
<th>ΔFull</th>
<th>Extra Structure</th>
<th>Pascal VOC (Faster RCNN)</th>
<th>ADE20K (UperNet)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AP_Box</td>
<td>ΔLoRand</td>
</tr>
<tr>
<td>FULL</td>
<td>198.58 M   100.00 %</td>
<td></td>
<td>x</td>
<td>84.43 %</td>
<td>- 2.69 %</td>
</tr>
<tr>
<td>FIXED</td>
<td>0.00 M      0.00 %</td>
<td>100.00 %</td>
<td>x</td>
<td>85.19 %</td>
<td>- 1.93 %</td>
</tr>
<tr>
<td>ADAPTER-B</td>
<td>32.04 M    16.13 %</td>
<td>- 83.87 %</td>
<td>√</td>
<td>80.93 %</td>
<td>- 6.19 %</td>
</tr>
<tr>
<td>ADAPTER-T</td>
<td>16.04 M    8.08 %</td>
<td>- 91.92 %</td>
<td>√</td>
<td>78.10 %</td>
<td>- 9.02 %</td>
</tr>
</tbody>
</table>

Our Methods

|               |                   |       |                |                |                  |
| LOAND         | 3.59 M    1.84 % | - 98.16 % | √              | 87.12 % | -        | 50.67 % | -        |
| LOAND+        | 7.19 M    3.62 % | - 96.38 % | √              | 87.63 % | + 0.51 % | 51.13 % | + 0.46 % |
| LOAND++       | 14.24 M   7.17 % | - 92.83 % | √              | 88.11 % | + 0.99 % | 51.87 % | + 1.20 % |

Table 2. Results of baselines and our methods on COCO benchmarks. Swin-B is employed as the pre-trained model here. We present the numbers and percentages of trainable backbone parameters on the left and all the performances on the right. ∗ denotes the trainable parameters in backbones.

<table>
<thead>
<tr>
<th>Swin-B (89M)</th>
<th>Trained+ Params %</th>
<th>ΔFull</th>
<th>Extra Structure</th>
<th>COCO (Cascade Mask R-CNN)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AP_Box</td>
</tr>
<tr>
<td>FULL</td>
<td>89.14 M   100.00 %</td>
<td></td>
<td>x</td>
<td>51.90 %</td>
</tr>
<tr>
<td>FIXED</td>
<td>0.00 M      0.00 %</td>
<td>100.00 %</td>
<td>x</td>
<td>15.30 %</td>
</tr>
<tr>
<td>ADAPTER-B</td>
<td>14.38 M    16.13 %</td>
<td>- 83.87 %</td>
<td>√</td>
<td>46.50 %</td>
</tr>
<tr>
<td>ADAPTER-T</td>
<td>7.20 M      8.08 %</td>
<td>- 91.92 %</td>
<td>√</td>
<td>43.20 %</td>
</tr>
</tbody>
</table>

Our Methods

|               |                   |       |                |                |                  |
| LOAND         | 2.39 M    2.76 % | - 97.24 % | √              | 51.10 % | -        | 44.10 % | -        |
| LOAND+        | 4.73 M    5.31 % | - 94.69 % | √              | 51.20 % | + 0.10 % | 44.30 % | + 0.20 % |
| LOAND++       | 9.32 M    10.46 % | - 89.54 % | √              | 51.50 % | + 0.40 % | 44.40 % | + 0.30 % |
formance. Tables 1 and 2 demonstrate that LoRand (standard) performs very closely to FULL on large benchmarks with only 1.84% to 2.76% trainable parameters. By tuning less than 3.6M backbone parameters, LoRand (standard) achieves 50.67% (mIoU) on ADE20K, and 51.10% (AP$_{Box}$) / 44.10% (AP$_{Mask}$) on COCO, which is only about 1.5% off on average compared to FULL. LoRand+ and LoRand++ further reduce the gap between these two paradigms to approximately 1% with slight parameter increases. For Swin-L, LoRand saves about 195M parameters per copy compared to FULL. For Swin-B, LoRand saves about 86 M. These results are interesting, which means we do not have to spend plenty of hardware resources to store these redundant parameters. Industrial service providers deliver thousands of model training tasks every day. With LoRand-Tuning, millions of gigabytes per year for model storage could be saved.

3) LoRand effectively broadens the potential of conventional parameter-efficient adapter architectures in dense predictions. From the results, we can draw similar conclusions to [24] that the standard adapter [22] performs worse than fine-tuning on dense predictions. Tables 1 and 2 illustrate that the ADAPTER’s performance is far from FULL, although it reduces 80% of trainable backbone parameters. Also adding new structures, LoRand achieves comparable performance to FULL by training fewer parameters than the ADAPTER. Overall, Tables 1 and 2 demonstrate the feasibility of parameter-efficient tuning paradigm in visual dense prediction tasks.

Comparisons with other fine-tuned backbone. We then show the comparisons of LoRand with some other remarkable fine-tuned backbones in Table 3. Table 3a shows the results based on UperNet and ADE20K, and 3b shows the results based on Cascade MASK R-CNN and COCO. Table 3 shows that LoRand (based on Swin-Transformer) can outperform most existing fine-tuned backbones with less than 2M parameters. Compared to these backbones, LoRand not only presents more robust and superior results but also saves massive hardware resources in this era of parameter explosion. Specifically, LoRand (Swin-T) exceeds COCO by 1.9% (AP$_{Box}$) and 1.2% (AP$_{Mask}$) with 80.12M fewer new backbone parameters than ResNeXt-101-64. Similarly, LoRand (Swin-L) surpasses 5.82% (mIoU) on ADE20K with 40.41M fewer trainable backbone parameters than ResNet-101.

Comparisons on different backbone scales. In addition to Swin-L and Swin-B, we also conduct extensive experiments on Swin-S and Swin-T. We illustrate the performance of baselines and LoRand on multiple backbones. Figure 5 shows the performance of the six methods on different backbone scales, which includes three Swin variants for each benchmark. As FIXED’s performance on COCO and ADE20K is too low to display, we only show FIXED’s re-

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Fine-Tuning Paradigm</th>
<th>AP$_{Box}$</th>
<th>AP$_{Mask}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoRand (Swin-T)</td>
<td>0.88 M</td>
<td>50.2 %</td>
<td>42.9 %</td>
</tr>
<tr>
<td>LoRand (Swin-S)</td>
<td>1.80 M</td>
<td>50.7 %</td>
<td>43.8 %</td>
</tr>
<tr>
<td>LoRand (Swin-B)</td>
<td>2.39 M</td>
<td>51.1 %</td>
<td>44.3 %</td>
</tr>
<tr>
<td>LoRand-Tuning</td>
<td>1.80 M</td>
<td>50.2 %</td>
<td>42.9 %</td>
</tr>
<tr>
<td>LoRand (Swin-T)</td>
<td>2.39 M</td>
<td>49.62 %</td>
<td>51.60 %</td>
</tr>
<tr>
<td>LoRand (Swin-B)</td>
<td>3.59 M</td>
<td>50.67 %</td>
<td>51.10 %</td>
</tr>
<tr>
<td>LoRand (Swin-L)</td>
<td>197 M</td>
<td>53.25 %</td>
<td>51.60 %</td>
</tr>
</tbody>
</table>

Table 3. Comparisons between LoRand-Tuning and Fine-Tuning on ADE20K.

(a) Comparisons between LoRand-Tuning and Fine-Tuning on COCO.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Trained Params$^*$</th>
<th>AP$_{Box}$</th>
<th>AP$_{Mask}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>12 M</td>
<td>39.97 %</td>
<td>42.78 %</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>25 M</td>
<td>44.85 %</td>
<td>44.01 %</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>44 M</td>
<td>49.30 %</td>
<td>49.30 %</td>
</tr>
<tr>
<td>DeiT-S</td>
<td>22 M</td>
<td>51.8 %</td>
<td>44.01 %</td>
</tr>
<tr>
<td>Swin-S</td>
<td>50 M</td>
<td>51.8 %</td>
<td>44.30 %</td>
</tr>
<tr>
<td>Swin-B</td>
<td>88 M</td>
<td>51.8 %</td>
<td>51.8 %</td>
</tr>
<tr>
<td>Swin-L</td>
<td>197 M</td>
<td>53.25 %</td>
<td>53.25 %</td>
</tr>
</tbody>
</table>

(b) Comparisons between LoRand-Tuning and Fine-Tuning on ADE20K.
Figure 5. Seven methods on different backbone scales. Figures show results on PASCAL VOC, COCO, and ADE20K from left to right. Swin-S, Swin-B, and Swin-L are employed as the pre-trained models for PASCAL VOC and ADE20K. Swin-T, Swin-S, and Swin-B are employed for COCO. FIXED’s performances are so low on COCO and ADE20K that they reduce the intuitiveness of the other six methods, so FIXED is only presented in PASCAL VOC comparisons.

Figure 6. Ablation Study for $\alpha$ and $\beta$. $\alpha$ ranges from 2, 4, 6, and $\beta$ ranges from 4, 8, 16. Figures from left to right present experiments on three benchmarks respectively. We only present AP\textsubscript{box} changes for COCO benchmark considering the strong correlation between the values of AP\textsubscript{box} and AP\textsubscript{mask} in COCO.

demonstrate that LoRand can outperform both FULL and traditional adapter structures in low-resource cases and perform very closely to FULL in large benchmarks.

4.3. Ablation Study

In this section, we ablate two key hyperparameters in LoRand: the LoRand branch number $\alpha$ and the kernel matrix dimension $\beta$. $\alpha$ affects the distributed decision-making of LoRand, while $\beta$ focuses on a single branch’s learning capability and consistency.

Several sets of ablation experiments are designed and implemented to investigate the effect of $\alpha$ and $\beta$ on the performance of LoRand. The ablation experiments were conducted on the same three benchmarks. In order to improve the upper limit of LoRand, our experiments are conducted on the largest backbone of each dataset (ADE20K/PASCAL VOC: Swin-L, COCO: Swin-B). The value sets of $\alpha$ and $\beta$ are $\{2, 4, 6\}$ and $\{4, 8, 16\}$. Figure 6 shows the results of ablation studies on three datasets. In most cases, LoRand’s performance increases slightly as $\alpha$ and $\beta$ become larger but hardly outperforms fine-tuning on large benchmarks. Besides, exponentially increasing the size of the LoRand does not result in an equivalent performance improvement and even leads to a reduction ($\alpha=6$ in VOC and COCO). Ablation studies demonstrate that larger LoRands have fewer gains both in parameter efficiency and performance. We have considered this trade-off when designing the LoRand standard, LoRand+, and LoRand++.

5. Conclusion

This paper presents LoRand, a parameter-efficient low-rank adapter for dense predictions, which completely shares the feature understanding of advanced pre-trained models and effectively transfers it to downstream tasks. LoRand performs on par with fine-tuning in COCO instance segmentation, ADE20K semantic segmentation, and PASCAL VOC object detection with only 1% to 3% trainable backbone parameters. Moreover, LoRand effectively avoids the disadvantages of the fine-tuning paradigm and delivers better performance in low-resource situations. We hope that parameter-efficient LoRand can save massive redundant storage resources and facilitate a unified training paradigm for vision and language.
References


[19] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 16000–16009, 2022. 2


