SubTuning: Efficient Finetuning for Multi-Task Learning

Gal Kaplun [1 2] **Andrey Gurevich** [1] **Tal Swisa** [1] **Mazor David** [1] **Shai Shalev-Shwartz** [1 3] **Eran Malach** [1 3]

Abstract

Finetuning a pretrained model has become a standard approach for training neural networks on novel tasks, resulting in fast convergence and improved performance. In this work, we study an alternative finetuning method, where instead of finetuning all the weights of the network, we only train a carefully chosen subset of layers, keeping the rest of the weights frozen at their initial (pretrained) values. We demonstrate that *subset finetuning* (or SubTuning) often achieves accuracy comparable to full finetuning of the model, and even surpasses the performance of full finetuning when training data is scarce. Therefore, SubTuning allows deploying new tasks at minimal computational cost, while enjoying the benefits of finetuning the entire model. This yields a simple and effective method for multi-task learning, where different tasks do not interfere with one another, and yet share most of the resources at inference time. We demonstrate the efficiency of SubTuning across multiple tasks, using different network architectures and pretraining methods.

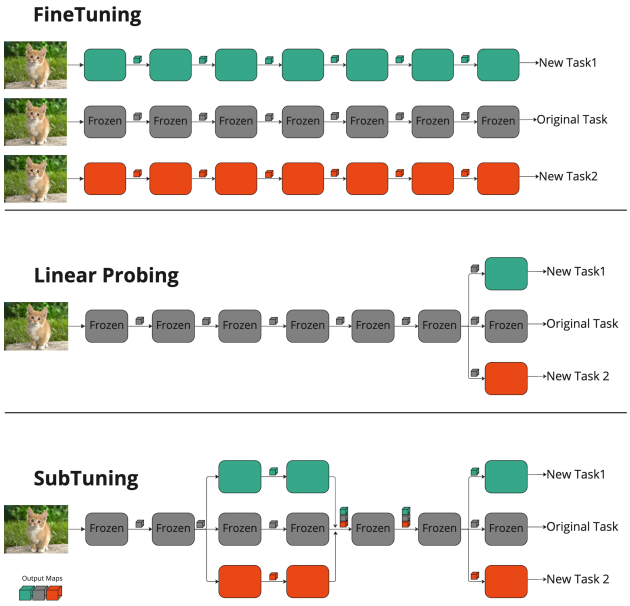

Figure 1. Illustration of finetuning, linear probing and SubTuning in a multi-task setting. In SubTuning we train only a subset of intermediate layers for each task. At inference time, the early layers are shared between all tasks while the tuned layers (green and red) are separate for each task. The output maps of the tuned layers are concatenated in the "batch" axis, and the final layers are applied to the outputs of all tasks, before splitting into task heads.

1. Introduction

Finetuning a large pretrained model has become a widely used method for achieving optimal performance on a diverse range of machine learning tasks in both Computer Vision and Natural Language Processing (Brown et al., 2020; Chowdhery et al., 2022; Wang et al., 2022; Yu et al., 2022). Traditionally, neural networks are trained "from scratch", where at the beginning of the training the weights of the network are randomly initialized. In finetuning, however, we use the weights of a model that was already trained on a different task as the starting point for training on the new task, instead of using random initialization. In this approach, we typically replace the final (readout) layer of the model by a new "head" adapted for the new task, and tune the rest of the model (the backbone), starting from the pretrained weights. The use of a pretrained backbone allows leveraging the knowledge acquired from a large dataset, resulting in improved performance on new tasks.

The finetuning process, while highly effective in improving performance on new tasks, does come with its own set of challenges. One major drawback in the context of multi-task learning (Caruana, 1997; Ruder, 2017) is that once the model is finetuned on a new task, its weights may no longer be suitable for the original source task (a problem known as *catastrophic forgetting* (McCloskey & Cohen, 1989)). Consider for instance the following multi-task setting, which serves as the primary motivation for this paper. Assume we have a large backbone network that was trained on some source task, and is already deployed and running as part

[1]Mobileye Global Inc. [2]Harvard University [3]Hebrew University, Jerusalem, Israel. Correspondence to: Gal Kaplun <galkaplun@g.harvard.edu>.

For github repository see:
https://github.com/talswisa/SubTuning.

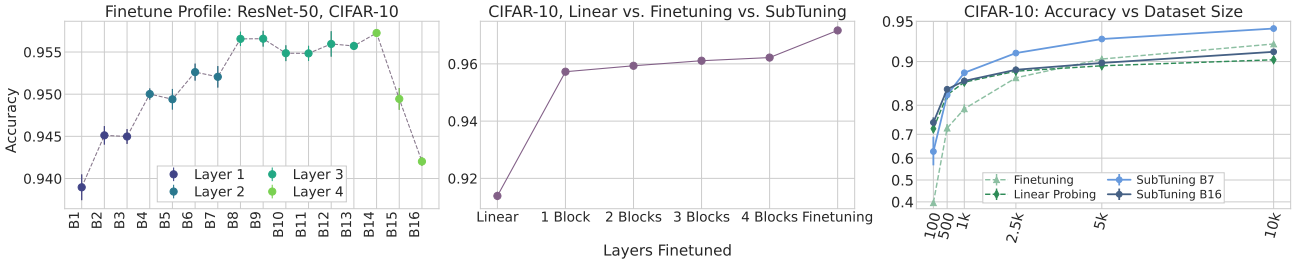

Figure 2. **Left.** The *Finetuning Profile* of ResNet-50 Pretrained on ImageNet and finetuned on CIFAR-10. On the x-axis we have 16 res-blocks where each Layer (with Capital L) corresponds to a drop in spatial resolution. **Middle.** Comparing linear probing, finetuning and SubTuning on CIFAR-10, ResNet-50. Even with few appropriately chosen residual blocks, SubTuning has comparable accuracy to Finetuning. **Right.** Evaluating SubTuning on *Varying Size* subsets of CIFAR-10. When choosing the right layer, SubTuning shows superior performance for every dataset size. We use logit scale for the y-axis to demonstrate differences across multiple accuracy scales.

of our machine learning system. When presented with a new task, we finetune our deployed backbone on this task, and want to run the new finetuned network in parallel to the old one. This presents a problem, as we must now run the same architecture twice, each time with a different set of weights. Doing so doubles the cost both in terms of compute (the number of multiply-adds needed for computing both tasks), and in terms of memory and IO (the number of bits required to load the weights of both models from memory). An alternative would be to perform multi-task training for both the old and new task, but this usually results in degradation of performance on both tasks, with issues such as data balancing, parameters sharing and loss weighting cropping up (Chen et al., 2018; Sun et al., 2020; Sener & Koltun, 2018). Another option is to use a linear layer, or a small network, on top of the original features, but this typically gives inferior performance compared to finetuning (e.g., see Kornblith et al. (2018)).

Our objective is therefore to finetune the model for new tasks with minimal added computational power and memory footprint, without sacrificing much in performance on target tasks. In this work, we study how finetuning can be made more efficient if instead of finetuning all the weights, we finetune only a small subset of intermediate layers. We show that by doing *subset finetuning*, or SubTuning (see Figure 1), we can efficiently deploy new tasks at inference time, with minimal cost in terms of compute, memory and IO.

The key for achieving optimal performance lies in a careful choice of the subset of layers that are finetuned. To do this, we study how each part of the network affects the overall performance as part of the finetuning process. We divide the network into blocks (such as residual blocks in the ResNet architecture or attention layers in a Transformer) and observe how training one block at a time affects the target accuracy (see left side of Figure 2). These experiments allow us to draw the *finetuning profile* of the network (see Section 2), which identifies the best consecutive[1] subset of layers to be trained. Interestingly, the optimal subset of layers can be a set of intermediate layers, where the exact choice of layers depends on the dataset, architecture and setting. Simply choosing the layers closest to the output, or otherwise layers with more parameters, may degrade performance. By choosing the best layers in each setting, we are able to achieve performance comparable to full finetuning of the model, with only a small subset of the layers being trained (see Figure 2 middle).

We demonstrate the efficiency of our method across different downstream tasks, network architectures (convolutional and transformer based networks) and pretraining methods (supervised and self-supervised) in the vision domain. Importantly, we show how SubTuning lets us deploy a network finetuned for a new task alongside the original backbone network, with minimal additional latency during inference time. We further show that SubTuning achieves superior accuracy compared to full finetuning when training on small datasets (see Figure 2 right), and that it significantly outperforms competing transfer learning methods such as Head2Toe (Evci et al., 2022) in a similar setting (see Table 1).

Our Contributions.

- We enhance our understanding of finetuning by introducing the *finetuning profile*, which serves as a crucial tool to illuminate the significance of different layers during finetuning, as detailed in Section 2.
- We propose **SubTuning**, a simple yet effective algorithm that selectively finetunes specific layers based on the *finetuning profile*. We demonstrate its effectiveness and computational efficiency at run-time, as outlined in Section 3.
- We investigate the interplay of SubTuning and dataset size, and observe that in low data regimes SubTuning outper-

[1]We finetune only consecutive blocks as this simplifies our implementation and better utilizes computational resources.

forms both linear probing, finetuning and Head2Toe (Evci et al., 2022), and show that it can also be used in the active learning setting, as presented in Section 4.

- We conduct a thorough ablation study to investigate the impact of initialization, pruning, and reusing frozen features on the performance of SubTuning, as described in Section 5.

1.1. Related Work

Multi-Task Learning. Neural networks are often used for solving multiple tasks. These tasks typically have similar properties, and solving them together allows sharing common features that may capture knowledge that is relevant for all tasks (Caruana, 1997). However, multi-task learning also presents significant challenges, such as negative transfer (Liu et al., 2019), loss balancing (Lin et al., 2022; Michel et al., 2022), optimization difficulty (Pascal et al., 2021), data balancing and shuffling (Purushwalkam et al., 2022). While these problems can be mitigated by careful sampling of the data and tuning of the loss function, these solutions are often fragile (Worsham & Kalita, 2020). In a related setting called *Continual Learning* (Rusu et al., 2016b; Rebuffi et al., 2017b; Kirkpatrick et al., 2017; Kang et al., 2022), adding new tasks needs to happen on-top of previously deployed tasks, while losing access to older data due to storage or privacy constraints, complicating matters even further.

Parameter Efficient Transfer Learning. In recent years, it became increasingly popular to finetune large pretrained models (Devlin et al., 2018; Radford et al., 2021; He et al., 2021b). As the popularity of finetuning these models grows, so does the importance of deploying them efficiently for solving multiple tasks. Thus there has been a growing interest, especially in the NLP domain, in Parameter Efficient Transfer Learning (PETL) (Rusu et al., 2016a; Sung et al., 2022; Evci et al., 2022; Zhang et al., 2019; Radiya-Dixit & Wang, 2020; Mallya & Lazebnik, 2018; Zaken et al., 2021; He et al., 2022) where we either modify a small number of parameters, add a few small layers or mask (Zhao et al., 2020) most of the network. Using only a fraction of the parameters for each task can help in avoiding catastrophic forgetting and can be a good solution for both multi-task learning and continual learning. These methods include Prompt Tuning (Li & Liang, 2021; Lester et al., 2021; Jia et al., 2022), adapters (Houlsby et al., 2019; Rebuffi et al., 2017a; Chen et al., 2022; Rebuffi et al., 2018), LoRA (Hu et al., 2021), sidetuning (Zhang et al., 2019), feature selection (Evci et al., 2022) and masking (Radiya-Dixit & Wang, 2020). Fu et al. (2022) and He et al. (2021a) (see also references therein) attempt to build a unified approach of PETL and suggest improved methods.

Our work draws inspiration from the PETL literature, and to some extent shares the core motivations of the papers mentioned above. That said, many of the PETL works operate in the NLP domain, where using models with billions of parameters has become a common practice, and hence their main objective is reduction in parameter count. Our work focuses on deployment of networks for computer vision, which are typically much smaller in terms of parameters, and so our primary focus is on the *computational efficiency* of the deployed model, rather than on its parameter efficiency. While some of the works above suggest methods that can also improve computational efficiency, unlike previous work, our approach is versatile and can be applied and appended to any architecture, and does not assume sparsity optimized hardware. Finally, our *finetuning profiles* offer unique insight into the mechanisms of finetuning, making our research not only practical but also scientifically illuminating. We also note that SubTuning is compatible with many other PETL methods, and composing SubTuning with methods like LoRA and Head2Toe is a promising research direction that we leave for future work.

In a recent study, Lee et al. (2022) investigated the impact of selective layer finetuning on small datasets and found it to be more effective than traditional finetuning. They observed that the training of different layers produced varied results, depending on the shifts in data distributions. Specifically, they found that when there was a label shift between the source and target data, later layers performed better, but in cases of image corruption, early layers were more effective.

While our work shares some similarities with Lee et al. (2022), the motivations and experimental settings are fundamentally different. Primarily, our research focuses on multi-task learning and aims to develop an efficient finetuning algorithm that can achieve high performance, while Lee et al. (2022) focuses on improved finetuning for small corrupted datasets. Additionally, Lee et al. (2022) finetunes large parts of the network, while in our work we finetune a carefully selected minimal subset of the layers.

2. The Finetuning Profile

The goal of this section is to conduct a comprehensive analysis of the significance of finetuning different components of the network. This analysis guides the choice of the subset of layers to be used for SubTuning. To accomplish this, we run a series of experiments in which we fix a specific subset of consecutive layers within the network and finetune only these layers, while maintaining the initial (pretrained) weights for the remaining layers.

For example, we take a ResNet-50 model that was trained on the ImageNet dataset, and finetune it on the CIFAR-10 dataset, replacing the readout layer of ImageNet (which has 1000 classes) by a readout layer adapted to CIFAR-10 (with 10 classes). As noted, in our experiments we do not finetune

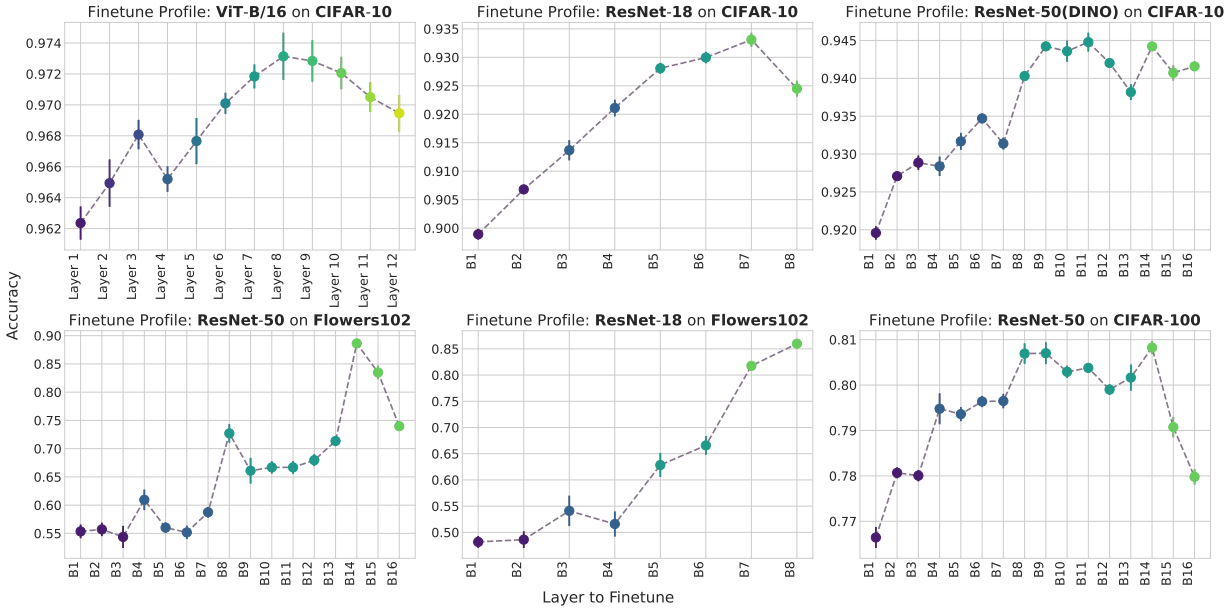

Figure 3. Finetuning profiles for different architectures, initializations and datasets.

all the weights of the network, but rather optimize only a few layers from the model (as well as the readout layer). Specifically, as the ResNet-50 architecture is composed of 16 blocks (i.e., *ResBlocks*, see Appendix A and He et al. (2015) for more details), we choose to run 16 experiments, where in each experiment we train only one block, fixing the weights of all other blocks at their initial (pretrained) values. We then plot the accuracy of the model as a function of the block that we train, as presented in Figure 2 left. We call this graph the *finetuning profile* of the network. Following a similar protocol (see Appendix A), we compute *finetuning profiles* for various combinations of architectures (ResNet-18, ResNet-50 and ViT-B/16), pretraining methods (supervised and DINO (Caron et al., 2021)), and target tasks (CIFAR-10, CIFAR-100 and Flower102). In Figure 3, we present the profiles for the different settings.

Results. Interestingly, our findings indicate that for most architectures and datasets, the importance of a layer cannot be predicted by simply observing properties such as the depth of the layer, the number of parameters in the layer or its spatial resolution. In fact, the same architecture can have very different *finetuning profiles* when trained on a different downstream task or from different initialization (see Figure 3). While we find that layers closer to the input tend to contribute less to the finetuning process, the performance of the network typically does not increase monotonically with the depth or with the number of parameters[2], and after a certain point the performance often starts *decreasing* when training deeper layers. For example, in the finetuning profile of ResNet-50 finetuned on the CIFAR-10 dataset (Figure 2 left), we see that finetuning `Block 13` results in significantly better performance compared to optimizing `Block 16`, which is deeper and has many more parameters.

To further investigate the behavior of finetuning, we also look into the effect of finetuning more consecutive blocks. We plot similar fintuning profiles, but this time training groups of 2 and 3 consecutive blocks together. In Figure 9 (in Appendix B) we present the finetuning profiles for training groups of 2 and 3 consecutive blocks, each time starting from a block deeper into the network. The results indicate that finetuning more layers improves performance, and also makes the finetuning profile more monotonic.

3. Efficient MTL with SubTuning

In the preceding section, we demonstrated that different layers have different impact on the overall performance of the finetuned model. These variations are encapsulated in the *fintetuning profile* of the network. Analyzing these profiles reveals that we can achieve high accuracy without the need to train all the parameters of the network, if we choose correctly which layers to train.

Based on these observations, we devise a simple yet efficient method for finetuning, referred to as SubTuning, where instead of finetuning all layers of the network, we only finetune a selected subset of layers. Although not mandatory, for computational reasons we always train subsets of *consecutive* layers. Additionally, since different tasks have different specifications (e.g., number of classes) we also train a linear readout layer over the last backbone layer for each new task

[2]In the ResNet architectures, deeper blocks have more parameters, while for ViT all layers have the same amount of parameters.

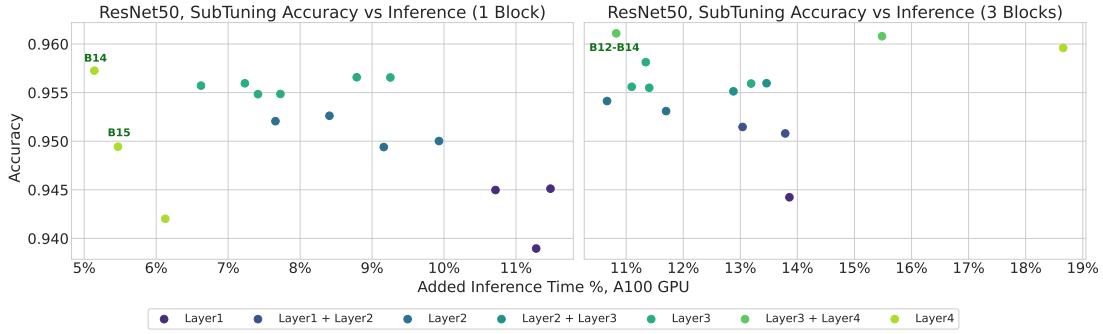

Figure 4. Inference vs CIFAR-10 accuracy on A100 gpu with batch size of 1 and input resolution of 224.

(see Figure 1). In this section, we will examine the advantages of SubTuning in terms of computational efficiency, both at inference time and during training.

3.1. Computationally Efficient Finetuning

We will now demonstrate how SubTuning improves the computational efficiency of the network at *inference time*, which is the primary motivation for this study. We discuss the improvements during *training time* in Section 3.2.

Let us consider the following setting of multi-task learning. We trained a network f_θ on some task. The network gets an input $\mathbf{x}$ and returns an output $f_\theta(\mathbf{x})$. We now want to train a new network on a different task by finetuning the weights θ, resulting in a new set of weights $\widetilde{\theta}$. Now, at inference time, we receive an input $\mathbf{x}$ and want to compute both $f_\theta(\mathbf{x})$ and $f_{\widetilde{\theta}}(\mathbf{x})$ with minimal compute budget. Since we cannot expect the overall compute to be lower than just running $f_\theta(\mathbf{x})$[3], we only measure the *additional* cost of computing $f_{\widetilde{\theta}}(\mathbf{x})$, given that $f_\theta(\mathbf{x})$ is already computed.

Since inference time heavily depends on various parameters such as the hardware used for inference (e.g., CPU, GPU, FPGA), the hardware parallel load (i.e., how many jobs the machine is currently running), the network compilation (i.e., kernel fusion) and the batch size, we will conduct a crude analysis of the compute requirements (see in depth discussion in Li et al. (2021)). The two main factors that contribute to computation time are: 1) **Computational cost**, or the number of multiply-adds (FLOPs) needed to compute each layer and 2) **IO**, which refers to the number of bits required to read from memory to load each layer's weights.

If we perform full finetuning of all layers, in order to compute $f_{\widetilde{\theta}}(\mathbf{x})$ we need to double both the computational cost and the IO, as we are now effectively running two separate networks, f_θ and $f_{\widetilde{\theta}}$, with two separate sets of weights. Note that this does not necessarily mean that the computation-time is doubled, since most hardware used for inference does significant parallelization, and if the hardware is not fully utilized when running $f_\theta(\mathbf{x})$, the additional cost of running $f_{\widetilde{\theta}}(\mathbf{x})$ in parallel might be smaller. However, in terms of additional compute, full finetuning is the least optimal thing to do.

Consider now the computational cost of SubTuning. Denote by N the number of layers in the network, and assume that the parameters $\widetilde{\theta}$ differ from the original parameters θ only in the layers ℓ_{start} through ℓ_{end} (where $1 \leq \ell_{\text{start}} \leq \ell_{\text{end}} \leq N$). Let us separate between two cases: 1) ℓ_{end} is the final layer of the network and 2) ℓ_{end} is some intermediate layer.

The case where ℓ_{end} is the final layer is the simplest: we share the entire compute of $f_\theta(\mathbf{x})$ and $f_{\widetilde{\theta}}(\mathbf{x})$ up until the layer ℓ_{start} (so there is zero extra cost for layers below ℓ_{start}), and then we "fork" the network and run the layers of f_θ and $f_{\widetilde{\theta}}$ in parallel. In this case, both the compute and the IO are doubled *only* for the layers between ℓ_{start} and ℓ_{end}.

In the second case, where ℓ_{end} is some intermediate layer, the computational considerations are more nuanced. As in the previous case, we share the entire computation before layer ℓ_{start}, with no extra compute. Then we "fork" the network, paying double compute and IO for the layers between ℓ_{start} and ℓ_{end}. For the layers after ℓ_{end}, however, we can "merge" back the outputs of the two parallel branches (i.e., concatenating them in the "batch" axis), and use the same network weights for both outputs. This means that for the layers after ℓ_{end} we double the compute (i.e., in FLOPs), but the IO remains the same (by reusing the weights for both outputs). This mechanism is illustrated in Figure 1.

More formally, let c_i be the computational-cost of the i-th layer, and let s_i be the IO required for the i-th layer. To get a rough estimate of how the IO and compute affect the backbone run-time, consider a simple setting where compute and IO are parallelized. Thus, while the processor computes layer i, the weights of layer $i+1$ are loaded into memory.

[3]Optimizing the compute budget of a single network is outside the scope of this paper.

The total inference time of the backbone is therefore:

$$\text{Compute} = \max(2s_{\ell_{\text{start}}}, c_{\ell_{\text{start}}-1}) + \sum_{i=\ell_{\text{start}}}^{\ell_{\text{end}}} 2\max(c_i, s_{i+1}) + \sum_{i=\ell_{\text{end}}+1}^{N-1} \max(2c_i, s_{i+1}) + 2c_N$$

Thus, both deeper and shallower layers can be optimal for SubTuning, depending on the exact deployment environment, workload and whether we are IO or compute bound. To give a concrete example, we conduct an experiment using a ResNet-50 backbone on an NVIDIA A100-SXM-80GB GPU with a batch size of 1 and input resolution 224. We finetune 1 and 2 consecutive res-blocks and plot the accuracy against the added inference cost, as seen in Figure 4. We see that using this methodology, we are able to achieve a significant improvement in performance, with minimal computational cost. However, it is important to mention that the exact choice of which layer gives the optimal accuracy/latency tradeoff can heavily depend on the deployment environment, as the runtime estimation may vary depending on factors such as hardware, job load, and software stack. While this is admittedly a limitation of our method, we argue that the flexibility of SubTuning allows us to use the same methodology across a wide range of deployment environments, providing a powerful tool for effective finetuning with small compute overhead.

3.2. Training Time Efficiency

In addition to inference considerations, SubTuning has also favorable properties at train time, especially when the subset of finetuned layers is closer to the output. For concreteness, let ℓ_{start} be the first layer we finetune. Then, during backpropagation, we only need to run the forward pass for layers 1 through $\ell_{\text{start}} - 1$, without computing the gradient for these layers, thus reducing both the memory footprint and the computational burden during training.

Furthermore, in many cases we can finetune even faster by running once a forward pass on all the data, storing the activations of $\ell_{\text{start}} - 1$ and training the selected layers on top of the stored frozen activations. Note that using frozen activation features does not allow using data augmentation during training time (although some augmentations, such as random crop and flip, can usually be performed on the saved features). However, in a setting where complex augmentations during training time are not required, saving features can significantly speed up the training.

4. SubTuning in Low-Data Regime

Thus far, we studied settings where SubTuning has shown comparable or inferior performance to full finetuning. While this is typically the case when downstream data is abundant, in low-data scenarios full finetuning often leads to suboptimal results, and simpler methods such as linear probing are more effective, as seen, e.g., in Evci et al. (2022).

We proceed to show that when training data is limited, SubTuning with appropriate layer choice performs significantly better than both linear probing and finetuning. The compact nature of SubTuning, in terms of parameter count, makes it suitable for the low-data regime. On the other hand, unlike linear probing, it has the ability to express complex functions by tuning parameters deep inside the network. Finally, we demonstrate the efficacy of SubTuning for pool-based Active Learning (AL) (McCallum et al., 1998), where a pool of unlabeled examples can be labeled upon demand.

Table 1. Comparing SubTuning with other methods, 1k datasets.

	CIFAR-100		Flowers102	
	ResNet-50	ViT-B/16	ResNet-50	ViT-B/16
Linear	48.5	55.0	84.8	72.1
Ours	**51.3**	**65.5**	**92.4**	**95.5**
Head2Toe	47.1	58.2	85.6	85.9
Finetune	33.2	44.3	85.2	54.1

4.1. Evaluating SubTuning in Low-Data Regimes

To thoroughly evaluate the performance of SubTuning in low-data regimes, we fix a pretrained model and run experiments using linear probing, SubTuning, and full finetuning on varying amounts of data. As an example, in Figure 2 (right), we use a ResNet-50 model pretrained on ImageNet to compare these three approaches on the CIFAR-10 dataset, with dataset sizes ranging from 100 to 10k examples. The results (see Figure 2 and Figure 12 in Appendix B), show that by carefully selecting the appropriate layer, SubTuning can surpass both finetuning and linear probing across a variety of dataset sizes. This illustrates that SubTuning is a flexible approach that allows us to adapt the model to the task and dataset size. We observe that in general, layers closer to the output work better when the dataset size is very small, but as the number of data points grow, earlier layers perform better.

We compare the performance of SubTuning to Head2Toe (Evci et al., 2022), a recent method for bridging the gap between linear probing and finetuning in the low-data regime, which operates by selecting features from layers deep in the network. We use an experimental setting similar to Evci et al. (2022), training on 1k examples from CIFAR-100 and Flowers102 datasets. We SubTune a single block (ResNet-50) or layer (ViT-B/16), and report the best performing block/layer along with the performance of Head2Toe, Linear Probing and full finetuning (as reported in Evci et al. (2022)). As seen in Table 1, in this setting SubTuning outperforms all

methods by a large margin[4].

4.2. Active Learning with SubTuning

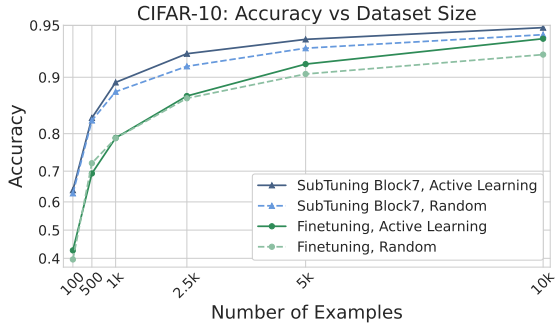

Figure 5. ResNet-50 pretrained on ImageNet with SubTuning on CIFAR-10 using Active Learning (AL) (see Section 4.2). We used logit scale for the y-axis to visualize the differences between multiple accuracy scales.

We saw that SubTuning is a superior method compared to both finetuning and linear probing when the amount of labeled data is limited. We now further explore the advantages of SubTuning in the pool-based AL setting, where a large pool of unlabeled data is readily available, and additional examples can be labeled to improve the model's accuracy. It is essential to note that in real-world scenarios, labeling is a costly process, requiring domain expertise and a significant amount of manual effort. Therefore, it is crucial to identify the most informative examples to optimize the model's performance (Katharopoulos & Fleuret, 2018).

A common approach in this setting is to use the model's uncertainty to select the best examples (Lewis & Gale, 1994; Joshi et al., 2009a; Campbell et al., 2000; Joshi et al., 2009b). The process of labeling examples in AL involves iteratively training the model using all labeled data, and selecting the next set of examples to be labeled using the model. This process is repeated until the desired performance is achieved or the budget for labeling examples is exhausted.

In our experiments (see details in Appendix A), we select examples according to their classification margin. We start with 100 randomly selected examples from the CIFAR-10 dataset. At each iteration we select and label additional examples, training with 500 to 10,000 labeled examples that were iteratively selected according to their margin. For example, after training on the initial 100 randomly selected examples, we select the 400 examples with the lowest classification margin and reveal their labels. We then train on the 500 labeled examples we have, before selecting another 500 examples to label to reach 1k examples. In Figure 5 we compare the performance of the model when trained on examples selected by our margin-based rule, compared to training on subsets of randomly selected examples (similar to the results in Section 4.1). We also compare our method to full finetuning with and without margin-based selection of examples. Evidently, we see that using SubTuning for AL outperforms full finetuning, and that the selection criterion we use gives significance boost in performance.

Efficient Active Learning with Fast Inference. SubTuning for AL not only improves performance, but also allows for a more efficient implementation of the example selection method within the AL algorithm. Normally, to select the next examples to label, the algorithm needs to compute the margin of the current model, which requires running a forward-pass of the model over all the unlabeled dataset. This can be a time-consuming and costly process, especially if the full model is trained in each iteration. However, when using SubTuning, we can speed this process up by caching the activations of the original network in the layer before the block that we use for SubTuning. These activations are computed once, and can be reused across different tasks and different iterations of the AL algorithm, significantly reducing the time and computation needed to compute the margins for all the examples. This highlights another positive property of SubTuning: it allows fast inference over an offline dataset. This is useful for gathering different statistical metrics, retrieval and hard negative mining.

5. Extensions and Ablations

In this section, we propose several enhancements to SubTuning, and also evaluate the significance of certain design choices. In Section 5.1 we present an extension to SubTuning that combines the original network's features with newly finetuned layers, resulting in improved accuracy. In Section 5.2, we explore techniques to reduce the cost of SubTuning by implementing pruning, a process that removes unnecessary channels, without compromising much on performance. Finally, in Section 5.3, we delve into the critical role of weight initialization in SubTuning. For the full experimental setup refer to Appendix A.

5.1. Siamese SubTuning

In the MTL setting (Section 3), we assume that $f_\theta(\mathbf{x})$ is already computed, and we want to minimize the cost of computing $f_{\widetilde{\theta}}(\mathbf{x})$, while preserving good performance. Since $f_\theta(\mathbf{x})$ is computed anyway, its features are available at no extra cost[5], and we can combine them with the new features. To achieve this, we concatenate the representations given by $f_\theta(\mathbf{x})$ and $f_{\widetilde{\theta}}(\mathbf{x})$ before inserting them into the classification

[4]Mean accuracy across 5 runs. Note, the settings in our and the H2T experiments are slightly different. E.g., H2T use subsets from VTAB-1k and we do random selection. Also, pretrained weights might differ due to library difference (TensorFlow vs PyTorch).

[5]Neglecting the extra cost of the increased linear layer.

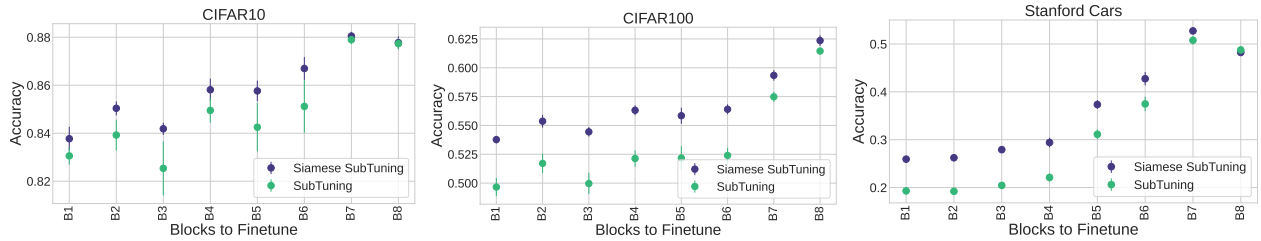

Figure 6. The impact of Siamese SubTuning on ResNet-18 when using 5,000 randomly selected training samples from each dataset.

head. We call this method Siamese SubTuning.

We evaluated the effectiveness of Siamese SubTuning (See illustraion in Figure 13 in Appendix B.4.1) on multiple datasets and found it to be particularly beneficial in scenarios where data is limited. As demonstrated in Figure 6, when finetuning on 5,000 randomly selected training samples from the CIFAR-10, CIFAR-100, and Stanford Cars datasets, Siamese SubTuning with ResNet-18 outperforms standard SubTuning. It is worth noting that both SubTuning and Siamese SubTuning significantly improve performance when compared to linear probing in this setting. For instance, linear probing on top of ResNet-18 on CIFAR-10 achieves 79% accuracy, where Siamese SubTuning achieves 88% accuracy in the same setting. See also ResNet-50 results in Figure 14 and Layer-wise (as opposed to block-wise) results are reported in Figure 15 in the Appendix B.4.1.

5.2. SubTuning with Pruning

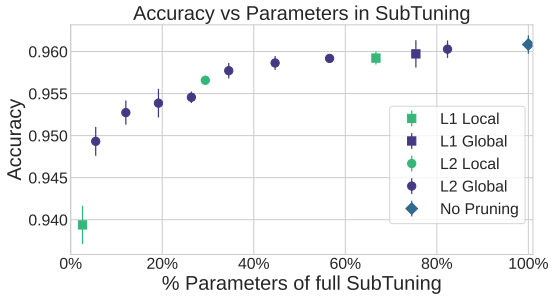

Figure 7. SubTuning with channel-wise pruning on the last 3 blocks of ResNet-50. We plot the accuracy vs the pruning rate of different pruning techniques (Global vs. Local and pruning norm). The figure shows the best values for each parameter fraction.

So far, we demonstrated the effectiveness of SubTuning in reducing the cost of adding new tasks for MTL while maintaining high performance on those tasks. To further optimize computational efficiency and decrease the model size for new tasks, we propose utilizing channel pruning on the SubTuned component of the model. We employ two types of pruning, local and global pruning, to reduce the parameter size and runtime of the model while preserving its accuracy. Local pruning removes an equal portion of channels for each layer, while global pruning eliminates channels across the network disregarding how many channels are removed per layer. Similar to Li et al. (2016), for both pruning techniques we prune the weights with the lowest L1 and L2 norms so that we can meet the target pruning ratio.

In Figure 7 we illustrate the effectiveness of combining channel pruning with SubTuning on the last 3 blocks of ResNet-50. Instead of copying the weights and then training the blocks, as we have done so far, we add an additional step of pruning before the training. This way we only prune the original, frozen, network once for all future tasks. Our results show that pruning is effective across different parameter targets, reducing the cost with only small performance degradation. For example, when using less than 3% of the last 3 blocks (about 2% of all the parameters of ResNet-50) we maintain 94% accuracy on the CIFAR-10 dataset. In the same setting, linear probing gets about 91% accuracy. For more pruning results see Figure 16 in the Appendix.

5.3. Effect of Random Re-initialization

We now show that initializing the weights of a model with the pretrained weights from a different task improves the performance and speed of training when SubTuning it on a new task. Specifically, in Figure 17 (in the Appendix) we finetune ResNet-50 pretrained on ImageNet, on the CIFAR-10 dataset. We compare this to randomly reinitializing the weights and show that, similar to standard finetuning, the pretrained weights led to a significant improvement in performance and faster convergence. We also find that the earlier the layers we finetuned, the greater the improvement is. This study demonstrates that using pretrained weights as initialization for SubTuning can lead to faster and better results. We note that even with longer training time of 80 epochs, random initialization achieves poor performance.

6. Discussion

The wide adoption of neural networks for solving a multitude of tasks has drastically changed software development methodologies in AI-centric applications and products.

More and more parts of the code are replaced by trained neural networks, instead of "traditional" algorithms. Some even predict that in many applications, all code will eventually be replaced by a single large neural network, completing the transition to Software 2.0 (Karpathy, 2017).

While the benefits of embedding neural networks into the software stack are clear, this also poses some immediate challenges. In traditional software development, multiple teams can develop each part of the system independently from one another, where interfaces between parts of the code are clearly defined and conflicts can be safely handled by version control systems (e.g., git). When replacing software with learned networks, such independence is hard to maintain. Independent teams building a single network for solving different tasks need to coordinate a single training cycle each time a task is changed, and every task can negatively affect any other task.

We believe that SubTuning can be a viable methodology for addressing this challenge. SubTuning allows developers to "fork" existing deployed networks and develop new tasks on top of the trained network, without interfering with the "code" of other teams that may also be using the network. This enables independent development of different tasks while maximizing performance, sharing of knowledge between tasks, and minimizing the computational cost of deploying new tasks. Thus, efficient finetuning methods like SubTuning can play a key role in transforming software development in the era of neural networks.

7. Conclusions and Future Work

In this paper we introduced SubTuning, an efficient finetuning method that allows deployment of new tasks, without interfering with other tasks that share the same pretrained network. We showed that finetuning only a small subset of the network's layers can reach performance that is competitive with finetuning all the layers of the network, and in some cases can even drastically improve performance compared to full finetuning. This opens up a few promising directions for future research.

First, we note that in this work we studied selective finetuning of a subset of *consecutive* layers. While this choice greatly simplifies our method, training a subset of non-consecutive layers can potentially improve performance even further. Therefore, an interesting direction for future work is to investigate the benefit from training arbitrary subsets of layers, as well as developing an algorithm for choosing the optimal subset.

Another promising direction for future research is to better understand the interplay between the architecture, the data, and the optimal subset of layers to be finetuned. As the study of the *finetuning profiles* shows, different architectures and datasets can have different "preferred" layers for finetuning. In this paper, the method for choosing the best layer is simply to try all the options of consecutive subsets of layers. However, it might be possible to find measures, such as the gradient of the loss or the change in the weights during training, that can assist in choosing the correct subset, without performing a full training cycle for each subset.

8. Acknowledgments

We would like to thank Preetum Nakkiran, Asaf Noy, Asaf Maman and Itay Benou for comments on early version of the draft. GK is currently supported by a Simons Investigator Fellowship, NSF grant DMS-2134157, DARPA grant W911NF2010021,and DOE grant DE-SC0022199. GK is also grateful for support from Oracle Labs and past support by the NSF, as well as the Packard and Sloan foundations and the BSF.

References

Balcan, M.-F., Broder, A., and Zhang, T. Margin based active learning. In *International Conference on Computational Learning Theory*, pp. 35–50. Springer, 2007.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

Campbell, C., Cristianini, N., Smola, A., et al. Query learning with large margin classifiers. In *ICML*, volume 20, pp. 0, 2000.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. Emerging properties in self-supervised vision transformers, 2021. URL https://arxiv.org/abs/2104.14294.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Chen, H., Tao, R., Zhang, H., Wang, Y., Ye, W., Wang, J., Hu, G., and Savvides, M. Conv-adapter: Exploring parameter efficient transfer learning for convnets, 2022. URL https://arxiv.org/abs/2208.07463.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International*

conference on machine learning, pp. 794–803. PMLR, 2018.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. Palm: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

Evci, U., Dumoulin, V., Larochelle, H., and Mozer, M. C. Head2toe: Utilizing intermediate representations for better transfer learning. *CoRR*, abs/2201.03529, 2022. URL https://arxiv.org/abs/2201.03529.

Fang, G. Torch-Pruning, 7 2022. URL https://github.com/VainF/Torch-Pruning.

Fu, Z., Yang, H., So, A. M.-C., Lam, W., Bing, L., and Collier, N. On the effectiveness of parameter-efficient fine-tuning. *arXiv preprint arXiv:2211.15583*, 2022.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021a.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. B. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021b. URL https://arxiv.org/abs/2111.06377.

He, X., Li, C., Zhang, P., Yang, J., and Wang, X. E. Parameter-efficient model adaptation for vision transformers, 2022. URL https://arxiv.org/abs/2203.16329.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751, 2019. URL http://arxiv.org/abs/1902.00751.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685.

Jia, M., Tang, L., Chen, B.-C., Cardie, C., Belongie, S., Hariharan, B., and Lim, S.-N. Visual prompt tuning, 2022. URL https://arxiv.org/abs/2203.12119.

Joshi, A. J., Porikli, F., and Papanikolopoulos, N. Multi-class active learning for image classification. In *2009 ieee conference on computer vision and pattern recognition*, pp. 2372–2379. IEEE, 2009a.

Joshi, A. J., Porikli, F., and Papanikolopoulos, N. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2372–2379, 2009b. doi: 10.1109/CVPR.2009.5206627.

Kang, M., Park, J., and Han, B. Class-incremental learning by knowledge distillation with adaptive feature consolidation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16071–16080, 2022.

Karpathy, A. Software 2.0, 2017. URL https://karpathy.medium.com/software-2-0-a64152b37c35.

Katharopoulos, A. and Fleuret, F. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Kornblith, S., Shlens, J., and Le, Q. V. Do better imagenet models transfer better? *CoRR*, abs/1805.08974, 2018. URL http://arxiv.org/abs/1805.08974.

Krause, J., Stark, M., Deng, J., and Fei-Fei, L. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

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

Lee, Y., Chen, A. S., Tajwar, F., Kumar, A., Yao, H., Liang, P., and Finn, C. Surgical fine-tuning improves adaptation to distribution shifts, 2022. URL https://arxiv.org/abs/2210.11466.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *CoRR*, abs/2104.08691, 2021. URL https://arxiv.org/abs/2104.08691.

Lewis, D. D. and Gale, W. A. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pp. 3–12, Berlin, Heidelberg, 1994. Springer-Verlag. ISBN 038719889X.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets, 2016. URL https://arxiv.org/abs/1608.08710.

Li, S., Tan, M., Pang, R., Li, A., Cheng, L., Le, Q. V., and Jouppi, N. P. Searching for fast model families on datacenter accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8085–8095, 2021.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190, 2021. URL https://arxiv.org/abs/2101.00190.

Lin, B., YE, F., and Zhang, Y. A closer look at loss weighting in multi-task learning, 2022. URL https://openreview.net/forum?id=OdnNBNIdFul.

Liu, S., Liang, Y., and Gitter, A. Loss-balanced task weighting to reduce negative transfer in multi-task learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 9977–9978, 2019.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization, 2017. URL https://arxiv.org/abs/1711.05101.

Mallya, A. and Lazebnik, S. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *CoRR*, abs/1801.06519, 2018. URL http://arxiv.org/abs/1801.06519.

Marcel, S. and Rodriguez, Y. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia*, pp. 1485–1488, 2010.

McCallum, A., Nigam, K., et al. Employing em and pool-based active learning for text classification. In *ICML*, volume 98, pp. 350–358. Citeseer, 1998.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Michel, P., Ruder, S., and Yogatama, D. Balancing average and worst-case accuracy in multitask learning, 2022. URL https://openreview.net/forum?id=H_qwVb8DQb-.

Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

Pascal, L., Michiardi, P., Bost, X., Huet, B., and Zuluaga, M. A. Optimization strategies in multi-task learning: Averaged or separated losses? *CoRR*, abs/2109.11678, 2021. URL https://arxiv.org/abs/2109.11678.

Purushwalkam, S., Morgado, P., and Gupta, A. The challenges of continuous self-supervised learning. *arXiv preprint arXiv:2203.12710*, 2022.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020.

Radiya-Dixit, E. and Wang, X. How fine can fine-tuning be? learning efficient language models. *CoRR*, abs/2004.14129, 2020. URL https://arxiv.org/abs/2004.14129.

Rebuffi, S., Bilen, H., and Vedaldi, A. Learning multiple visual domains with residual adapters. *CoRR*, abs/1705.08045, 2017a. URL http://arxiv.org/abs/1705.08045.

Rebuffi, S., Bilen, H., and Vedaldi, A. Efficient parametrization of multi-domain deep neural networks. *CoRR*, abs/1803.10082, 2018. URL http://arxiv.org/abs/1803.10082.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017b.

Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *CoRR*, abs/1606.04671, 2016a. URL http://arxiv.org/abs/1606.04671.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016b.

Sener, O. and Koltun, V. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

Sun, X., Panda, R., Feris, R., and Saenko, K. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33:8728–8740, 2020.

Sung, Y.-L., Cho, J., and Bansal, M. Lst: Ladder side-tuning for parameter and memory efficient transfer learning, 2022. URL https://arxiv.org/abs/2206.06522.

Wang, W., Bao, H., Dong, L., Bjorck, J., Peng, Z., Liu, Q., Aggarwal, K., Mohammed, O. K., Singhal, S., Som, S., and Wei, F. Image as a foreign language: Beit pretraining for all vision and vision-language tasks, 2022. URL https://arxiv.org/abs/2208.10442.

Worsham, J. and Kalita, J. Multi-task learning for natural language processing in the 2020s: where are we going? *Pattern Recognition Letters*, 136:120–126, 2020.

Yu, J., Wang, Z., Vasudevan, V., Yeung, L., Seyedhosseini, M., and Wu, Y. Coca: Contrastive captioners are image-text foundation models, 2022. URL https://arxiv.org/abs/2205.01917.

Zaken, E. B., Ravfogel, S., and Goldberg, Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *CoRR*, abs/2106.10199, 2021. URL https://arxiv.org/abs/2106.10199.

Zhang, J. O., Sax, A., Zamir, A., Guibas, L. J., and Malik, J. Side-tuning: Network adaptation via additive side networks. *CoRR*, abs/1912.13503, 2019. URL http://arxiv.org/abs/1912.13503.

Zhao, M., Lin, T., Jaggi, M., and Schütze, H. Masking as an efficient alternative to finetuning for pretrained language models. *CoRR*, abs/2004.12406, 2020. URL https://arxiv.org/abs/2004.12406.

A. Experimental Setup

For experiments throughout the paper we used a fixed experimental setting presented in Table 2. We focus on short training with 10 epochs, using the AdamW (Loshchilov & Hutter, 2017) optimizer on image resolution of 224 using random resized crop to transform from lower to larger resolution. We also employ random horizontal flip and for Flowers102 we use random rotation up to probability 30%. We make two exceptions to this setting. One, in Section 4, due to scarce data, we use a 50 epoch training. The second is, in Subsection 5.3 we report both the 10 epoch and a longer, 80 epoch, training results. This is done to prove that random weight runs are not under performing due to short training time. We run our experiments on the CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), Flower102 (Nilsback & Zisserman, 2008) and Standford Cars (Krause et al., 2013) datasets.

Table 2. Training Parameters. For ViT-B/16 we use two sets of parameters. One for a full length datasets and the other for small datasets with 1k training examples (Table 1).

	ResNet	ViT-B/16 full datasets	ViT-B/16 1k examples
Learning Rate	0.001	0.0001	0.001
Weight Decay	0.01	0.00005	0.01
Batch Size	256	256	256
Optimizer	AdamW	AdamW	AdamW
Scheduler	Cosine Annealing	Cosine Annealing	Cosine Annealing

Pretrained Weights For ResNet-50, ResNet-18 (He et al., 2015) and ViT-B/16 (Dosovitskiy et al., 2020) we use pretrained weights using the default TorchVision implementation (Marcel & Rodriguez, 2010) pretrained on ImageNet (Deng et al., 2009). For the DINO ResNet-50 (Caron et al., 2021), we used the official paper github's weights https://github.com/facebookresearch/dino.

Finetuning Profiles. To generate the finetuning profiles we only train the appropriate subset of subsequent residual blocks (for ResNets) and Attention layers (For ViT) in addition to training an appropriate linear head. For example, for ResNet-18, there are 8 residual blocks (ResBlocks), 2 in each layer or spatial resolution (see full implementation here: link.). In general, each such block consists of a few Convolution layers with a residual connection that links the input of the block and the output of the Conv-layers. Similarly, ResNet-50 has 16 blocks, [3, 4, 6, 3], for layers [1, 2, 3, 4] respectively. For ViT-B/16, there are naturally 12 attention layers and we train one (or few) layer at a time.

Inference Time Measurement. We measure inference time on a single NVIDIA A100-SXM-80GB GPU with a batch size of 1 and input resolution 224. We warm up the GPU for 300 iteration and run 300 further iterations to measuring the run time. Since measuring inference time is inherently noisy, we make sure the number of other processes running on the GPU stays minimal and report the mean time out of 10 medians of 300. We attach figures for absolution times in Figure 10.

Pruning. We use the Torch-Pruning library (Fang, 2022) to apply both local and global pruning, using L1 and L2 importance factors. We conduct a single iteration of pruning, varying the channel sparsity factor between 0.1 and 0.9 in increments of 0.1, and selecting the highest accuracy value for every 5% range total SubTuning params.

Active Learning In our experiments, we select examples according to their classification margin. At every iteration, after SubTuning our model on the labeled dataset, we compute the classification margin for any unlabeled example (similar to the method suggested in Balcan et al. (2007); Joshi et al. (2009a); Lewis & Gale (1994)). That is, given some example x, let $P(y|x)$ be the probability that the model assigns for the label y when given the input x[6]. Denote by y_1 the label with the maximal probability and by y_2 the second-most probable label, namely $y_1 = \max_y P(y|x)$ and $y_2 = \max_{y \neq y_1} P(y|x)$. We define the classification margin of x to be $P(y_1|x) - P(y_2|x)$, which captures how confident the model is in its prediction (the lower the classification margin, the less confident the model is). We select the examples that have the smallest classification margin (examples with high uncertainty) as the ones to be labeled.

[6]We focus on classification problems, where the model naturally outputs a probability for each label given the input. For other settings, other notions of margin may apply.

In our AL experiments we start with 100 randomly selected examples of the CIFAR-10 dataset. At each iteration we select and label additional examples, training with 500, 1000, 2500, 5000, and 10,000 labeled examples that were iteratively selected according to their margin. That is, after training on 100 examples, we choose the next 400 examples to be the ones closest to the margin, train a new model on the entire 500 examples, use the new model to select the next 500 examples, and so on. In Figure 5 we compare the performance of the model when trained on examples selected by our margin-based rule, compared to training on subsets of randomly selected examples (similar to the results presented in Section 4.1). We also compare our method to full finetuning with and without margin-based selection of examples.

B. Additional Experiments

B.1. Additional Finetuning Profiles

In this subsection we provide some more SubTuining profiles. We validate that longer training does not affect the ViT results, reporting the results in Figure 8. In Figure 9 we provide SubTuning results for 2 and 3 consecutive blocks. We show that using more blocks improves the classification accuracy, and makes the choice of later blocks more effective.

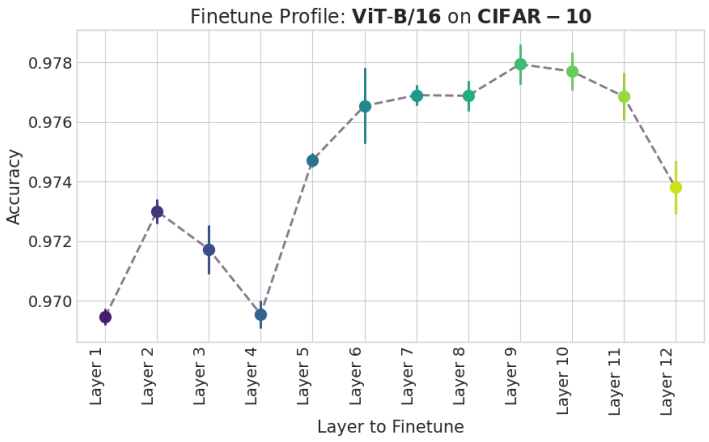

Figure 8. Finetuning profile trained with ViT-B/16 on Cifar10 trained for 40 epochs.

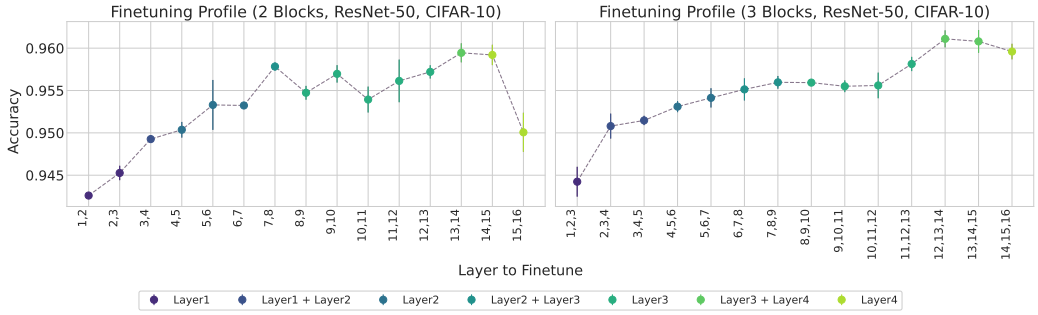

Figure 9. Finetuning profiles of ResNet-50 pretrained on ImageNet on CIFAR-10 with 2 blocks (*Left.*) and 3 blocks (*Right.*.)

B.2. Computational Efficiency

In this subsection we provide some more inference time results. In Figure 10 we provide the absolute results for SubTuning inference time for a different number of consecutive blocks. In Figure 11 we provide the accuracy vs the added inference time for 2 consecutive blocks. We can see that using blocks 13 and 14 yields excellent results both in running time and accuracy.

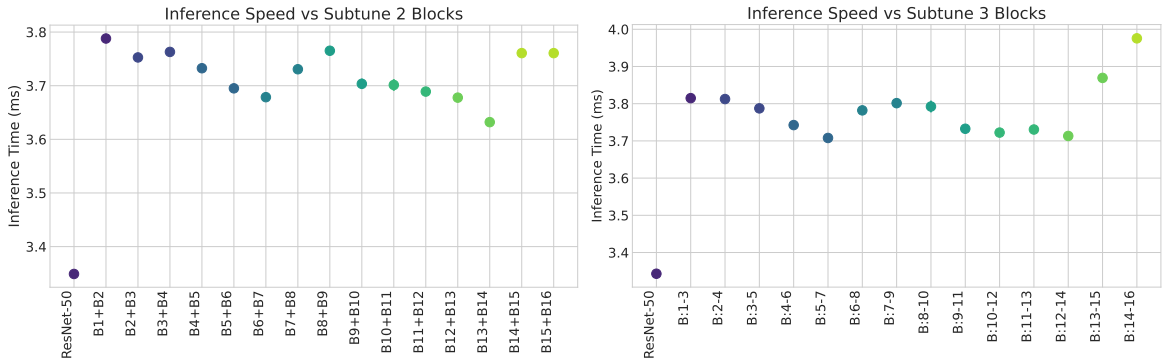

Figure 10. Absolute inference times for A100 GPU SubTuning on 2 and 3 blocks.

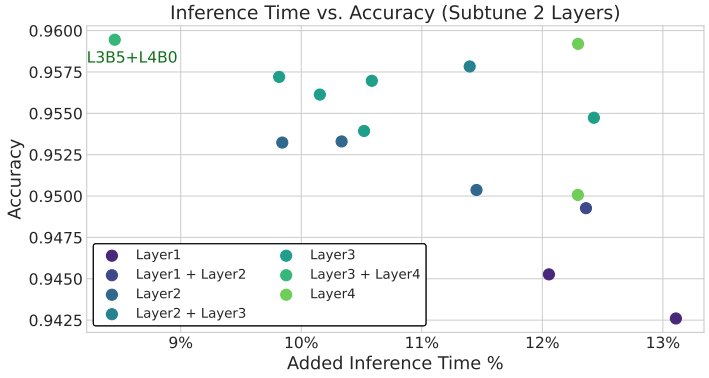

Figure 11. Accuracy vs inference time of two consecutive block SubTuning.

B.3. Low Data Regime

In this subsection we analyze each SubTuning block performance for different number of training examples. From Figure 12 we can see that for lower data regimes, later blocks in the network yield excellent performance, but when data become more abundant the better performing blocks are in an earlier stage of network.

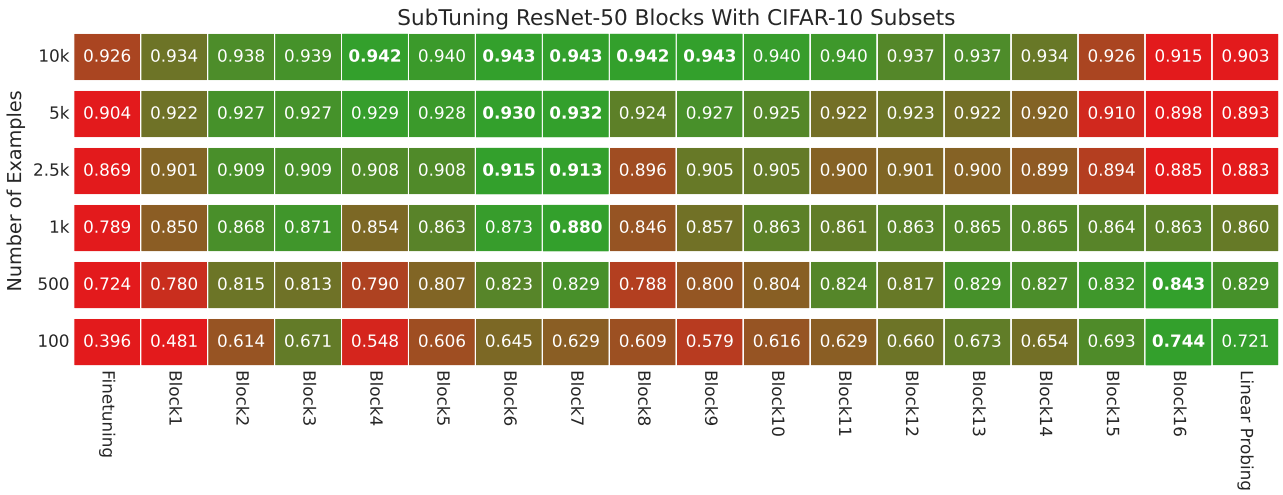

Figure 12. Single Block SubTuning of ResNet-50 on CIFAR-10. We run five trainings for 50 epochs for each dataset size and Block, and report the mean accuracy for each dataset size and Block. We also report finetuning and linear probing accuracy for comparison.

B.4. Extensions and Ablations

B.4.1. Siamese SubTuning

In this subsection we provide some extra experiments on the Siamese SubTuning setting. In Figure 13 we illustrate the Siamese SubTuning algorithm.

Furthermore, we expand our comaprison of SubTuning and Siamese SubTuning. We perform all the experiments on 5,000 randomly selected training samples from CIFAR10, CIFAR100 and Stanford Cars datasets. In Figure 14 we provide the results for resblocks of ResNet50. In Figure 15 we provide the results of full ResNet layers SubTuning, that is SubTuning a few consecutive blocks that are applied to the same resolution. As can be seen from the aforementioned figures, Siamese SubTuning adds a performance boost in vast majority of architectures, datasets and blocks choices.

Figure 13. Illustration of SubTuning vs Siamese SubTuning. Note that the difference is that the new tasks now get the original features as input.

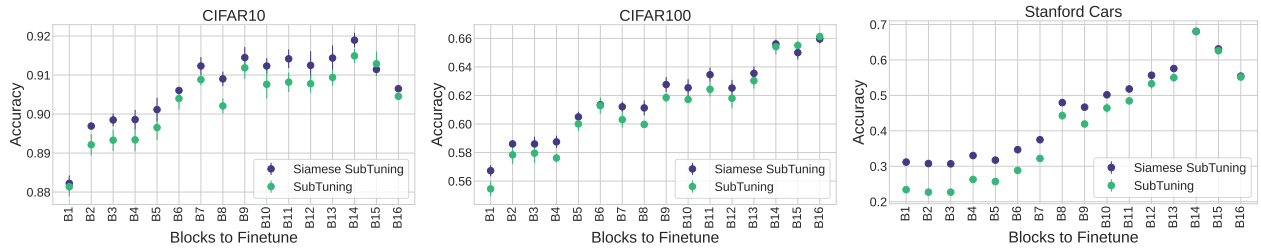

Figure 14. Siamese SubTuning on for ResNet50 on CIFAR-10 (*left.*), CIFAR-100 (*middle.*) and Standford Cars (*right.*). We use 5,000 randomly selected training samples from each dataset.

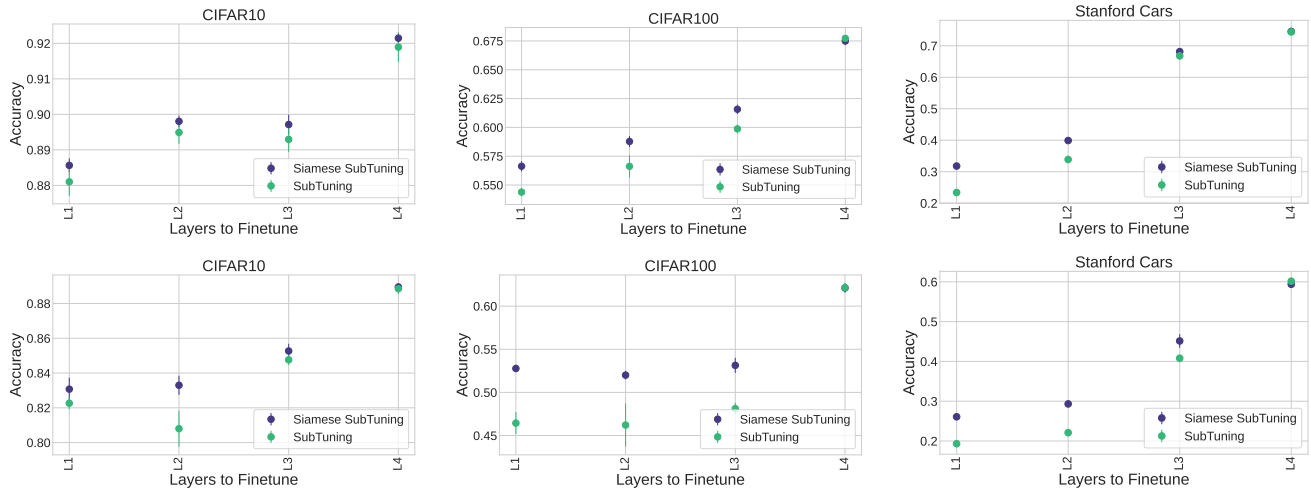

Figure 15. The impact of Siamese SubTuning of whole ResNet Layers (a group of blocks applied to the same resolution). Results for 5,000 randomly selected training samples from each dataset are presented for ResNet50 (**Top**) and ResNet18 (**Bottom**).

B.4.2. PRUNING

In Figure 16 we show all the pruning results for Local or Global pruning with L1 or L2 norms and varying channel sparsity factor between 0.1 and 0.9 in increments of 0.1. We do not have a specific goal in performance or parameter ratio thus we provide results for multiple fractions of the total SubTuning parameters and accuracy values. Even though there are slight differences between the methods all of them yield good results in reducing the complexity of the SubTuning model with only minor accuracy degradation.

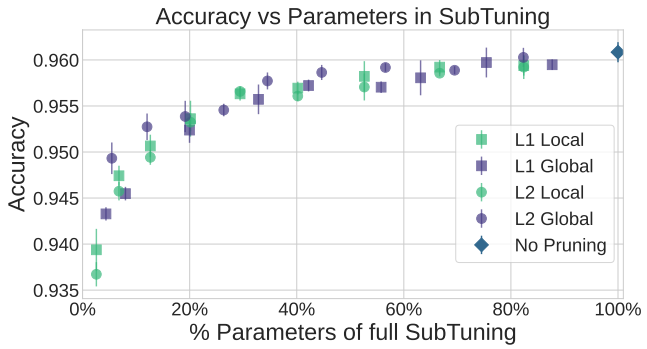

Figure 16. Full results of SubTuning with channel-wise pruning on the last 3 blocks of ResNet-50. We plot the accuracy vs the pruning rate of different pruning techniques (Global vs. Local and pruning norm) for different pruning rates.

B.4.3. EFFECT OF RANDOM RE-INITIALIZATION

In Figure 17 we illustrate the importance of weight initialization in SubTuning. We show that random re-initialization of the weights in SubTuning causes significant degradation in performance and even numerical instability.

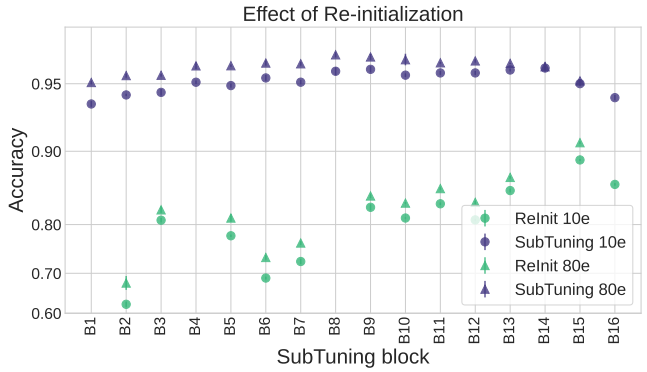

Figure 17. The effects of longer training and weight reinitializion on the *Finetuning Profile* of ResNet-50 pretrained on ImageNet and finetuned on CIFAR-10. For randomly re-initialized the weights, we encountered some optimization issues when training the first block on each resolution of the ResNet model, i.e. blocks 1, 4, 8 and 14. We use a logit scale for the y-axis, since it allows a clear view of the gap between different scales.