TINY ADAPTERS FOR VISION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision Transformers (ViTs) have become one of the dominant architectures in computer vision and pretrained ViT models are commonly adapted to new tasks via fine-tuning of its parameters. Recent works in NLP proposed a variety of parameter-efficient transfer learning methods such as adapters to avoid the prohibitive storage cost of fine-tuning.

In this work, we start from the observation that adapters perform poorly when the dimension of adapters is small and we propose a training algorithm that addresses this issue. We start from large adapters which can be trained easily and iteratively reduce the size of every adapter. We introduce a scoring function that can compare neuron importance across layers and consequently allow automatic estimation of the hidden dimension of every adapter. Our method outperforms existing PET methods in terms of the trade-off between accuracy and trained parameters across two benchmarks. We will release our code publicly upon acceptance.

1 INTRODUCTION

Transformers for vision problems have recently attracted growing attention due to their remarkable performance Liu et al. (2021b); Touvron et al. (2021); Dosovitskiy et al. (2020). In particular, the image representation learned with large-scale pretrained Vision Transformers (ViTs) has shown a promising ability for learning new tasks Dosovitskiy et al. (2020). The commonly adopted strategy to learn new tasks consists in fully or partially fine-tuning a pretrained network. However, in the case of multiple tasks, this approach requires training numerous separate instances of the models and induces significant storage costs.

Parameter-Efficient Training (PET) approaches have recently been developed in order to adapt large-pretrained models to new tasks with less training power Hu et al. (2021); He et al. (2021). Among these, adapters Houlsby et al. (2019) and their variants Mahabadi et al. (2021); He et al. (2021); Karimi Mahabadi et al. (2021) are frequently utilized and adapted to many architectures for Natural Language Processing (NLP) tasks. In a nutshell, adapters are tiny modules injected into transformer blocks, which enable efficient adaption of the data representation to the downstream task. Adapters offer similar performance to full fine-tuning (*i.e.* updating all parameters) while requiring a very limited number of trainable parameters Houlsby et al. (2019); Pfeiffer et al. (2021).

When it comes to vision tasks, PET approaches are mostly explored for convolutional neural networks Berriel et al. (2019); Rebuffi et al. (2018; 2017); Mallya et al. (2018). On the contrary, several PET approaches have been proposed in the context of NLP tasks. In this case, adapters are commonly implemented with Multi-Layer-Perceptron (MLP) layers equipped with residual connections Houlsby et al. (2019). Multi-layer adapters offer sufficient representation power to adapt to the new tasks but require a significant number of trainable parameters subject to their bottleneck dimension, which provides a simple trade-off between performance and parameter efficiency Houlsby et al. (2019). Nevertheless, they suffer from two main weaknesses. First, the performance drops when the size of the multi-layer adapters is too small (as also confirmed by our experiments). Therefore, these adapters cannot be employed in scenarios where the available storage is limited. Second, the hyper-parametrization of adapters is complex, since the hidden layer dimensions must be specified for every adapters in every layer. As well, the hidden dimension depends on the downstream task.

In this work, we propose a training scheme named TINA (Fig. 1), that addresses these two limitations of adapters. Our strategy allows a more efficient parameter allocation since additional parameters are predominantly allocated to layers really requiring adaptation to the new task. More specifically, we start from adapters with hidden spaces of high dimension and we iteratively reduce the dimension by identifying the neurons that can be dropped in every adapter. To identify in which layer

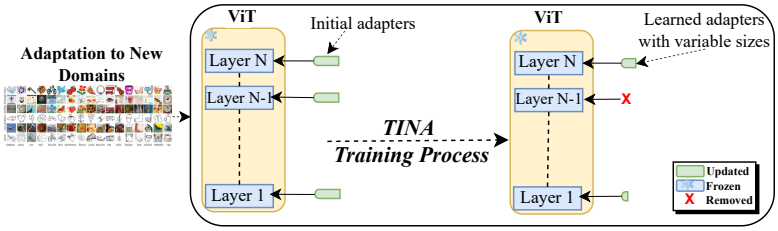


Figure 1: TINy adapters (TINA) are layer-wise small blocks injected into ViTs to efficiently adapt to new domains by estimating the best rank for each adapter weights. TINA reduces the number of parameters and removes completely injected adapters for some layers if necessary.

adaptation must be primarily performed, we introduce a novel scoring criterion that allows comparison of neuron importance across different adapters. In short, the contributions of this work are:
• We propose a novel iterative training scheme for learning tiny adapters in the case of ViTs.
• We propose a novel scoring function to compare neuron importance across adapters, which can be included in various magnitude pruning methods. This allows us to automatically estimate the hidden dimension of adapters for ViTs, leading to more efficient parameter allocation.
• Finally, we compare the proposed approach with multiple PET methods designed for NLP using a total of 10 datasets. From these experiments, we draw several conclusions: (i) we observe a discrepancy between the rankings obtained in vision and NLP benchmarks; (ii) we demonstrate that our approach obtains the best performance in terms of accuracy among methods with similar numbers of parameters; (iii) our ablation study validates the positive impact of our adaptive strategy to automatically estimate the hidden dimension of adapters.

2 Related Work

Vision Transformers. Initially proposed for NLP tasks, Transformers Vaswani et al. (2017) have been recently proposed for vision tasks. The image is split into a sequence of patches that are projected into token embeddings, and fed to transformer encoders that employ self-attention to learn rich image representations Dosovitskiy et al. (2020). With sufficient training data, ViTs are able to outperform ConvNets on various image classification benchmarks Guo et al. (2022). Nevertheless, their high number of parameters induces high storage cost, which hinders complete finetuning for each new task and naturally motivates this study. Several variants of ViTs have been proposed in the literature, out of which Swin Liu et al. (2021b) is probably the most popular thanks to its high performance on various vision tasks. Swin employs local attention applied to different regions of the input image from one layer to another, unlike the initial ViT architecture Dosovitskiy et al. (2020) which uses global attention. Despite we evaluate our approach using Swin, our tiny adapters can be added to any ViT architecture.
Network Pruning. When referred to deep neural networks, pruning consists of reducing the number of parameters of a pre-trained model Han et al. (2015a); Cun et al. (1990). It can be roughly categorized into two groups: (i) unstructured pruning, which removes the least significant weights (according to certain criteria like weight magnitude Han et al. (2015b) or gradient magnitude Molchanov et al. (2019)) without a specific structure to be followed; (ii) structured pruning, which focuses in removing model sub-structures, like channels Tartaglione et al. (2018); He et al. (2017) or attention heads Michel et al. (2019). A key focus of these works resides in identifying redundant connections whose removal brings the least perturbation to the overall performance.
This work does not introduce a new pruning method, although TINA does use iterative pruning. In this regard TINA is unique in two key aspects. First, pruning methods traditionally reduce the cardinality of the parameters of a network trained on a specific task, while TINA reduces the number of parameters injected via adapters to adapt the model to a new task without changing the original model parameters. Second, we leverage on the adapter's structure to design an on-purpose importance metric that accounts simultaneously parameters from the two layers in adapters, while pruning typically considers parameters of every layer independently.
Efficient Transformers Finetuning. The lack of the typical CNN inductive biases in ViTs makes them difficult to be finetuned on new tasks without using (very) large datasets. For this reason, Liu

et al. (2021a) addresses this by adding a regularization term during finetuning to extract additional information in self-supervised manner by the model. However, it requires updating all the parameters and storing one copy of the fine-tuned model per task. This causes substantial storage and deployment costs and hinders the applicability of large-scale models to real-world applications. In order to address this issue, three types of approaches have been proposed for NLP tasks: (i) only updating newly added parameters (added either to the input or model) Pfeiffer et al. (2021); Houlsby et al. (2019)Chen et al. (2022); (ii) sparsely updating a small number of parameters of the model Hu et al. (2021); Ben Zaken et al. (2022); (iii) performing low-rank factorization for the weights to be updated Karimi Mahabadi et al. (2021). Inspired by prompting techniques in NLP, Jia et al. (2022) introduced *VPT*, a method that learns prompts for efficient finetuning of ViTs. Although prompting achieves good performance, it lacks flexibility since prompt sizes must be equal to the dimensions of the corresponding layers in the pretrained ViT. Thus, prompts are not suitable for downstream tasks that strongly differ form the pre-training task Chen et al. (2022). Among these approaches, adapters, belonging to the first category, showed promising results in comparison with full model fine-tuning. Our work falls into the first category of approaches. More precisely, TINA is based on the adapters proposed in Houlsby et al. (2019), but we provide a specific training algorithm that enables small size adapters' training. Furthermore, while the other state-of-the-art approaches adopt fixed size adapters in every layer Chen et al. (2022) In our preliminary results, we show different layers require different adapter sizes to adapt efficiently to the new tasks (see Appendix A.2.1). TINA dynamically estimates the adapter size for each layer and even removes them, if necessary. In this manner, we reduce the number of trainable parameters and improve the performance.

3 Proposed Method

In this section, we start with the description of adapters Houlsby et al. (2019) and their practical benefits. Then, we introduce our proposed method to estimate the hidden dimension for each adapter layer that can effectively train ViTs reaching higher performance with fewer parameters.

3.1 Preliminaries

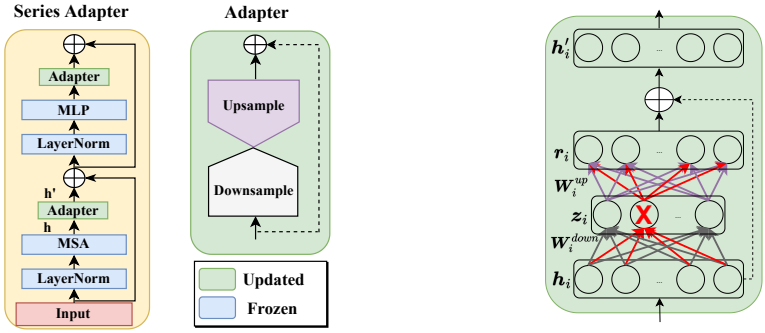
We assume that we have at our disposal a pretrained ViT network and that our goal is to adapt this network to learn a classifier for a new task. We have access to an annotated dataset for the targeted downstream task. To this aim, we employ small modules, referred to as *adapters* that are added to the layers of the pre-trained model in order to adapt it to the downstream task while keeping the weights of the original model frozen. Therefore, adapting the network consists in jointly training the linear classifier (commonly referred to as *head*) and the parameters of the adapters. For the sake of simplicity, we no longer mention the linear classifier parameters when describing our training procedure, but it has to be kept in mind that they are jointly learned with the adapters.

Formally, ViT architectures such as the original ViT Dosovitskiy et al. (2020) or Swin Liu et al. (2021b) employ layers composed of two main sub-layers: a multi-head self-attention layer and a feedforward layer, each of them preceded by layer normalization. A residual connection is applied across each of the sub-layers. We insert two adapters after each of these sub-layers. The adapter is always applied directly to the output of the target sub-layer, as show in Fig. 2a. The internal structure of adapters is visually summarized in Fig. 2b. Considering the i^{th} adapter added to our pretrained ViT, let $\boldsymbol{h}_i \in \mathbb{R}^{m_i}$ denote its input of size m_i Following Houlsby et al. (2019), adapters employ a first fully connected layers that down-project $\boldsymbol{h}_i$ into $\boldsymbol{z}_i \in \mathbb{R}^{n_i}$ with some non-linear activation $\phi(\cdot)$. This layer is parametrized by a linear projection matrix $\boldsymbol{W}_i^{down} \in \mathcal{R}^{m_i \times n_i}$. Then, a second fully connected layer with parameters $\boldsymbol{W}_i^{up} \in \mathcal{R}^{n_i \times m_i}$ up-samples $\boldsymbol{z}_i$ producing as output $\boldsymbol{r}_i \in \mathbb{R}^{m_i}$. Finally, a residual skip-connection is employed inside the adapter module such that, if $\boldsymbol{r}_i$ is always close to 0, the adapter module degenerates to an identity function. To summarize, given the input vector $\boldsymbol{h}_i$, the output vector $\boldsymbol{h}_i'$ is calculated as:

$$\boldsymbol{h}_i' = \boldsymbol{W}_i^{up} \phi\left(\boldsymbol{W}_i^{down} \boldsymbol{h}_i\right) + \boldsymbol{h}_i. \tag{1}$$

The total number of parameters in adapters is equal to $2 \cdot n_i \cdot m_i$. Since m_i is fixed, we generally specify n_i such that $n_i \ll m_i$ to obtain a low number of trainable parameters. We define the compression rate σ_i of an adapter as follows $\sigma_i = \frac{m_i}{n_i}$.

In previous works Houlsby et al. (2019); Rücklé et al. (2021), adapters have a unique hidden dimension n_i for every adapter Pfeiffer et al. (2021); Houlsby et al. (2019). This choice, however,



(a) Adapters injected into ViT model. (b) Internal structure of adapters with TINA.

Figure 2: Illustration of adapter structure injected into ViT, and our approach to adjust adapter's size. MSA and MLP indicate multi-head self-attention and feedfoward blocks, respectively.

can potentially be sub-optimal as early layers may focus on general patterns and late layers on task-specific ones Zhang et al. (2020). Allowing the model to dynamically adjust the adapter's hidden dimension n_i (or equivalently, σ_i), and identifying where they should be injected, can help adaptation to the downstream task proficiently.

3.2 Overall Procedure of TINA

Let $\boldsymbol{W}^{ViT}$ be the initial parameters of the ViT model which are frozen through the whole adaptation process. Our goal is to learn $\boldsymbol{W}^{ada}$, the set containing the adapter parameters $\boldsymbol{W}_i^{ada}$ of every i^{th} ViT sub-layer. In previous works Houlsby et al. (2019); Rebuffi et al. (2018), $\boldsymbol{W}_i^{ada}$ is straightforwardly learned with stochastic-gradient-descent-based optimization. However, in our experiments (see Sec. 4) we show that this approach does not perform well in the case of tiny adapters (small n_i values). To develop our training scheme, we start from the observation that, with the existing optimizers, sequentially training and pruning a large network is a successful strategy to find small networks with good performance, while directly training small networks usually suffers from optimization issues Frankle & Carbin (2018). Therefore, we propose to start from large adapters and adopt an iterative pruning strategy that iteratively reduce their dimensions -detailed in Algorithm 1-.

Algorithm 1 TINA

1: **procedure** TINA ($\boldsymbol{W}^{ViT}$, $\boldsymbol{W}^{ada}$, ρ, σ^{target})
2: Learn $\boldsymbol{W}^{ada}$ ▷ $\boldsymbol{W}^{ViT}$ is maintained frozen
3: **while** $\sigma < \sigma^{target}$ **do** ▷ while compression target is not met
4: $\boldsymbol{W}^{ada} \leftarrow$ top$(1-\rho)$ fraction of neurons in $\boldsymbol{W}^{ada}$ ▷ according to their score (Sec. 4.2)
5: Fine-tune the remaining $\boldsymbol{W}^{ada}$ on the target dataset ▷ $\boldsymbol{W}^{ViT}$ is maintained frozen
6: **end while**
7: **return** $\boldsymbol{W}^{ada}$
8: **end procedure**

We initialize every adapter with a hidden dimension proportional to its input dimensions. We start from constant compression rates σ_i for every layer and equal to σ^0. In our first training stage (line 2), we learn the adapter parameters $\boldsymbol{W}^{ada}$ via cross-entropy minimization using stochastic gradient descent. Then, we propose to estimate a score that measures the importance of each adapter's neurons. The estimation of this score is detailed in Sec. 3.3. This score is used to select the **neurons** that have the smallest impact on the adapter outputs. More precisely, we remove the bottom fraction ρ of neurons from $\boldsymbol{W}^{ada}$ (line 4). The remaining ones will constitute the new adapter configuration and the hidden space sizes n_i are updated accordingly.
If the achieved average compression rate σ is still lower than the target σ^{target}, another compression iteration follows; otherwise the achieved configuration will be returned and the method stops. Note that the number of training cycles C is given by: $C = \left\lceil \frac{\log(\sigma^0) - \log(\sigma^{target})}{\log(\rho)} - 1 \right\rceil$, where $\lceil \cdot \rceil$ denotes

the ceiling function. Therefore, our training scheme stops after a deterministic number of iterations that can be computed in advance. Besides, while we employ a stopping criteria based on a specific target compression rate, a target performance on a validation dataset could be used.

3.3 IMPORTANCE SCORE IN TINY ADAPTIVE ADAPTERS (TINA)

In this section, we present the importance score function that we use in our training algorithm. Our design of the scoring function is motivated by the observation that, if an entire row in $\boldsymbol{W}_i^{down}$ and an entire column in $\boldsymbol{W}_i^{up}$ are equal to zero, then our adapter is strictly equivalent to an adapter with a smaller dimension n_i. Therefore, we propose a novel scoring function to employ the sum of the L^1 norm of the corresponding row in $\boldsymbol{W}_i^{down}$ and the corresponding column in $\boldsymbol{W}_i^{up}$. More precisely, our importance score is formulated as follows:

$$\mathcal{I}^{ij} = \frac{1}{n_i + m_i} \left(\sum_{k=1}^{m_i} \left| W_i^{down}[j,k] \right| + \sum_{k=1}^{n_i} \left| W_i^{up}[k,j] \right| \right), \tag{2}$$

where $[\cdot,\cdot]$ denotes the matrix indexing operator. This importance score can be interpreted as a "look-ahead" strategy, where we observe, besides the output of a specific j-th neuron in the hidden space, also the impact of such an output in the next layer. Note that this formulation is based only on *the magnitude of parameters belonging to the same neuron* of down-sampling, and its corresponding up-sampling neuron, and not on the magnitude of activations. This makes the importance score more computationally efficient since activation-based scoring would depend on the input images, and consequently, statistics should be gathered at the batch or at the dataset level. This would induce important additional computation. Furthermore, this choice is empirically supported by many works in the literature, like Chauvin (1988); Han et al. (2015b); Molchanov et al. (2017); Renda et al. (2020).We also provide more details that motivates equation 2 in the Appendix, at Sec. A.4.

Importantly, $\mathcal{I}^{ij}$ is normalized by the total number in the sums to allow fair comparison across adapters with different input and hidden layer sizes.

4 EXPERIMENTS

We provide the details about the datasets and our experimental setup.

Datasets. We evaluate our methods using the protocol previously adopted by Liu et al. (2021a), which consists of ten datasets for image classification tasks divided into two benchmarks. The first benchmark is known as *DomainNet* Pan et al. (2019). It contains six different visual domains, which makes the fine-tuning experiments non-trivial. Since *DomainNet* does not have a labeled testing set, we use the validation dataset for testing, as in Pan et al. (2019). The second benchmark contains CIFAR-10/CIFAR-100 Krizhevsky et al., Oxford Flowers102 Nilsback & Zisserman (2008) and SVHN Netzer et al. (2011), which are widely used as low-regime training datasets. Contrarily to DomainNet, these datasets are not single task oriented, but contain a larger variety of domains/tasks. We refer to them as belonging to the *Multi-task* benchmark.

Implementation Details. We follow the training protocol adopted by Liu et al. (2021a). We conduct our experiments with the official pretrained model Swin-T ($\sim$27M parameters) Liu et al. (2021b) trained on ImageNet-1K. In all our experiments, we use the AdamW Loshchilov & Hutter (2019) optimizer with a cosine decay learning-rate scheduler for 80 epochs, preceded by 20 epochs of linear warm-up. In all the experiments, the images are resized to the same fixed resolution (224×224). Contrarily to Liu et al. (2021a) that trains every baseline for 100 epochs only, we train them for 500 epochs to allow fair comparison with our approach that requires multiple training cycles C. With TINA, ρ is set to 50%, namely we half the number of neurons in the adapters, at each 100 epochs.

4.1 MAIN RESULTS

We compare our proposed method TINA with multiple PETs methods that were initially for NLP transformers. We include the following methods. • *Full fine-tuning*: it fine-tunes all parameters of the model. • *Att/MLP fine-tune*: we only tune the Attention/MLP layers and the classification head. *Linear-probe*: all parameters are frozen except for the task-specific classification layer. • *Adapters*

Table 1: Results on the DomainNet benchmark Pan et al. (2019).

Method	# Params (M) ↓	Trained (%) ↓	Clipart	Infograph	Painting	Quickdraw	Real	Sketch	Mean ↑
Full fine-tuning	27.8	100	79.16	48.29	74.64	75.88	86.21	73.26	**72.90**
Att-blocks	8.93	32.14	48.36	75.38	73.28	86.13	72.81	72.57	79.44
MLP-blocks	17.54	63.12	79.23	48.11	75.02	74.82	86.35	73.29	72.80
Linear prob	0.27	0.95	62.89	33.96	64.93	42.95	81.69	54.24	56.77
PHM-Adapter	0.47	1.72	75.79	44.62	72.49	66.62	83.73	68.51	68.62
Compacter	0.41	1.44	75.25	44.19	72.09	66.01	83.42	67.99	68.16
LoRa (Q, K)	0.42	1.51	74.53	43.65	71.81	64.53	83.81	67.09	67.57
BitFit	0.34	1.22	72.14	41.07	70.00	60.39	82.43	64.43	65.08
VPT (10 tokens)	0.32	1.15	61.91	24.87	57.05	57.09	76.94	55.12	55.49
VPT (100 tokens)	0.71	2.57	64.33	18.65	63.20	59.40	79.65	56.41	56.94
AdaptFormer-64	0.84	3.06	73.76	42.38	71.11	63.41	83.23	66.14	66.67
AdaptFormer-256	2.54	9.24	75.32	43.74	72.16	66.00	83.88	67.68	68.13
Adapters ($n_i = 47$)	1.37	4.90	76.15	45.28	73.04	67.86	84.83	69.17	69.39
Adapters ($n_i = 23$)	0.68	2.47	75.28	44.17	72.41	66.44	83.98	68.02	68.38
Adapters ($n_i = 1$)	0.30	1.07	72.12	41.30	69.93	59.95	82.49	64.18	65.00
Adapters ($\sigma = 32$)	1.37	4.90	77.42	46.51	74.06	69.81	85.30	70.84	**70.65**
TINA (1 cycle)	0.80	2.89	76.83	46.00	73.76	67.93	85.05	69.61	69.86
TINA (2 cycles)	0.53	1.92	76.83	45.45	73.11	66.67	84.42	69.05	69.26
TINA (3 cycles)	0.40	1.43	75.44	44.60	72.59	64.73	83.87	68.05	68.21
TINA (4 cycles)	**0.30**	**1.07**	74.38	43.52	71.50	63.41	83.12	67.46	67.23

Table 2: Results on the Multi-task benchmark.

Method	# Params (M) ↓	Trained (%) ↓	CIFAR100	CIFAR10	Flowers	SVHN	Mean ↑
Full fine-tuning	27.8	100	88.13	98.50	97.35	96.59	**95.14**
Att-blocks	8.93	32.14	88.03	98.41	97.79	95.99	95.05
MLP-blocks	17.54	63.12	88.44	98.47	96.50	96.14	94.89
Linear prob	0.27	0.95	75.58	91.84	76.80	55.26	74.87
PHM-Adapter	0.47	1.72	84.17	96.48	89.18	93.32	90.78
Compacter	0.41	1.44	83.95	96.26	88.43	92.67	90.32
LoRa (Q, K)	0.42	1.51	83.87	96.41	87.07	91.87	89.81
BitFit	0.34	1.22	83.56	96.14	87.85	90.29	89.46
VPT (10 tokens)	0.33	1.20	67.69	90.99	22.77	85.11	66.64
VPT (100 tokens)	0.52	1.88	72.53	93.03	34.88	86.70	71.78
AdaptFormer-64	0.66	2.38	83.79	96.93	90.50	92.45	90.91
AdaptFormer-256	2.98	8.55	84.74	97.23	92.13	94.97	92.27
Adapters ($n_i = 47$)	1.37	4.90	85.04	97.52	92.72	96.35	92.91
Adapters ($n_i = 23$)	0.68	2.47	85.18	97.57	92.16	95.81	92.68
Adapters ($n_i = 1$)	0.30	1.07	82.60	96.03	89.77	88.80	89.30
Adapters ($\sigma = 32$)	1.37	4.90	85.59	97.49	94.80	96.27	93.53
TINA (1 cycle)	0.80	2.89	87.12	97.98	96.59	96.98	**94.67**
TINA (2 cycles)	0.53	1.92	86.33	97.49	96.73	96.48	94.26
TINA (3 cycles)	0.40	1.43	85.22	97.11	96.81	95.60	93.69
TINA (4 cycles)	**0.30**	**1.07**	84.07	97.11	96.81	93.94	92.98

Houlsby et al. (2019): we add adapters with $\sigma = 32$ to have adapters with hidden dimensionality proportional to the input dimension m_i. We also include variants where the size of every adapter is fixed over all the layers: $n_i = 47$, and $n_i = 23$. These baselines are considered to emphasize the effect of parameter allocation throughout the layers on the final performance. • *BitFit* Ben Zaken et al. (2022): all the weights are frozen, the biases are learned. By not storing intermediate activations, this method enables substantial memory savings. • *PHM-Adapter* Zhang et al. (2021): the weights of the adapters are learned using parameterized hyper-complex multiplication layers (PHM) layers. • *Compacter* Karimi Mahabadi et al. (2021): adapter weights are learned using PHM layers. • *LoRa* Hu et al. (2021): trainable low-rank matrices are employed to approximate the updates for the model. Updates are applied mainly to the key k and the query q of the attention blocks. • *AdaptFormer* Chen et al. (2022): introduces a small module like Adapters, but only after MLP network with a scaling parameters s applied to the output of the injected modules. • *VPT* Jia et al. (2022): fine-tuning learnable parameters (i.e. prompts) injected into the embedding space.

Discussion. Table 1 reports the number of trained parameters and the average accuracy across datasets in the DomainNet benchmark, while Table 2 contains the results achieved for the Multi-task benchmark. For both, the number of trained parameters is reported in millions, and the average top-1 accuracy on the datasets is reported in the rightest column. We observe that *full fine-tuning*

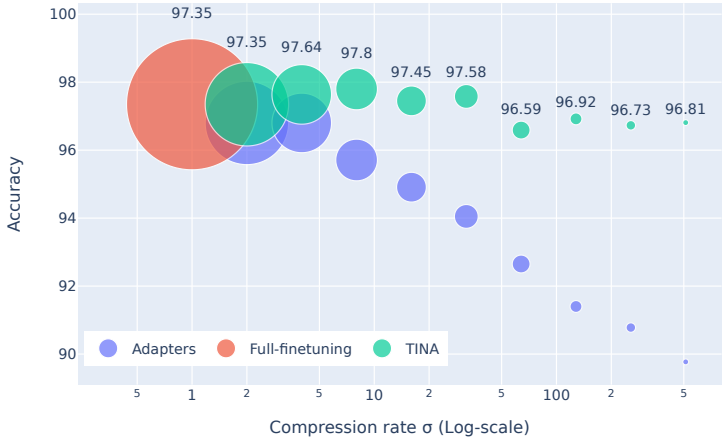


Figure 3: Comparison of top-1 accuracy versus compression rate σ on VGG-Flowers. Size of blob markers represents the number of trainable parameters.

has generally the highest accuracy, but it requires a huge number of parameters to be finetuned for each dataset. Among the vanilla fine-tuning baselines, we observe that tuning the parameters of the *attention/MLP* layer turns out to be surprisingly effective. Nevertheless, it still requires a high number of task-specific parameters, compared to other PET approaches. *Linear probing* does not perform well illustrating the need to change the feature representations when adapting to new tasks. *PHM*, *LoRA*, and *Compacter* are effective methods to get on-par performance with *full-model fine-tune* while adjusting less than 2% of the parameters. Contrarily to what is observed for NLP tasks Houlsby et al. (2019), PETs on visual tasks do not reach *full fine-tuning* performance on any dataset with low number of trainable parameters (smaller than 2%). VPT does not perform well, indicating that injecting tokens into the embedding space do not help much if the pre-training dataset is different from the downstream task. Generally speaking, all PET methods maintain similar performance rankings on all the tasks. This suggests that the choice of the best adaptation strategy does not depend on the downstream task.

Adapters outperform all PET methods in terms of accuracy (69.39% for DomainNet, 92.91% for Multi-task) but just with a higher number of trainable parameters (1.37M, 4.90% of the total) for $\sigma = 32$. *Adapters* outperform *AdaptFormer* with fewer parameters (92.91% with 1.37M parameters, versus 92.27% with 2.98M parameters). This result indicates that adapting the representations after both MSA and MLP blocks, as done in *Adapters* (see Fig2a), allows better adaptation than acting only on the MLP block via a parallel branch (as in *AdaptFormer*).

When comparing *adapter* with uniform and proportional parameter distribution, we observe that allocating parameters proportionally to the layer dimension performs better. Indeed, adapters with $\sigma = 32$ outperform adapters with $n_i = 47 \forall i$ (70.65% vs 69.39% in DomainNet, 93.53% vs 92.91% in Multi-task). This suggests that the last layers, which have higher dimensionality, are more task-specific, and consequently require more adaptation. We also show that reducing the size of adapters ($n_i = 23$) hurts the performance with a drop which, despite being marginal for Multi-task (0.23%) is more consistent in DomainNet (1.01%). This emphasizes that training tiny adapters in a vanilla fashion, leads to unsatisfying performance and motivates our specific training procedure.

TINA versus Vanilla training. From Table 1 we observe that, in the DomainNet benchmark, TINA outperforms methods with similar trained parameters, in all the compression ranges. In particular, in the most challenging one (with 0.30M parameters), TINA outperforms the closest approach, BitFit, which trains 0.34M parameters, showing a gain in average accuracy larger than 2%.

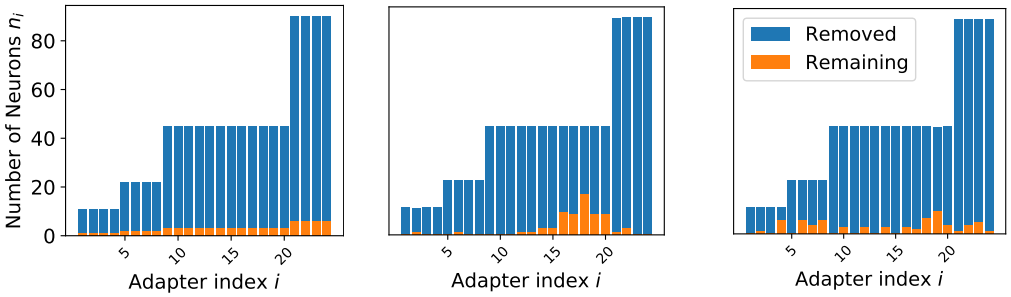
Looking at the Multi-task benchmark (Table 2), we observe that *TINA* significantly reduces the number of parameters by $4\times$ (0.40M, 1.43%) while outperforming all PET methods in the Multi-task benchmark. In particular, *TINA* outperforms *adapters*-$n_i = 47$ despite having less parameters, demonstrating that our iterative training procedure improves the parameter efficiency. To further emphasize on the performance gap between the two approaches, we introduce Fig. 5, illustrating the performance as a function of the number of trainable parameters. We observe the significant performance gap between vanilla adapters compared to adapters trained with TINA approach.

Table 3: Comparative performance analysis for neuron selection on VGG-Flowers. $L^1(w)$ and $L^1(a)$ denote magnitude pruning of the parameters and the activations respectively.

Method	Pruning Method	Score	Iterative	Scaling	σ 32	64	128	256	512
Vanilla Adapters	-	-	-	-	94.80	90.12	89.42	88.85	86.03
Baselines	Vanilla L^1	$L^1(w)$			-	95.46	95.06	94.28	93.79
	Local	$L^1(w)$			-	96.10	95.57	96.15	96.23
	Local	$L^1(w)$	✓		-	96.41	96.65	96.72	96.72
	Global	$L^1(a)$		✓	-	96.10	94.28	93.80	93.25
	Global	$L^1(a)$	✓	✓	-	96.13	95.15	95.77	95.72
	Global	$\mathcal{I}_0$			-	94.88	95.28	95.66	95.45
	Global	$\mathcal{I}$		✓	-	96.10	95.82	96.34	96.50
TINA	Global	$\mathcal{I}$	✓	✓	-	**96.59**	**96.92**	**96.73**	**96.81**

4.2 Ablation Study

Importance score for TINA. Next, we move on to our design choice of dimensionality reduction inside adapters throughout the adaptation cycles. We observe the contribution of various components of TINA with different setups. • *Vanilla Adapters* corresponds to injecting adapters with a compression rate σ. • *Vanilla* L^1: we select same percentage of neurons for each adapter applied to down-sampling layer only. • *Local neuron selection*: we select the same percentage of neurons for each adapter ($\rho = 50\%, 75\%$, etc...), applied to down-sampling, and up-sampling layers independently.• *Global neuron selection*: we select a specific amount of neurons ρ to be removed. The amount of selected neurons per adapter is calculated using $\mathcal{I}$ as in equation 2, if scaling is applied. We also evaluate our scoring function without the scaling factor $n_i + m_i$. This variant of our score is denoted by $\mathcal{I}_0$. • *Global* with $L^1(a)$: we select a specific amount of neurons ρ to be removed according to the magnitude of the neuron activation. • *TINA*: our proposed method as in Alg. 1.
To compare the different methods we proceed as follows. When using an iterative method, we always start from the baseline model where $\sigma = 32$. When using a non-iterative method: we start with adapters of $\sigma_0 = \sigma_{target}/(1-\rho)$, and prune once only after the first cycle. Training continues for $C-1$ cycles to guarantee fair comparison with iterative methods. Results are reported in Table 3.
Discussion. Table 3 summarizes the performance of local and global neuron selection for adapters. Firstly, we notice the drop of performance for vanilla adapters, as we reduce the number of parameters -higher values of σ-. Secondly, we notice the advantage of using global over local neuron selection. *Local neuron* selection method still has all the adapters injected but with low dimensionality n_i. On the contrary, *global neuron selection* method emphasizes on finetuning adapters at specific layers, where other adapters are removed completely.
At the last cycles, TINA identifies important adapters for the adaptation, and focuses on finetuning them. Thus, the model performance improves by adjusting specific latent representations of the model tailored for the downstream task while training adapters with fewer parameters. In Table 3, for $\sigma = 64$ we observe that TINA (using either local/global neuron selection) achieves better performance than a vanilla training of adapters on VGG-Flowers dataset with a performance gap of 6.14%. Interestingly, this gap in performance expands, as we compare smaller adapters $\sigma = 256, 512$. When comparing to a vanilla L^1 importance scoring, we see the beneficial effect of considering parameters both in the downsampling and in the upsampling for the adapters, with consistent gains in performance for all the explored compression rates, ranging from 0.5% to more than 3%. We can observe that the performance gap is especially clear with high compression rates. Finally, scaling the importance score as in equation 2 boosts the performance of *Global* for every σ value by about 1%.
Parameter allocation analysis with TINA. Fig. 4 illustrates the distribution of the removed and remaining neurons using TINA on VGG-Flowers, and CIFAR-10. We observe the difference between local neuron selection that removes uniformly neurons from adapters and *global neuron selection*. We witness that the latter totally removes some adapters from the model (see layers $4, 5, 7, 8$ on VGG-Flowers) and allocates many parameters to some other adapters. We provide normalized plots for VGG-Flowers, and CIFAR100 illustrating the distribution of removed neurons (Fig. 9, and 10)
Furthermore, *global neuron selection* adapts differently to each dataset as shown in Fig. 4. The distribution of removed neurons is different for CIFAR-10, where fewer adapters have been removed completely with respect to VGG-Flowers. For VGG-Flowers, only adapters at late stages are kept,



(a) Local pruning: all datasets. (b) Global pruning: VGG-Flowers . (c) Global pruning: CIFAR-10.

Figure 4: Layer-wise analysis of adapter's neurons distribution at 4th cycle. Bar plots represent number of neurons n_i at each adapter i using local, global pruning for VGG-Flowers and CIFAR-10, respectively.

Table 4: Performance analysis of our method on different ViTs backbones.

	Method	# Params (M) ↓	Trained (%) ↓	CIFAR-100	CIFAR-10	VGG-Flowers	SVHN	Mean ↑
ViT-B-16	Finetune	85.90	100	91.22	99.01	99.32	97.68	**96.81**
	Adapters	0.96	0.89	89.39	98.02	97.69	94.17	94.82
	TINA	0.62	0.54	89.86	98.09	98.75	94.94	95.41
	TINA	**0.37**	**0.32**	89.84	98.17	98.85	95.32	**95.55**
Swin-S	Finetune	48.80	100	90.12	98.88	98.37	98.16	**96.38**
	Adapters	0.41	4.88	89.05	98.48	94.60	97.25	94.84
	TINA	0.23	2.75	88.86	98.53	96.16	97.22	95.19
	TINA	**0.11**	**1.32**	88.62	98.50	96.68	96.94	**95.18**
CvT	Finetune	19.65	100	90.01	98.68	97.98	98.09	**96.19**
	Adapters	0.78	4.00	86.68	97.91	88.93	96.96	92.62
	TINA	0.47	2.40	86.47	97.98	93.28	97.17	**93.73**
	TINA	**0.28**	**1.44**	85.87	97.77	94.31	96.67	93.66

which may indicate that early layer's representations are suitable for this dataset. However for CIFAR-10, remaining adapters are widespread through all layers of ViT.

ViT variants with TINA. We analyze the performance of TINA using different ViT backbones. TINA is injected into Swin-S, ViT Dosovitskiy et al. (2020) and CVT Wu et al. (2021), that improves ViTs by introducing convolutions into ViT to yield the best of both designs .
We train the 3 baselines *Finetuning*, *Adapters*, and *TINA* for 3 cycles. We report the best score for the last cycle in Table 4. TINA achieves on-par performance with respect to full model finetuning with a gap of 1.2%, 1.2% and 1.4% for ViT-B/16, Swin-S and CvT, respectively. By finetuning less than 1.5% of parameters including the head classifier.
TINA outperforms vanilla adapters with 4 times fewer parameters on all ViT backbones (ViT, Swin-T and CvT) models. These experiments show that TINA do generalize to various ViT backbones.

5 Conclusion

In this work we propose TINA, a training algorithm to learn TINy Adapters for the problem of ViT finetuning. Rather than directly training adapters with few parameters, we propose to start with large and over-parametrized adapters, and then, iteratively select the more important neurons in every adapter. Our training procedure estimates the hidden dimension for each adapter which reduces the number of trainable parameters and even removes adapters at certain layers when necessary. Experimentally we demonstrate the greater performance of our training scheme with respect to vanilla adapters and show that our method achieves good performance with unprecedentedly low numbers of trainable parameters. Our ablation study validates the positive impact of our adaptive strategy to estimate the hidden dimension of each adapter.

REFERENCES

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1.

Rodrigo Berriel, Stephane Lathuillere, Moin Nabi, Tassilo Klein, Thiago Oliveira-Santos, Nicu Sebe, and Elisa Ricci. Budget-aware adapters for multi-domain learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 382–391, 2019.

Yves Chauvin. A back-propagation algorithm with optimal use of hidden units. *Advances in neural information processing systems*, 1, 1988.

Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. Adaptformer: Adapting vision transformers for scalable visual recognition, 2022. URL https://arxiv.org/abs/2205.13535.

Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pp. 598–605. Morgan Kaufmann, 1990.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

Jianyuan Guo, Kai Han, Han Wu, Yehui Tang, Xinghao Chen, Yunhe Wang, and Chang Xu. Cmt: Convolutional neural networks meet vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12175–12185, 2022.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015a. doi: 10.48550/ARXIV.1510.00149.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015b.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. 2021.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning, 2022. URL https://arxiv.org/abs/2203.12119.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).

Yahui Liu, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco De Nadai. Efficient training of visual transformers with small datasets. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021a.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. pp. 10012–10022, 2021b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *ACL/IJCNLP*, pp. 565–576, 2021.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 67–82, 2018.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, volume abs/1608.08710. OpenReview.net, 2017.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11256–11264, 2019.

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

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

Yingwei Pan, Yehao Li, Qi Cai, Yang Chen, and Ting Yao. Multi-source domain adaptation and semi-supervised domain adaptation with focus on visual domain adaptation challenge 2019. 2019.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.39.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30, 2017.

Sylvestre-Alvise Rebuffi, Andrea Vedaldi, and Hakan Bilen. Efficient parametrization of multi-domain deep neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8119–8127, 2018. doi: 10.1109/CVPR.2018.00847.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. Adapterdrop: On the efficiency of adapters in transformers. In *EMNLP (1)*, 2021.

Enzo Tartaglione, Skjalg Lepsøy, Attilio Fiandrotti, and Gianluca Francini. Learning sparse neural networks via sensitivity-driven regularization. *Advances in neural information processing systems*, 31, 2018.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22–31, 2021.

Aston Zhang, Yi Tay, SHUAI Zhang, Alvin Chan, Anh Tuan Luu, Siu Hui, and Jie Fu. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with $1/n$ parameters. In *International Conference on Learning Representations*, 2021.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Michael C. Mozer, and Yoram Singer. Identity crisis: Memorization and generalization under extreme overparameterization. In *International Conference on Learning Representations*, 2020.

A APPENDIX

In this appendix, we provide: (1) additional experimental results to further analyze the proposed TINA approach. (2) justify in more detail our choice for the design of the importance score (3) provide details regarding the datasets used in our experiments.

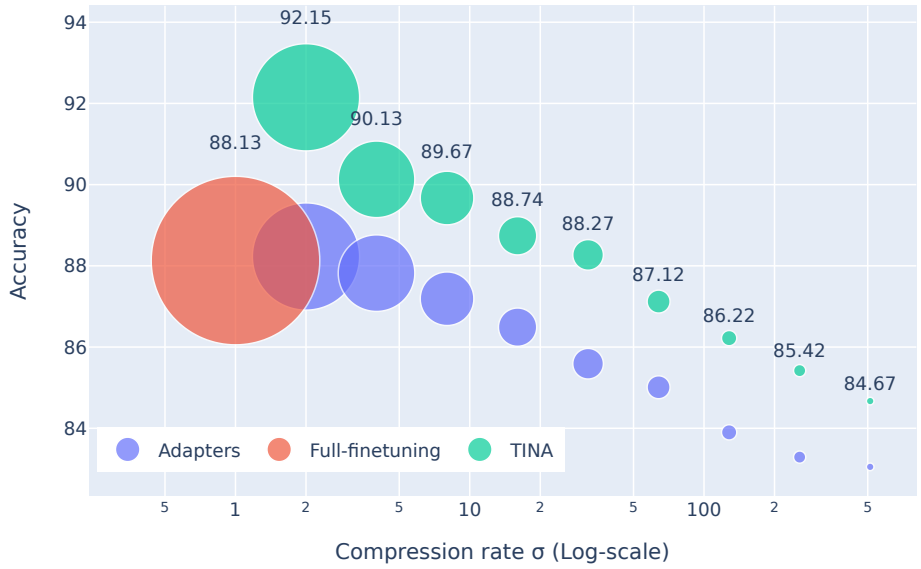


Figure 5: Comparison of top-1 accuracy of vanilla adapters, and TINA with respect to compression rate σ on CIFAR-100 dataset. Size of blob markers represents the number of trainable parameters. We notice that at $\sigma = 2, 4, 8$, TINA outperforms full finetuning.

A.1 LOCAL VERSUS GLOBAL NEURONS SELECTION

Table 5: A comparative performance analysis of local and global neuron selection on VGG-Flowers.

Method	**Pruning Method**	**Selection**	**Iterative**	**Scaling**	σ 32	64	128	256	512	1024	2048	4096
Vanilla adapters					94.80	90.12	89.42	88.85	86.03	86.09	85.14	85.14
Baselines	Local	✓			-	96.10	95.57	96.15	96.23	96.76	95.83	95.83
	Local	✓	✓		-	96.41	96.65	96.72	96.72	**96.81**	**96.83**	94.54
	Global	✓			-	94.88	95.28	95.66	95.45	95.56	96.03	96.03
	Global	✓		✓	-	96.10	95.82	96.34	96.50	96.15	96.03	96.03
TINA	Global	✓	✓	✓	-	**96.59**	**96.92**	**96.73**	**96.81**	96.55	96.47	**96.17**

Here below, we provide extra experiments for adapters with different sizes for VGG-Flowers, CIFAR-10, and CIFAR-100.

Tables 5and 6 report the performance of TINA algorithm with respect to vanilla training -*Baseline*- on VGG-Flowers, CIFAR-10 and CIFAR-100. TINA outperforms vanilla adapters on all the adapters with a significant performance gap. Interestingly, this gap in performance expands, as we compare smaller adapters $\sigma = 256, ..., 4096$ (higher compression ratio).

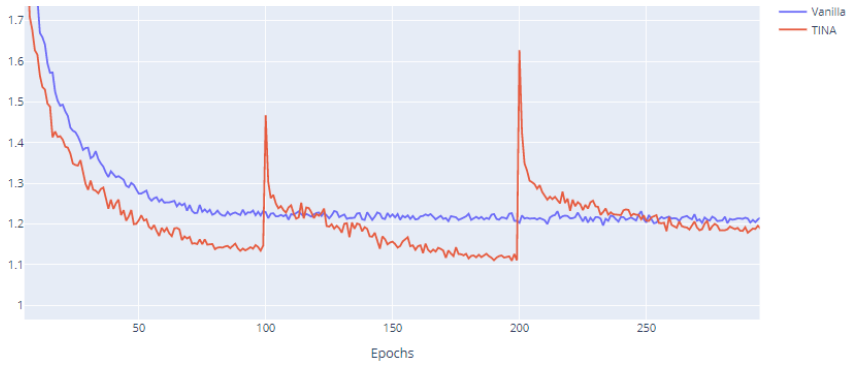
Furthermore, these results emphasizes on the usefulness of each component of the importance score for TINA. We notice that applying global neuron selection, normalization and iterative training outperforms local neuron selection on almost all adapter sizes for VGG-Flowers (Table 5) and CIFAR-100 (Table 6). This indicates that each component of the importance score of TINA is important to boost the performance and reduce the parameters.

Table 6: A comparative performance analysis of local and global neuron selection on CIFAR-10 and CIFAR-100.

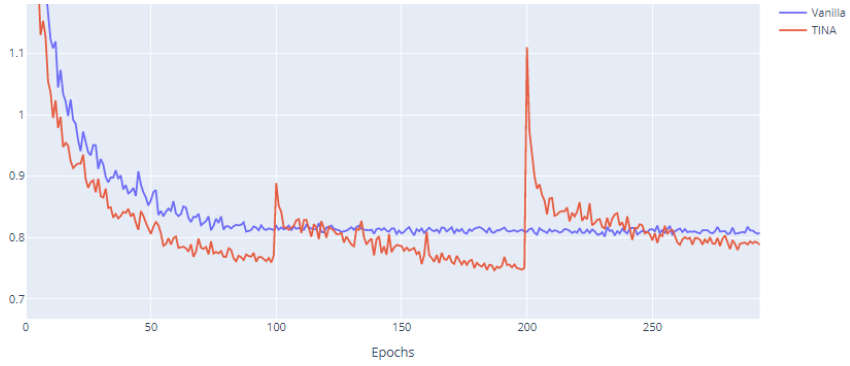
Method	Pruning Method	Selection	Iterative	Scaling	σ 32	64	128	256	512	1024	2048	4096
CIFAR-10												
Vanilla adapters	-				97.89	97.39	97.06	96.60	96.53	96.27	82.06	82.06
	Local	✓	✓		-	**98.06**	97.92	**97.64**	97.11	**96.91**	**96.44**	93.49
TINA	Global	✓	✓	✓	-	97.98	97.92	97.49	**97.15**	96.71	96.04	**95.57**
CIFAR-100												
Vanilla adapters	-				86.22	85.02	84.33	83.54	82.26	82.19	83.17	82.19
Baseline	Local	✓	✓		-	86.88	86.15	85.30	84.06	83.71	**83.35**	78.44
TINA	Global	✓	✓	✓	-	**87.12**	**86.22**	**85.42**	**84.67**	**83.25**	82.67	**82.10**

A.2 Vanilla versus TINA Training for Adapters

To validate the greater optimization performance of TINA, in Figs. 6a, 6b, we show the training loss curves of vanilla, and TINA training of adapters for CIFAR-100, and SVHN, respectively. At the end of training, the two models (*i.e.* vanilla training and TINA) have similar numbers of training parameters.



(a) Training loss on CIFAR-100.



(b) Training loss on SVHN.

Figure 6: Training loss curves of finetuning adapters with vanilla, and TINA training.

We notice that the loss of the training using TINA algorithm is much smoother than vanilla training resulting in adapters that generalize well on the downstream task as previously shown in Fig. 5. Furthermore, we notice spikes in the training loss of TINA, due to the removal of neurons after each cycle. Eventually, sequential training and neuron selection is a successful strategy to find small networks while maintaining good performance, since directly training small networks does not provide similar results Frankle & Carbin (2018).

A.2.1 IMPACT OF THE PARAMETER ALLOCATION

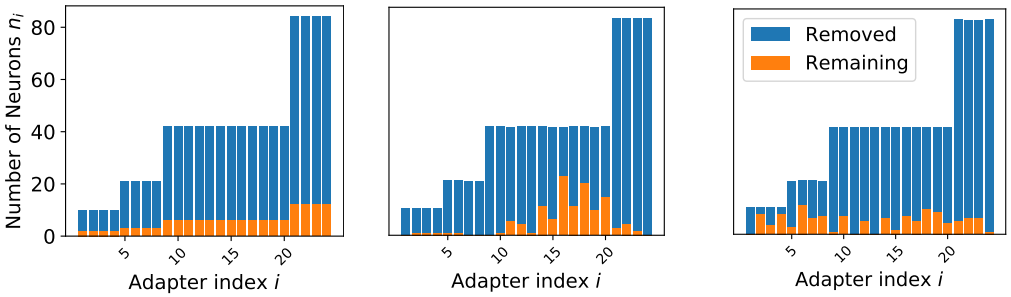
In this ablation study, we validate the idea that every layer of a ViT needd to be adapted differently in order to learn a new task. The Swin-B vision transformer model that we use in all our experiments consists of 4 stages. Therefore, we propose to evaluate the performance when we vary the adapter size in each stage. The results are reported in table 7. First, it shows that the size of adapters has a strong effect on the model performance. In general, the best performances are achieved when using adapters with higher number of parameters. Furthermore, bigger sizes of adapters are not sufficient for better performance, but subject to which stage they are injected into. We observe that adding adapters to late stages (*i.e.* III and IV), boosts the performance better than injecting them into early stages: adapters with $\sigma_i = 128$ added to (III, IV) stages rather than (I, II) improves the performance from *95.29%*, *73.38%* to *97.40%*, *87.04%* on CIFAR-10, and VGG-Flowers, respectively.

Table 7: Effect of adapters compression rate σ_i on adapters performance in terms for top-1 accuracy (%) for CIFAR-10 and VGG-Flowers datasets. Compression rate $\sigma = \infty$ equivalent to not adding adapter. We vary the σ_i value for each ViT stage (I, II, III, IV)

σ_i in each Swin Stage				Dataset	
I	**II**	**III**	**IV**	**CIFAR-10**	**VGG-Flowers**
128	128	32	32	**97.91**	**89.90**
32	32	128	128	97.64	87.05
128	32	128	32	97.84	89.45
32	128	32	128	97.72	88.49
128	128	128	32	97.79	89.22
256	128	64	32	**97.91**	89.84
32	64	128	256	97.43	86.90
128	128	∞	∞	95.29	73.38
∞	∞	128	128	97.40	87.04
∞	128	128	∞	96.97	84.65
128	∞	∞	128	96.53	80.84

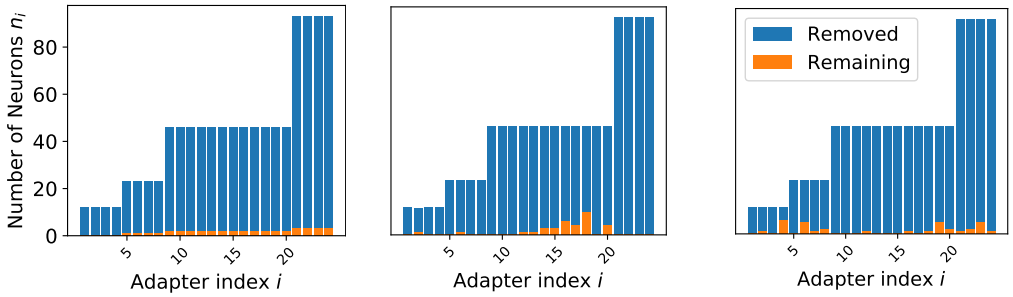
A.3 ILLUSTRATIONS OF LOCAL VERSUS GLOBAL NEURON SELECTION

Figures 7 and 8 show additional illustrations of the distribution of the removed and remaining neurons using TINA on VGG-Flowers, and CIFAR-10. We show the learned adapters at different cycles for both local and global neuron selection methods. We also complete these visualizations (Figures 9 and 10) with histograms where we show the percentages of remaining neurons. Overall, these experiments show that our method is capable of obtaining different parameter allocations that are specific to every task.



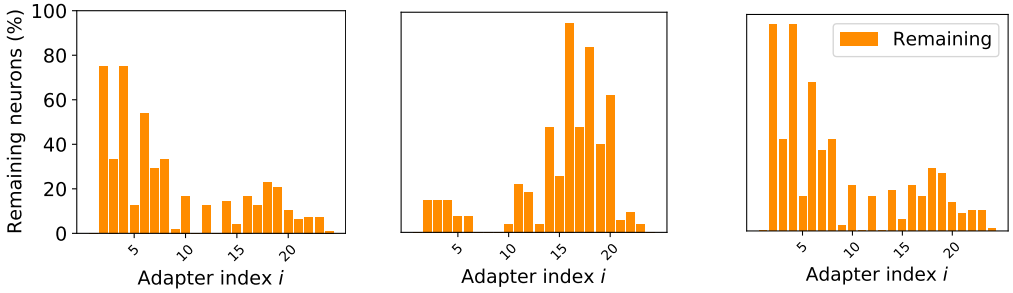
(a) Local pruning: all datasets. (b) Global pruning: VGG-Flowers . (c) Global pruning: CIFAR-10.

Figure 7: Layer-wise analysis of adapter's neurons distribution at 3^{rd} cycle. Bar plots represent number of neurons n_i at each adapter i using local, global neuron selection for VGG-Flowers and CIFAR-10, respectively.



(a) Local pruning: all datasets. (b) Global pruning: VGG-Flowers . (c) Global pruning: CIFAR-10.

Figure 8: Layer-wise analysis of adapter's neurons distribution at 5^{th} cycle. Bar plots represent number of neurons n_i at each adapter i using local, global neuron selection for VGG-Flowers and CIFAR-10, respectively.



(a) Local pruning: all datasets. (b) Global pruning: VGG-Flowers . (c) Global pruning: CIFAR-10.

Figure 9: Layer-wise analysis of adapter's neurons distribution at 3^{rd} cycle. **Normalized** bar plots represent **percentage (%) of remaining neurons** n_i at each adapter i using local, global neuron selection for VGG-Flowers and CIFAR-10, respectively.

A.3.1 Impact of ρ

In this section, we investigate the effect of the hyper-parameter ρ.

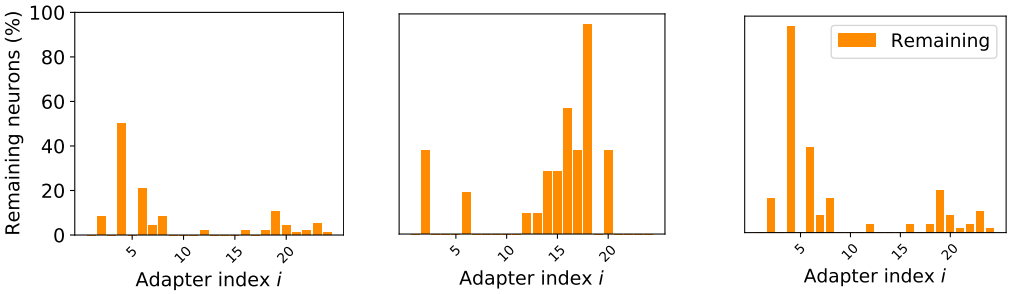
In figure 11. We notice that higher values of ρ hurts the performance, because we remove many parameters after each cycle, but we reduce the size of adapters significantly. On the other hand, if ρ is small (i.e 25%), we maintain good performance on VGG-Flowers dataset, but it requires higher training cycles C to reach the target compression rate σ_{target}.

We have a trade-off between the performance, and training budget in order to reach the σ_{target}. Removing too much parameters at each cycles, hurts performance. Maintaining good performance requires higher number of training cycles C.

A.3.2 Series Adapters Versus Parallel Adapters

In this experiment, we would like to investigate two different designs for adapters as illustrated in Figure 12. Traditional adapters known as series adapters (SA) Houlsby et al. (2019), and another variant parallel adapters (PA) He et al. (2021). The parallel configuration PA acts as a perturbation on the base network, the serial configuration more directly changes the hidden representations fed into the next layer. In this way, the parallel adapters are less prone to the loss of the 'knowledge' stored in the base model. Also, they are more efficient during training, where we can benefit from parallel operations. On the other hand, SA increases the latency of the model at inference time.

According to results in Table 8, SA outperforms PA with 1.5% on Quickdraw dataset. It indicates that acting directly on the feature allows more efficient adaptation to the new task. Furthermore, we



(a) Local pruning: all datasets. (b) Global pruning: VGG-Flowers . (c) Global pruning: CIFAR-10.

Figure 10: Layer-wise analysis of adapter's neurons distribution at 5^{th} cycle. **Normalized** bar plots represent **percentage (%) of remaining neurons** n_i at each adapter i using local, global neuron selection for VGG-Flowers and CIFAR-10, respectively.

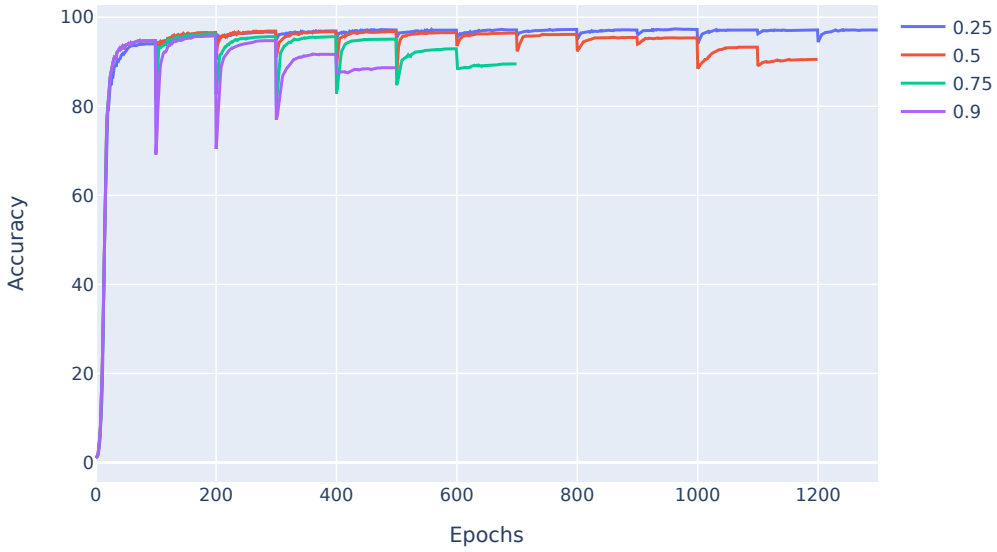


Figure 11: Analysis of TINA performance on VGG-Flowers dataset with different values of ρ. If ρ is very high, the drop in performance is significant, but it requires less C training cycles to reach σ_{traget}.

notice that both PA, and SA benefits from the higher sizes of adapters, where over-parametrization eases the optimization process and improves performance.

A.4 Magnitude assumption as importance score

In this section we provide more insights on the importance score employed within TINA. In particular, under Gaussian input assumption for adapters and imposing weight decay at training time, we will see that, towards a better choice of parameters to be removed, considering just $\boldsymbol{W}_i^{down}$ is sub-optimal, and $\boldsymbol{W}_i^{up}$ should be accounted as well. We drop the adapter index i for abuse of notation, as we will always refer to the same adapter.

Let us have an m-dimensional input $\boldsymbol{h}$, whose elements are distributed according to a Gaussian $\mathcal{N}(\mu_k, \Sigma_k)$. We assume the adapter has already been trained; hence we consider, in the down-sampling phase all the w_{jk}^{down} as constants. Form the property of linear expectations, we know that, before reaching the non-linear activation, the post-synaptic potential is still a Gaussian random

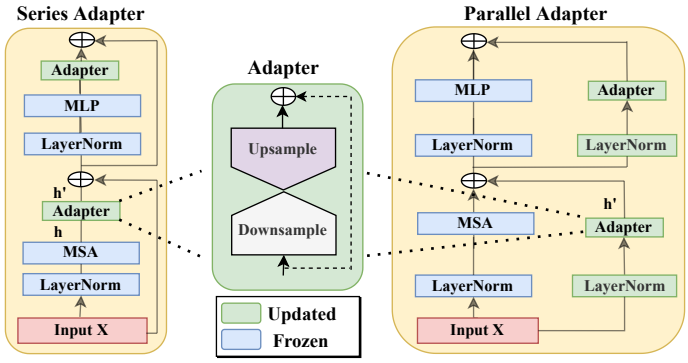


Figure 12: Two different designs for Adapter. Series Adapters (SA) Houlsby et al. (2019), and our proposed variant parallel adapters (PA). PA does not alter the representation of the pretrained model.

Table 8: Comparison between Series Adapters (SA) and Parallel Adapters (PA)with different compression ratio σ on Quickdraw with adapters. The number of parameters is reported in million and includes the linear classifier parameters

Method	$\sigma = 32$	$\sigma = 64$	$\sigma = 128$	$\sigma = 256$
SA	**69.29**	**67.45**	**65.96**	**64.32**
# Params (in M)	1.4	0.8	0.5	0.4
PA	67.91	66.0	64.49	62.92
# Params (in M)	1.4	0.8	0.6	0.4

variable, having average

$$\mu_j^{down} = \sum_{k=1}^{m} W_{jk}^{down} \cdot \mu_k \tag{3}$$

and variance

$$\Sigma_j^{down} = \sum_{k=1}^{m} W_{jk}^{down} \cdot \left[W_{jk}^{down} \Sigma_{kk} + 2 \sum_{k'<k} W_{jk'}^{down} \Sigma_{kk'} \right] \tag{4}$$

where Σ_{ab} indicates an element of the covariance matrix for the input of the adapter. For sake of easier tractation, if we assume $\Sigma_{kk'} = 0 \; \forall k \neq k'$, equation 4 simply reduces to

$$\Sigma_j^{down} = \sum_{k=1}^{m} \left(W_{jk}^{down} \right)^2 \Sigma_{kk}. \tag{5}$$

In transformers, the commonly-used activation function is the Gaussian error linear unit (GELU), whose analytical expression is

$$\phi(x) = x \cdot \frac{1}{2} \left[1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right] \tag{6}$$

where $\text{erf}(\cdot)$ is the error function. For values close to zero, or larger than zero, it can be approximated to the identity function, while for values much lower than zero, it asymptotically tends to zero. Let us focus on the first scenario: we can approximate the post-synaptic potential to the output of the non-linearity, saying that the output

$$z_j \approx \mathcal{N}(\mu_j^{down}, \Sigma_j^{down}). \tag{7}$$

At this point, the signal undergoes an up-sampling: following-up on the same approach adopted for the down-sampling, we find that the output $\boldsymbol{r}$ still follows an Gaussian distribution having average

$$\mu_l^{up} = \sum_{j=1}^{n} W_{jl}^{up} \mu_j^{down} = \sum_{j=1}^{n} W_{jl}^{up} \sum_{k=1}^{m} W_{jk}^{down} \cdot \mu_k \tag{8}$$

and variance

$$\Sigma_l^{up} = \sum_{j=1}^{n} \left(W_{jl}^{up}\right)^2 \Sigma_j^{down} = \sum_{j=1}^{n} \left(W_{jl}^{up}\right)^2 \sum_{k=1}^{m} \left(W_{jk}^{down}\right)^2 \Sigma_{kk}. \tag{9}$$

$$\begin{aligned} \mu_{l,\bar{a}}^{up} &= \mu_l^{up} - W_{al}^{up} \sum_{k=1}^{m} W_{ak}^{down} \cdot \mu_k \\ \Sigma_{l,\bar{a}}^{up} &= \Sigma_l^{up} - (W_{al}^{up})^2 \sum_{k=1}^{m} \left(W_{ak}^{down}\right)^2 \Sigma_{kk}. \end{aligned} \tag{10}$$

In order to assess the impact of removing a whole neuron in the embedding space, we can write the KL-divergence of the distribution for r_l with and without the a-th neuron in the embedding space

$$\begin{aligned} D_{\mathrm{KL}}(r_l, r_{l,\bar{a}}) = &\log\left(\frac{\Sigma_l^{up} - (W_{al}^{up})^2 \sum_{k=1}^{m} \left(W_{ak}^{down}\right)^2 \Sigma_{kk}}{\Sigma_l^{up}}\right) + \\ &+ \frac{(\Sigma_l^{up})^2 + \left(\mu_l^{up} - \mu_l^{up} + W_{al}^{up} \sum_{k=1}^{m} W_{ak}^{down} \cdot \mu_k\right)}{2 \cdot \left[\Sigma_l^{up} - (W_{al}^{up})^2 \sum_{k=1}^{m} \left(W_{ak}^{down}\right)^2 \Sigma_{kk}\right]^2} - \frac{1}{2}. \end{aligned} \tag{11}$$

According to equation 10, we can rewrite equation 11 as

$$\begin{aligned} D_{\mathrm{KL}}(r_l, r_{l,\bar{a}}) = &\log\left(1 - \frac{(W_{al}^{up})^2 \sum_{k=1}^{m} \left(W_{ak}^{down}\right)^2 \Sigma_{kk}}{\Sigma_l^{up}}\right) + \\ &+ \frac{(\Sigma_l^{up})^2 + \left(W_{al}^{up} \sum_{k=1}^{m} W_{ak}^{down} \cdot \mu_k\right)}{2 \cdot \left[\Sigma_l^{up} - (W_{al}^{up})^2 \sum_{k=1}^{m} \left(W_{ak}^{down}\right)^2 \Sigma_{kk}\right]^2} - \frac{1}{2}. \end{aligned} \tag{12}$$

Let us now investigate which is the a-th neuron which, when removed, causes the least perturbation at the output r_l (or in other words, such that $D_{\mathrm{KL}}(r_l, r_{l,\bar{a}})$ is as low as possible). Looking at the argument of the logarithm, we ask $\frac{(W_{al}^{up})^2 \sum_{k=1}^{m} (W_{ak}^{down})^2 \Sigma_{kk}}{\Sigma_l^{up}} = 0$ and, since we can safely assume Σ_l^{up}, we need to select a such that $(W_{al}^{up})^2 \sum_{k=1}^{m} \left(W_{ak}^{down}\right)^2 \Sigma_{kk} = 0$. Considering that also $\Sigma_{kk} > 0 \; \forall k$, we satisfy the condition if either:

- $W_{ak}^{down} = 0 \; \forall k$, namely the L^1 norm for $\boldsymbol{W}_{-a}^{down}$ is zero;
- $W_{al}^{up} = 0$. Considering though that this condition needs to be satisfied for all the l outputs of the adapters, we ask $W_{al}^{up} = 0 \; \forall l$ or, in other words, the L^1 norm for $\boldsymbol{W}_{a-}^{up}$ is also zero.

We observe that, when either of the two conditions are met, the KL divergence is zero as

$$D_{\mathrm{KL}}(r_l, r_{l,\bar{a}}) = \log(1) + \frac{(\Sigma_l^{up})^2}{2 \cdot (\Sigma_l^{up})^2} - \frac{1}{2} = 0$$

We can also assume that, if either $W_{ak}^{down} = 0 \; \forall k$ or $W_{al}^{up} = 0 \; \forall l$, the norm of the non-zero parameters associated to some neuron a are small when training with any weight penalty regularizer (as the contribution to the output is zero, the signal is either not forward or back-propagated, leaving the weight penalty term the only update for these parameters). This further motivates our proposed importance score given in equation 2.

A.5 Details about Datasets

In Table 9, we report different statistics that capture the diversity of the datasets we use in our experiments.

Table 9: Datasets used in our empirical analysis

	Dataset	Train Size	Test Size	Classes
Multi-task	CIFAR-10	50000	10000	10
	CIFAR-100	50000	10000	100
	Oxford Flowers	2040	6149	102
	SVHN	73257	26032	10
DomainNet	Clipart	33525	14604	345
	Infograph	36023	15582	345
	Painting	50416	21850	345
	Quickdraw	120750	51750	345
	Real	120906	52041	345
	Sketch	48212	20916	345

A.6 Baselines Details: PHM, Compacter, and LoRA

In order to have, a comparable number of trainable parameters for the baselines listed in Table 1 and Table 2. We set *Compacter* $n = 12$, and for *PHM*, $n = 12$.

In LoRa, the budget is expressed with a rank parameter r. We evaluate different rank values for LoRa and reported in our experiments the configuration leading to the best performance. For this preliminary experiment, we considered the Quickdraw dataset as a proxy to select the best r. The result is presented in Table 10. Rank $r = 8$ gives the best performance with a satisfying number of trainable parameters.

Table 10: Effect of rank dimension r on Quickdraw with LoRa fine-tuning.

Method	**n = 4**	**n = 8**	**n = 16**	**n = 32**
Accuracy (%)	59.72	**62.89**	59.7	61.87
# of Params	349209	419865	561177	843801