Self-Masking Networks for Unsupervised Adaptation

Alfonso Taboada Warmerdam[1][0000−0002−1767−367X], Mathilde Caron[2][0000−0001−6594−6698], and Yuki M. Asano[1][0000−0002−8533−4020]

[1] University of Amsterdam, Amsterdam
[2] Google Research, Grenoble

Abstract. With the advent of billion-parameter foundation models, efficient fine-tuning has become increasingly important for the adaptation of models to downstream tasks. However, especially in computer vision, it can be hard to achieve good performance when access to quality labeled data is lacking. In this work, we propose a method adapting pretrained generalist models in a self-supervised manner by learning binary masks. These **s**elf-supervised **m**asking **n**etworks (SMNs) are up to 79x more efficient to store and significantly improve performance on label-efficient downstream tasks. We validate the usefulness of learning binary masks as a fine-tuning method on 8 datasets and 3 model architectures, and we demonstrate the effectiveness of SMNs in 3 label-efficient settings.

Keywords: Fine-Tuning · Self-Supervised Learning · Masking.

1 Introduction

Recent advancements in large-scale pretraining of visual encoders, such as CLIP [33], DINO [7] or MAE [20], have shown remarkable and generalizable performances across a variety of computer vision tasks. Conventionally, these networks are fine-tuned for specific downstream tasks by adjusting their weights through gradient descent, either by training an additional layer on top of the

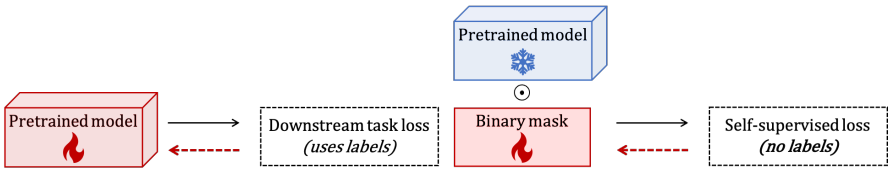


Fig. 1: **Conceptual comparison between two adaptation mechanisms:** Standard full-finetuning *versus* self-supervised self-masking (Ours).

pretrained network or by fine-tuning the entire network. One limitation of full-fine-tuning is that it necessitates a copy of all the fine-tuned weights to be stored for each downstream task. This leads to significant memory requirements, especially as vision models are reaching parameter counts of >22 Billion [12]. In contrast, simpler methods like linear probing are often limited in how well they can adapt the base model, as they only have access to the frozen deep representations.

Previous attempts at solving this challenge include adding light-weight learnable adapters [35], prompt learning [29,2,25], where additional inputs are learned, side-tuning [44] and model soups [42], wherein another network is trained and then 'fused' with the frozen model, and head-to-toe adaptation [15], which uses intermediate features from all layers of the pretrained network to train a classification head. However, these attempts are usually only evaluated in a supervised setting, while in practice the availability of labels for downstream task adaptation can be lacking.

In this paper, we explore the potential of finding subnetworks within trained networks in a self-supervised manner in order to both reduce the memory needs of fine-tuned models and adapt effectively to downstream tasks in label-sparse situations. The memory requirements compared to standard fine-tuning techniques are reduced because masks, with their binary weights, are more compact than full copies of the network weights. Hence, it is possible to adapt a model to thousands of downstream tasks by storing thousands of cheap binary weights and only one copy of the original model weights. To further explore the potential of this method, we also explore the possibility of chaining multiple adapted models together to yield 'model cascades': Here, we train multiple binary masks on different, coherent subsets of the downstream dataset in a self-supervised manner in order to increase downstream accuracy with a negligible increase in model size. This is done in a fully self-supervised manner.

2 Related work

Frozen Network Adaptation. Recent work has shown promising results in adapting pretrained models without modifying the core network weights, which would in theory preserve the original models' performance while enabling it to cater to novel tasks. One such technique is the application of lightweight feature adapters [18,45], which introduce trainable parts into the network while keeping the majority of the model frozen. Other methods include fine-tuning only the bias parameters of pretrained models [5,4], learning low-rank adaptations [24], or learning additional inputs to one or multiple layers in a pretrained vision transformer, such as Visual Prompting [3,13,26], which only learns additional inputs, or [25], which also fine-tunes a linear classifier on top of the model.

Masking Neural Networks. Mallya *et al.* [31] were the first to use the pass-through trick, a method that learns subnetwork masks directly through gradient descent. They applied this method to pretrained models to achieve domain adaptation, the new method outperformed their previous work [32] and reached

performance very close to conventional fine-tuning. Ramanujan *et al.* [34] found that a similar technique is also able to find well-performing domain-specific subnetworks on random, untrained networks, with implications regarding the Lottery Ticket Hypothesis [17,30]. However, they found that the base network had to be significantly wider in order to perform as well as standard, weight-modifying gradient descent. In follow-up work, Wortsman *et al.* [43] expand on this approach to teach a random, untrained network thousands of tasks. A similar technique, in combination with training the actual neural network weights has also been applied as a pruning technique for CNNs and vision transformers [38,27] as well as for domain generalization [8].

Our masking method is functionally equivalent to that of [31], however, we remove redundant hyperparameters (see section 3.2) and we simplify the scaling necessary to keep the variance constant when neural network weights are masked. We do this by scaling the network weights during forward propagation rather than the gradients (see Eq. 1). This is similar to Ramanujan *et al.* [34]'s method, but since they fix the percentage of weights that will be masked beforehand, they can instead scale the weights once during initialization.

Other fields where this technique has been applied include Neural Architecture Search [41], where the pass-through trick is used to determine where to add or remove a connection, and the design of novel neural network architectures such as neural networks with binary weights (-1 or 1) and ternary weights (-1, 0 or 1) [9,10,28,46]. These works have demonstrated the effectiveness of the pass-through trick for learning certain kinds of discrete components through gradient descent. With regard to sparsity in general, [22] have provided a good overview of the different use-cases and methods for sparsity in neural networks.

Self-supervised Learning on Restricted Domains. Recently, self-supervised approaches have gained traction as a way to improve the performance of computer vision models both in general and in label sparse situations [7,6,21]. These approaches enable a model backbone to be trained without the use of labels, where the focus has mostly been on learning good foundation models. However, some work has also shown that self-supervised learning can be used to improve performance on downstream tasks in computer vision, either through self-supervised knowledge distillation [16,14], or full-network adaptation [36], which has also been done in natural language processing [19,23]. We build on the idea of self-supervised adaptation and apply it to mask-learning with the simple yet effective self-supervised SwAV loss [6].

3 Method

Our goal is to adapt pretrained networks in a storage-efficient manner without labels. In a nutshell, our method learns a binary mask for a pretrained network with a self-supervised loss. In this section, we will recall the background on learning masks for neural networks followed by our contributions of provably removing unnecessary hyperparameters and show how this method can be used to learn multiple expert models for a given domain to improve adaptation performance.

3.1 Background: Network Masking

Subnetworks are represented by a binary mask M which indicates the active weights and the ones that are zeroed out. To learn an appropriate mask, each weight is assigned a corresponding score s, initialized to a value higher than a threshold μ [34]. At inference, a weight θ_i is then considered 'active' if its score is larger than μ, otherwise, the mask is set to zero and the weight is unused:

$$\theta_i = \frac{\theta_i}{\alpha} \cdot M_{\theta_i} \tag{1}$$

,

where $M_{\theta_i} = \mathbb{I}[S_{\theta_i} > \mu]$, and $\alpha = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[S_{\theta_i} > \mu]}$ is a scaling term to keep the variance of a weight matrix with N parameters constant when weights are deactivated [34].

Pass-through trick for training. Since the gradient of the score is lost when a weight is deactivated, the score cannot be learned directly. Instead, the gradient with respect to the mask is used to update the scores: $S^t = S^{t-1} - \lambda \frac{d\mathcal{L}^{t-1}}{dM^{t-1}}$, where $\mathcal{L}^{t-1}$ denotes the loss at time step $t-1$, and λ the learning rate. The key to implementing this in practice is to set the gradient of the score with respect to the loss to be the gradient of the mask with respect to the loss, $\frac{d\mathcal{L}^t}{dS^t} = \frac{d\mathcal{L}^t}{dM^t}$.

Which results in

$$S^t = S^{t-1} - \lambda \frac{d\mathcal{L}^{t-1}}{dS^{t-1}}, \tag{2}$$

which is the standard Stochastic Gradient Descent (SGD) update equation for parameter S. This approach, called the pass-through trick, thus allows standard SGD algorithms to update the score and learn a binary mask.

Intuitively, this method is able to optimize the loss as follows: The update equation makes sure that if increasing the mask would reduce the loss, the score is increased, and vice versa. Through multiple gradient descent iterations, the score then decreases for weights that increase the loss, eventually reaching the threshold and deactivating the weight. Conversely, if a deactivated weight would now reduce the loss when active, the score will increase, and the weight will eventually be reactivated.

3.2 Hyperparameter-free masking

The existing approach can learn a mask for a network given some loss function, this comes however at the cost of introducing two additional hyperparameters when compared to standard full-fine-tuning: the threshold μ and the initial score value S^0. We show that it is possible to remove *both* hyperparameters:

Theorem 1. *Translation invariance of threshold and initialization. Shifting the score initialisation S^0 and the threshold μ by an equal amount does not affect SGD-based training without weight-decay.*

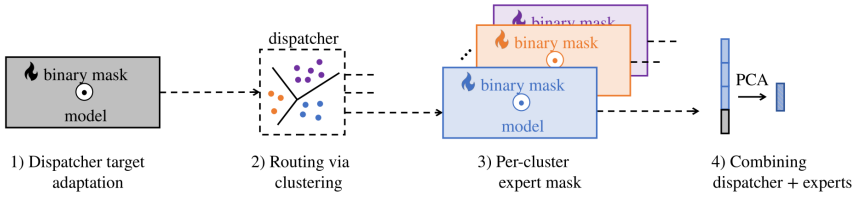


Fig. 2: **Cascade models** work by complementing the root model's features with those of expert models which are tailored to specific parts of the training distribution.

The proof follows from the idea that if the initial mask remains the same after translating both parameters equally, then the gradient will also be the same and so will the mask in the next training step.

Proof. As detailed in Section 3.1 , the mask $M^t_{\theta_i}$ for a given weight θ_i on the training step t is given by

$$M^t_{\theta_i} = \mathbb{I}[S^t_{\theta_i} > \mu]$$

. For ease of notation, we omit θ_i from this point on, i.e. $M^t_{\theta_i} = M^t$ and $S^t_{\theta_i} = S^t$. Now, given the update equation (Eq. 2) , S^t can be formulated as

$$S^t = S^0 - \sum_{\hat{t}=0}^{t-1} \lambda^{\hat{t}} g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}})$$

where $g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}})$ is the gradient of the the weight θ_i's score given the mask $\mathcal{M}^{\hat{t}}$ and mini-batch $\mathcal{B}^{\hat{t}}$. Note that the gradient can be fully determined by these two variables (plus all constant parameter such as the model weights). Note also that we assume the computation of the gradient to be deterministic, i.e. the same gradient will be computed for the same input, and the input $\mathcal{B}^{\hat{t}}$ is only dependent on $\hat{t}$. Combining these two equations we get

$$\begin{aligned} M^0 &= [S^0 > k] && (3)\\ &= [S^0 + a > k + a] && (4)\\ M^t &= [S^0 - \sum_{\hat{t}=0}^{t-1} \lambda^{\hat{t}} g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}}) > k] && (5)\\ &= [S^0 + a - \sum_{\hat{t}=0}^{t-1} \lambda^{\hat{t}} g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}}) > k + a] && (6) \end{aligned}$$

.

Eq. 4 indicates that replacing S^0_i with $S^0 + a$ and k with $k + a$ will not change M^0, since this is true for every individual mask, the full network mask $\mathcal{M}^0$ is also invariant to this change. This means that the gradient $g_i(\mathcal{M}^0, \mathcal{B}^0)$ does not

change, and consequently, M^1 is also invariant to this change, as indicated by Eq. 6. The same reasoning can be applied recursively to M^2 and so on. Thus, by induction, translating the initial score and threshold by equal amounts will not change any of the network masks during training with SGD w/o weight decay.

This eliminates the threshold parameter μ and leaves us with only the set of initialization scores S^0. However, even these can be subsumed into an already existing hyperparameter for training, the learning rate:

Theorem 2. *Learning rate and score initialization equivalence. Scaling the score initialization S^0 by a factor α is equivalent to scaling the learning rate λ by a factor $\frac{1}{\alpha}$.*

Proof. The equation for SGD with weight decay is given below:

$$S^t = S^{t-1} - \lambda^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \gamma S^{t-1} \right)$$

Say we scale the learning rates and scores by α, and the weight decay by $1/\alpha$. I.e. we replace λ^t with $\lambda^t \alpha$, S^{t-1} with $S^{t-1}\alpha$ and γ with $\frac{\gamma}{\alpha}$ for some $\alpha \in \mathbb{R}^+$, then we arrive at:

$$\begin{aligned} &\alpha S^{t-1} - \alpha\lambda^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \frac{\gamma}{\alpha} \alpha S^{t-1} \right) && (7) \\ &\quad = \alpha \left(S^{t-1} - \lambda^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \gamma S^{t-1} \right) \right) \\ &\quad = \alpha S^t && (8) \end{aligned}$$

In other words, the update equation then provides the same updated score, except it is also scaled by α, like the input score.

Similarly to the previous proof, the initial masks $M^0 = [S^0 > 0]$ are invariant to the scale change $M^0 = [\alpha S^0 > 0]$, so the replacement of S^0 with αS^0, combined with the other replacements, does not change the gradient $g_i(\mathcal{M}^0, \mathcal{B}^0)$. Combined with eq 8, this means that the updated parameter after the first SGD step is only different in scale when compared to what it would have been without the scale change ($= \alpha S^t$). Apply this reasoning recursively and it can be seen through induction that the network masks will be the same during training as for the original learning rate, score initialization and weight decay.

Practical implications Note that, in the interest of keeping the proofs simple, momentum is omitted from the derivations. However, we ran the experiment in Appendix A, which validates these proofs experimentally, with momentum enabled. This indicates that this invariance also holds if momentum is included. Also note that other SGD settings, which we left disabled in our research, may break the proven invariances. The translation invariance for instance *does not* hold if weight decay is enabled.

These two theorems indicate that, if we keep weight decay disabled, the hyperparameter combination $\lambda = 50$, $S^0 = 1.0$, $\mu = 0.0$ is equivalent to the drastically different hyperparameter combination $\lambda = 100$, $S^0 = 2.5$, $\mu = 0.5$ for example. In theory this means that masking the same model under exactly

the same conditions, except choosing a different equivalent hyperparameter combination, will result in exactly the same output masks. For this reason, we run all our masking experiments with $\mu = 0$ and $S_0 = 1$ with no loss of expressiveness, provided that weight decay is disabled (which we also disable). We show in Appendix A that this theoretical result also applies in practice.

3.3 Label-free adaptation

Since eq. (2) can work with any loss function $\mathcal{L}$. We can make this approach work on arbitrary data without any annotations by utilizing a self-supervised loss based on optimal-transport clustering [1,6]:

$$\ell(x_t, x_s)) = -\sum_k \mathbf{\Phi}(x)_s^{(k)} \log \mathbf{p}_t^{(k)}, \qquad \text{with} \quad \mathbf{p}_t^{(k)} = \frac{\exp\left(\frac{1}{\tau}\mathbf{\Phi}(x_t)^\top \mathbf{c}_k\right)}{\sum_{k'} \exp\left(\frac{1}{\tau}\mathbf{\Phi}(x_t)^\top \mathbf{c}_{k'}\right)} \tag{9}$$

where x_t, x_s are two augmented views of an image x, c_k are a set of entropy-regularised prototypes learned with Sinkhorn-Knopp [11] and, in this case the visual encoder Φ is parameterised with a set of frozen weights θ and the learnable masking scores S.

3.4 Model Cascades

Besides training separate masks for adapting a single model efficiently to several domains, we next show how this approach can be used to obtain a set of fine-grained adapted models on a single dataset without supervision. To this end, we first adapt a pretrained model in the manner described above on some target dataset $\mathcal{D}$ to obtain the adapted network Φ_0. Next, the deep embeddings of this network are used to obtain a non-parametric routing function R via clustering into K sets. Finally, starting from the adapted network's initialization, each subset $\mathcal{D}_k$ is used to learn an 'expert' network. This follows from the findings of self-supervised models performing [14] and training [39] better in-domain. At inference, the R (*e.g.* a trained k-means module) simply routes based on the proximity to the nearest centroid to a single expert.

Combining Embeddings. Since the experts are trained independently of each other and from the dispatcher (besides starting off with the same weight scores), each model may produce wildly different embeddings for the same data point. Care must thus be taken regarding how these embeddings are combined. In addition, we require the model cascade to provide image embeddings of the same dimensionality as the original adapted model (the dispatcher). This is necessary in order to be able to do a fair comparison using a linear probe between the embedding quality from just the dispatcher versus the full cascade. Namely, the number of parameters of the supervised evaluation component, *i.e.* the weight matrix of the linear probe is the same for just the dispatcher and the cascade.

We thus decide to combine the individual models' embeddings with Principal Component Analysis (PCA) applied to a concatenation of the embeddings,

ensuring that the output dimensionality is the same as that of the dispatcher alone. We evaluate two different ways to concatenate the embeddings. Which are **unconditional** and **conditional** concatenation.

In the **unconditional** case, the dispatcher embedding $D(x)$ and all expert embeddings $E_1(x), \ldots, E_K(x)$ are concatenated for each datapoint. The concatenated embedding $\overline{e}$ is then given by

$$\overline{e} = [D(x), E_1(x), ..., E_K(x)] \tag{10}$$

In contrast, for the **conditional** case, only the dispatcher embedding $D(x)$ and one expert embedding $E_n(x)$ are combined for each datapoint. In this case, the expert is chosen by the router R by simply picking the cluster closest to the datapoint. The concatenated embedding $\overline{e}$ is then given by

$$\overline{e} = [D(x), ..., E_i(x), ...], \text{ where } i = R(x) \tag{11}$$

The embedding is padded to have the same length as the unconditional concatenation, where the embeddings from each expert are placed in different dimensions. Effectively, the concatenated embedding is the same as for the unconditional case, except that the expert embeddings from other clusters are zero'ed out. Consequently, only two forward passes (dispatcher and one expert) are needed to embed a new datapoint, as opposed to K.

Dimensionality Reduction. As these two approaches increase the dimensionality of the embeddings by a factor 6, we reduce dimensionality back to the original size ($F = 2048$) for a fair comparison to the original model. To this end, we first center the features such that $e'_i = \overline{e_i} - c_i$, where c_i is the mean of the i'th feature over the (concatenated) training set. The final cascade embedding is then given by $e^* = \text{diag}(1/S)V^T\overline{e}$, where $V \in \mathbb{R}^{K \cdot F \times F}$ is the matrix of eigenvectors with the F largest eigenvalues of the covariance matrix of the centered training set embeddings and S the vector of eigenvalues. We divide by the eigenvalues to make sure all features have approximately equal variance, which is expected by the linear probe algorithm applied afterwards in some experiments.

4 Experiments

4.1 Datasets and implementation

We compare the performance of the found subnetworks with standard full-fine-tuning in the supervised and self-supervised adaptation setting. In all our experiments we use SGD with a momentum of 0.9 and a batch size of 64. For finding the subnetworks, we set the hyperparameters $\mu = 0$, $S_0 = 1$ and use no weight decay, this configuration is used for all models, in all settings. We run experiments on the default ResNet-18 and ResNet-50 models from the TIMM repository [40] (supervised pretraining), SwAV [6] and DINO [7] (self-supervised pretraining), as well as Vision Transformers (ViT-B/32) from CLIP [33] (cross-modal contrastive pretraining). We use the datasets from [29].

Table 1: **Masking is a viable strategy for downstream task adaptation.** We report top-1 accuracy for nine image classification benchmarks with four different methods for downstream task transfer. For this experiment, we use supervised masks. We report the memory storage (in MegaBits, uncompressed) required to store the weights for a downstream task adaptation using the given method in the *Size* column. This excludes having to store the original model weights as these only need to be stored once.

Method	Size	CIFAR10	CIFAR100	DTD	EUROSAT	FLOWERS	PETS	SUN397	UCF101
Pretrained model: ResNet-18$_{Supervised}$									
k-NN	n/a	0.826	0.589	0.587	0.896	0.677	0.877	0.465	0.596
FFT	368	**0.950**	0.759	**0.692**	**0.969**	**0.963**	**0.889**	**0.534**	**0.689**
Mask	12	0.949	**0.760**	0.660	**0.969**	0.955	0.852	0.479	0.659
Pretrained model: ResNet-50$_{SwAV}$									
k-NN	n/a	0.832	0.497	0.693	0.754	0.728	0.726	0.535	0.604
FFT	736	**0.965**	**0.817**	**0.736**	**0.977**	**0.987**	**0.892**	**0.623**	**0.675**
Mask	23	0.962	0.798	0.709	0.974	0.967	0.863	0.482	0.628
Pretrained model: ViT-B/32$_{CLIP}$									
k-NN	n/a	0.909	0.694	0.666	0.858	0.818	0.768	**0.687**	0.753
FFT	2752	0.958	0.821	0.723	**0.979**	**0.974**	0.885	0.640	0.809
Mask	86	**0.971**	**0.834**	**0.738**	0.978	0.973	**0.891**	0.668	**0.815**

Supervised ResNets. For the supervised adaptation baseline on the ImageNet-pretrained ResNet18 model from the TIMM repository, we use the most common hyperparameters from [37] for every dataset on that model architecture, which is a learning rate of 0.001 and a weight decay of 5e-4. However, we use a simple cosine learning rate decay rather than a step decay. For the SwAV-pretrained Resnet-50 model, we use a learning rate of 0.15 and a weight decay of 1e-6. The learning rate was determined by taking the original learning rate the model was trained on and scaling it by the new batch size ($0.15 = 0.6 \cdot 64/256$). For finding the subnetworks of these models, we use a learning rate of 50 and a cosine learning rate decay with a linear warmup up to epoch 40. Both for finding the subnetworks and for the baselines of these models, we train with a standard cross-entropy loss and a new random linear head for 150 epochs. We only train or mask the convolutional and downsampling layers.

Supervised Transformers. For the CLIP-pretrained vision transformer baseline and linear probe, we use the results from [2,29], which uses the original head from the CLIP model and a cosine similarity metric between the image and text embeddings as the output logits, before using a cross-entropy loss. For finding the subnetworks of this model, we use the same loss and number of epochs (32) but stick to a simple cosine learning rate decay schedule with a learning rate of 10.

We only mask the projections and the Multilayer Perceptrons (MLP) after each attention block. All supervised experiments use data augmentations from [29].

Self-Supervised ResNets. The self-supervised experiments are done using the augmentations and other default settings from SwAV [7], except with a learning rate of 0.15, batch size of 64, no warm up and 500 prototypes. For finding subnetworks with self-supervision, we keep the prototypes and linear head trainable using the aforementioned hyperparameters (which are thrown away regardless), we only find a mask for the backbone, using the same parameters as for the supervised experiments. We train for 150 epochs and start the queue after 30 epochs.

Model Cascade. For the dispatcher in the Model Cascade, we train CIFAR100 for 150 epochs and INAT500 for 54 epochs, which corresponds to approximately the same number of steps. The experts are also trained for approximately the same number of steps (≈ 117188), based on the formula: $\mathcal{E} = 50000/\mathcal{D} * 150$, where $\mathcal{D}$ is the dataset size, which varies for each cluster, and $\mathcal{E}$ is the number of epochs, rounded to the nearest integer. We start the queue after the first $1/5$ of the epochs. For the router R we choose a 5-way Gaussian Mixture Model applied on dimensionality-reduced feature vectors using the 20 largest components of a Principal Component Analysis.

Table 2: **Comparison of our masking mechanism (SMN) with Ramanujan *et al.* [34]'s topk% method.** For completeness, we run the method of Ramanujan *et al.*with the sparsity-level that was found by our method. We report top-1 accuracy on different downstream tasks using different evaluation methods. In the last row, we report the sparsity levels found by our method on the different datasets. All numbers are reported by masking with a self-supervised loss, starting from a ResNet-50 pretrained with SwAV.

Method	Sparsity	CIFAR10	CIFAR100	DTD	EUROSAT	FLOWERS	SUN397	UCF101
k-NN evaluation								
topk%	50%	0.891	0.564	0.465	0.968	0.403	0.460	0.439
topk%	found	0.908	0.630	0.671	**0.973**	0.903	**0.519**	**0.572**
SMN	found	**0.921**	**0.656**	**0.674**	0.971	**0.920**	0.518	0.549
Linear probe evaluation								
topk%	50%	0.913	0.678	0.536	0.980	0.800	0.565	0.548
topk%	found	0.940	0.747	**0.733**	**0.984**	**0.985**	0.630	0.690
SMN	found	**0.950**	**0.769**	0.714	0.983	0.983	**0.631**	**0.697**
Sparsity level:		91.4%	92.5%	98.8%	96.7%	98.6%	94.4%	95.7%

Evaluation. Linear probes are done with logistic regression on the unaugmented training set embeddings, as done by [7]. k-NN evaluations are done with the

default settings from the same paper, with 200 neighbors and a temperature of 0.1. Source code is available at: https://github.com/alvitawa/UnsupervisedMasking.

4.2 Masking is a viable adaptation strategy

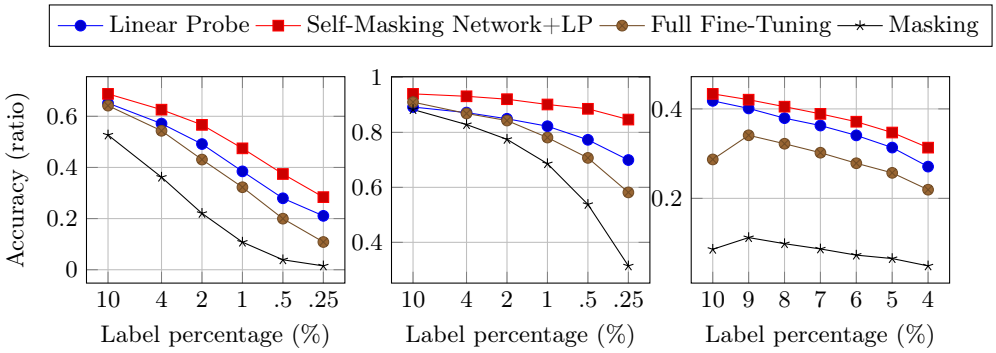
In this section, we validate that masking weights of a pretrained network is a viable solution to adapt it to a given downstream task by comparing our masking adaptation method to more standard transfer techniques such as nearest neighbors classification (k-NN) and full fine-tuning of the model parameters (FFT). For this experiment, we consider masking with a supervised loss to prevent confounding factors compared with full fine-tuning. For each of these methods, in addition to reporting the accuracy obtained of the downstream tasks, we also report the memory storage required to deploy the transfer technique. For k-NN, we consider that the memory requirement is none. However, one could argue that it is still required to store the embeddings of the training set of the dataset, which can quickly be at the order of magnitude of the gigabit for large datasets such as iNaturalist and embeddings of high dimension like ViT-B/32_{CLIP}.

Results comparing k-NN, full fine-tuning (FFT) and supervised masking adaptation (Mask) strategies are reported in table 1. We observe that masking produces good transfer performance on a large set of downstream tasks, and is even competitive with full fine-tuning. We see in table 1 that this observation is consistent both across network architectures (convolutional neural network and vision transformers) and pretraining paradigms (supervised, self-supervised and image-text contrastive). Overall, results in table 1 show that even though not widely adopted by the deep learning community, masking is a viable and storage-efficient adaptation technique. In addition, compared to full fine-tuning it requires 32 times less memory storage to deploy when looking at the uncompressed number of bits. However, in practice it can require 78.8 times less memory to store the masks due to the fact that the masks are more easily compressible than a full copy of model weights. This is shown experimentally in Appendix C.

4.3 Self-supervised adaptation with self-masking

Though efficient on the memory storage axis, masking, as well as other typical transfer learning approaches, arguably lack efficiency in terms of *label* utilization. Indeed, transferring a network to a downstream task is in many cases a process requiring access to many labels in the downstream domain to achieve good performance. In this section, we explore adaptation to a downstream domain in a label-efficient manner, by learning the transfer mask without using any labels.

Comparison of different masking strategies. In table 2, we compare the performance of our Self-Masking Network (SMN) with our masking method and that of Ramanujan *et al.* [34]'s topk% method, which activates a fixed percentage of the weights with the highest scores rather just weights with scores larger than the threshold. We find that our masking method (threshold masking) outperforms topk% masking when the sparsity is set to 50%, which is the sparsity level they



(a) CIFAR100 (b) CIFAR10 (c) SUN397

Fig. 3: **Low-shot adaptation with self-supervised self-masking.** We report top-1 accuracy after transferring to downstream tasks in a low-shot setting (% of labeled data used). We compare different adaptation techniques: linear probing, full fine-tuning, self-masking with a supervised objective and self-masking in a self-supervised manner. The pretrained network is ResNet-50_{SwAV}.

use in their paper. We also find that if we use the sparsity which was automatically found by our method, their method can come close to and is sometimes able to outperform our method. However, since Ramanujan *et al.*'s method is not able to find this sparsity by itself, we argue that our masking method is more practical.

Low-shot transfer. We propose to transfer to different downstream tasks by using only a fraction of the training labels, while keeping access to all training *images*. Results are presented in fig. 3. We observe that self-supervised self-masking outperforms conventional fine-tuning methods in all three datasets tested under low-shot conditions. We also observe in fig. 3 that the difference in performance is maintained or even exacerbated as fewer labels are used. This is likely due to the SMN's ability to utilize the unlabeled data for downstream adaptation. Overall, self-supervised masking is able to overcome the shortcomings of supervised masking in situations where few labeled data is available, providing a solution for both label-efficient and storage-thrifty downstream task transfer.

4.4 Model Cascade

In Table 3, we evaluate our cascade SMN's. We observe that the cascade mechanism allows for improving the performance of self-supervised masking by large margins, *i.e.* +3.8 points on CIFAR-100 and +2.6 points on iNat500 in linear probing evaluation. This comes at the cost of an increase in storage requirement (×6). However, the storage requirement is still significantly (×5) smaller than

Table 3: **Self-masking cascade.** We compare vanilla self-masking with conditional and unconditional self-masking cascades. We also report the (uncompressed) storage requirements relative to the cost of storing the original pretrained model in 32-bit floating point, which we denote by β. We exclude the cost of storing the original model (of size β) as well as the PCA and GMM parameters, which are negligible. The pretrained backbone used is ResNet-50_{SwAV}.

Method	Storage	CIFAR100	iNat500
k-NN evaluation			
Self-masking	$\beta/32$	0.656	0.263
Self-masking + cascade (conditional)	$6\beta/32$	0.752	0.395
Self-masking + cascade (unconditional)	$6\beta/32$	**0.778**	**0.424**
Linear probe evaluation			
Self-masking	$\beta/32$	0.769	0.524
Self-masking + cascade (conditional)	$6\beta/32$	0.793	0.521
Self-masking + cascade (unconditional)	$6\beta/32$	**0.807**	**0.550**

alternative transfer techniques like full finetuning, and does not use supervision for domain adaptation. Finally, we also observe that the unconditional variant of the cascade performs better than the conditional model. An explanation is that the experts learned from neighboring domains still capture useful features that are not completely orthogonal to each other and so benefit overall performances. Overall, this experiment shows that masking the same model multiple times using self-supervision through our cascade mechanism enables more information to be extracted from the training set, resulting in better downstream accuracy.

5 Conclusion

In this work, we started with the problem of adapting a pretrained model to a novel domain without knowledge of the final task or other manual annotations. To this end, we proposed Self-Masking, a simple approach for learning lightweight binary masks in an unsupervised manner. We have shown the benefits of Self-Masking are particularly pronounced for the important semi-supervised setting, where both large amounts of unlabeled data and small amounts of labeled data are available. Finally, we have shown how our approach can be used to produce high-performing model cascades, by selecting domains and training expert models without any supervision. We believe this work will gain importance with the increase in parameter counts of vision models.

References

1. Asano, Y.M., Rupprecht, C., Vedaldi, A.: Self-labelling via simultaneous clustering and representation learning. In: International Conference on Learning Representations (ICLR) (2020)
2. Bahng, H., Jahanian, A., Sankaranarayanan, S., Isola, P.: Exploring visual prompts for adapting large-scale models. arXiv preprint arXiv:2203.17274 (2022)
3. Bahng, H., Jahanian, A., Sankaranarayanan, S., Isola, P.: Visual prompting: Modifying pixel space to adapt pre-trained models. arXiv preprint arXiv:2203.17274 (2022)
4. Ben Zaken, E., Goldberg, Y., Ravfogel, S.: 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. Association for Computational Linguistics, Dublin, Ireland (May 2022). https://doi.org/10.18653/v1/2022.acl-short.1, https://aclanthology.org/2022.acl-short.1
5. Cai, H., Gan, C., Zhu, L., Han, S.: Tinytl: Reduce memory, not parameters for efficient on-device learning. Advances in Neural Information Processing Systems **33**, 11285–11297 (2020)
6. Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., Joulin, A.: Unsupervised learning of visual features by contrasting cluster assignments. Advances in Neural Information Processing Systems (NeurIPS) (2020)
7. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: Proceedings of the International Conference on Computer Vision (ICCV) (2021)
8. Chattopadhyay, P., Balaji, Y., Hoffman, J.: Learning to balance specificity and invariance for in and out of domain generalization. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IX 16. pp. 301–318. Springer (2020)
9. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. Advances in neural information processing systems **28** (2015)
10. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)
11. Cuturi, M.: Sinkhorn distances: Lightspeed computation of optimal transport. Advances in neural information processing systems **26** (2013)
12. Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A., Caron, M., Geirhos, R., Alabdulmohsin, I., et al.: Scaling vision transformers to 22 billion parameters. arXiv preprint arXiv:2302.05442 (2023)
13. Elsayed, G.F., Goodfellow, I., Sohl-Dickstein, J.: Adversarial reprogramming of neural networks. arXiv preprint arXiv:1806.11146 (2018)
14. Ericsson, L., Gouk, H., Hospedales, T.M.: How well do self-supervised models transfer? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5414–5423 (2021)
15. Evci, U., Dumoulin, V., Larochelle, H., Mozer, M.C.: Head2toe: Utilizing intermediate representations for better transfer learning. In: International Conference on Machine Learning. pp. 6009–6033. PMLR (2022)
16. Fang, Z., Wang, J., Wang, L., Zhang, L., Yang, Y., Liu, Z.: Seed: Self-supervised distillation for visual representation. International Conference on Learning Representations (ICLR) (2021)

17. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. International Conference on Learning Representations (ICLR) (2019)
18. Gao, P., Geng, S., Zhang, R., Ma, T., Fang, R., Zhang, Y., Li, H., Qiao, Y.: Clip-adapter: Better vision-language models with feature adapters. arXiv preprint arXiv:2110.04544 (2021)
19. Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., Smith, N.A.: Don't stop pretraining: Adapt language models to domains and tasks. Proceedings of ACL (2020)
20. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16000–16009 (2022)
21. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9729–9738 (2020)
22. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. The Journal of Machine Learning Research **22**(1), 10882–11005 (2021)
23. Howard, J., Ruder, S.: Universal language model fine-tuning for text classification. arXiv preprint arXiv:1801.06146 (2018)
24. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models. International Conference on Learning Representations (ICLR) (2021)
25. Jia, M., Tang, L., Chen, B.C., Cardie, C., Belongie, S., Hariharan, B., Lim, S.N.: Visual prompt tuning. In: Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII. pp. 709–727. Springer (2022)
26. Kloberdanz, E., Tian, J., Le, W.: An improved (adversarial) reprogramming technique for neural networks. In: Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part I 30. pp. 3–15. Springer (2021)
27. Lagunas, F., Charlaix, E., Sanh, V., Rush, A.M.: Block pruning for faster transformers. Empirical Methods in Natural Language Processing (2021)
28. Liu, B., Li, F., Wang, X., Zhang, B., Yan, J.: Ternary weight networks. In: ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1–5. IEEE (2023)
29. Loedeman, J., Stol, M., Han, T., Asano, Y.M.: Prompt generation networks for efficient adaptation of frozen vision transformers. arxiv preprint arxiv:2210.06466 (2022)
30. Malach, E., Yehudai, G., Shalev-Schwartz, S., Shamir, O.: Proving the lottery ticket hypothesis: Pruning is all you need. In: International Conference on Machine Learning. pp. 6682–6691. PMLR (2020)
31. Mallya, A., Davis, D., Lazebnik, S.: 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)
32. Mallya, A., Lazebnik, S.: Packnet: Adding multiple tasks to a single network by iterative pruning. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 7765–7773 (2018)

33. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PMLR (2021)
34. Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., Rastegari, M.: What's hidden in a randomly weighted neural network? In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
35. Rebuffi, S.A., Bilen, H., Vedaldi, A.: Learning multiple visual domains with residual adapters. Advances in neural information processing systems **30** (2017)
36. Reed, C.J., Yue, X., Nrusimha, A., Ebrahimi, S., Vijaykumar, V., Mao, R., Li, B., Zhang, S., Guillory, D., Metzger, S., et al.: Self-supervised pretraining improves self-supervised pretraining. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 2584–2594 (2022)
37. Salman, H., Ilyas, A., Engstrom, L., Kapoor, A., Madry, A.: Do adversarially robust imagenet models transfer better? Advances in Neural Information Processing Systems (NeurIPS) (2020)
38. Sanh, V., Wolf, T., Rush, A.: Movement pruning: Adaptive sparsity by fine-tuning. Advances in Neural Information Processing Systems (NeurIPS) (2020)
39. Tian, Y., Henaff, O.J., van den Oord, A.: Divide and contrast: Self-supervised learning from uncurated data. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10063–10074 (2021)
40. Wightman, R.: Pytorch image models. https://github.com/rwightman/pytorch-image-models (2019). https://doi.org/10.5281/zenodo.4414861
41. Wortsman, M., Farhadi, A., Rastegari, M.: Discovering neural wirings. Advances in Neural Information Processing Systems (NeurIPS) (2019)
42. Wortsman, M., Ilharco, G., Gadre, S.Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A.S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al.: Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In: International Conference on Machine Learning. pp. 23965–23998. PMLR (2022)
43. Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., Farhadi, A.: Supermasks in superposition. Advances in Neural Information Processing Systems (NeurIPS) (2020)
44. Zhang, J.O., Sax, A., Zamir, A., Guibas, L., Malik, J.: Side-tuning: a baseline for network adaptation via additive side networks. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16. pp. 698–714. Springer (2020)
45. Zhang, R., Fang, R., Zhang, W., Gao, P., Li, K., Dai, J., Qiao, Y., Li, H.: Tip-adapter: Training-free clip-adapter for better vision-language modeling. arXiv preprint arXiv:2111.03930 (2021)
46. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. International Conference on Learning Representations (ICLR) (2017)