Self-Masking Networks for Unsupervised Adaptation

Anonymous Author(s)
Affiliation
Address
email

Abstract

With the advent of billion-sized 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 self-masking networks (SMNs) are up to 32x 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 a wealth of 10 datasets and 3 model architectures, and we demonstrate the effectiveness of SMNs in 3 label-efficient settings.

1 Introduction

Recent advancements in large-scale pretrained networks, such as CLIP [1] or MAE [2], have demonstrated remarkable and generalizable performance 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 pretrained network or by fine-tuning the entire network. One limitation of this approach is that it necessitates a full copy of the fine-tuned weights to be stored for each downstream task, leading to significant memory requirements.

YA: add a paragraph: about previous attempts at this (learning adapters, prompt learning / linear probing / head-to-toe adaptation) and how they do not fully solve the problem.

In this work, we investigate the potential of identifying domain-specific subnetworks that achieve good performance on downstream tasks, without modifying the original pretrained network weights. This approach can be implemented in a self-supervised or supervised manner and involves training a mask that selectively deactivates certain network weights. The mask is trained separately for each downstream task, enabling the network to dynamically adapt to different tasks.

Moreover, this method can provide practical advantages in some scenarios, such as reducing memory requirements compared to standard fine-tuning techniques, as masks are smaller in size than full copies of the network weights. We take advantage of this property to train a self-supervised self-ensemble that can significantly improve performance on a downstream domain while barely increasing the memory required to store and use the model.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

2 Related Work

2.1 Continual Learning

Continual learning seeks to enable a single neural network to learn and perform multiple tasks sequentially without forgetting previously learned tasks. Several works have done this by identifying task-specific subnetworks that are efficient in terms of memory storage.

Mallya, Davis, and Lazebnik 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, their method reached performance very close to conventional fine-tuning. Ramanujan et al. found that a similar technique is also able to find well performing domain specific subnetworks on random, untrained networks. 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 a follow up work, Wortsman et al. expand on this approach to teach a random, untrained network thousands of tasks.

2.2 Pruning and Neural Network Architectures

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 Sanh, Wolf, and Rush[9].

Other fields where this technique has been applied include Neural Architecure Search[16], where Wortsman, Farhadi, and Rastegari use the pass-through trick 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)[4, 8]. These works have demonstrated the effectiveness of the pass through trick for learning certain kinds of discrete components through gradient descent.

2.3 Self-supervised Learning

Recently, self-supervised approaches have gained traction as a way to improve performance of computer vision models both in general and in label sparse situations[2, 3, 6]. These approaches enable a model backbone to be trained without the use of labels. The focus has mostly been on learning good foundation models, but some work has also shown that self supervised learning can also be used to improve performance on downstream tasks in computer vision [13] and natural language processing [5, 7]. Given the increasing importance of label-free machine learning, it is important to investigate whether the pass-through trick can be used in combination with self-supervised learning.

YA: generally: it's good to add some contextualisation behind a subsection: e.g. "Compared to these works we XXX." or "We use insights in from these works and instead apply them to YYY".

3 Method

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. A weight is active if its score is above the threshold, otherwise it is zeroed out.

3.1.1 Forward Pass

Formally, the consequent weight θ^t that is used for the forward pass of the training step t is then given by the formula

$$\theta^t = \frac{\overline{\theta} \cdot M^t}{\alpha} \tag{1}$$

where $\overline{\theta}$ is the fixed, pre-trained weight, and M^t is the value of the mask on that training step, as given by

$$M^t = \begin{cases} 1 & \text{if} \quad S^t > \mu \\ 0 & \text{if} \quad S^t \leq \mu \end{cases} \tag{2}$$

in terms of the score S^t and the threshold μ. The term α is a scaling term necessary to keep the variance of a weight matrix or vector constant when some weights are deactivated. This ensures that the original network's properties with regards to the change in variance after a transformation are largely preserved, i.e. the variance remains unchanged after a transformation. The scaling factor is given by

$$\alpha = \sqrt{\frac{1}{IJ} \sum_{i,j}^{I,J} M_{ij}} \tag{3}$$

, where the mask values for weights in the same weight matrix or vector are given by M_{ij}.

3.1.2 Backward Pass

All weights thus start out activated, and then the score is updated during gradient descent using the gradient of the loss with respect to the mask. If increasing the mask would reduce the loss, the score is increased, and vice versa. Through multiple gradient descent iterations, the score 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.

The following equation[1] describes this updating mechanism

$$S^t = S^{t-1} - \lambda \frac{d\mathcal{L}^{t-1}}{dM^{t-1}} \tag{4}$$

where $\mathcal{L}^{t-1}$ denotes the loss at time step $t-1$, and λ is 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

$$\frac{d\mathcal{L}^t}{dS^t} = \frac{d\mathcal{L}^t}{dM^t} \tag{5}$$

Which results in

$$S^t = S^{t-1} - \lambda \frac{d\mathcal{L}^{t-1}}{dS^{t-1}} \tag{6}$$

which is the standard Stochastic Gradient Descent (SGD) update equation for some parameter. This approach, called the passthrough trick, thus allows standard SGD algorithms to update the score in the manner shown.

3.1.3 Hyperparameters μ and S^0

Seemingly regrettably, this method introduces two additional parameters when compared to standard full-finetuning: the threshold μ and the initial score value S^0. However, if a constant initialization for S^0 is assumed, S^0 and μ can be set to arbitrary values with no loss of generality, as long as $S^0 > \mu$ and weight decay is not used in SGD. This is derived from proofs A.1.1 and A.1.2. The first proof shows that shifting the threshold and the score initialization by the same amount results in the same mask after training; thus, we arbitrarily set $\mu = 0$. The second proof demonstrates that scaling the score initialization by a factor α is equivalent to scaling the learning rate by a factor $\frac{1}{\alpha}$, so we set $S^0 = 1$.

[1] Momentum is also used in practice; see the appendix for the full equation

3.2 Model Cascades

Besides training separate masks for several tasks (on the same backbone), we also experiment with training multiple masks for the same task, in order to improve performance through a type of model ensembling. To do this, first, a model is adapted using SWAV[3] to a downstream dataset using the 'mask learning' method described above. Then, the whole training set is embedded using the adapted model, and the centered embeddings are clustered into five clusters, using a dimensionality reduction through Principal Component Analysis (PCA) followed by a gaussian mixture model (GMM) to create the cluster assignments. Finally, a new 'expert' mask is trained for each cluster, starting from the weight scores, prototypes and head of the original adapted model, which we will refer to as the 'dispatcher'.

3.2.1 Combining Embeddings

Since the experts are trained independently from 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 neccessary 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 another PCA application, but this time it is applied to a concatenation of the embeddings, and 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 $E_1(x), \ldots, E_5(x)$ are concatenated for each datapoint. The concatenated embedding $\overline{e}$ is then given by

$$\overline{e_i} = \begin{cases} D(x)_i & \text{if} \quad i < F \\ E_{\lfloor i/F \rfloor}(x)_{i \bmod F} & \text{otherwise} \end{cases} \tag{7}$$

where $\overline{e_i}$ is the i'th feature and F is the dimensionality of the embedding from the individual models.

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 that corresponds to the cluster the datapoint x is predicted to belong to, based on the GMM. The concatenated embedding $\overline{e}$ is then given by

$$\overline{e_i} = \begin{cases} D(x)_i & \text{if} \quad i < F \\ E_n(x)_{i \bmod F} & \text{if} \quad \lfloor i/F \rfloor = n \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

. Effectively, the concatenated embedding is the same as for the unconditional case, except that the expert embeddings from other clusters are zeroed out. This means that they do not have to be computed during evaluation. Consequently, only two forward passes (dispatcher and one expert) are needed to embed a new datapoint, as opposed to six.

Dimensionality Reduction Then, concatenated embeddings are centered 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 $e^* = \mathrm{diag}(1/S)V^T\overline{e}$, where $V \in \mathbb{R}^{(6 \cdot F) \times F}$ is the matrix of eigenvectors with the F largest eigenvalues of the covariance matrix of the centered training set embeddings, and S is the vector of eigenvalues.

4 Experiments

4.1 Datasets and implementation

We compare the performance of the found subnetworks with standard full-finetuning 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.

For the supervised adaptation baseline on the ImageNet-Pretrained Resnet18 model from the TIMM repository, we use the most common hyperparameters from [14] for every dataset on that model architecture, which is a learning rate of 0.001 and a weight decay of 0.0005. However, we use a simple cosine learning rate decay rather than a step decay. For the SWAV-Pretrained Resnet50 model, we use a learning rate of 0.15 and a weight decay of 0.000001. The learning rate was determined by taking the original learning rate the model was trained on and scaling it by the new batch size ($0.15 = 0.6 * 64/256$). 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 submask the convolutional and downsampling layers.

For the CLIP-Pretrained vision transformer baseline and linear probe, we use the results from [1, 10], which uses the original head from the CLIP model and a cosine similarity metric between the image and text embeddings as the model outputs, 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, besides using the hyperparameters already mentioned for the finding of subnetworks. For this we use a learning rate of 10. We only submask the projections and the Multilayer Perceptrons (MLP) after each attention block. All supervised experiments use data augmentations from [10].

The self supervised experiments are done using the default settings from SWAV [2], 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 anyways), we only find a mask for the backbone, using the same parameters as for the supervised experiments. We We train for 117187.5 steps, rounded to the nearest epoch. This corresponds to training a dataset of 50000 samples for 150 epochs with a batch size of 64. The formula to determine the number of epochs based on the dataset size $\mathcal{D}$ is then $50000/\mathcal{D} * 150$. We start the queue after the first 1/5 of the epochs.

Linear probes are done with logistic regression on the unaugmented training set embeddings, as done in [2].

4.2 Supervised adaptation

Table 1: **YA: MASKING IS VIABLE STRATEGY FOR FINETUNING.**Accuracy for ResNet models with a Linear Probe (+LP), a learned subnetwork (+M) and after Full Fine-Tuning (+FFT).

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	inat	oxfordpets	sun397	ucf101
rn18-timm+LP	0.87	0.59	0.62	0.92	0.93	0.42	**0.89**	0.49	0.66
rn18-timm+M	**0.95**	**0.76**	0.66	**0.97**	**0.96**	**0.50**	0.85	0.48	0.66
rn18-timm+FFT	**0.95**	**0.76**	**0.69**	**0.97**	**0.96**	**0.50**	**0.89**	**0.53**	**0.69**
rn50-swav+LP	0.91	0.64	**0.76**	0.73	0.98	0.60	0.88	**0.66**	**0.78**
rn50-swav+M	**0.96**	0.80	0.71	0.97	0.97	0.64	0.86	0.48	0.63
rn50-swav+FFT	**0.96**	**0.82**	0.74	**0.98**	**0.99**	**0.66**	**0.89**	0.62	0.67

Table 2: Accuracy for CLIP vision transformer with a Linear Probe (+LP), a learned subnetwork (+M) and after Full Fine-Tuning (+FFT).

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	oxfordpets	sun397	ucf101
vitb32-clip+LP	0.950	0.800	**0.746**	0.953	0.969	**0.892**	**0.750**	**0.833**
vitb32-clip+M	**0.971**	**0.834**	0.738	0.978	0.973	0.891	0.668	0.815
vitb32-clip+FFT	0.958	0.821	0.723	**0.979**	**0.974**	0.885	0.640	0.809

Table 3: Percentage of weights that are zeroed-out after masking different architectues.

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	inat	oxfordpets	sun397	ucf101
rn18-timm+M	15.234%	35.675%	7.448%	6.555%	6.896%	48.770%	4.420%	32.039%	10.558%
rn50-swav+M	10.460%	22.821%	4.927%	4.316%	5.495%	34.575%	3.810%	18.936%	8.334%
vitb32-clip+M	0.394%	1.268%	0.104%	0.122%	0.146%	5.227%	0.084%	0.704%	0.207%

4.3 Self-supervised adaptation

It can be seen from Table 4 that the self supervised methods are able to adapt the backbone to the new task for most tested datasets, with the Self-Masking Network performing somewhere in between the original network and the Self-Training Network, which gives it better performance in the cases where self-supervised adaptation is actually degrading the performance of the model. However, as can be seen from Table 5, the self-supervised adaptations rarely perform better than their supervised counterparts.

Table 4: KNN Accuracies comparison of our Self-Masking (SMN) and Self-Training (STN) networks with the pretrained, unadapted backbone. Caron et al. [3]

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	oxfordpets	sun397	ucf101
rn50-swav	0.83	0.50	**0.69**	0.58	0.73	**0.73**	**0.54**	**0.60**
rn50-swav+SMN	0.921	0.656	0.67	0.97	0.92	0.64	0.52	0.55
rn50-swav+STN	**0.937**	**0.707**	**0.69**	**0.98**	**0.96**	0.63	0.50	0.53

Table 5: KNN Accuracies comparison of our Self-Masking (SMN) and Self-Training (STN) networks with standard full fine-tuning (FFT) and masking (M).

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	oxfordpets	sun397	ucf101
rn50-swav+FFT	**0.96**	**0.80**	**0.71**	**0.98**	**0.97**	**0.89**	**0.59**	**0.67**
rn50-swav+STN	0.937	0.707	0.69	**0.98**	0.96	0.63	0.50	0.53
rn50-swav+M	**0.96**	**0.80**	**0.70**	**0.97**	**0.96**	**0.86**	0.47	**0.65**
rn50-swav+SMN	0.921	0.656	0.67	**0.97**	0.92	0.64	**0.52**	0.55

Table 6: Accuracies comparison of our Self-Masking (SMN) and Self-Training (STN) networks with standard full fine-tuning (FFT) and masking (M).

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	oxfordpets	sun397	ucf101
rn50-swav+LP	0.91	0.64	**0.76**	0.73	0.98	**0.88**	**0.66**	**0.78**
rn50-swav+SMN+LP	0.95	**0.77**	0.71	0.98	0.98	0.83	0.63	0.70
rn50-swav+STN+LP	**0.96**	**0.77**	0.72	**0.99**	**0.99**	0.75	0.59	0.65

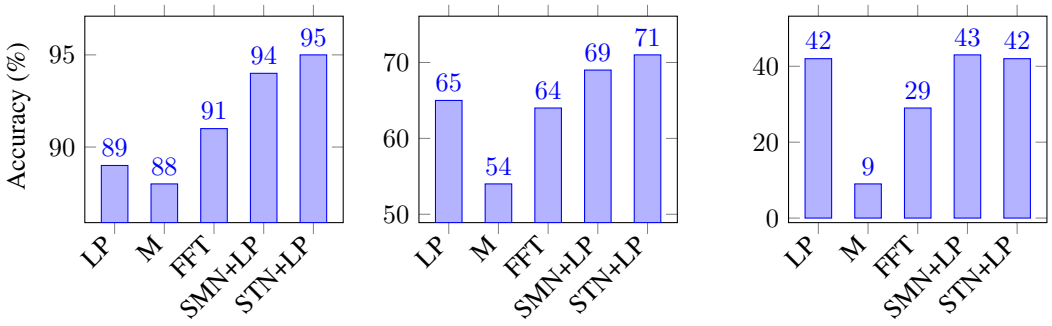
Table 7: Accuracies comparison of our Self-Masking (SMN) and Self-Training (STN) networks with standard full fine-tuning (FFT) and masking (M).

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	oxfordpets	sun397	ucf101
rn50-swav+M	**0.96**	**0.80**	**0.71**	0.97	0.97	**0.86**	0.48	0.63
rn50-swav+SMN+LP	0.95	0.77	**0.71**	**0.98**	**0.98**	0.83	**0.63**	**0.70**
rn50-swav+FFT	**0.96**	**0.80**	0.71	0.98	0.97	**0.89**	**0.59**	**0.67**
rn50-swav+STN+LP	**0.96**	0.77	**0.72**	**0.99**	**0.99**	0.75	**0.59**	0.65

4.4 Label Sparsity

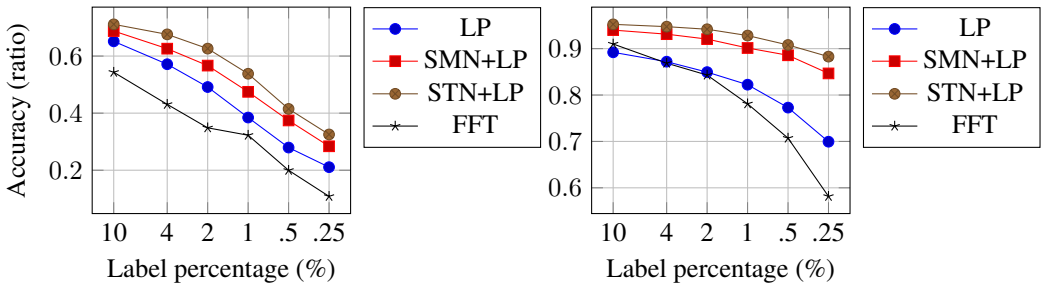
In Figure 1 it can be seen that our self-supervised methods outperform conventional fine tuning methods in all three datasets tested under label-sparse conditions. In addition, Figure 2 shows that the difference in performance is maintained or even excacerbated as fewer and fewer labels are used.

Figure 1: ResNet-50 (SWAV); Top@1 accuracy when comparing different ways to fine-tune the model backbone using only 10% of labeled data. From left to right: Linear Probing (LP), Masking (M), Full Fine-Tuning (FFT), Self-Masking Network trained on 100% of the unlabeled data with a linear probe on the remaining 10% (SMN+LP), a similarly applied Self-Training Network (STN+LP).



(a) CIFAR-10 (b) CIFAR-100 (c) SUN-397

Figure 2: ResNet-50 (SWAV); Top@1 accuracy of linear probe with different label sparsities (% of labeled data used) on ImageNet-Pretrained backbone (LP) versus our Self-Masking Network (SMN) and Self-Training Network (STN).



(a) CIFAR-100 (b) CIFAR-10

4.5 Model Cascade

Finally, Table 8 shows promising gains in performance when applying the unconditional Model Cascade for both datasets tested, but only shows a performance gain in the conditional case for cifar100. This shows that masking the same model multiple times using self-supervision enables more information to be extracted from the training set.

4.6 Ablations

5 Discussion

Limitations.

Conclusion.

Table 8: Comparison of the conditional and unconditional Self-Masking Cascades (SMC) with our Self-Masking Network (SMN), Self-Training Network (STN), supervised fine-tuning (FFT) and supervised Masking (M). Size is given relative to the cost of storing the original, pretrained model in 32-bit floating point (=β), excluding the cost of storing the original model, and the PCA and GMM parameters, which should be negligible. The number of forward passes needed to compute the embeddings is given.

Method	cifar100	inat	size	forward passes
STN+LP	0.77	0.52	β	1
SMN+LP	0.77	0.52	$1/32\beta$	1
SMC+LP (cond.)	0.79	0.52	$6/32\beta$	2
SMC+LP (uncond.)	0.81	0.55	$16/32\beta$	6

Table 9: **Ablations.** We ablate the key components of our method: xx, yy, zz. We evaluate via kNN evaluation.

(a) varying number of prototypes

	cifar10	dtd	sun397
50			
500			
5000			

(b) Varying the initialiation

	cifar10	dtd	sun397
MoCo-v2			
SwAV			
Superv.			

(c) Keeping layers frozen.

	cifar10	dtd	sun397
none			
only BNs			
first half			

A Appendix

A.1 Proofs

A.1.1 Translation invariance of threshold and score initialization

Recall the formula given for determining the value of the mask M_i^t for the i'th weight of the network

$$M_i^t = [S_i^t > \mu] \tag{9}$$

where S_i^t, given the update equation (6), can be formulated as

$$S_i^t = S_i^0 - \sum_{\hat{t}=0}^{t-1} \gamma^{\hat{t}} g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}}) \tag{10}$$

where $g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}})$ is the gradient of the i'th score given the mask $\mathcal{M}^{\hat{t}}$ and 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

$$M_i^0 = [S_i^0 > k] \qquad = [S_i^0 + a > k + a] \tag{11}$$

$$M_i^t = [S_i^0 - \sum_{\hat{t}=0}^{t-1} \gamma^{\hat{t}} g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}}) > k] \qquad = [S_i^0 + a - \sum_{\hat{t}=0}^{t-1} \gamma^{\hat{t}} g_i(\mathcal{M}^{\hat{t}}, \mathcal{B}^{\hat{t}}) > k + a] \tag{12}$$

.

For the base case we then have that replacing S_i^0 with $S_i^0 + a$ and k with $k + a$ will not change M_i^0 for any i, which means the network mask $\mathcal{M}^0$ is also invariant to this change. Consequently, M_i^1 is also invariant to this change because of eq. 12 and because the gradient $g_i(\mathcal{M}^0, \mathcal{B}^0)$ does not change. The same reasoning can be applied recursively to M_i^2 and so on. Thus, by induction, translating the initial score and threshold by the same amount will not change any of the network masks during training (under simple sgd without weight decay).

A.1.2 Scale invariance of learning rate and score initialization

Equation for sgd with weight decay:

$$S_i^t = S_i^{t-1} - \gamma^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \lambda S_i^{t-1} \right) \tag{13}$$

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

$$\alpha S_i^{t-1} - \alpha\gamma^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \frac{\lambda}{\alpha} \alpha S_i^{t-1} \right) = \alpha \left(S_i^{t-1} - \gamma^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \lambda S_i^{t-1} \right) \right) \tag{14}$$

$$= \alpha S_i^t \tag{15}$$

.

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 $M_i^0 = [S_i^0 > 0]$ are invariant to the scale change $M_i^0 = [\alpha S_i^0 > 0]$, so the replacement of S_i^0 with αS_i^0, combined with the other replacements, does not change the gradient $g_i(\mathcal{M}^0, \mathcal{B}^0)$. Combined with eq 15, 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 ($= \alpha S_i^t$). 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.

Note that momentum is ommited from these proofs, but we verified experimentally that this invariance also holds if it is enabled.

References

[1] Hyojin Bahng et al. *Exploring Visual Prompts for Adapting Large-Scale Models*. June 3, 2022. DOI: 10.48550/arXiv.2203.17274. arXiv: 2203.17274 [cs]. URL: http://arxiv.org/abs/2203.17274 (visited on 05/06/2023). preprint.

[2] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. May 24, 2021. DOI: 10.48550/arXiv.2104.14294. arXiv: 2104.14294 [cs]. URL: http://arxiv.org/abs/2104.14294 (visited on 11/22/2022). preprint.

[3] Mathilde Caron et al. *Unsupervised Learning of Visual Features by Contrasting Cluster Assignments*. Jan. 8, 2021. arXiv: 2006.09882 [cs]. URL: http://arxiv.org/abs/2006.09882 (visited on 12/03/2022). preprint.

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. *BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations*. Apr. 18, 2016. DOI: 10.48550/arXiv.1511.00363. arXiv: 1511.00363 [cs]. URL: http://arxiv.org/abs/1511.00363 (visited on 11/22/2022). preprint.

[5] Suchin Gururangan et al. "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL 2020. Online: Association for Computational Linguistics, July 2020, pp. 8342–8360. DOI: 10.18653/v1/2020.acl-main.740. URL: https://aclanthology.org/2020.acl-main.740 (visited on 12/14/2022).

[6] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. Mar. 23, 2020. arXiv: 1911.05722 [cs]. URL: http://arxiv.org/abs/1911.05722 (visited on 11/22/2022). preprint.

[7] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2018. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 328–339. DOI: 10.18653/v1/P18-1031. URL: https://aclanthology.org/P18-1031 (visited on 12/14/2022).

[8] Itay Hubara et al. "Binarized Neural Networks". In: (), p. 9.

[9] François Lagunas et al. *Block Pruning For Faster Transformers*. Sept. 10, 2021. arXiv: 2109.04838 [cs]. URL: http://arxiv.org/abs/2109.04838 (visited on 11/22/2022). preprint.

[10] Jochem Loedeman et al. "Prompt Generation Networks for Efficient Adaptation of Frozen Vision Transformers". Oct. 12, 2022. arXiv: 2210.06466 [cs]. URL: http://arxiv.org/abs/2210.06466 (visited on 10/25/2022).

[11] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. *Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights*. Mar. 16, 2018. DOI: 10.48550/arXiv.1801.06519. arXiv: 1801.06519 [cs]. URL: http://arxiv.org/abs/1801.06519 (visited on 11/22/2022). preprint.

[12] Vivek Ramanujan et al. "What's Hidden in a Randomly Weighted Neural Network?" Mar. 30, 2020. arXiv: 1911.13299 [cs]. URL: http://arxiv.org/abs/1911.13299 (visited on 07/27/2022).

[13] Colorado J. Reed et al. "Self-Supervised Pretraining Improves Self-Supervised Pretraining". In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2022, pp. 2584–2594. URL: https://openaccess.thecvf.com/content/WACV2022/html/Reed_Self-Supervised_Pretraining_Improves_Self-Supervised_Pretraining_WACV_2022_paper.html (visited on 04/07/2023).

[14] Hadi Salman et al. *Do Adversarially Robust ImageNet Models Transfer Better?* Dec. 7, 2020. arXiv: 2007.08489 [cs, stat]. URL: http://arxiv.org/abs/2007.08489 (visited on 01/23/2023). preprint.

[15] Victor Sanh, Thomas Wolf, and Alexander M Rush. "Movement Pruning: Adaptive Sparsity by Fine-Tuning". In: (), p. 12.

[16] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. *Discovering Neural Wirings*. Nov. 16, 2019. arXiv: 1906.00586 [cs]. URL: http://arxiv.org/abs/1906.00586 (visited on 11/22/2022). preprint.

[17] Mitchell Wortsman et al. "Supermasks in Superposition". In: (), p. 12.