Appendix of Self-Masking Networks for Unsupervised Adaptation

Anonymous Author(s)
Affiliation
Address
email

A Proofs

A.1 Translation invariance of threshold and score initialization

As described in Section **??**, 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 (**??**), 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 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

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

.

For the base case we then have that replacing S^0_i with $S^0 + a$ and k with $k + a$ will not change M^0 for any i, which means the network mask $\mathcal{M}^0$ is also invariant to this change. Consequently, M^1 is also invariant to this change because of eq. 1 and because the gradient $g_i(\mathcal{M}^0, \mathcal{B}^0)$ does not change. The same reasoning can be applied recursively to M^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.2 Scale invariance of learning rate and score initialization

Equation for SGD with weight decay:

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

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

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:

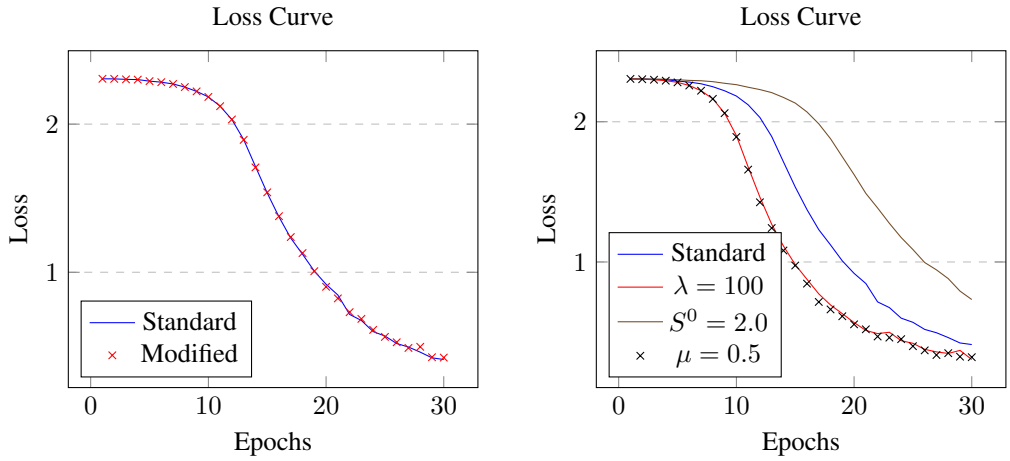
$$\begin{aligned} \alpha S^{t-1} - \alpha\lambda^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \frac{\gamma}{\alpha} \alpha S^{t-1} \right) &= \alpha \left(S^{t-1} - \lambda^t \left(g_i(\mathcal{M}^{t-1}, \mathcal{B}^{t-1}) + \gamma S^{t-1} \right) \right) \\ &= \alpha S^0 \end{aligned} \tag{2}$$

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 2, 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.

Note that, in the interest of keeping the proofs simple, momentum is ommited from these proofs. However, we ran the experiment in Appendix A.3 with momentum enabled, indicating that this invariance also holds if momentum is included. 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.

A.3 Experimental verification



(a) Equivalent configurations

(b) One at a time

Figure 1: Comparison of the loss for standard training ($\lambda = 50$, $S^0 = 1.0$, $\mu = 0.0$) with equivalent but distinct hyperparemeters ($\lambda = 100$, $S^0 = 2.5$, $\mu = 0.5$). Shown is the progression of the loss as well as the average value of the scores during training. The plot on the right shows how the curves would differ when applying standard training, except changing one of the hyperparameterers at a time. These experiments were run on CIFAR-10, with the supervised masking algorithm.

Figure 1 shows an experimental verification of the conclusions of these proofs. It can be seen that both experiments have a very similar loss curve, which is an indication that the models are behaving

the same. Note that we controlled the most important random components such as the data sampling, however some randomness from CUDA operations could still be present, in addition to differences in behaviour of floating point arithmetic under the different scales. For this reason we cannot expect in practice that both hyperparameter settings produce exactly the same masks.

Notice that it can be seen in Figure 1b that only doubling the learning rate gives the same loss curve as only increasing the threshold to 0.5. This is in line with proofs A.1 and A.2, as increasing the threshold by 0.5 is equivalent to reducing the score initialization by 0.5, effectively halving it, which is in turn equivalent to doubling the learning rate.

B Sparsities

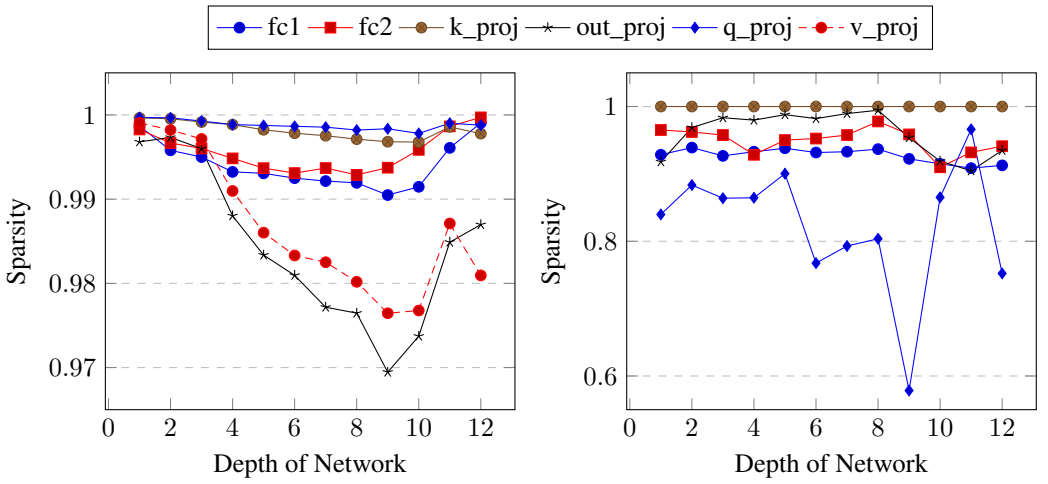
B.1 Supervised

Table 1 shows the sparsities found by the supervised masking experiments shown in Table **??**. Notably, the CLIP vision transformer deactivates very few weights, especially for some datasets. An explanation could be the fact that this model was trained with a pre-aligned classification head, thanks to CLIP's textual embeddings, thus requiring less invasive alterations. However, further analysis is necessary to determine whether another factor is responsible, such as the different architecture.

The vision transformer is further analyzed in Figure 2. It would appear that the weights in the middle layers generally get masked the most. The biases however do not show a similar pattern. Interestingly though, the biases of the key projections (**key_proj**) are left completely unchanged by the algorithm. A similar analysis for the ResNet-50 model can be seen in appendix B.2, at Figure 3.

Table 1: Percentage of weights that are active after masking different architectues in the supervised setting.

Model+Method	cifar10	cifar100	dtd	eurosat	flowers	inat	oxfordpets	sun397	ucf101
rn18-timm+M	84.8%	64.3%	92.6%	93.4%	93.1%	51.2%	95.6%	68.0%	89.4%
rn50-swav+M	89.5%	77.2%	95.1%	95.7%	94.5%	65.4%	96.2%	81.1%	91.7%
vitb32-clip+M	99.6%	98.7%	99.9%	99.9%	99.9%	94.8%	99.9%	99.3%	99.8%



(a) Weights

(b) Biases

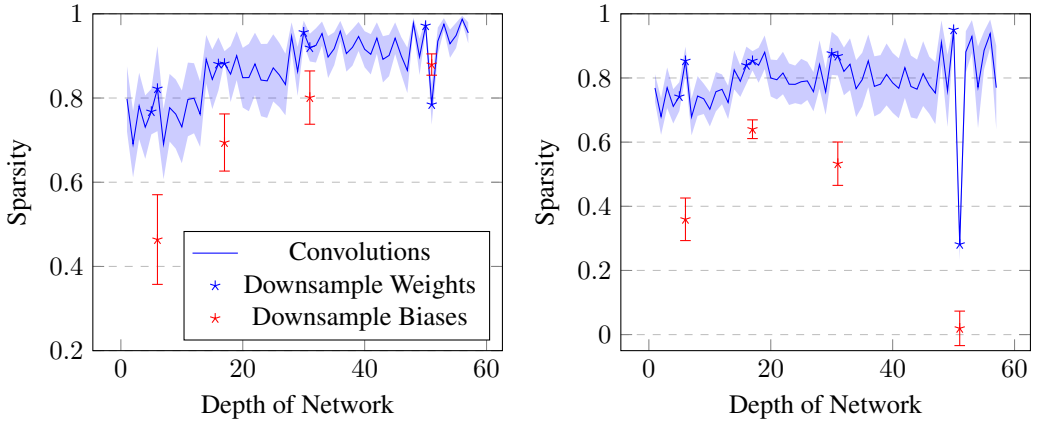
Figure 2: Sparsity levels found across layers by our masking algorithm, when applied to the ViT-B/32_{CLIP} model. Displayed is the average across the datasets CIFAR-10, CIFAR-100, SUN397 and DTD.

Table 2: **Updating Ramanujan *et al.* 's method [1], such that it decreases the number of active weights as training progresses.** The progression is a linear progression with respect to the percentage of active weights, starting at 100% and going down to the 'found' sparsity. All numbers are reported by masking with a self-supervised loss, starting from a ResNet-50 pretrained with SwAV.

Masking mechanism	Sparsity	DTD	EUROSAT	FLOWERS	UCF101
k-NN evaluation					
Ramanujan *et al.* [1]	progressive	0.658	0.957	0.895	0.549
Ramanujan *et al.* [1]	found	0.671	**0.973**	0.903	**0.572**
Threshold (Ours)	found	**0.674**	0.971	**0.920**	0.549
Linear probe evaluation					
Ramanujan *et al.* [1]	progressive	0.706	0.981	0.983	0.681
Ramanujan *et al.* [1]	found	**0.733**	**0.984**	**0.985**	0.690
Threshold (Ours)	found	0.714	0.983	0.983	**0.697**
Sparsity level found by our method:		98.8%	96.7%	98.6%	95.7%

B.2 Comparisons

Figure 3 shows shows the sparsities found at different layers of the network by our self-supervised masking method in a comparison between self-supervised and supervised masking. The corresponding overall sparsities can be seen in Table 2 of the main paper. Notably, earlier ResNet blocks appear to get masked more, but within a block, the pattern reverses, with later layers seeming to get masked more. Also, the last downsampling layer is masked heavily. This appears to be generally true for both the supervised and self-supervised settings. In addition, these effects seem to remain across datasets, as can be seen from the relatively small standard deviations.



(a) Self-Supervised (b) Supervised

Figure 3: Sparsity levels found across layers by our Self-Masking Networks on a SwAV-pretrained ResNet-50, compared to our supervised masking algorithm. Average across the datasets CIFAR-100, CIFAR-10, SUN397, and DTD. Standard deviations included.

B.3 Progressive Sparsity

Table 2 shows an additional experiment, where we progressively deactivate more weights during training, using a fixed schedule, allowing Ramanujan *et al.* 's method to start with every weight activated. Unfortunately, this only seems to degrade its performance.

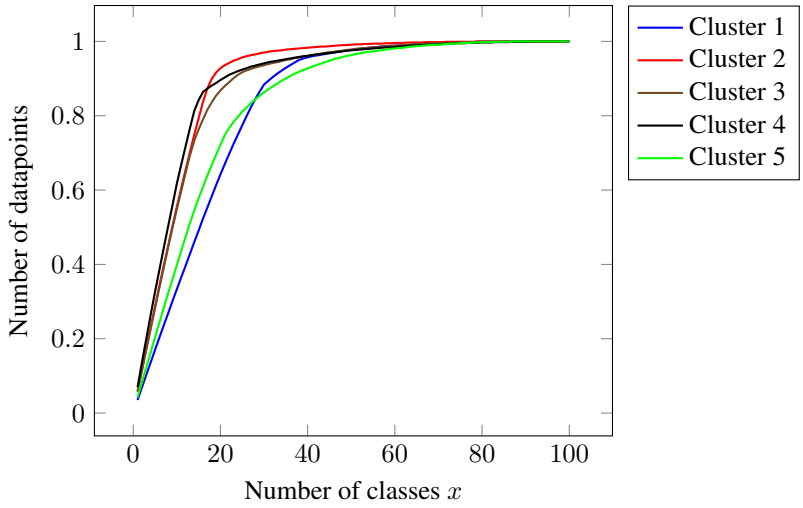


Figure 4: Proportion of datapoints made up of the top x most common classes for each of the five cascade clusters (training set).

Table 3: **Variance in k-NN accuracies with the Threshold Masking Mechanism.** This table shows the results of five separate runs of the same experiment on the "dtd" and "ucf101" datasets, including the mean across experiments and the standard deviation of the results.

Dataset	Run	R1	R2	R3	R4	R5	Mean	STD
k-NN evaluation								
DTD	Threshold (Ours)	0.685	0.678	0.670	0.678	0.671	0.676	0.005
UCF101	Threshold (Ours)	0.569	0.576	0.557	0.574	0.568	0.569	0.007

C Clusters

Finally, we show in Figure 4, for each cluster of the Cascade model, the cumulative distribution of datapoints belonging to the most common classes in each cluster. This plot shows that the dispatcher is effective at creating dataset splits that have more homogenous distributions. For example, it can be read from the graph that, for each cluster, the top 20 classes represent at least 60% of the datapoints in that cluster, and three of these clusters are even more homogenous, with the top 20 classes representing at least 80% of the datapoints.

D Error Bars

Due to computational and time constraints, we were only hable to run every experiment in the main paper once. However, to give an idea of the level of noise in our results, we show the variance of our self-supervised learning experiment from Table 2 from the main paper of our thresholding method in Table 3 by running it 5x on two different datasets.

E Broader Impact

Our technique could help with reducing storage and network tansfer costs. However, it is not more computationally efficient and thus the (more important) energy cost of running these masked models remains equal, or increases if running the Cascade model. Another interesting aspect of mask-learning in general through the passthrough trick is that it enables the learning of *discrete* components like text rather than only being able to learn images. Hence, the passthrough trick could be used to perform text reconstruction attacks. Existing approaches to privacy-preserving model training thus should also protect against this.

F Source Code

Source code will be published upon publication of this paper.

References

[1] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.