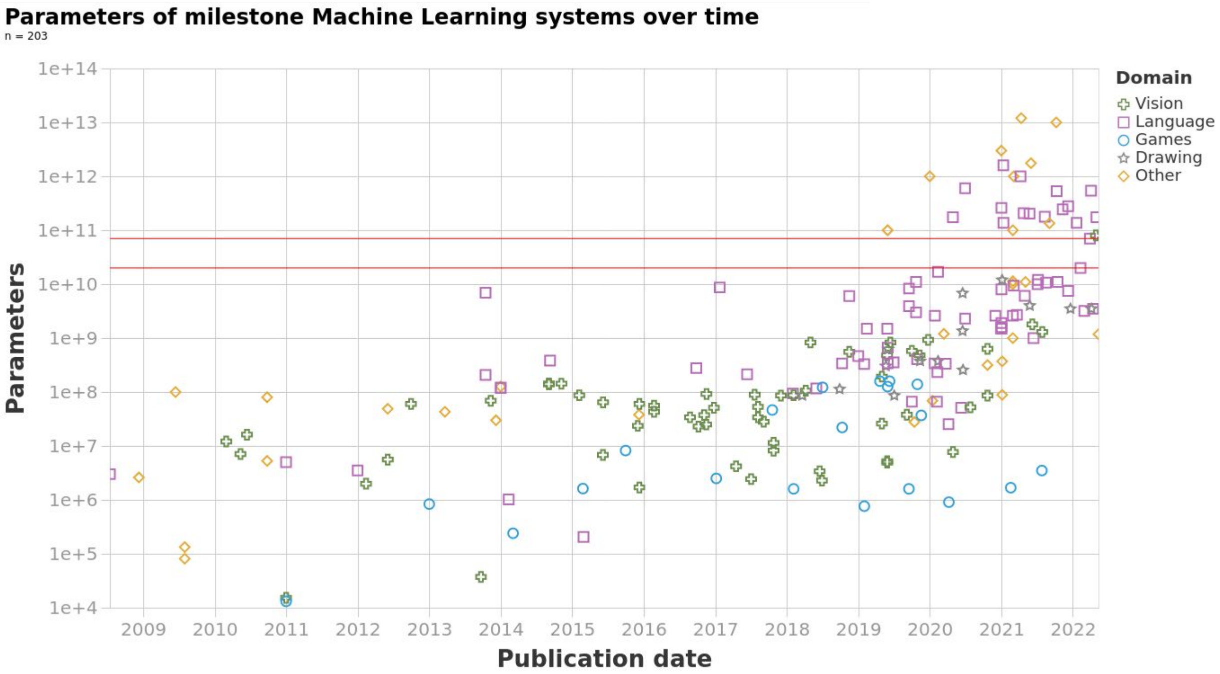
Self-Masking Networks for Unsupervised Adaptation

Alfonso Taboada Warmerdam
Mathilde Caron
Yuki Asano

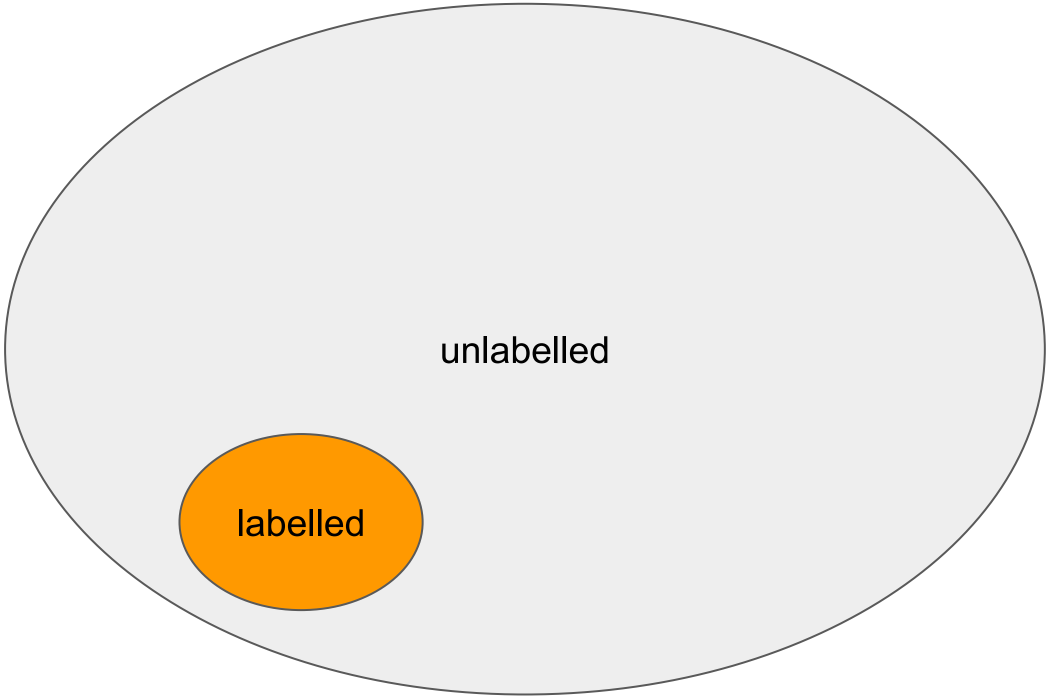
Introduction

Increasing model sizes



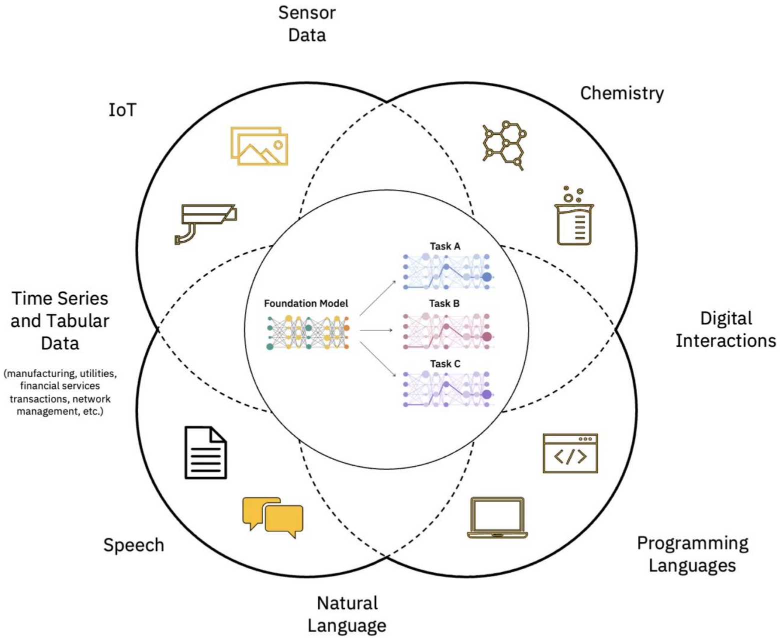
Introduction

Scarcity of labelled data



Introduction

Efficacy of fine-tuning



Introduction

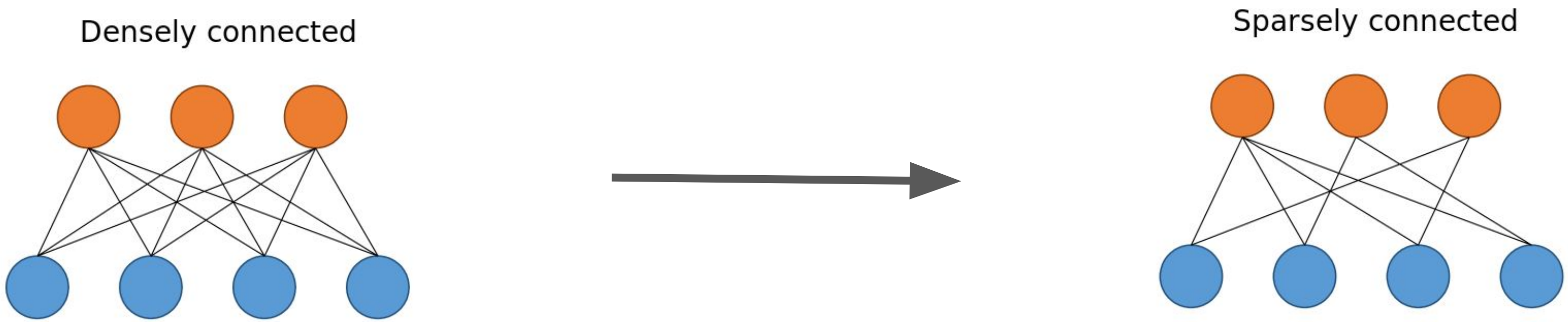
Label- and parameter-efficient training

Self-Masking Networks for Unsupervised Adaptation

- 79x more efficient to store fine-tuned parameters
- significantly improve performance on label-efficient downstream tasks

Method

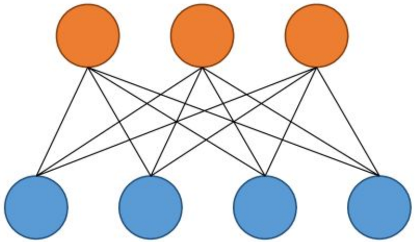
Masking



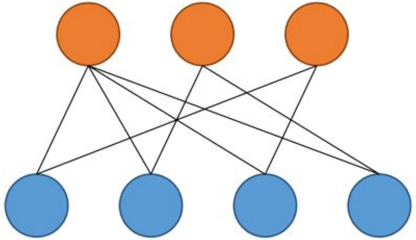
Method

Masking

Densely connected



Sparsely connected

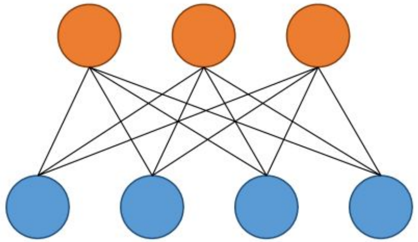


$$\begin{bmatrix} \theta_{00} & \theta_{01} & \theta_{02} \\ \theta_{10} & \theta_{11} & \theta_{12} \\ \theta_{20} & \theta_{21} & \theta_{22} \end{bmatrix} \times \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}}_{\text{MASK}} = \begin{bmatrix} \theta_{00} & 0 & 0 \\ 0 & \theta_{11} & 0 \\ \theta_{20} & \theta_{21} & 0 \end{bmatrix}$$

Method

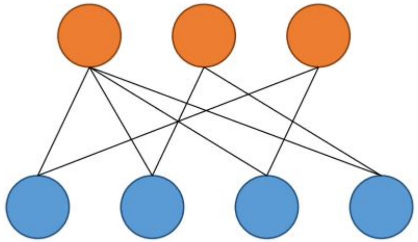
Masking

Densely connected



Very cheap to store 🥰

Sparsely connected

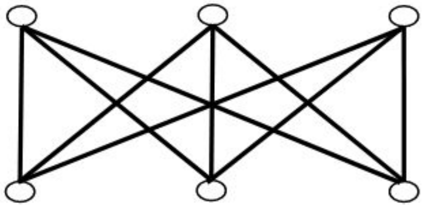


$$\begin{bmatrix} \theta_{00} & \theta_{01} & \theta_{02} \\ \theta_{10} & \theta_{11} & \theta_{12} \\ \theta_{20} & \theta_{21} & \theta_{22} \end{bmatrix} \times \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}}_{\text{MASK}} = \begin{bmatrix} \theta_{00} & 0 & 0 \\ 0 & \theta_{11} & 0 \\ \theta_{20} & \theta_{21} & 0 \end{bmatrix}$$

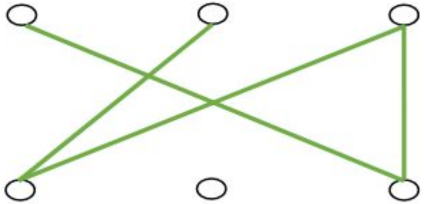
Method

Masking, how?

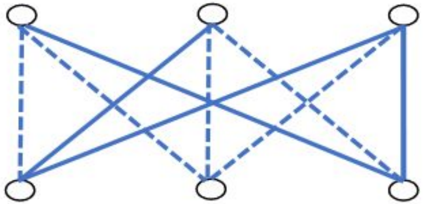
Assign a score
for each weight (edge)



Forward: Remove
edges with low score



Backward: Update all
scores



Method

Masking, how?

Standard training (simplified SGD):

$$\theta^{t+1} = \theta^t - \lambda \frac{d\mathcal{L}^t}{d\theta^t}$$

Submasking:

$$\theta^t = \overline{\theta} \cdot M^t$$

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

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

The trick to implement this:

$$\frac{d\mathcal{L}}{dS} = \frac{d\mathcal{L}}{dM}$$

Method

Masking, how?

Standard training (simplified SGD):

$$\theta^{t+1} = \theta^t - \lambda \frac{d\mathcal{L}^t}{d\theta^t}$$

Submasking:

$$\theta^t = \overline{\theta} \cdot M^t$$

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

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

The trick to implement this:

$$\frac{d\mathcal{L}}{dS} = \frac{d\mathcal{L}}{dM}$$

[1] Piggyback: Adapting a single network to multiple
tasks by learning to mask weights.

Method

Masking, how?

Standard training (simplified SGD):

$$\theta^{t+1} = \theta^t - \lambda \frac{d\mathcal{L}^t}{d\theta^t}$$

Extra parameters:
μ (threshold)
S^0 (initial score)

Submasking:

$$\theta^t = \overline{\theta} \cdot M^t$$

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

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

The trick to implement this:

$$\frac{d\mathcal{L}}{dS} = \frac{d\mathcal{L}}{dM}$$

[1] Piggyback: Adapting a single network to multiple
tasks by learning to mask weights.

Method

Hyperparameter-free masking

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.*

$$\begin{tikzcd}
S^0 \arrow[r, "SGD"] \arrow[d, "> \mu"] & S^1 \arrow[r, "SGD"] \arrow[d, "> \mu"] & S^2 \arrow[r, "SGD"] \arrow[d, "> \mu"] & S^3 \arrow[d, "> \mu"] \ \dots \arrow[r, "SGD"] & S^n \arrow[d, "> \mu"] \\
M^0 & M^1 & M^2 & M^3 & M^n
\end{tikzcd}$$

Method

Hyperparameter-free masking

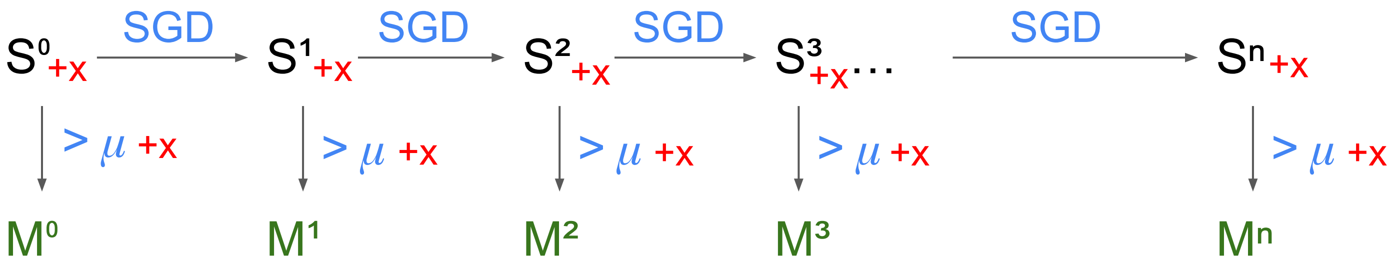
Theorem 1. *Translation invariance of threshold and initialization. Shifting the score initialisation S^0 and the threshold μ by an equal amount does not affect SGD-based training without weight-decay.*

$$
\begin{tikzcd}
S^0_{+x} \arrow[r, "\text{SGD}"] \arrow[d, "> \mu\ +x"] & S^1_{+x} \arrow[r, "\text{SGD}"] \arrow[d, "> \mu\ +x"] & S^2_{+x} \arrow[r, "\text{SGD}"] \arrow[d, "> \mu\ +x"] & S^3_{+x} \cdots \arrow[r, "\text{SGD}"] \arrow[d, "> \mu\ +x"] & S^n_{+x} \arrow[d, "> \mu\ +x"] \\
M^0 & M^1 & M^2 & M^3 & M^n
\end{tikzcd}
$$

Method

Hyperparameter-free masking

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.*



Method

Hyperparameter-free masking

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.*

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 $\frac{1}{\alpha}$.*

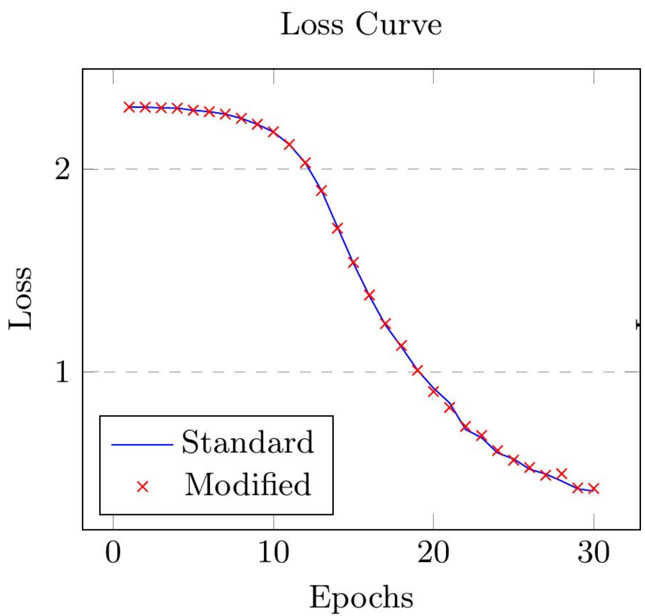
Method

Hyperparameter-free masking

$\lambda = 50, S^0 = 1.0, \mu = 0.0$
standard

$\cong$

$\lambda = 100, S^0 = 2.5, \mu = 0.5.$
modified



(a) Equivalent configurations

Method

Hyperparameter-free masking

$\lambda = 50$, $S^0 = 1.0$, $\mu = 0.0$
standard

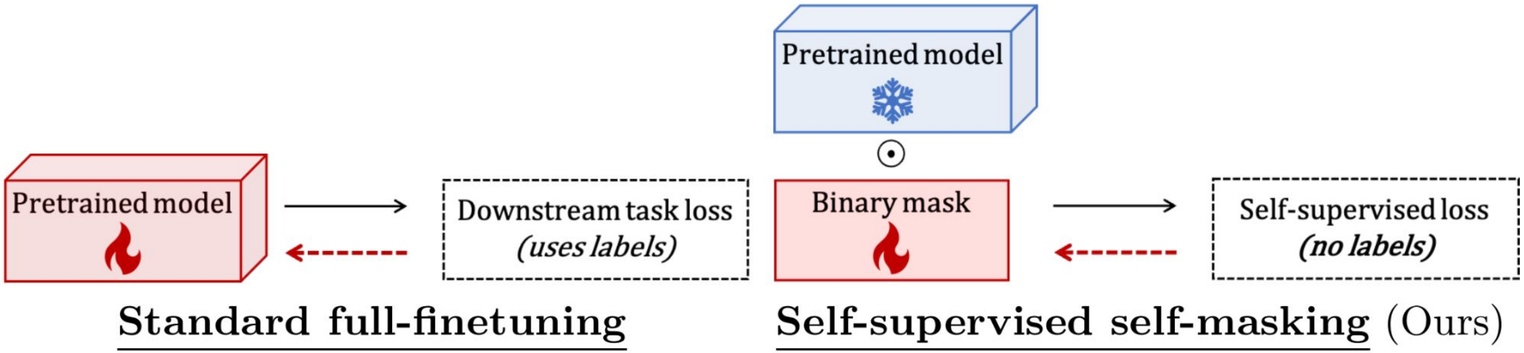
$\cong$

$\lambda = 100$, $S^0 = 2.5$, $\mu = 0.5$.
modified

Neat!

Method

Self-supervised masking



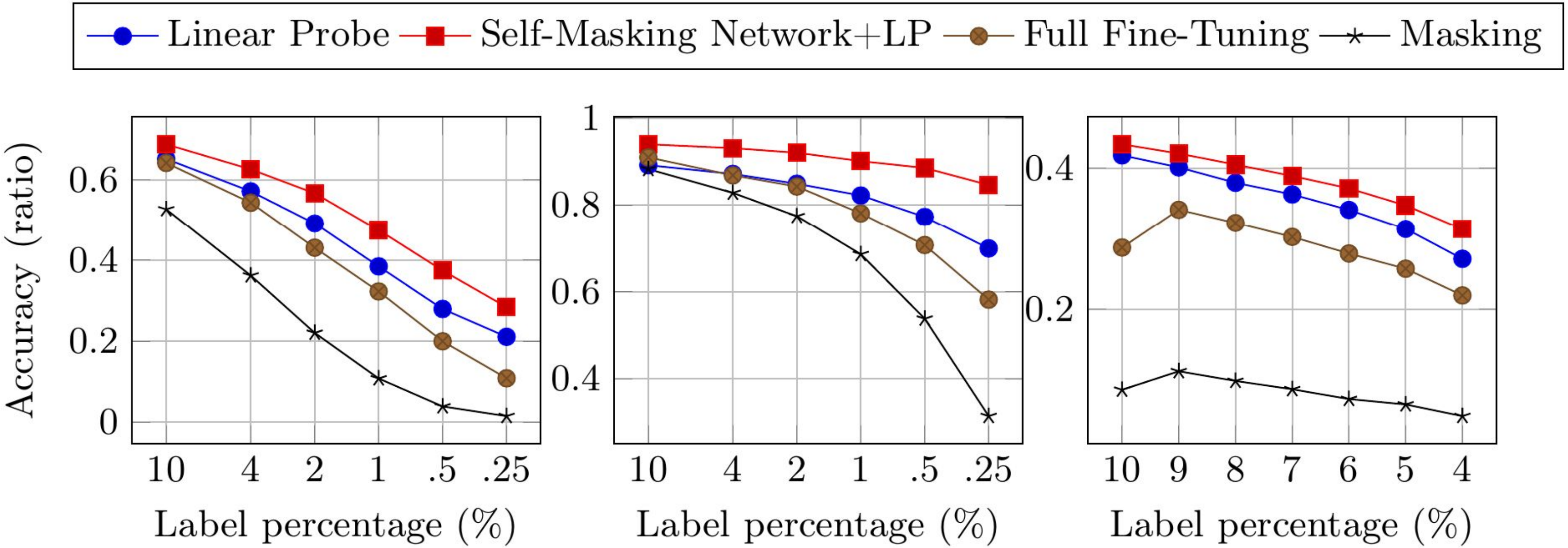
Results

Supervised Fine-tuning

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**

Results

Self-supervised label-efficient masking



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

Conclusion

Self-masking networks

- Very efficient to store parameters
- Comparable performance to FFT in label-rich supervised settings
- Bad performance if only trained on small labelled dataset
- Great performance if fine tuned using self-supervision first
- Simple method with few additional parameters

Try it

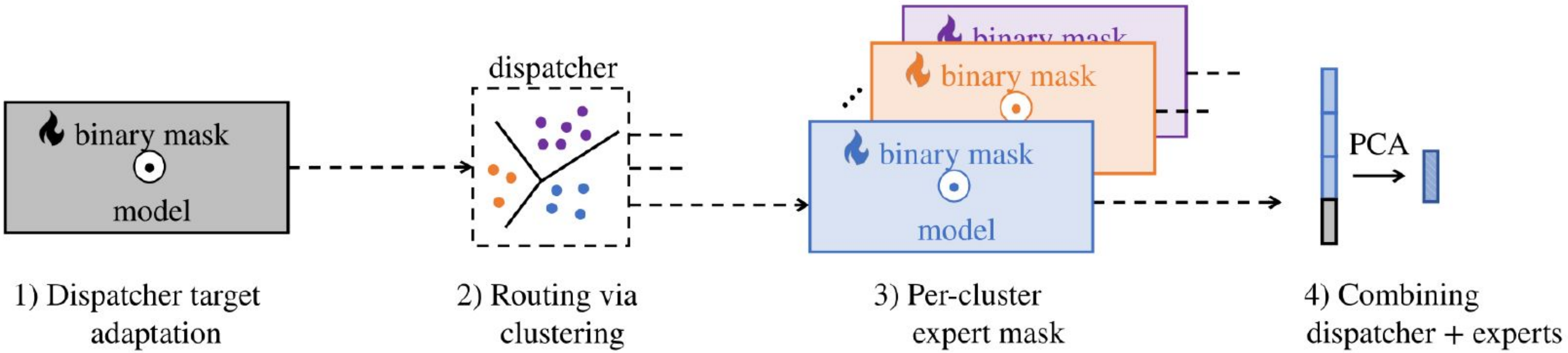
Self-masking networks

https://github.com/alvitawa/UnsupervisedMasking

END

Bonus

Self-masking cascade



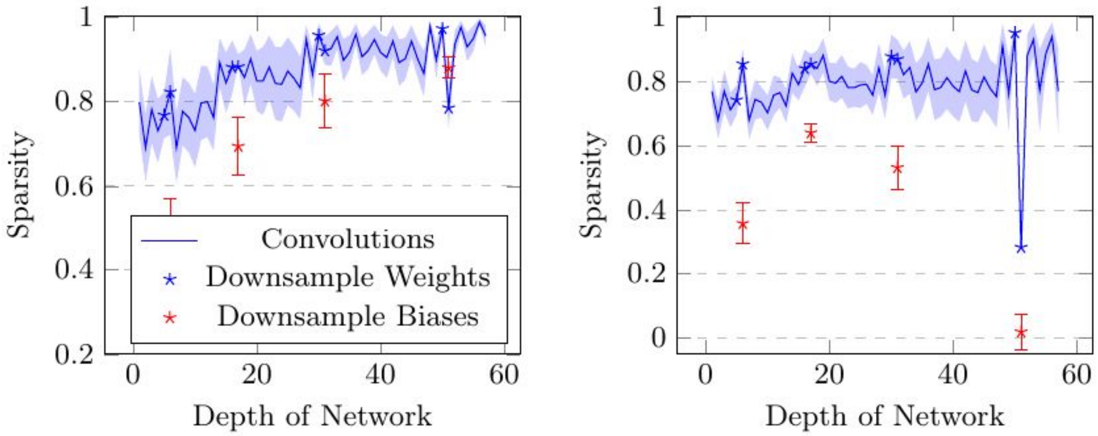
Bonus

Self-masking cascade

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**

Bonus

Sparsity



(a) Self-Supervised (b) Supervised

Fig. 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.

Bonus

Sparsity

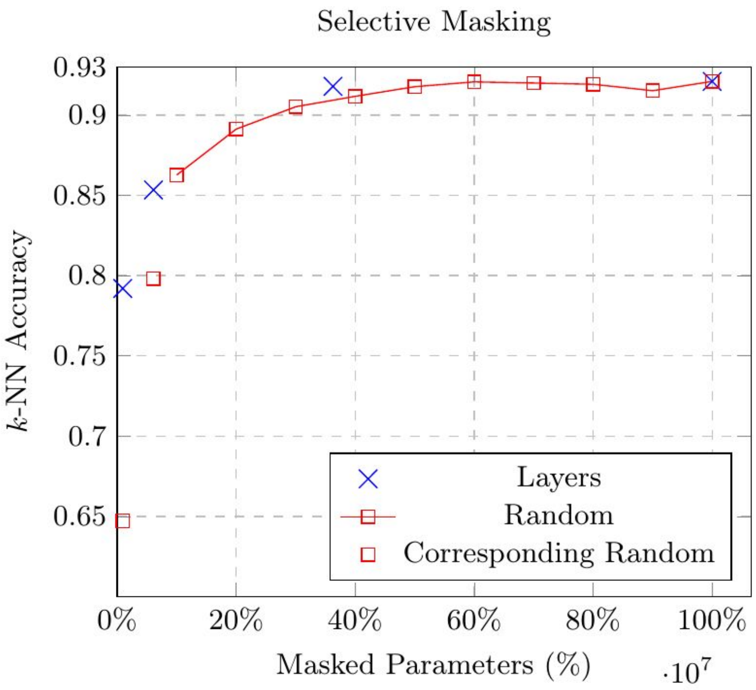
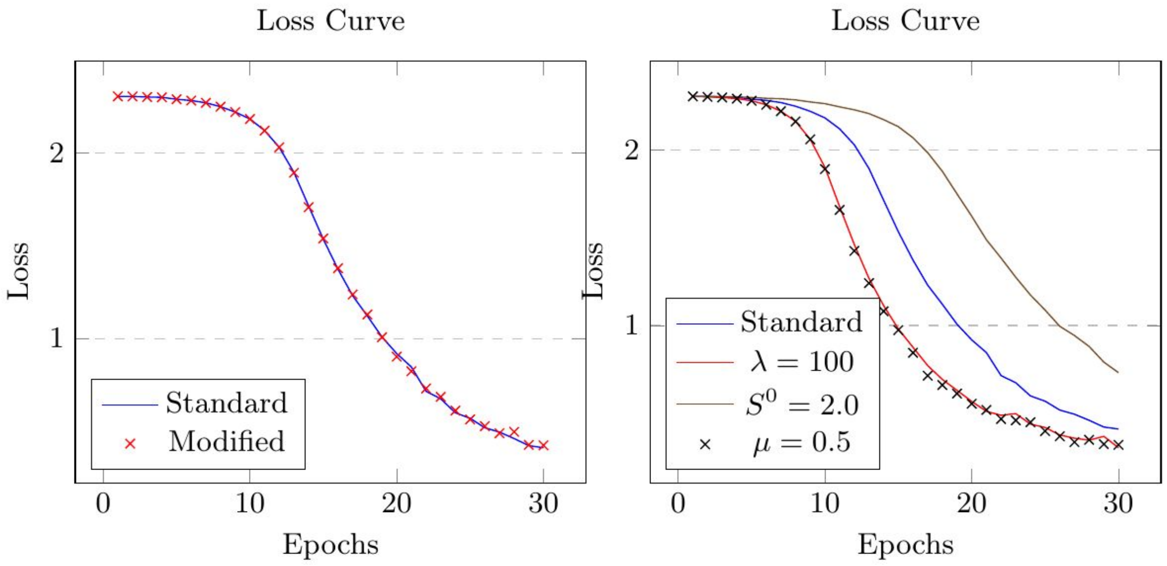


Fig. 4: Accuracy when only masking the first N ResNet-50 layers (L1, L2, L3, L4) with corresponding experiments where random weights are masked instead, and accuracy when training 10%, 20%, ... up to 90% of masks randomly across all weights in each weight matrix. The resulting total number of trainable masks is given on the x-axis. These experiments where run with a ResNet-50 from SWaV.

Thank you for listening!

Alfonso Taboada Warmerdam
avtwarmerdam@gmail.com

Bonus slides (Q&A)



(a) Equivalent configurations

(b) One at a time

Fig. 1: Left: Comparison of the loss for standard training ($\lambda = 50$, $S^0 = 1.0$, $\mu = 0.0$) with equivalent but distinct hyperparameters ($\lambda = 100$, $S^0 = 2.5$, $\mu = 0.5$). Shown is the progression of the loss during training. Right: How the curves would differ when applying standard training, except changing only one of the hyperparameters at a time (doubling the learning rate, doubling the score initialization or shifting the threshold with 0.5). These experiments were run on CIFAR-10, with the supervised masking algorithm.

Ablations

Table 1: **Ablations.** We ablate the key components of our Self-Masking Network: the number of prototypes, network initialization, and the layers that are masked. We evaluate via k-NN evaluation. The row marked with an asterisk (*) indicates the configuration used in the rest of the paper.

(a) Varying prototypes

	CIFAR10	DTD	SUN397
50	0.510	0.644	0.503
500*	**0.921**	**0.674**	**0.518**
5000	0.920	0.640	0.496

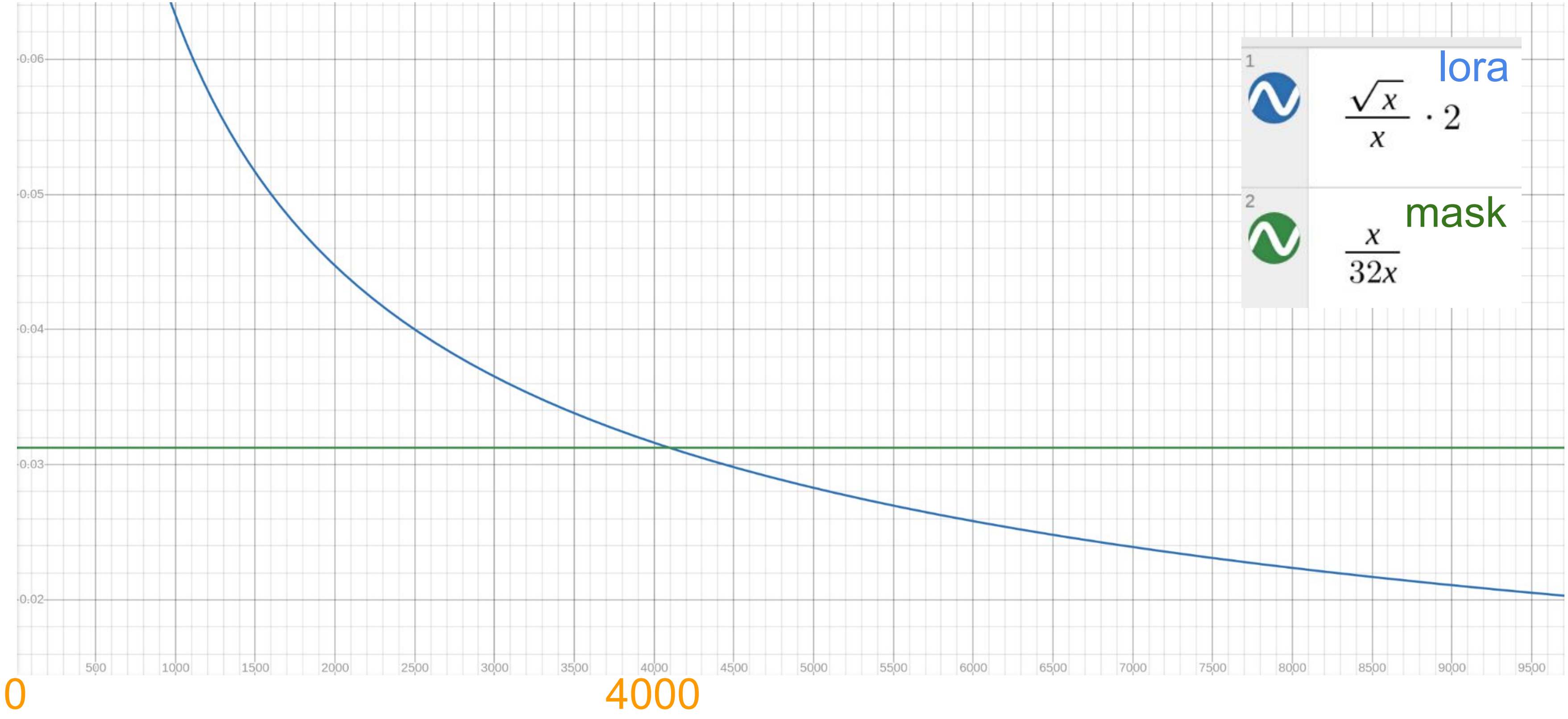
(b) Keeping layers frozen.

	CIFAR10	DTD	SUN397
none	0.560	0.434	0.177
BNs*	**0.921**	**0.674**	**0.518**
biases	0.681	0.505	0.266

(c) Varying the initialiation

	CIFAR10	DTD	SUN397
DINO	0.915	**0.695**	**0.529**
SwAV*	**0.921**	0.674	0.518
TIMM	0.900	0.623	0.505

LoRA


0.06
0.05
0.04
0.03
0.02
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
6000
6500
7000
7500
8000
8500
9000
9500
1
lora
$\frac{\sqrt{x}}{x} \cdot 2$
2
mask
$\frac{x}{32x}$
0
4000

79x

32-bit

1/((1-0.8332)/(32*(1-0.077))) = 177,0743405275779

16-bit estimate

1/((1-0.8332)/(16*(1-0.077))) = 88,53717026378897

Mask compression

Table 2: Compressing learned masks (using the Self-Masking method) with different off-the-shelf compression methods vs compressing the weights after Full Fine-Tuning (cifar100 dataset).

(a) Masked

Method	Masks	Reduction (%)
gzip	23462592	78.32
bz2	23462592	79.24
lzma	23462592	80.73
lz4	23462592	59.06
snappy	23462592	62.57

(b) Trained

Method	f32's	Reduction (%)
gzip	23462592	6.99
bz2	23462592	4.69
lzma	23462592	7.70
lz4	23462592	-0.39
snappy	23462592	-0.0046

Mask compression

Table 3: Same as table 2, but with SUN397 dataset.

(a) Masked

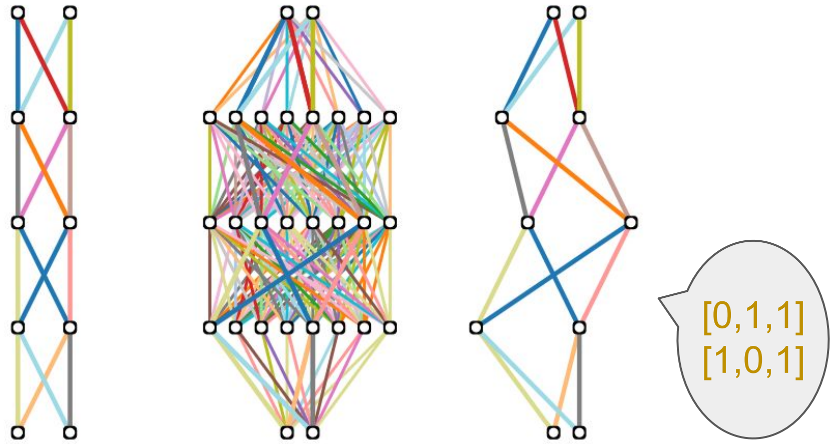
Method	Masks	Reduction (%)
gzip	23462592	81.25
bz2	23462592	82.11
lzma	23462592	83.32
lz4	23462592	62.52
snappy	23462592	66.54

(b) Trained

Method	f32's	Reduction (%)
gzip	23462592	6.99
bz2	23462592	4.69
lzma	23462592	7.69
lz4	23462592	-0.39
snappy	23462592	-0.0046

Randomly weighted neural networks - Ramanujan et Al[1]

- Tested on CNNs and ResNets
- Same performance as normal neural networks if the architecture is widened
- Example result of Ramanujan et al:
  - 6 Layer CNN -> ~89.5% accuracy
  - Subnetwork of 6 layer CNN with 2x the width -> ~89.3% accuracy
  - The subnetwork having the same number of active weights as in the dense CNN
  - (on cifar10)



A neural network τ which achieves good performance

Randomly initialized neural network N

A subnetwork τ' of N

[1] Whats hidden in a randomly weighted neural network?

Pytorch

```
class GetSubnet(autograd.Function):
    @staticmethod
    def forward(ctx, scores):
        out = scores.clone()

        out[out <= 0] = 0
        out[out > 0] = 1

        return out

    @staticmethod
    def backward(ctx, g):
        # send the gradient g straight-through on the backward pass.
        return g
```

Pytorch

```
class MaskedLinear(nn.Module):
        def __init__(self, in_ft, out_ft, score_init):
            super().__init__()
            self.W = nn.Linear(in_ft, out_ft, bias=False)
            self.W.weight.requires_grad = False
            self.S = nn.Parameter(torch.ones(out_ft, in_ft)*score_init)

        def forward(self, x):
            M = GetSubnet.apply(self.S)
            out = x @ (self.W.weight * M).T / torch.sqrt(M.mean())
            return out
```

spadework.ai