A Verification of Hyperparameter-free masking

In practice, it is not feasible to control the conditions perfectly, and randomness from CUDA operations or mini-batch sampling, as well as differences in behavior of floating point arithmetic under different scales will result in slightly different results. We show in this section that performance of the models remain generally unaffected, indicating that these invariances also hold in practice.

Figure 1a shows an experimental verification of the proofs in Section **??**. 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 behavior 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 the theorems posited in the main paper, 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 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

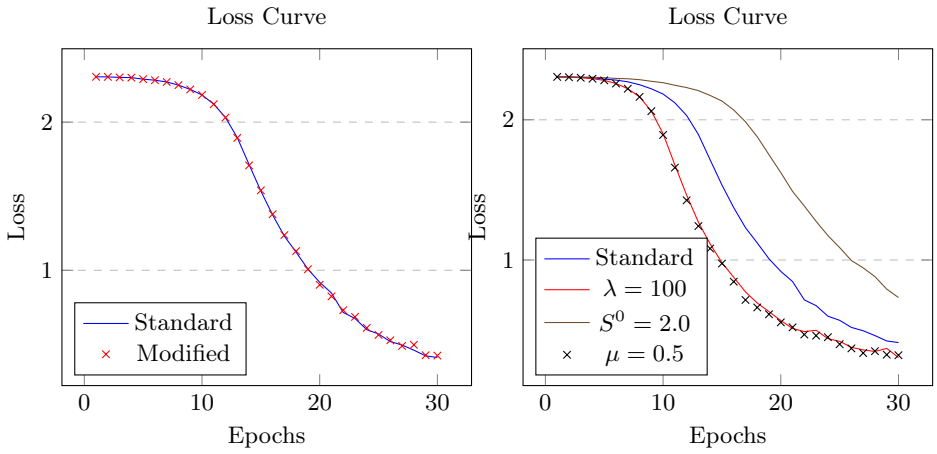
	CIFAR10	DTD	SUN397
50	0.510	0.644	0.503
500*	**0.921**	**0.674**	**0.518**
5000	0.920	0.640	0.496

(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



(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.

In this section, we ablate several components of our model.

Number of prototypes. We find that as long as our model is provided with enough capacity of 500 or more prototypes, the exact number does not matter and we achieve good performances, echoing previous self-supervised clustering findings [1,3]

Excluding weights from masking. Finally, in table 1b we evaluate what happens when the masking strategy is applied to all weights ('none' are frozen), all excluding the batch-norms (our setup) or all except the bias terms. We find that, as in the original masking formulations [13,10], it is essential to not apply masking to the batch-norm statistics, this could be due to the drastic effect zeroing out the batch norm weight can have, which disables a node completely.

Pretrained weights. In table 1c, we evaluate whether our self-masking can be run on differently pretrained backbones. Despite our loss utilising a self-supervised clustering formulation, we find it performs just as well on a teacher-distillation pretrained method, DINO [4] outperforming the supervisedly pretrained TIMM weights.

C Mask compression

It turns out that simply compressing the masks with off-the-shelf approaches can significantly reduce the storage costs. In particular, masks are more easily compressible than neural network weights, which further reduces the storage costs of masks when compared to a full set of fine-tuned weights. We found that, after compressing both, the masks take up 1.305% of the storage costs when compared to the full-fine-tuned network weights using the models trained on cifar100 (SMN vs self-sup fine-tuning) and 1.2695% using the models trained on SUN397. If neither are compressed at all, the masks take up 3.125% the storage cost of the full fine-tuned weights (using f32). See the full experiment results below in Tables 2 and 3. Overall, compressing the masks reduces their storage cost by up to 83%.

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

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

D Found Sparsities

In this section we document the sparsities found by our method in detail.

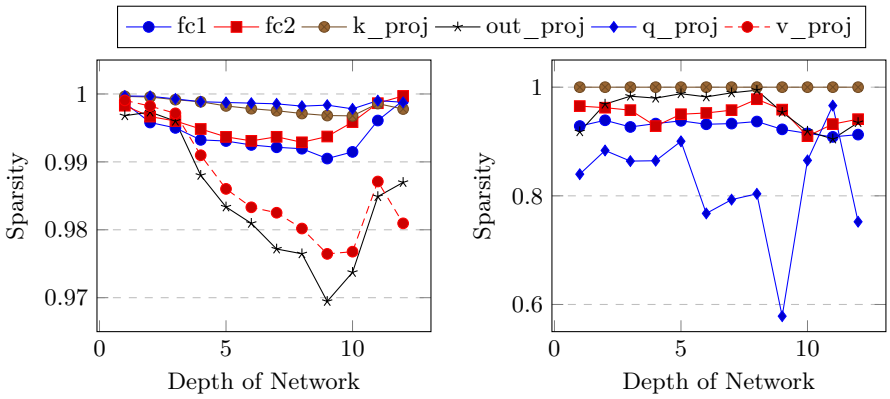
D.1 Supervised sparsities

Table 4 shows the sparsities found by the supervised masking experiments shown in Table 1 . 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 Figure 3.

Table 4: Percentage of weights that are active after masking different architectues in the supervised setting.

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



(a) Weights (b) Biases

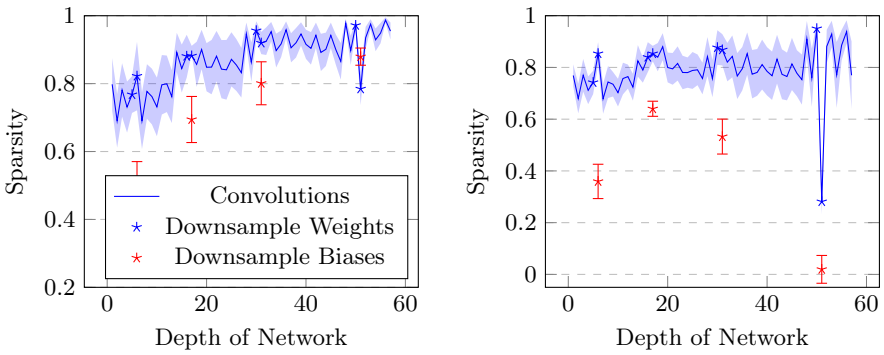
Fig. 2: Sparsity levels found across layers by our masking algorithm, when applied to the ViT-B/32_{CLIP} model. Displayed is the average across the datasets CIFAR-10, CIFAR-100, SUN397 and DTD.

D.2 Comparison

Figure 3 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 . 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.

E Progressive Sparsity

Table 5 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.



(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.

Table 5: **Updating Ramanujan *et al.*'s method [13], 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.* [13]	progressive	0.658	0.957	0.895	0.549
Ramanujan *et al.* [13]	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.* [13]	progressive	0.706	0.981	0.983	0.681
Ramanujan *et al.* [13]	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%

F Selective masking

This section evaluates whether the storage cost can be further reduced by selecting a smaller subset of the weights to be masked, and leaving the others unchanged. Following the results from Appendix D.2, where earlier layers where masked more than subsequent layers, we see what performance we get when only masking the first layer (L1), the first and second (L1 + L2) and the first, second and third (L1 + L2 + L3) layers. The results are compared to the baseline of masking all layers (L1+L2+L3+L4). In addition, we perform separate experiments where we randomly freeze each individual mask, corresponding to a single weight, with probability p, such that we run 9 additional experiments where 10%, 20%, ... up to 90% of masks are trained. The results are shown in Figure 4. In addition, two experiments are shown where only 0.9% and 6.1% of weights are masked, corresponding to the number of parameters in only the first layer or only the first and second layers.

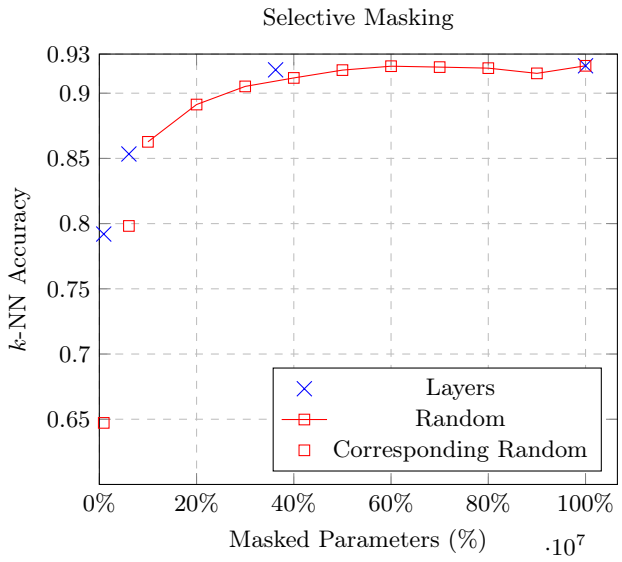


Fig. 4: Accuracy when only masking the first N ResNet-50 layers (L1, L2, L3, L4) with corresponding experiments where random weights are masked instead, and accuracy when training 10%, 20%, ... up to 90% of masks randomly across all weights in each weight matrix. The resulting total number of trainable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

It can be seen that, for this dataset, the model is able to maintain its accuracy with little loss in accuracy when training a mask on only half of the parameters. The graph also shows that, on this experimental setting, masking the first N layers yields higher accuracies than randomly masking the same number of

weights across the whole network. It is thus possible to further reduce storage costs of the masks with little loss of accuracy by at least 50% using this technique. Corresponding to an up to 64x reduction of storage costs of the fine-tuned model when compared to full fine-tuning.

Additionally, we run the same experiment on Cifar-100, but here we compare the random selection of trainable masks across layers with a similar method where we select the top-k% masks corresponding to weights with the highest magnitude. The results are displayed in Figure 5, where it can be seen that the maximum magnitude approach provides some benefit when only a tiny proportion of weights are trained, however, by at this level, the accuracy is already far below the baseline when masking all weights. Surprisingly though, a small gain in accuracy appears to be made when only masking around half the weights as opposed to all.

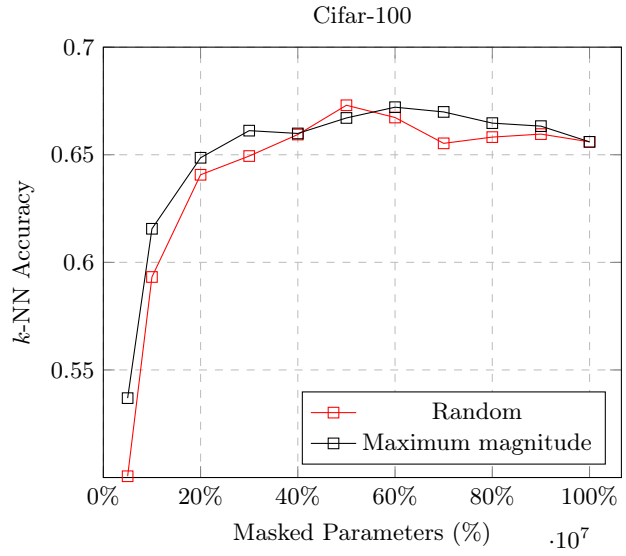


Fig. 5: Accuracy when training 5%, 10%, 20%, ... up to 90% of masks randomly across all weights in each weight matrix, compared to training 5%, 10%, 20%, ... up to 90% of the masks corresponding to the weights with the highest magnitudes.

G Dispatcher Clusters

In Figure 6 we compare the final embeddings of the original pretrained model, the domain adapted and the cascade model on a subset of CIFAR100. We find that our self-mask adapted embeddings create a few more distinct clusters, but a massive difference is not visible when compared to the cascade embeddings.

(a) **Original embeddings.** (b) **Adapted embeddings.** (c) **Cascade embeddings.**

Fig. 6: We compare the embeddings from (a) the original pretrained model, (b) the adapted model and (c) the cascade model on first 20 classes of CIFAR100. Ground-truth classes coded by color.

Both however are a clear improvement over the unadapted embeddings from the pre-trained model.

We also show in Figure 7, 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.

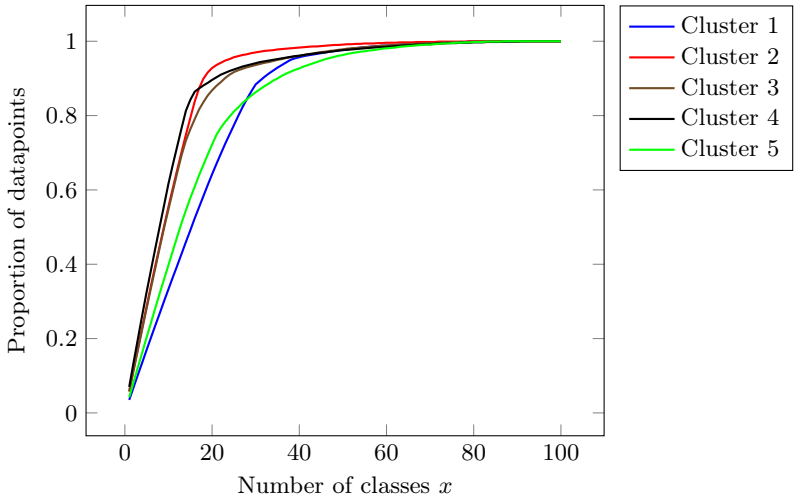


Fig. 7: Proportion of datapoints made up of the top x most common classes for each of the five cascade clusters (training set).

H Error Bars

Due to computational and time constraints, we were only able to run every experiment once. However, to give an idea of the level of noise in our results, we show the variance of our thresholding method by running it 5x on two different datasets. The results can be seen in Table 6.

Table 6: **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

I Datasets

We use datasets from [9], namely CIFAR100 & CIFAR10 [8], Oxford Flowers [11], Food101 [2], EuroSAT [7], SUN397 [16], UCF101 [14], Oxford-IIIT Pets [12], DTD [5]. For the Model Cascade, we also use a fine-grained subset of iNaturalist [15], iNatLoc500 from [6].

J Compute cost

At least 24GB of GPU memory is necessary to run the most demanding experiments. On an A100 gpu, most experiments (training a single model) take less than 24h. Self-supervised fine-tuning is usually slower than supervised fine-tuning. Masking one of the smaller datasets with the supervised algorithm only takes about 6 hours. Masking was not noticeably slower than full fine-tuning. So the costs of training one model/mask will vary between $12 and $48 at $2 per GPU-hour. This applies to the ResNet50 and ViT-B models. ResNet18 is a lot cheaper/faster.

K 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 pass-through trick is that it enables the learning of *discrete* components like text rather than only being able to learn images. Hence, the pass-through trick could be used to perform text reconstruction attacks. Existing approaches to privacy-preserving model training thus should also protect against this.

L Source Code

Source code is available at https://github.com/alvitawa/UnsupervisedMasking.

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. Bossard, L., Guillaumin, M., Van Gool, L.: Food-101–mining discriminative components with random forests. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13. pp. 446–461. Springer (2014)
3. Caron, M., Bojanowski, P., Joulin, A., Douze, M.: Deep clustering for unsupervised learning of visual features. In: Proceedings of the European conference on computer vision (ECCV). pp. 132–149 (2018)
4. 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)
5. Cimpoi, M., Maji, S., Kokkinos, I.: Sammy, mohamed, and andrea vedaldi. Describing textures in the, wild. In, CVPR **2** (2014)
6. Cole, E., Wilber, K., Van Horn, G., Yang, X., Fornoni, M., Perona, P., Belongie, S., Howard, A., Aodha, O.M.: On label granularity and object localization. In: Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part X. pp. 604–620. Springer (2022)
7. Helber, P., Bischke, B., Dengel, A., Borth, D.: Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **12**(7), 2217–2226 (2019)
8. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
9. 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)
10. 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)
11. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing. pp. 722–729. IEEE (2008)
12. Parkhi, O.M., Vedaldi, A., Zisserman, A., Jawahar, C.: Cats and dogs. In: 2012 IEEE conference on computer vision and pattern recognition. pp. 3498–3505. IEEE (2012)
13. 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)
14. Soomro, K., Zamir, A.R., Shah, M.: Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402 (2012)
15. Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P., Belongie, S.: The inaturalist species classification and detection dataset. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8769–8778 (2018)
16. Xiao, J., Hays, J., Ehinger, K.A., Oliva, A., Torralba, A.: Sun database: Large-scale scene recognition from abbey to zoo. In: 2010 IEEE computer society conference on computer vision and pattern recognition. pp. 3485–3492. IEEE (2010)