A CAP algorithm description

The section below illustrates in the CAP pruning algorithm step-by-step. Prunable model weights $\mathbb{R}^d$ are partitioned into blocks of fixed size B. Below $\rho_i^{(B)}$ denotes the saliency scores for weight i^{th} inside a block it belongs to and ρ_i is the score across the whole model. The steps of the algorithm are listed below:

Algorithm 1 CAP pruning algorithm

```
$\rho_i$ - saliency scores for weights
Accumulate Fisher inverse blocks $\mathbf{F}$
for each block do
    $\text{err} = 0$
    for element in a block do
        Select the weight $w_i$ with smallest score $\rho_i^{(B)}$ (using the (2) for $\rho_i$)
        Prune $w_i$
        Update remaining weights in the block via (2)
        $\text{err}+ = \rho_i^{(B)}$
        $\rho_i \leftarrow \text{err}$
        Save current state of the block for later merging
        Update Fisher inverse block
    end for
end for
Sort the scores $\rho_i$ in ascending order
Mark the weights with smallest scores $\rho_i$ as pruned
for each block do
    Load the saved state of the block with the weights marked pruned and all remaining alive.
end for
```

B Training details

Augmentation/regularization recipe

Table 4: Summary of the augmentation and regularization procedures used in the work.

Procedure	DeiT	light1
Weight decay	0.05	0.03
Label smoothing ε	0.1	0.1
Dropout	✗	✗
Stoch.Depth	0.1	0.0
Gradient Clip.	✗	1.0
H.flip	✓	✓
RRC	✓	✓
Rand Augment	9/0.5	2/0.5
Mixup alpha	0.8	0.0
Cutmix alpha	1.0	0.0
Erasing prob.	0.25	0.0
Erasing count	1	0
Test crop ratio	0.9	0.9

For the gradual pruning experiments (with 300 epochs) we have used cyclic learning schedule, with high learning rate directly after the pruning step with gradual decrease up to the next pruning step. For both DeiT-Tiny and DeiT-Small model during the additional fine-tuning for 100 epochs we've applied cosine annealing schedule with $\eta_{\max} = 5 \cdot 10^{-5}$, $\eta_{\min} = 1 \cdot 10^{-5}$ and all other parameters the same as in the Table 5.

Table 5: Hyperparameters of the schedules used in gradual pruning.

Model	Prune freq	LR sched $\{f_{\text{decay}}, \eta_{\max}, \eta_{\min}\}$	Augm	Batch size	Epochs
DeiT-Tiny	20	$\{$cyclic_linear, $5 \cdot 10^{-4}$, $1 \cdot 10^{-5}\}$	*light1*	1024	300
DeiT-Small	20	$\{$cyclic_linear, $5 \cdot 10^{-4}$, $1 \cdot 10^{-5}\}$	*deit*	1024	300

C Post-Pruning Recovery

The choice of augmentation parameters and learning rate schedule is critical for high performance. For example, reducing the level of augmentation during fine-tuning for smaller models, e.g. DeiT-Tiny, significant improves performance, whereas larger models, e.g. the 4x larger DeiT-Small, requires strong augmentations for best results even during fine-tuning. See Figure 5 for an illustration; the augmentation procedure is described in detail in B.

Moreover, the choice of cyclic learning rate (LR) schedule is critical as well. To illustrate this, we compare convergence obtained when using a *cosine annealing* schedule, which is very popular for pruning CNNs [24, 38, 32], from $\eta_{max} = 5 \cdot 10^{-4}$ to $\eta_{min} = 10^{-5}$, while performing pruning updates 2 times more frequently (one update per 10 epochs) than in our standard setup from the following section 4.2. The results are provided in Figure 5, where cosine annealing (no cycles) is in red. All experiments use the CAP pruner, and highlight the importance of the learning rate and augmentation schedules for recovery.

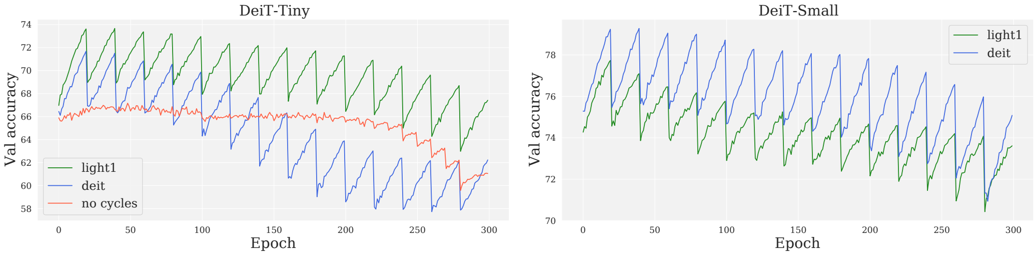

Figure 5: Ablations of the training setting on DeiT-Tiny (up) and DeiT-Small (down). Green curves correspond to the *light1* [40] augmentation recipe, blue curves to the *deit* [42] recipe. The red curve follows training with a single (acyclic) cosine annealing schedule, as in [24, 38].

D Additional Results for One-Shot Pruning

In this section we present comparison of Global Magnitude (GM), WoodFisher (WF) and Correlation Aware (CAP) pruners in one-shot pruning setting for DeiT models [42] of different size (i.e DeiT-Tiny, DeiT-Small, DeiT-Base) to study the scaling behavior of ViT sparsification.

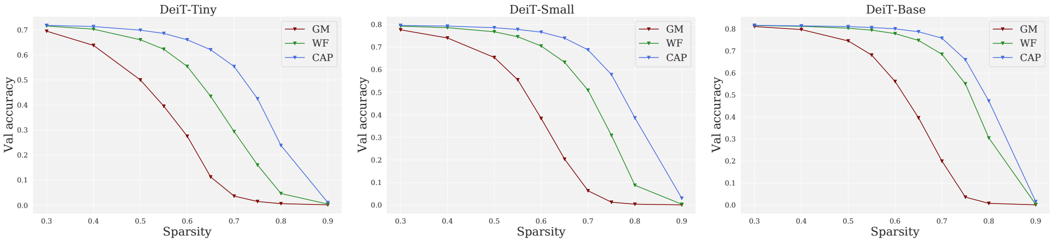

Figure 6: One-shot pruning of different DeiT versions.

Notice, that the gap between magnitude pruning and second order methods is very pronounced for all models, whereas the difference in performance between CAP and WF decreases with increase of the size of model. Nevertheless, CAP performs still noticeably better than WF, especially in high sparsity regime.

Pruning and finetuning. In most of the practical setups one cannot achieve both high compression rate and maintain performance of the dense model in one-shot setup. The full retraining procedure allows to achieve high sparsity but is rather expensive in the terms of compute. One can be interested to have something in between - moderately sparse model, close in performance to the original model but without the need of long and expensive training.

In the experiments below we prune all models to 50% sparsity, and fine-tune for 20 epochs. In addition to ViT/DeiT, we also consider similar models based on variants of self-attention [27, 1], and compare against a GM baseline. We use a linearly-decaying learning rate schedule between $\eta_{max} = 10^{-4}$ to $\eta_{max} = 10^{-5}$ and the DeiT training recipe [42]. The results are given in Table 6, and show that CAP can almost fully-recover accuracy in this setup for all models; the gaps from GM and WF (see DeiT-Small 75 and 90%) are still very significant.

Table 6: One-shot + fine-tuning on ImageNet-1k.

Model	Method	Sparsity (%)	Top1-Accuracy (%)
DeiT-Small	Dense	0	79.8
	GM	50	79.0
	CAP		**79.5**
	GM	75	74.3
	WF		75.8
	CAP		**76.9**
	GM	90	45.6
	WF		59.3
	CAP		**65.1**
DeiT-Base	Dense	0	81.8
	GM	50	81.5
	CAP		**81.6**
	GM	75	80.1
	WF		80.2
	CAP		**81.0**
	GM	90	68.1
	WF		69.2
	CAP		**76.3**

Model	Method	Sparsity (%)	Top1-Accuracy (%)
ConvNext-Small	Dense	0	83.1
	GM	50	82.5
	WF		82.5
	CAP		**82.8**
	GM	75	80.7
	WF		81.0
	CAP		**81.9**
	GM	90	70.9
	WF		73.2
	CAP		**78.2**
XCiT-Small	Dense	0	82.0
	GM	50	81.7
	CAP		**81.9**
Swin-Tiny	Dense	0	81.3
	GM	50	80.6
	CAP		**80.9**

E Experiments with other models

In the main part of the text, we considered only gradual pruning of ViT models, but the proposed method is applicable to any architecture for image classification, such as convolutional neural network (CNN) or a ViT-CNN hybrid. We have selected recently proposed EfficientFormer [26] as a member of ViT-CNN hybrid family and trained it using the same setting and hyperparameters as for DeiT-Small. Two CNN architectures - ResNet50-D [2] and EfficientNetV2-Tiny [41] [3], considered in this work were trained with the use of augmentation and regularization procedure described in the recent PyTorch blog post. Differently from most of the prior art we have used the ResNet50-D trained with the modern recipe from timm repository.

For ResNet50-D we prune all convolutional weights except the first convolution and we keep the classification layer dense. In EfficientNetv2-Tiny we do not prune depthwise convolutions since they do not contribute much to the total number of parameters and FLOPs but they are important to the model performance. We have set the block size to be 256 for ResNet50-D and 16 for EfficientNetV2-Tiny while keeping all the other hyperparameters of CAP the same as for DeiT experiments. Such a small block size was chosen for EfficientNetV2-Tiny due to the fact that it is the largest common divisor of the prunable weights.

First of all, we conducted comparison between one-shot pruning methods for ResNet50-D. We compare between Uniform and Global magnitude pruning, WoodFisher with block size of 256, M-FAC with block size of 2048 and CAP with uniform and global sparsity. One can observe that CAP outperforms all previous methods even when comparing uniform sparsity with global sparsity. Contrary to the case of DeiT where there is no much difference in performance between uniform and global magnitude pruning for ResNet50-D global sparsity turns out to be much better. This results is quite expectable since CNN are not uniform and deeper layers are mode wide than those close to the input.

---

[2] `resnet50d` checkpoint with 80.5 % accuracy for dense model

[3] `efficientnetv2_rw_t` checkpoint with 82.3 % accuracy for dense model

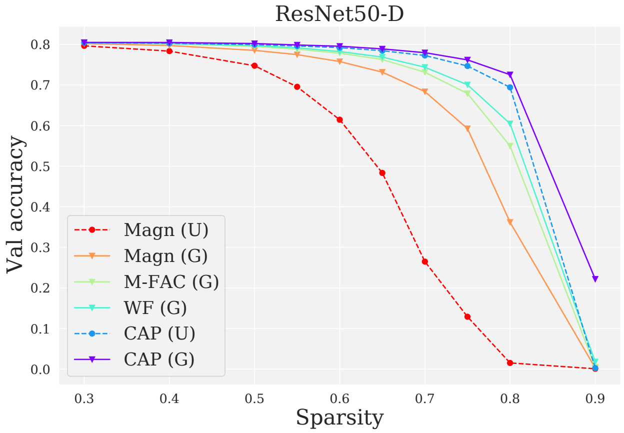

Figure 7: One-shot pruning of ResNet50-D model on ImageNet dataset.

Next we carried out one-shot + finetuning experiments with ResNet50-D keeping setup the same as for DeiT models in Table 6. We have selected 75 % and 90 % as the difference between methods becomes pronounced only at sufficiently high sparsity. Notably, the performance of Global magnitude and WF is roughly the same, the initial difference after one-shot pruning between WF and Global Magnitude vanishes during the finetuning procedure, whereas there is still a gap in performance between CAP and other methods.

Table 7: One-shot + fine-tuning on ImageNet.

Model	Method	Sparsity (%)	Top1-Accuracy (%)
ResNet-50D	Dense	0	80.5
	GM	75	79.0
	WF		79.0
	CAP		**79.2**
	GM	90	74.7
	WF		74.8
	CAP		**75.2**

Finally we conducted gradual pruning runs following the same sparsification schedule as for DeiT-models. The EfficientFormer and EfficientNet models despite being already very optimized and parameter efficient can be still compressed with small drop in accuracy.

F Scaling behaviour of pruning with respect to model size.

To study the scaling behavior with respect to model size we took all variants of the ConvNext2 family of models [48] since it covers wide range of model sizes except for the large one due to the memory and compute constraints. The smallest model from the family - ConvNext2-Atto has 3.7M parameters whereas the largest considered ConvNext2-Large has 198M parameters. All the models were pruned to 50% in one-shot. We observed that the relative accuracy drop (difference between accuracy of the dense and sparse model) initially decreases with increase of model size and then reaches a plateau. CAP consisently outpeforms WF across all scales and the difference is the most pronounced for the smallest model.

Table 8: Gradual pruning on ImageNet. Parentheses followed by the upwards directed arrow denote additional fine-tuning for 100 epochs.

Model	Method	Sparsity (%)	Top1-Accuracy (%)
EffFormer-L1	Dense	0	78.9
	CAP	50	78.0
		60	77.4
		75	76.4
		90	72.4 (72.8 ↑)
ResNet-50D	Dense	0	80.5
	CAP	50	79.8
		60	79.7
		75	79.2 (79.6 ↑)
		90	77.1 (77.5 ↑)
EffNetV2-Tiny	Dense	0	82.4
	CAP	50	81.0
		60	80.6
		75	79.6 (80.0 ↑)
		90	75.0

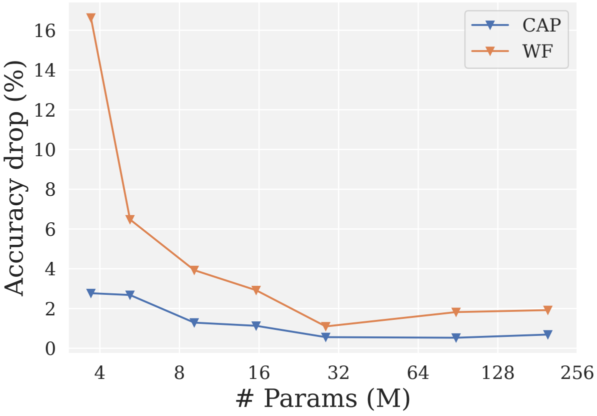

Figure 8: **Left**: CAP vs WoodFisher for pruning of ConvNext2 family.

G Timings

Any algorithms involving second order loss information are believed to require tremendous amounts of compute. Time required for calculation of pruning scores and the OBS update comprises collection of grads, Fisher inverse rank-1 updates and additional pruning iteration for CAP. We have measured the time of single pruning step for DeiT-Small and present the results in Table 9. All measurements were done on a single RTX A6000 GPU with 48GB of memory. One can observe that the amount of time needed to perform a pruning update is not very large, especially when compared to the duration of typical training procedure of modern computer vision models on ImageNet that usually takes several days on a multi-gpu node. Note that the additional step for CAP adds small fraction of total computational time relative to other steps of the OBS method.

Table 9: Minutes per pruning step for DeiT-Small.

Model	Method	Time (minutes)
DeiT-Small	Fast WoodFisher [22]	20
	CAP	23

H Composite compression

In addition to weight pruning one can decrease storage and inference cost with the help of other compression approaches: quantization (casting weights and activations to lower precision) and token pruning specific for the transformer architecture.

H.1 Quantization-Aware Training

Weight quantization is done in the following way - one takes sparse checkpoint and then runs quantization aware training (QAT). We ran QAT training for 50 epochs with linearly decaying learning rate schedule from $\eta_{\text{max}} = 10^{-4}$ to $\eta_{\text{min}} = 10^{-5}$. Models are quantized to 8-bit precision. In all experiments performed accuracy of quantized model almost reproduces the accuracy of the sparse model stored in full precision.

Table 10: ImageNet-1K top-1 accuracy for sparse models after QAT training.

Model	Sparsity (%)	Accuracy (%)
DeiT-Tiny	75	72.2
DeiT-Small	75	77.7
DeiT-Base	75	81

H.2 Token Pruning

There are different approaches for token pruning proposed in the literature. In this work we follow the one from [34]. Specifically, in DynamicViT one selects the ratio of tokens being pruned at each step with the lowest importance score, predicted by the model itself. Following the main setup from the paper we prune tokens after 3^{rd}, 6^{th}, 9^{th} block, and the token pruning ratio after each block is $\rho = 0.2$ (i.e 20% least improtant tokens are pruned).

Table 11: ImageNet-1K top-1 accuracy for sparse models with token pruning.

Model	Method	Sparsity (%)	Top1-Accuracy (%)
DynamicViT-Tiny	CAP	50	72.0
		60	71.6
		75	70.2
DynamicViT-Small	CAP	50	79.5
		60	79.4
		75	78.7

H.3 Semi-structured sparsity.

While CPUs can utilize sparsity patterns of arbitrary form to speed-up the computations at the present time modern GPU accelerators can handle only restricted form of unstructured sparsity, namely the $N : M$ sparsity pattern that enforces exactly N non-zero values for each block of M weights. Namely, since the introduction of Ampere architecture NVIDIA GPUs have special kernels that can work with $2 : 4$ sparse matrices [29]. One can integrate the $N : M$ sparsity in the CAP framework without significant changes. The only difference with the original CAP approach is that while running the CAP iterations one doesn't prune a given weight in case in a group of M weights to which this weights belongs to there are $M - N$ zero weights. Since the sparsity pattern is significantly constrained compared to generic unstructured sparsity pattern drop in performance after doing pruning step and consequent fine-tuning is more challenging than it would be for unconstrained sparsity. In experiments below we prune models to $2 : 4$ sparsity either in one-shot setting and one-shot+finetune. We apply shorter (10 epochs) and longer (50 epochs) finetuning procedure with linearly decaying learning rate schedule. According to the results in the Table 12 CAP significantly outperforms competitive methods for one-shot pruning, although the drop in performance is quite large for all methods. After finetuning procedure difference between different methods decreases. Nevertheless, there is some gap in performance between second order methods and magnitude pruning even after relatively long finetuning.

To demonstrate practical benefits from 2:4 sparsity pattern we compiled both sparse and dense models via TensorRT engine and compared the throughput. The inference was executed on Nvidia T4 GPU

Table 12: Semi-structured $2:4$ pruning of ViT models.

Model	Method	Epochs	Top1-Accuracy (%)
DeiT-Tiny	Dense		72.2
	GM	0	24.4
	WF		44.1
	CAP		**55.9**
	GM	10	68.8
	WF		71.1
	CAP		**71.5**
	GM	50	72.5
	WF		**72.7**
	CAP		**72.7**
DeiT-Small	Dense		79.8
	GM	0	53.6
	WF		67.8
	CAP		**72.0**
	GM	10	77.9
	WF		**78.1**
	CAP		78.0
	GM	50	78.6
	WF		**79.0**
	CAP		**79.0**
DeiT-Base	Dense		81.8
	GM	0	66.4
	WF		73.7
	CAP		**78.1**
	GM	10	81.2
	WF		**81.3**
	CAP		**81.3**
	GM	50	**81.7**
	WF		81.6
	CAP		**81.7**

with batch size of 64 in half precision. Sparsity allows for small but certain speedup for models of different scale.

Table 13: Speedup factors for $2:4$ sparsity.

Model	Speedup
DeiT-Tiny	1.07
DeiT-Small	1.07
DeiT-Base	1.10

I CAP/WF hyperparameters

Following the oBERT's directions [22] on identifying the optimal set of hyperparameters via one-shot pruning experiments, we conduct a grid search over the three most important hyperparameters:

- Number of grads collected for Fisher inverse
- Dampening constant λ
- Block size

The more grads are collected, the more accurate is the empirical Fisher inverse estimate, however, more compute is required at the same time. We chose $N = 4096$ as a point from which further increase of Fisher samples doesn't improve performance a lot. Dependence of the one-shot pruning performance at different sparsities vs number of grads is presented on Figure 9.

The next parameter to be studied is the dampening constant λ in. This constant regularizes the empirical Fisher matrix and allows to avoid instabilities in computation of the inverse. However, this constant decreases the correlation between different weights and in the limit $\lambda \to \infty$ OBS reduces to magnitude pruning. The optimal dampening constant for CAP ($\lambda_{\rm opt} = 10^{-8}$) is smaller than the one for WoodFisher ($\lambda_{\rm opt} = 10^{-6}$), i.e CAP remains numerically and computationally stable with smaller amount of regularization compared to WF (we observed that for $\lambda < 10^{-7}$ WF performance starts to deteriorate rapidly).

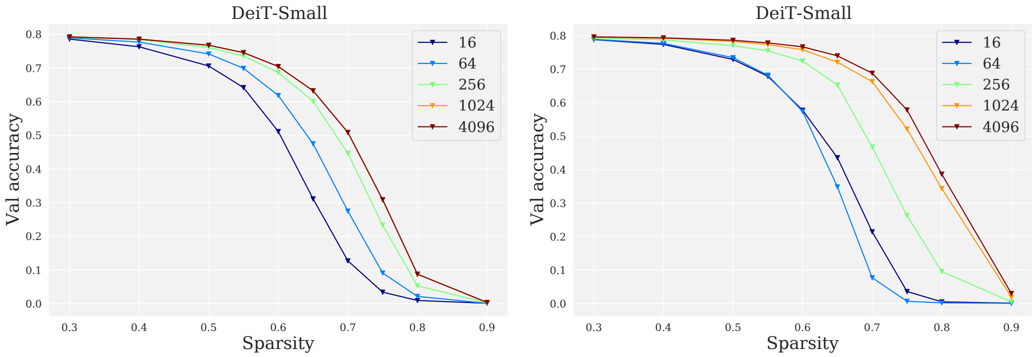

Figure 9: **Left**: One-shot pruning performance of WoodFisher. **Right**: One-shot pruning performance of CAP.

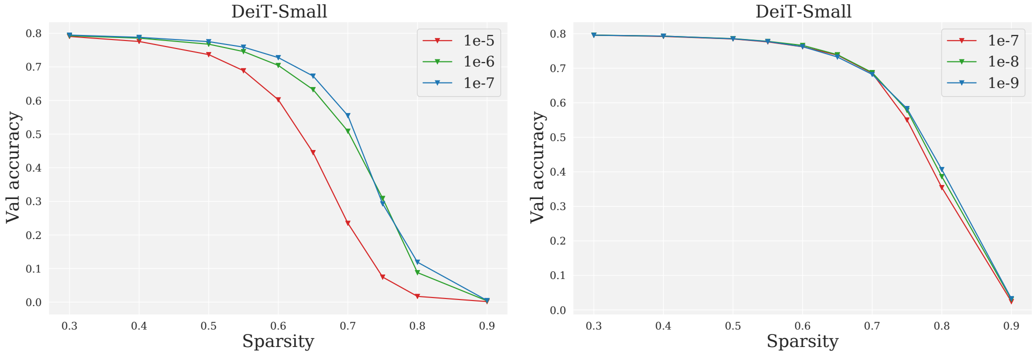

Figure 10: **Left**: One-shot pruning performance of WoodFisher. **Right**: One-shot pruning performance of CAP.

And the last but not the least important parameter is the block size in [38]. The larger the block size is, the more correlations between different weights are taken into account. However, as mentioned in 2 the computational and storage cost scales with the block size. Moreover, for a fixed number of gradients in the Fisher estimate matrix with larger block sizes is likely to be worse conditioned. Therefore, one would like to work with smaller block sizes but not to keep the approximation as accurate as possible. We've selected block size according to the accuracy-efficiency trade-off.

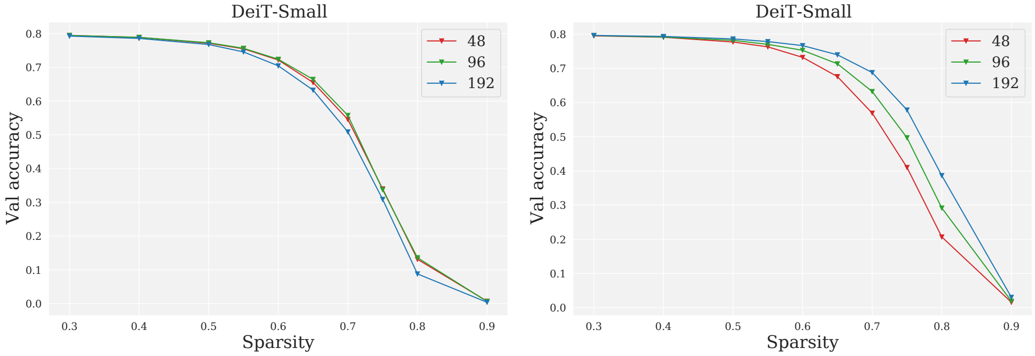

Figure 11: **Left**: One-shot pruning performance of WoodFisher. **Right**: One-shot pruning performance of CAP.

In addition, we've studied the benefit from application of multiple recomputations in the one-shot pruning setting for WoodFisher and CAP. Since the assumption of static Fisher matrix $\mathbf{F}(\mathbf{w}^*)$ doesn't hold in general, we expect that multiple recomputations are likely to result in higher one-shot accuracy in accordance with the result from [13]. This is indeed the case. The gain from recomputations is more pronounced for WoodFisher, since CAP already performs implicit Fisher inverse updates in its operation. Yet, the efect is not vanishing even for the case of CAP.

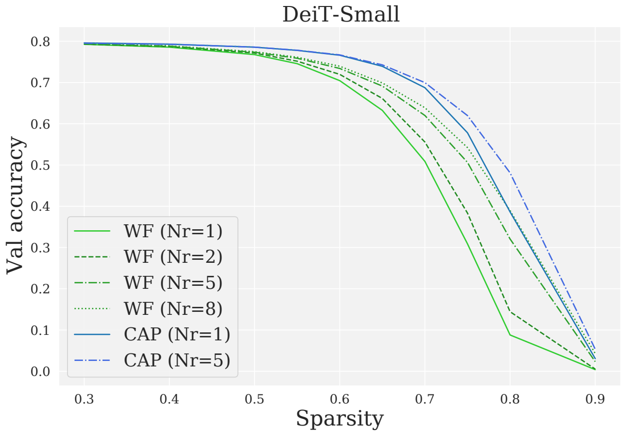

Figure 12: One-shot performance for WF and CAP with different number of recomputations N_r.

J Details and hyperparameter choices for other pruning methods.

In this section we provide some additional details about methods compared on Figures 2 and 7.
The popular Movement Pruning [37] computes the weight saliency scores during the training procedure, hence it is not a one-shot pruning method by the definition. We have observed that use of the naive elementwise product of gradient and weights (i.e $\rho_i = w_i \odot \nabla_{w_i} \mathcal{L}(w)$) leads to a poor performance, significantly below even the Magnitude Pruning baseline. However, the following first order pruning criterion:

$$\rho_i = \sum_{k=1}^{N} \| w_i^{(k)} \odot \nabla_{w_i} \mathcal{L}^{(k)}(w) \| \tag{9}$$

allows to get reasonable saliency scores that produce more accurate sparse models than Magnitude Pruning. However, its performance is still inferior to any of the second order pruning methods. This method is denoted as GrW (Gradient times weight) on Figures 2 and 7.
M-FAC Pruner proposed in [13] is a pruner leveraging second order information that doesn't require an explicit construction of Fisher Inverse matrix. Therefore, unlike WoodFisher and CAP that require $O(Bd)$ memory computation and storage cost of this method is constant with respect to the block size and one can take into account correlations between larger groups of weights for free. Following the original paper we chose block size of 2k as the best performing one. However, one can see from Figures 2 and 7 that smaller block sizes turn out to perform better. A possible explanation of this phenomenon is that the Fisher Inverse estimate becomes too noisy and unstable for large blocks.

K Execution latency.

In addition to the plot throughput vs accuracy shown in the main part we present in this section execution latency per sample vs latency when running models on the DeepSparse engine. The results are presented on Figure 13.

L Comparison with AC/DC training

In addition to the sparse training from scratch with periodic updates of the sparsity weights with some saliency criterion for weight elimination and regrowth [10] one can consider alternating compressed/decompressed training (AC/DC), proposed in [32]. Namely one switches between dense stages with standard unconstrained training of the model, and sparse stages when the model is pruned to the target sparsity level and trained with the frozen sparsity mask until the beginning of the next dense stage, when the sparsity mask is removed. This procedure produces both accurate dense and sparse models.

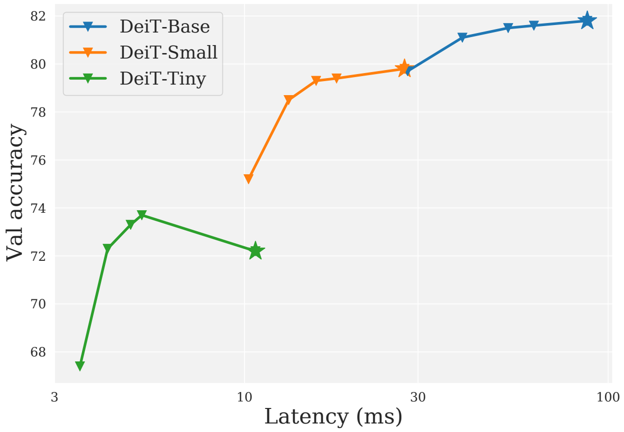

Figure 13: Accuracy vs latency on ImageNet-1k.

Following the original paper we use magnitude pruning as a saliency criterion for weight pruning. The augmentation and regularization pipeline follows the settings from [42]. All models with AC/DC were trained for 600 epochs in total with first pruning step at epoch 150 followed by 7 sparse stages 25 epochs long each, and 6 dense stages of the same length. The last dense stage lasts 50 epochs and the last sparse is 75 epochs long. Learning rate is gradually decayed from $\eta_{\text{max}} = 5 \cdot 10^{-4}$ to $\eta_{\text{min}} = 10^{-6}$ with cosine annealing. Initial warm-up phase with linearly increasing learning rate is 20 epochs. We compare AC/DC with CAP models finetuned for additional 100 epochs.

Table 14: AC/DC vs CAP (finetuned for additional 100 epochs) on ImageNet-1k.

Model	Method	Sparsity (%)	Top1-Accuracy (%)
DeiT-Small	CAP	60	79.9
	AC/DC		**80.4**
	CAP	75	**79.0**
	AC/DC		**79.0**
	CAP	90	**75.8**
	AC/DC		72.0

One can observe that at low sparsity AC/DC achieves higher accuracy for the same sparsity target (even outperforming the dense baseline by 0.6%), whereas for 75% performance of both methods is equal, and CAP outperforms AC/DC at higher sparsity. However, one should note, that CAP uses computational budget (including the training of original model) of 440 epochs for 60% sparsity, 520% for 75% and 700% for 90% vs 600 epochs used in AC/DC.

M One-shot pruning of DETR

The approach presented in the paper is not limited to the image classification task, but can be applied to other computer vision tasks, such as object detection. We chose the DeTR model [4] with ResNet50 backbone and ran one-shot pruning procedure with global magnitude, WoodFisher and CAP pruner. Specifically, we pruned all convolutional layers in the CNN backbone expect the first one and all linear projections in transformer encoder and decoder blocks while keeping the detection heads dense. The results are presented in Table 15. Following the standard protocol we used bbox mAP for evaluation. One can observe, that difference between the second order methods and magnitude pruning is very pronounced even for relatively small sparsity of 50%, and CAP outperforms WF pruner.

Table 15: One-shot pruning of DeTR.

Model	Method	Sparsity (%)	bbox mAP
DeTR	Dense	0	0.42
	GM	50	0.16
	WF		0.36
	CAP		**0.38**

N Proof of Theorem 1

Theorem N.0. *Let $\mathcal{S}$ be a set of samples, and let $\nabla_{\ell_1}(\mathbf{w}^*), \ldots, \nabla_{\ell_m}(\mathbf{w}^*)$ be a set of gradients with $i \in \mathcal{S}$, with corresponding empirical Fisher matrix $\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)$. Assume a sparsification target of k weights from $\mathbf{w}^*$. Then, a sparse minimizer for the the constrained squared error problem*

$$\mathit{min}_{\mathbf{w}'} \frac{1}{2m}\sum_{i=1}^{m}\Big(\nabla_{\ell_i}(\mathbf{w}^*)^\top \mathbf{w}' - \nabla_{\ell_i}(\mathbf{w}^*)^\top \mathbf{w}^*\Big)^2 \ \textit{s.t. } \mathbf{w}' \textit{ has at least } k \textit{ zeros}, \tag{10}$$

is also a solution to the problem of minimizing the Fisher-based group-OBS metric

$$\mathit{argmin}_{Q,|Q|=k}\ \frac{1}{2}\cdot \mathbf{w}^{*\top}_{\mathbf{Q}}\Big(\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)_{[Q,Q]}\Big)^{-1}\mathbf{w}^*_{\mathbf{Q}}. \tag{11}$$

Proof. We start by examining the unconstrained squared error function in Equation (10), which we denote by $\mathcal{G}$. Clearly, the function $\mathcal{G}$ is a d-dimensional quadratic in the variable $\mathbf{w}'$, and has a minimum at $\mathbf{w}^*$. Next, let us examine $\mathcal{G}$'s second-order Taylor approximation around $\mathbf{w}^*$, given by

$$(\mathbf{w}' - \mathbf{w}^*)^\top\Big(\frac{1}{m}\sum_{i=1}^{m}\nabla_{\ell_i}(\mathbf{w}^*)^\top\nabla_{\ell_i}(\mathbf{w}^*)\Big)(\mathbf{w}' - \mathbf{w}^*), \tag{12}$$

where we used the fact that $\mathbf{w}^*$ is a minimum of the squared error, and thus the function has 0 gradient at it. However, by the definition of the empirical Fisher, this is exactly equal to

$$(\mathbf{w}' - \mathbf{w}^*)^\top\widehat{\mathbf{F}}(\mathbf{w}^*)(\mathbf{w}' - \mathbf{w}^*). \tag{13}$$

The Taylor approximation is exact, as the original function is a quadratic, and so the two functions are equivalent. Hence, we have obtained the fact that, under the empirical Fisher approximation, a k-sparse solution minimizing Equation 10 will also be a k-sparse solution minimizing Equation 1. However, the question of finding a k-sparse solution minimizing Equation 1 is precisely the starting point of the standard OBS derivations (see e.g. [38] or [22]), which eventually lead to the formula in Equation (11). This concludes the proof. ☐

O Augmentation choice for Empirical Fisher

We compared the performance of CAP with Empirical Fisher computed on image-label pairs where the validation transforms were applied to images (i.e center crop with resize) and the same set of augmentations used for training and finetuning (RandAugment transforms, Label smoothing, e.t.c.). We observed that the sparsity solution obtained without augmenting samples for Empirical Fisher estimate turns out to be strongly overfitting. We point out that in both cases we use the same population size for Empirical Fisher.

Figure 14 illustrates this result: using validation augmentation (red) yields better training loss but degenerates in terms of validation accuracy. A possible explanation is that CAP chooses an overfitting solution which the model is unable to escape during finetuning.

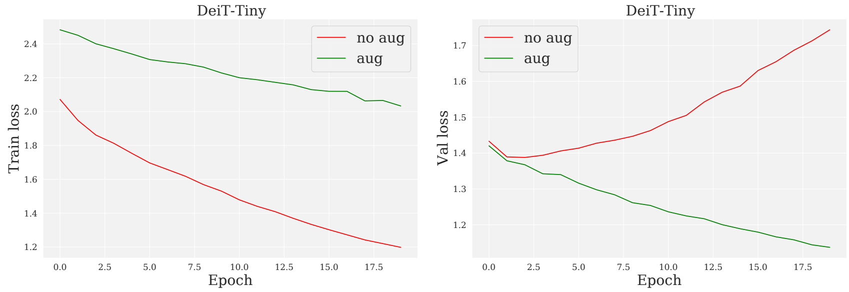

Figure 14: Training (**left**) and validation loss (**right**) for one-shot + finetuning.

P Fisher matrix structure

In order to validate the necessity of taking into account the correlations between weights, one has to make sure that the empirical Fisher matrix used as proxy for Hessian is non-diagonal. We have visualized an average block of empirical Fisher for a particular layer on Figure 15 from DeiT-Tiny and ConvNext-Small models. One can see, Fisher matrix exhibits a pronounced non-diagonal structure, which justifies the need of a careful and thorough treatment of weight correlations.

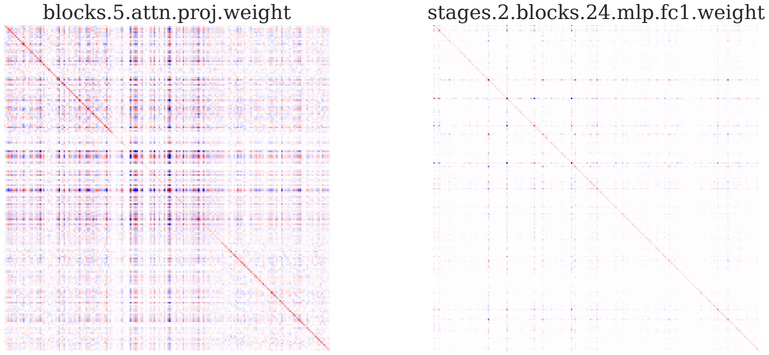

Figure 15: **Left**: empirical Fisher block for a weight from DeiT-Tiny. **Right**: empirical Fisher block for a weight from ConvNext-Small.

Q Broader impact

Compressed models are not expected to exhibit more malicious and potentially harmful behavior compared to the dense models. However, they may face the same issues like the original models in safety-critical applications such as susceptibility to adversarial attacks and distribution shifts.

R Limitations

The proposed method is mostly suitable for small and medium sized models (up to order of $\sim$ 100M). For larger models the compute and storage cost associated with the estimate of empirical Fisher becomes prohibitively expensive. Compression of the largest models considered is this paper requires 2-4 high-end GPUs (A100 with 80GiB). Pruning models to high sparsity requires significant amount of training. Search for fast and efficient procedures for the recovery of compressed models is left as potential direction for further research.