

Supplementary Material for

SOSP: Efficiently Capturing Global Correlations by

Second-Order Structured Pruning

A Additional Data and Experiments

A.1 Results for Medium Pruning Rates for Comparing Global Pruning Methods

Table 3: Comparison of SOSP to other global pruning methods for moderate pruning rates. The setting is exactly the same as in Tab. 1. For final accuracies after fine-tuning see App. A.3. * denotes the baseline model.

Dataset	Cifar10			Cifar100		
Method	Test acc. (%)	Reduct. in weights (%)	Reduct. in MACs (%)	Test acc. (%)	Reduct. in weights (%)	Reduct. in MACs (%)
VGG-Net*	94.18	-	-	73.45	-	-
NN Slimming	92.84	80.07	42.65	71.89	74.60	38.33
NN Slim. + L_1	93.79	83.45	49.23	72.78	76.53	39.92
C-OBDD	94.04 \pm 0.12	82.01 \pm 0.44	38.18 \pm 0.45	72.23 \pm 0.15	77.03 \pm 0.05	33.70 \pm 0.04
EigenDamage	93.98 \pm 0.06	78.18 \pm 0.12	37.13 \pm 0.41	72.90 \pm 0.06	76.64 \pm 0.12	37.40 \pm 0.11
SOSP-I (ours)	93.99 \pm 0.17	85.75 \pm 0.74	45.96 \pm 4.29	73.17 \pm 0.11	82.68 \pm 0.04	44.87 \pm 0.61
SOSP-H (ours)	93.73 \pm 0.16	87.29 \pm 0.21	57.74 \pm 2.57	73.11 \pm 0.19	79.20 \pm 0.35	51.61 \pm 0.98
ResNet-32*	95.30	-	-	76.8	-	-
C-OBDD	95.11 \pm 0.10	70.36 \pm 0.39	66.18 \pm 0.46	75.70 \pm 0.31	66.68 \pm 0.25	67.53 \pm 0.25
EigenDamage	95.17 \pm 0.12	71.99 \pm 0.13	70.25 \pm 0.24	75.51 \pm 0.11	69.80 \pm 0.11	71.62 \pm 0.21
SOSP-I (ours)	95.06 \pm 0.07	72.33 \pm 0.50	67.36 \pm 0.80	75.33 \pm 0.11	63.83 \pm 0.17	74.28 \pm 0.08
SOSP-H (ours)	95.22 \pm 0.12	72.85 \pm 0.40	67.85 \pm 0.37	75.52 \pm 0.20	69.31 \pm 0.36	71.60 \pm 0.38
DenseNet-40*	94.58	-	-	74.11	-	-
NN Slim. + L_1	94.32	35.52	-	73.76	35.45	-
SOSP-I (ours)	94.42 \pm 0.03	32.21 \pm 0.16	22.03 \pm 0.13	73.46 \pm 0.05	31.38 \pm 0.09	29.98 \pm 0.55
SOSP-H (ours)	94.41 \pm 0.12	34.78 \pm 0.67	26.14 \pm 0.13	73.60 \pm 0.17	34.15 \pm 0.13	28.23 \pm 0.09

A.2 Comparison of Accuracies Achieved by SOSP-I with and without Cross-Structure Correlations

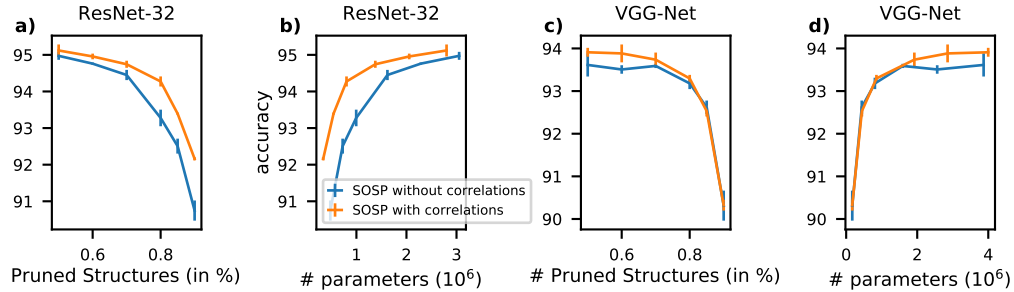


Figure 5: Comparison of the accuracies achieved by vanilla SOSP-I and a variation of SOSP-I, where all off-diagonal terms of the Hessian are set to zero, for ResNet-56 and DenseNet-40 on Cifar10. The results on both networks suggest that cross-structure correlations can significantly improve pruning performance.

451 A.3 Mean and Standard Deviation of Final Accuracies for the Global Pruning Comparison

Table 4: Mean and standard deviation of the final accuracies after the full fine-tuning step of both SOSP methods on CIFAR10 and CIFAR100 for VGG-Net, ResNet32 and Densenet40.

Dataset	CIFAR10						CIFAR100					
Pruning Ratio	moderate			high			moderate			high		
Method	Test acc (%)	Reduction in weights (%)	Reduction in MACs (%)	Test acc (%)	Reduction in weights (%)	Reduction in MACs (%)	Test acc (%)	Reduction in weights (%)	Reduction in MACs (%)	Test acc (%)	Reduction in weights (%)	Reduction in MACs (%)
VGG-Net(Baseline)	94.18	-	-	92.53 ± 0.13	97.79 ± 0.02	83.52 ± 0.29	73.45	-	-	63.87 ± 0.06	97.83 ± 0.04	87.02 ± 0.20
SOSP-I	93.88 ± 0.21	85.75 ± 0.74	45.96 ± 4.29	92.59 ± 0.19	97.81 ± 0.01	86.32 ± 0.29	72.93 ± 0.28	82.68 ± 0.04	44.87 ± 0.61	64.24 ± 0.55	97.81 ± 0.01	86.32 ± 0.29
SOSP-H	93.65 ± 0.16	87.29 ± 0.21	57.74 ± 2.57	-	-	-	72.94 ± 0.28	79.20 ± 0.35	51.61 ± 0.98	-	-	-
ResNet-32(Baseline)	95.30	-	-	92.19 ± 0.07	95.47 ± 0.33	94.07 ± 0.66	76.8	-	-	65.97 ± 0.52	92.69 ± 0.07	95.63 ± 0.13
SOSP-I	94.96 ± 0.08	72.33 ± 0.50	67.36 ± 0.80	91.97 ± 0.04	95.26 ± 0.10	94.45 ± 0.40	75.10 ± 0.10	63.83 ± 0.17	74.28 ± 0.08	67.37 ± 0.26	94.08 ± 0.21	95.06 ± 0.14
SOSP-H	95.17 ± 0.12	72.85 ± 0.40	67.85 ± 0.37	-	-	-	75.39 ± 0.27	69.31 ± 0.36	71.60 ± 0.38	-	-	-
DenseNet-40(Baseline)	94.58	-	-	94.07 ± 0.07	47.00 ± 0.10	36.35 ± 0.12	74.11	-	-	72.10 ± 0.10	45.22 ± 0.10	42.05 ± 1.16
SOSP-I	94.29 ± 0.04	32.21 ± 0.16	22.03 ± 0.13	94.15 ± 0.08	49.39 ± 0.65	38.86 ± 0.70	72.47 ± 0.47	31.38 ± 0.09	29.98 ± 0.55	72.09 ± 0.44	48.58 ± 0.22	42.05 ± 0.35
SOSP-H	94.28 ± 0.11	34.78 ± 0.67	26.14 ± 0.13	-	-	-	72.88 ± 0.43	34.15 ± 0.13	28.23 ± 0.09	-	-	-

452 A.4 Tabular Data and Mean and Standard Deviation for Layer-Wise Pruning Comparison

453 This section provides additional data complementing the results of Fig. 1. The numerical data of
454 Fig. 1 is shown in Tab. 5 and the mean and standard deviation of the final accuracies for both SOSP
455 algorithms is shown in Tab. 6 and Fig. 6.

Table 5: Pruning results of ResNet-56 and DenseNet-40 on CIFAR-10. *Gap* denotes the difference between the accuracy of the pruned model and the baseline accuracy. *PR* denotes the pruning ratio, i.e. the percentage drop in MACs or parameters.

Model	Top-1(Gap)%	Parameters(PR)	MACs(PR)
ResNet-56	93.88(0.0)	0.85M(0%)	125M(0%)
GAL Lin et al. (2019)	92.98(0.28)	0.75M(12%)	78M(38%)
SOSP-I (ours)	94.22(-0.44)	0.74M (13%)	110M (13%)
SOSP-H (ours)	94.25(-0.47)	0.73M (15%)	109M (14%)
HRank Lin et al. (2020)	93.52(-0.26)	0.71M(17%)	89M(29%)
SOSP-I (ours)	93.85(0.03)	0.54M (36%)	84M (33%)
SOSP-H (ours)	93.71(0.17)	0.52M (40%)	79M (37%)
FPGM He et al. (2019)	93.01(0.58)	0.49M(42%)	81M(36%)
HRank Lin et al. (2020)	93.17(0.09)	0.49M(42%)	63M(50%)
SOSP-I (ours)	93.25(0.53)	0.36M (58%)	60M (53%)
SOSP-H (ours)	93.27(0.51)	0.33M (61%)	54M (57%)
GALLin et al. (2019)	90.36(2.10)	0.29M(66%)	50M(60%)
HRank Lin et al. (2020)	90.72(2.54)	0.27M(68%)	32M(74%)
SOSP-I (ours)	92.35(1.53)	0.19M (78%)	36M (71%)
SOSP-H (ours)	91.97(1.91)	0.18M(79%)	31M (75%)
SOSP-I (ours)	90.80(3.08)	0.11M (87%)	22M (82%)
SOSP-H (ours)	90.78(3.10)	0.11M(87%)	21M (83%)
DenseNet-40	94.58(0.0)	1.04M(0%)	283M(0%)
SOSP-I (ours)	94.46(0.12)	0.88M(17%)	255M(10%)
SOSP-H (ours)	94.63(-0.05)	0.86M(19%)	245M(14%)
GAL Lin et al. (2019)	94.29(0.52)	0.67M(36%)	183M(35%)
HRank Lin et al. (2020)	94.24(0.57)	0.66M(37%)	167M(41%)
SOSP-I (ours)	94.35(0.22)	0.72M(32%)	221M(22%)
SOSP-H (ours)	94.43(0.15)	0.69M(35%)	209M(26%)
SOSP-I (ours)	94.14(0.44)	0.56M(47%)	180M(36%)
SOSP-H (ours)	94.27(0.31)	0.54M(49%)	172M(39%)
HRank Lin et al. (2020)	93.68(1.13)	0.48M(54%)	110M(61%)
GAL Lin et al. (2019)	93.53(1.28)	0.45M(57%)	128M(55%)
Zhao et al. (2019)	93.16(0.95)	0.42M(60%)	156M(45%)
SOSP-I (ours)	93.70(0.88)	0.42M(60%)	141M(50%)
SOSP-H (ours)	94.74(0.84)	0.40M(62%)	138M(51%)

Table 6: Mean and standard deviations of the accuracies after fine-tuning of SOSP for ResNet-56 and DenseNet-40.

Model	Top-1%	Pruned Parameters (in %)	Pruned MACs (in %)
ResNet-56	93.39	0	0
SOSP-I	94.22 \pm 0.10	12.95 \pm 0.15	12.78 \pm 0.09
SOSP-H	94.25 \pm 0.14	14.90 \pm 0.12	13.11 \pm 0.36
SOSP-I	93.85 \pm 0.09	36.36 \pm 0.11	33.31 \pm 0.12
SOSP-H	93.71 \pm 0.11	39.70 \pm 0.39	36.86 \pm 0.70
SOSP-I	93.25 \pm 0.16	58.19 \pm 0.28	52.53 \pm 0.21
SOSP-H	93.27 \pm 0.06	60.95 \pm 0.29	56.74 \pm 0.31
SOSP-I	92.35 \pm 0.25	77.98 \pm 0.46	71.04 \pm 0.21
SOSP-H	91.97 \pm 0.11	79.29 \pm 0.17	75.07 \pm 0.62
SOSP-I	90.8 \pm 0.18	86.68 \pm 0.14	81.79 \pm 0.23
SOSP-H	90.78 \pm 0.14	87.31 \pm 0.29	83.42 \pm 0.15
DenseNet-40	94.16	0	0
SOSP-I	94.46 \pm 0.11	16.59 \pm 0.22	10.00 \pm 0.72
SOSP-H	94.63 \pm 0.15	18.57 \pm 0.63	13.52 \pm 2.13
SOSP-I	94.35 \pm 0.04	32.21 \pm 0.16	22.03 \pm 0.13
SOSP-H	94.43 \pm 0.11	34.78 \pm 0.67	26.14 \pm 1.16
SOSP-I	94.14 \pm 0.07	47.00 \pm 0.10	36.35 \pm 0.12
SOSP-H	94.27 \pm 0.08	49.39 \pm 0.65	38.86 \pm 0.70
SOSP-I	94.70 \pm 0.08	60.32 \pm 0.31	50.24 \pm 0.80
SOSP-H	94.74 \pm 0.08	62.24 \pm 0.74	51.28 \pm 1.07

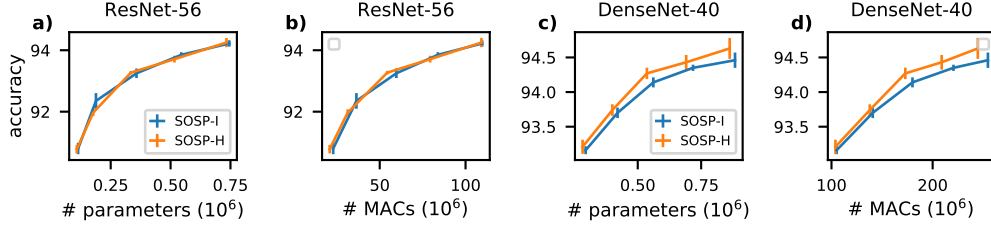


Figure 6: Mean and standard deviation plots of the accuracies after fine-tuning of SOSP for ResNet-56 and DenseNet-40 on Cifar10.

456 A.5 Pruning at Initialization vs Pruning a Pretrained Neural Network

457 In this section we investigate further why pruning a randomly initialized network tends to achieve
458 lower accuracies compared to pruning a pretrained network (Lee et al., 2018; van Amersfoort et al.,
459 2020). We compare the final accuracies of pruning and fine-tuning a randomly initialized and
460 pretrained network (see Fig. 7). While pruning a pretrained network leads to considerably higher
461 accuracies compared to pruning a randomly initialized network, the random baseline curves show the
462 same difference. Thus, to be able to compare pruning before and after training one needs to device
463 settings that enable a fair comparison, ensuring similar accuracies for random pruning across both
464 settings.

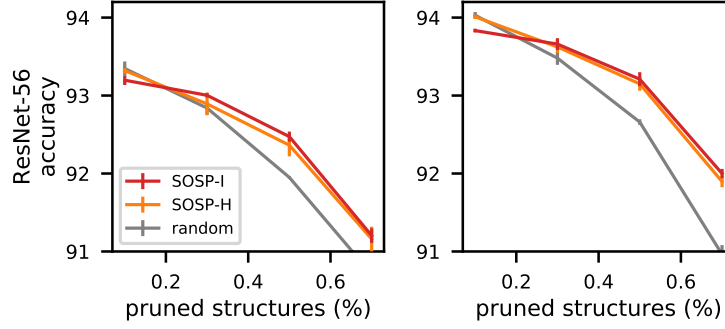


Figure 7: Comparison of pruning and fine-tuning a randomly initialized (left) and a pretrained (right) network for ResNet-56 on Cifar10. Random pruning is a baseline which selects uniformly at random structures s and adds them to the mask M until the predefined pruning ratio is reached.

465 A.6 Expand-init data

466 This section provides the experimental results of the corresponding expand-procedure at initialization
 467 (see Sec. 3.4). The results of Fig. 3 indicate that architectural bottlenecks exist not only for pretrained
 468 networks but also for randomly initialized networks. Therefore, we devise also an expand scheme
 469 before training. In this scheme the mask for the expand-procedure is not calculated for a pretrained
 470 network but for a randomly initialized network. Following the expand scheme the network is pruned
 471 and then only fine-tuned once. The results are displayed in Fig. 8.

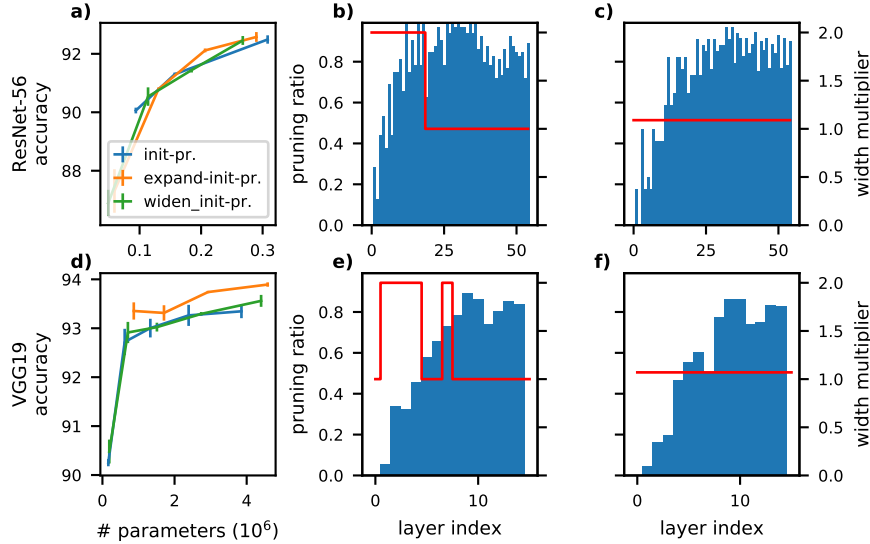


Figure 8: We remove architectural bottlenecks found by SOSp starting with a randomly initialized network. The width of blocks and layers with low pruning ratios in the train-pruning scheme (Fig. 3e and h) are expanded by a width multiplier of 2 (b, e). As a baseline, we again uniformly expand all layers in the network by a factor 1.1 (c, f). The layer-wise pruning ratios of the enlarged network models are shown as bar plots in (b, c, e, f). The average and standard deviation of the test accuracy across 3 trials are shown over the number of model parameters (a, d). Note that the full ResNet-56 and VGG models have $0.86 \cdot 10^6$ and $20 \cdot 10^6$ parameters, respectively.

B Approximative Second-Order Derivatives

Here we derive our approximations of the second-order loss derivatives, see Eq. (6) which is based on Eq. (5), and especially the particular matrix structures of R_n in these expressions, which allow for a more efficient matrix multiplication.

Recall that our approximation is based on omitting from the exact loss derivative those terms that involve the (expensive) second-order derivatives $\nabla_{\theta}^2 f_{\theta}(x_n)$ of the NN outputs $f_{\theta}(x_n) \in \mathbb{R}^D$. We however still include second-order couplings due to the loss function ℓ . Our approximation is thus to approximate the NN output

$$f_{\theta'}(x) \approx f_{\theta'}^{lin}(x) := f_{\theta}(x) + \phi(x) \cdot (\theta' - \theta) \quad (10)$$

to first order, where $\phi(x) := \nabla_{\theta} f_{\theta}(x) \in \mathbb{R}^{D \times P}$ is the first-order derivative of the NN output, and then to approximate the second-order derivatives of the NN loss as follows:

$$\nabla_{\theta}^2 \ell(f_{\theta}(x_n), y_n) \approx \nabla_{\theta}^2 \ell(f_{\theta}^{lin}(x_n), y_n). \quad (11)$$

We now compute this approximation $\nabla_{\theta}^2 \ell(f_{\theta}^{lin}(x_n), y_n)$ for both the *squared loss* $\ell(f, y) := \frac{1}{2} \|f - y\|^2$ as well as for the *cross-entropy loss* $\ell(f, y) := -\log \sigma(f)_y$, where $\sigma : \mathbb{R}^D \rightarrow \mathbb{R}^D$ denotes the softmax function. In this computation we use the following facts:

$$f_{\theta}^{lin}(x_n) = f_{\theta}(x_n), \quad (12)$$

$$\nabla_{\theta} f_{\theta}^{lin}(x_n) := \nabla_{\theta'} f_{\theta'}^{lin}(x_n)|_{\theta'=\theta} = \phi(x_n) = \nabla_{\theta} f_{\theta}(x_n), \quad (13)$$

$$\nabla_{\theta}^2 f_{\theta}^{lin}(x_n) := \nabla_{\theta'}^2 f_{\theta'}^{lin}(x_n)|_{\theta'=\theta} = 0, \quad (14)$$

which follow directly from (10).

B.1 Second-Order Approximation for Squared Loss

For the squared loss, we obtain:

$$\nabla_{\theta}^2 \ell(f_{\theta}^{lin}(x_n), y_n) = \nabla_{\theta}^2 \left[\frac{1}{2} (f_{\theta}^{lin}(x_n) - y_n)^T (f_{\theta}^{lin}(x_n) - y_n) \right] \quad (15)$$

$$= (f_{\theta}^{lin}(x_n) - y_n)^T (\nabla_{\theta}^2 f_{\theta}^{lin}(x_n)) + (\nabla_{\theta} f_{\theta}^{lin}(x_n))^T (\nabla_{\theta} f_{\theta}^{lin}(x_n)) \quad (16)$$

$$= \phi(x_n)^T \phi(x_n) \quad (17)$$

$$= \phi(x_n)^T R_n \phi(x_n), \quad (18)$$

where $R_n := 1_{D \times D}$ is here the $D \times D$ -identity matrix. This is Eq. (5) for the squared loss.

Due to this diagonal form of R_n , the matrix multiplication $(\phi(x_n)\theta_s)^T R_n (\phi(x_n)\theta_{s'})$ in Eq. (6) has, for each pair (s, s') , a computational complexity *linear* in the dimension D (which is the number of NN outputs), rather than quadratic:

$$(\phi(x_n)\theta_s)^T R_n (\phi(x_n)\theta_{s'}) = (\phi(x_n)^T \theta_s)(\phi(x_n)\theta_{s'}) \quad (19)$$

$$= \sum_{j=1}^D (\phi(x_n)_j \theta_s)_j (\phi(x_n)\theta_{s'})_j, \quad (20)$$

where $(\phi(x_n)\theta_s)_j \in \mathbb{R}$ denotes the j -th component of the vector $\phi(x_n)\theta_s \in \mathbb{R}^D$.

It is for this reason that the overall computational complexity of SOSPI is linear in D (see Sect. 2.1), rather than of order $O(D^2)$.

B.2 Second-Order Approximation for Cross-Entropy Loss

Note that the first derivatives of the softmax-function $\sigma : \mathcal{R}^D \rightarrow \mathcal{R}^D$, defined by $\sigma(f)_i := e^{f_i} / \sum_{k=1}^D e^{f_k}$ are:

$$\frac{\partial}{\partial f_j} \sigma(f)_i = -\sigma(f)_i (\sigma(f)_j - \delta_{ij}). \quad (21)$$

498 We can thus compute for the cross-entropy loss, where $\delta_{y \cdot} \in \mathbb{R}^D$ denotes the vector with entry 1 in
 499 component y and entries 0 everywhere else:

$$\nabla_{\theta}^2 \ell(f_{\theta}^{lin}(x_n), y_n) = -\nabla_{\theta}^2 [\log \sigma(f_{\theta}^{lin}(x_n))_{y_n}] \quad (22)$$

$$= \nabla_{\theta} [(\nabla_{\theta} f_{\theta}^{lin}(x_n))^T (\sigma(f_{\theta}^{lin}(x_n)) - \delta_{y \cdot})] \quad (23)$$

$$= (\nabla_{\theta} f_{\theta}^{lin}(x_n))^T (\nabla_{\theta} \sigma(f_{\theta}^{lin}(x_n))) + (\nabla_{\theta}^2 f_{\theta}^{lin}(x_n))^T (\sigma(f_{\theta}^{lin}(x_n)) - \delta_{y \cdot}) \quad (24)$$

$$= (\nabla_{\theta} f_{\theta}^{lin}(x_n))^T \cdot (\nabla_{\theta} \sigma(f_{\theta}^{lin}(x_n))). \quad (25)$$

500 To compute $\nabla_{\theta} \sigma(f_{\theta}^{lin}(x_n))$ in (25), we consider its i -th component:

$$\nabla_{\theta} \sigma(f_{\theta}^{lin}(x_n))_i = \sum_{j=1}^D \frac{\partial \sigma(f)_i}{\partial f_j} \Big|_{f=f_{\theta}^{lin}(x_n)} \cdot \nabla_{\theta} (f_{\theta}^{lin}(x_n))_j \quad (26)$$

$$= \sum_{j=1}^D \sigma(f_{\theta}^{lin}(x_n))_i (\delta_{ij} - \sigma(f_{\theta}^{lin}(x_n))_j) \cdot \nabla_{\theta} (f_{\theta}^{lin}(x_n))_j \quad (27)$$

$$= \sum_{j=1}^D (R_n)_{ij} \cdot \nabla_{\theta} (f_{\theta}^{lin}(x_n))_j \quad (28)$$

$$= (R_n \cdot \nabla_{\theta} f_{\theta}^{lin}(x_n))_i, \quad (29)$$

501 where $R_n \in \mathbb{R}^{D \times D}$ is the matrix with entries

$$(R_n)_{ij} = \sigma(f_{\theta}^{lin}(x_n))_i \delta_{ij} - \sigma(f_{\theta}^{lin}(x_n))_i \sigma(f_{\theta}^{lin}(x_n))_j \quad (30)$$

$$= \sigma(f_{\theta}(x_n))_i \delta_{ij} - \sigma(f_{\theta}(x_n))_i \sigma(f_{\theta}(x_n))_j. \quad (31)$$

502 Thus, plugging back into (25),

$$\nabla_{\theta}^2 \ell(f_{\theta}^{lin}(x_n), y_n) = (\nabla_{\theta} f_{\theta}^{lin}(x_n))^T R_n (\nabla_{\theta} f_{\theta}^{lin}(x_n)) \quad (32)$$

$$= \phi(x_n)^T R_n \phi(x_n), \quad (33)$$

503 which is Eq. (5) for the cross-entropy loss.

504 Note that R_n is a sum of a diagonal matrix R_n^{diag} (first part in Eq. (31)) and a rank-1 matrix
 505 $R_n^{\text{rank-1}}$ (second part in Eq. (31)). Due to this special matrix form, the matrix multiplication
 506 $(\phi(x_n)\theta_s)^T R_n (\phi(x_n)\theta_{s'})$ in Eq. (6) has, for each pair (s, s') , a computational complexity *linear*
 507 in the dimension D (the number of NN outputs), rather than quadratic. For the diagonal part
 508 $(R_n^{\text{diag}})_{ij} = \sigma(f_{\theta}(x_n))_i \delta_{ij}$, the reason for this is similar to the one for the squared error:

$$(\phi(x_n)\theta_s)^T R_n^{\text{diag}} (\phi(x_n)\theta_{s'}) = \sum_{i,j=1}^D (\phi(x_n)\theta_s)_i (R_n^{\text{diag}})_{ij} (\phi(x_n)\theta_{s'})_j \quad (34)$$

$$= \sum_{j=1}^D (\phi(x_n)\theta_s)_j \sigma(f_{\theta}(x_n))_j (\phi(x_n)\theta_{s'})_j. \quad (35)$$

509 For the rank-1 part $(R_n^{\text{rank-1}})_{ij} = -\sigma(f_{\theta}(x_n))_i \sigma(f_{\theta}(x_n))_j$, the $O(D)$ -efficient computation is

$$(\phi(x_n)\theta_s)^T R_n^{\text{rank-1}} (\phi(x_n)\theta_{s'}) = \sum_{ij} (\phi(x_n)\theta_s)_i (R_n^{\text{rank-1}})_{ij} (\phi(x_n)\theta_{s'})_j \quad (36)$$

$$= - \sum_{ij=1}^D (\phi(x_n)\theta_s)_i \sigma(f_{\theta}(x_n))_i (\phi(x_n)\theta_{s'})_j \sigma(f_{\theta}(x_n))_j \quad (37)$$

$$= - \left(\sum_{i=1}^D (\phi(x_n)\theta_s)_i \sigma(f_{\theta}(x_n))_i \right) \cdot \left(\sum_{j=1}^D (\phi(x_n)\theta_{s'})_j \sigma(f_{\theta}(x_n))_j \right). \quad (38)$$

Again, for these reasons, the overall computational complexity of SOSP-I is linear in D (see Sect. 2.1), instead of quadratic in D . This can make a significant difference for some datasets (e.g. $D = 1000$ classes on ImageNet).

An alternative $O(D)$ -efficient way of computing our second-order approximation follows by continuing from Eq. (25), noting that $\nabla_{\theta} f_{\theta}^{lin}(x_n) = \nabla_{\theta} f_{\theta}(x_n)$ by (13):

$$\begin{aligned}\nabla_{\theta}^2 \ell(f_{\theta}^{lin}(x_n), y_n) &= (\nabla_{\theta} f_{\theta}(x_n))^T \cdot (\nabla_{\theta} \sigma(f_{\theta}(x_n))) \\ &= \phi(x_n)^T \cdot \phi^{\sigma}(x_n),\end{aligned}\tag{39}$$

where we defined $\phi^{\sigma}(x_n) := \nabla_{\theta} (\sigma(f_{\theta}(x_n))) \in \mathbb{R}^{D \times P}$. Thus, each term in the sum (6) can be written as

$$(\phi(x_n)\theta_s)^T (\phi^{\sigma}(x_n)\theta_{s'}) = \sum_{j=1}^D (\phi(x_n)\theta_s)_j (\phi^{\sigma}(x_n)\theta_{s'})_j,\tag{41}$$

which again has complexity $O(D)$. For this it is necessary to pre-compute each $\phi^{\sigma}(x_n)$ in addition to $\phi(x_n)$, but both have the same complexity.

We finally note that our approximation of the second-order derivatives (i.e., of the Hessian) is somewhat different from the approximation made in (Peng et al., 2019) for the cross-entropy case: While we dropped all second-order derivatives $\nabla_{\theta}^2 f_{\theta}(x_n)$ of the *pre*-softmax activations, Peng et al. (2019) dropped all second-order derivatives of the *post*-softmax activations, i.e. all terms $\nabla_{\theta}^2 (\sigma(f_{\theta}(x_n)))$. A consequence of this difference is that our approximation (33) (or (41)) does not depend on the labels y_n (this is already apparent from our intermediate step Eq. (25)), whereas the approximation made in (Peng et al., 2019) does depend on the labels y_n . The fact that our second-order approximation is independent of the y_n is similar to the second-order approximation in the squared-loss case (see App. B.1 above, and also (Peng et al., 2019)). Note furthermore that the first-order terms in the loss approximation (see e.g. in Eq. (2) or (3)) are the same in our method as in (Peng et al., 2019), and these do depend on the labels y_n (cf. the expression in square bracket in Eq. (23)).

C Second-Order Approximation Corresponds to Output-Correlation

The purpose of this section is to provide a better intuition of the second-order components of our loss approximation. First, we reformulate the expression for the second-order terms in Eq. 6. We use the fact that for any ReLU-NN without batch normalization layers it holds almost everywhere that

$$\nabla_{\theta} f_{\theta}(x)\theta_s = f_{\theta^s}(x),\tag{42}$$

where θ^s is the weight vector, where all structures of the layer that contains structure s are set to zero except for the weights of structure s itself. The identity is straightforward to derive, therefore we show the relation for a feed-forward neural network with zero biases, but the proof for a convolutional neural network is almost identical.

Let f_{θ} be a fully-connected neural network with L layers and $f_{\theta}(x) = W_L \mathbf{1}_{h^{L-1}(x) \geq 0} W_{L-1} \dots \mathbf{1}_{h^1(x) \geq 0} W_1 x$, where W_l are the weight-matrices and $h^l(x)$ the output functions of the l -th layer and $\mathbf{1}_{h^l(x) \geq 0}$ the diagonal matrix with the step function on the diagonal elements corresponding to the components of $h^l(x) \geq 0$. Now assuming that structure s is contained in the i -th layer, the only non-vanishing components of the vector θ_s are the once associated with structure s . Thus, one can evaluate the gradient to get

$$\nabla_{\theta} f_{\theta}(x)\theta_s = W_L \mathbf{1}_{h^{L-1}(x) \geq 0} W_{L-1} \dots \mathbf{1}_{h^i(x) \geq 0} W_i^s \dots \mathbf{1}_{h^1(x) \geq 0} W_1 x,\tag{43}$$

where W_i^s is the weight matrix of layer i where all components are set to zero except for those belonging to structure s . Using this, one directly receives $\nabla_{\theta} f_{\theta}(x)\theta_s = f_{\theta^s}(x)$

Next, applying this identity to Eq. 6 gives

$$\begin{aligned}
\theta_s^T H(\theta) \theta_{s'} &\approx \frac{1}{N'} \sum_{n=1}^{N'} (\phi(x_n) \theta_s)^T R_n (\phi(x_n) \theta_{s'}) \\
&= \frac{1}{N'} \sum_{n=1}^{N'} (\nabla_{\theta} f_{\theta}(x_n) \theta_s)^T R_n (\nabla_{\theta} f_{\theta}(x_n) \theta_{s'}) \\
&= \frac{1}{N'} \sum_{n=1}^{N'} f_{\theta_s}(x_n) R_n f_{\theta_{s'}}(x_n),
\end{aligned}$$

where the explicit form of the R_n matrix is provided in the previous section. From this relation one can now see that the second-order components of our pruning objective correspond to output-correlations. This connection could also explain, why our second-order pruning methods do not improve the pruning performance at initialization, but do improve performance after training, since at initialization different structures may not have as strong output-correlations as the learned features after training.

D Counting of Parameters and MACs

Here we provide details on how we compute the number of parameters of our pruned NNs, and the number of MACs required for the evaluation of a pruned NN on one input point. The description is specific to the ResNets and DenseNets used in our work; the main complication arises for the ResNet architecture (in particular for the residual connections), as we describe below.

While our parameter and MAC counting is exact, it is somewhat intricate. We do not know whether this exact counting has been implemented in other papers on pruning as well (see the comparisons in Tables 1 and 2), since these other works did not elaborate on their counting. Therefore, the comparability with the counts from other papers is not necessarily given. When we compare our exact counting to a more straightforward but approximative counting, we find that our exact counts for ResNet50 (Table 2) yield substantially higher parameter and MAC numbers than the straightforward approximate counting. The straightforward approximate counting would yield MAC-pruning-ratios that are 12-18 percentage points higher (better) than our exact numbers. We now explain our exact counting first.

The number of parameters of a convolutional layer l equals the number F_l^{in} of input filters of the layer times the number F_l^{out} of output filters times the kernel size K_l^{wh} (which equals the kernel width times the kernel height): $C^l = F_l^{in} \cdot F_l^{out} \cdot K_l^{wh}$. In addition to this count, there are $2F_l^{out}$ parameters for the batchnorm layer following each convolutional layer (our networks do not have bias terms in the convolutional layers, which would add another F_l^{out}).

The subtlety is now that, within the chain of convolutional layers in the ResNet architecture, the number F_{l+1}^{in} of input filters into the following convolutional layer $l+1$ does *not necessarily* equal the number of output filters F_l^{out} of the present convolutional layer l . Namely, this can happen if the output of layer l is added to the output of a residual connection to compute the input into $l+1$. At such a point in the chain of convolutional layers, the number of input filters F_{l+1}^{in} into layer $l+1$ depends on both the output filters of layer l *as well as* on the output filters of the residual connection.

More precisely, only those filters can be removed from the input into layer $l+1$ which are absent from *both* the output of layer l *as well as* from the output of the residual connection. (Note, thus, that the number F_{l+1}^{in} of input filters into layer $l+1$ *cannot* be determined by only knowing the number of output filters F_l^{out} and the number of output filters of the residual connection. Rather, the answer depends on *which* output filters have been pruned.) To determine F_{l+1}^{in} , two kinds of residual connections have to be distinguished:

- (a) **Residual connection is an identity skip connection.** In this case, the output filters of the residual connection are exactly the output filters of a previous layer: either the output filters of layer $l' := l-2$ (when the skip connection is in a “BasicBlock”) or of layer $l' := l-3$ (when the skip connection is in a “BottleneckBlock”). F_{l+1}^{in} thus equals the number of filters that are un-pruned in the output of both layer l' and un-pruned in the output of layer l .

(b) **Residual connection is a downsampling layer.** As we exclude downsampling layers from pruning (see Sec. 3), the number of output filters of a downsampling layer in the pruned network equals the number of outputs of the downsampling layer in the original network. Therefore, if the output of layer l is added to the output of a downsampling layer, then F_{l+1}^{in} in the pruned network takes the same value as in the original (un-pruned) network, i.e. F_{l+1}^{in} is the original number of input filters into layer $l + 1$.

The same reasoning and procedure applies to determining the number of input filters into any downsampling layer (i.e., this number is the same as the number of inputs into the convolutional layer that has the same input as the downsampling layer; note, a downsampling layer is a convolutional layer as well, but not part of the “chain of convolutional layers”), and also to determine the number of inputs (“input neurons”) into the final fully-connected layer (i.e., the number of “input neurons” into the last fully-connected layer is the same as would be the number of input filters into a convolutional layer at this stage).

Having determined the number of input and output filters (neurons) into all channels in this way, our parameter count sums up the parameter numbers for all convolutional layers (incl. downsampling layers), batchnorm layers, and fully-connected layers (incl. bias terms). This is the exact number of parameters needed to specify the pruned NN and also to build the pruned NN, since these parameters specify all surviving filters (and the fully-connected layers).

Our exact MAC count is similar, also based on the “true” F_l^{in} and F_l^{out} as just determined. The MAC count of a convolutional layer is $M_l = C_l \cdot S_l^w \cdot S_l^h$, where S_l^w and S_l^h are the numbers of width-wise and height-wise applications of each filter; for our networks, S_l^w equals the spatial picture width at layer l divided by the width-stride, and similarly for S_l^h .

Finally – and as mentioned above – we briefly describe the more straightforward but approximate counting method, that would yield pruning ratios that can appear quite a bit better. In this approximate way of counting, we take, within the chain of convolutional layers in the ResNet architecture, as input filters into layer $l + 1$ exactly the output filters from layer l , i.e. $F_{l+1}^{in} := F_l^{out}$. Consequently for any downsampling layer, we take as its number of input filters the number of output filters of its preceding convolutional layer, and as its number of output filters we take the number of input filters into the following convolutional layer (as determined by the previous sentence). We compute the number of parameters and MACs then according to the same formulas as in the exact method, but with potentially other values for F_l^{in} and F_l^{out} for all convolutional layers (incl. the downsampling layers, and the number of input nodes into the last fully-connected layer). Note, this approximate count is actually the exact count for a version of the pruned network where the size of the residual connections (identity skip connections and downsampling layers) has been adapted in a “natural” way to the sizes of the pruned convolutional layers in the chain of convolutional layers. In particular, this count can therefore be computed by just knowing the numbers of pruned filters in each convolutional layer, instead of knowing exactly which of the filters in each convolutional layer have been pruned.

It is apparent from the way of approximate counting just described that its parameter and MAC count will be smaller or equal to the exact count (described further above); both counts coincide for the original (un-pruned) network. For pruned networks, however, the difference in both counts can be substantial, esp. for the ResNet50 network (Table 2). For example, whereas the MAC-pruning-ratio of our SOSP-I(0.3) method is 15% (see Table 2), its MAC-pruning-ratio according to the approximate counting would equal 27%. In case that the competitor papers used this simpler approximate counting (which we do not know), we should also use this approximate counting to evaluate our method, which would thus appear more favorable, especially on the ImageNet experiments and especially on ResNet-50 (Table 2).

On a final note, we remark that the paper Tang et al. (2020) introducing the SCOP method, mentioned a discrepancy between the theoretically computed number of MACs and the experimentally measured value for this quantity, hinting at least at some inconsistency in the counting.