Appendix *Interspace Pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs*

A. Structure of the Appendix

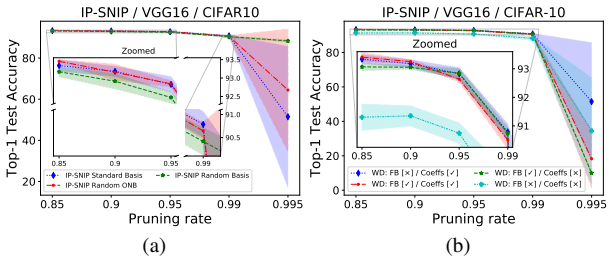
The Appendix is divided into the following Sections:

- A Describes the structural organization of the Appendix.
- B Contains ablation studies for IP methods which are not shown in the main body of the text.
- C Discussion about storing unstructured sparse networks.
- D Gives detailed information about the computations of FB-CNNs including backpropagation formulas. Especially, a comparison between the number of FLOPs required to evaluate and train a convolutional layer in the spatial and interspace representation is drawn. Finally, details on the real time measurements of sparse speed ups are given.
- E Computes transformation rules between the spatial and interspace coefficients, their gradients and Hessian matrices.
- F Here, the computations of the pruning scores used for experiments in the main body of the text are proposed.
- G Shows three different Algorithms to initialize FB-CNNs, including the standard initialization scheme used in the main body of the text. Further, details of implementations of the pruning methods are proposed.
- H Describes used training setups, hyperparameters, datasets and evaluation procedure for experiments in the main body of the text.
- I Presents network architectures, used in the experimental evaluation.
- J Concludes the Appendix with a mathematical proof of Theorem 1.

B. Additional ablations

B.1. Using different initializations for the interspace.

For SP-SNIP, the problem of vanishing gradients occurs, see Fig. A3(a). Filters which are spatially too sparse induce a vanishing gradient for high pruning rates. As shown in Fig. 1, IP leads to less zeros in the spatial representation of filters than SP *after* training. But, a pruned CNN has a spatially sparse topology *before* training if a standard initialization is used. This seems not to be the optimal initial situation for training FBs jointly with their coefficients. To analyze different starting conditions for IP, we initialized the interspace with standard, random `ONB` and `random` initializations. For details on these different initialization schemes, see Sec. G.



(a) (b)

Figure A1. Both: IP-SNIP for a VGG16 trained on CIFAR-10. (a) Standard, random `ONB` and `random` initialization are compared. (b) Weight decay applied [✓] / not applied [✗] on FBs and FB coefficients.

Experimental results can be seen in Fig. A1(a) for a VGG16 trained on CIFAR-10. For lower pruning rates, starting with $\mathcal{B}$ and a random ONB behaves similar. For high pruning rates, random ONBs are even better suited to be used. With them, the forward and backward dynamics of a pruned network are not impaired by spatially sparse filters at the beginning of training. Using non-orthonormal FBs leads to worse results than ONBs for lower pruning rates. Elements of a random basis are likely to be more similar to each other than those of ONBs. This redundancy worsens performance for lower pruning rates, but significantly improves results for higher sparsity.

B.2. Top-5 accuracy for PaI on ImageNet

Figures A2(a), (b) and (c) show the top-5 test accuracies for the PaI ImageNet experiment with a ResNet18. Using IP instead of SP again improves results significantly as already shown and discussed for top-1 test accuracies in Figs. 3b, 3d and 3f and Sec. 5.2, respectively. Similar to the top-1 accuracy, random PaI reaches better top-5 results than SNIP, GraSP and SynFlow.

B.3. Impact of weight decay.

Weight decay (WD) [A.15] reduces the network's capacity by shrinking parameters smoothly during training. Due to the bias-variance trade-off [A.9], WD can help to increase the network's generalization ability. To find the best way to combine WD and IP, we tested all combinations of WD turned on/off for FBs and their coefficients. For this purpose, we used IP-SNIP on VGG16 and CIFAR-10, see Fig. A1(b). For lower p, not using WD at all yields the worst performance whereas the best results are obtained by applying WD on both, FBs and FB coefficients. For higher p, applying WD on the FBs reduces the network's capacity too much. On average, using WD on the FB coefficients but not on the FBs themselves leads to the best results.

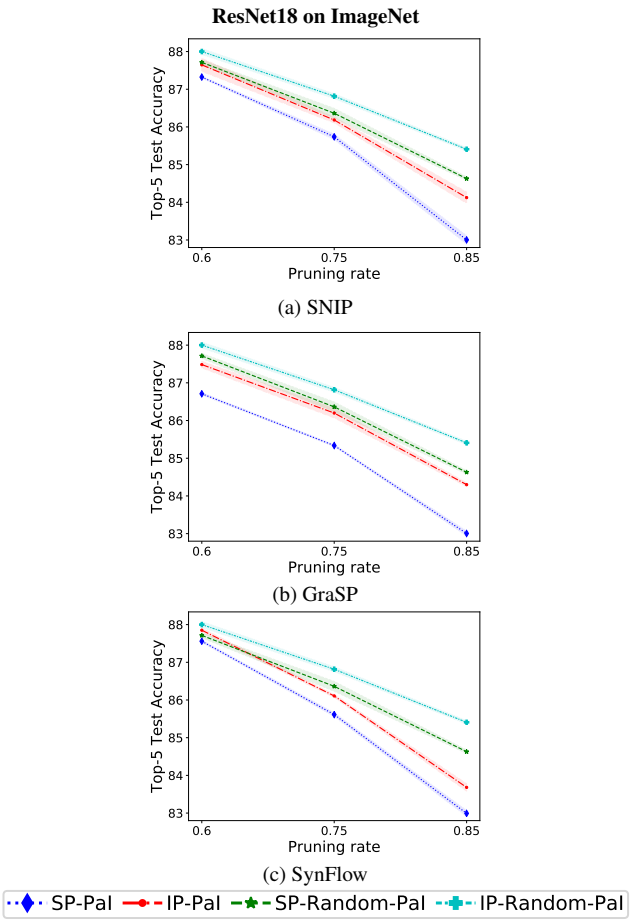


Figure A2. Comparison between top-5 test accuracies for SP on SNIP (a), GraSP (b) and SynFlow (c), and their adapted IP methods for a ResNet18 on ImageNet.

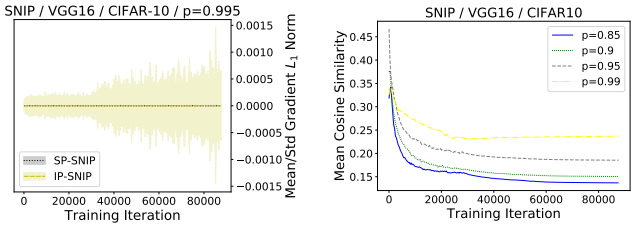


Figure A3. (a) Mean and std of gradient L_1 norm for IP- and SP-SNIP for $p = 0.995$. (b) The mean cosine similarity of all elements of the `coarse` FB for a VGG16 pruned with IP-SNIP and trained on CIFAR-10 is tracked over training for varying pruning rates.

B.4. Similarity of filter bases

In Fig. A3(b), the development of the cos similarity for the `coarse` FB $\mathcal{F}$ is tracked at training time for different pruning rates for IP-SNIP with a VGG16 trained on CIFAR-10. For $\mathcal{F}$, `random` initialization is used. The cos similarity

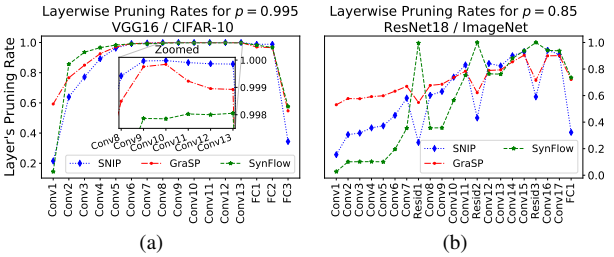


Figure A4. Layerwise pruning rates for PaI methods on (a) VGG16 on CIFAR-10 and (b) ResNet18 on ImageNet.

of $\mathcal{F}$ is the sum of all absolute values of cos similarities of distinct elements in $\mathcal{F}$, i.e.

$$\frac{2}{K^2 \cdot (K^2 - 1)} \sum_{j=1}^{K^2} \sum_{k=j+1}^{K^2} \frac{|\langle g^{(j)}, g^{(k)} \rangle|}{\|g^{(j)}\|_2 \cdot \|g^{(k)}\|_2} \,. \tag{A.1}$$

It therefore measures how similar two elements in $\mathcal{F}$ are on average. Figure A3(b) shows that the bases have approximately the same similarity at the beginning of training for all pruning rates. For lower pruning rates, the final similarity is much smaller than for higher ones. Therefore, we assume that increasing the FB to more than 9 filters for lower pruning rates might reduce the number of needed FB coefficients, as there is "enough space" left between the 9 basis filters. On the other hand, for high pruning rates we should be able to reduce the elements in the FB, since the basis elements tend to assimilate, *i.e.* "do not need the whole space". Experimental justifications of these assumptions are shown in Sec. B.6.

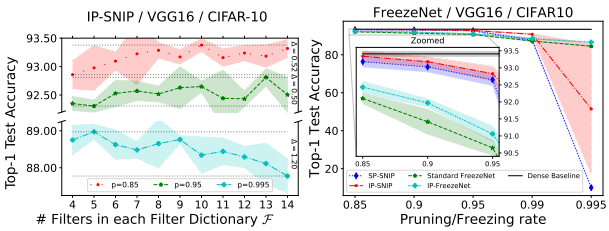
B.5. Layer wise pruning rates for PaI

As shown in Fig. A4(a), SNIP has the problem of pruning big layers too much. For the VGG16, convolutional layers 9 and 10 are pruned almost completely. This will lead to a vanishing gradient, see for example Fig. A3(a). With IP, the gradient flow can be increased, but if a layer is pruned completely, even an adaptive basis can not repair the damage.

SynFlow tends to fully prune 1×1 convolutional residual connections in ResNets. As shown in Fig. A4(b), all three residual connections are pruned completely. Consequently, IP- and SP-SynFlow show worse results than IP-/SP-SNIP and GraSP for ResNets, see for example Fig. 3f. Both, SNIP and GraSP prune residual connections even less than surrounding layers.

B.6. Generalizing filter bases

Up to now, we discussed experiments where FBs $\mathcal{F}$ formed bases. But, the spanning system $\mathcal{F} \subset \mathbb{R}^{K \times K}$ does not need to form a basis. The interspace can also be spanned by an overcomplete $\mathcal{F}$, *i.e.* $\#\mathcal{F} > K^2$ or an undercomplete $\mathcal{F}$ with $\#\mathcal{F} < K^2$. This leads to the more generalized



(a) (b)

Figure A5. (a): Different sizes of FDs for varying pruning rates p. (b): IP and SP versions of freezing parameters compared to pruning them. Frozen/pruned parameters are selected before training by the SNIP criterion.

formulation of *filter dictionaries* (FDs) which include all sizes of $\#\mathcal{F}$. Of course, a FB defines a FD with $\#\mathcal{F} = K^2$ elements which are additionally assumed to be linearly independent.

As discussed in Sec. D, undercomplete FDs can be used to reduce the number of computations needed for a 2D FB convolution. However, overcomplete FDs might lead to representations of filters needing less coefficients, see [A.28, A.5, A.3]. It is not clear which elements of a basis $\mathcal{B}$ should be removed to obtain an undercomplete FD, or added for overcomplete ones. Thus, we initialized all elements of the FDs randomly in this experiment.

A VGG16 contains 3×3 filters, thus a FB has 9 filters. Figure A5(a) shows IP-SNIP for a VGG16 trained on CIFAR-10. Reported results are those with the best validation accuracy from `coarse`, `medium` and `fine` FD sharing. Here, p measures the pruning rate for IP with a FB, *i.e.* $\mathcal{F} = 9$. For $\#\mathcal{F} \neq 9$, the number of non-zero FD coefficients is equal to $\#\mathcal{F} = 9$. Thus, the representation of a filter in the interspace spanned by its dictionary is more sparse if $\#\mathcal{F} > 9$ and less sparse if $\#\mathcal{F} < 9$ compared to $\#\mathcal{F} = 9$.

More than 9 elements in a FD improve results if coefficients are not too sparse, *e.g.* $\#\mathcal{F} = 10$ for $p = 0.85$ or $\#\mathcal{F} = 13$ for $p = 0.95$. Using more sophisticated methods to determine initial FDs might help to exploit overcomplete FDs better. Since $\#\mathcal{F} > 9$ increases the sparsity of FD coefficients, the performance for high pruning rates drops drastically for overcomplete FDs compared to bases.

If undercomplete FDs are used, performance worsens for lower pruning rates. Here, the capacity of the network is too low as the interspace is only $\#\mathcal{F}$ dimensional. Due to only few non-zero FB coefficients, this is not a limiting factor for high pruning rates anymore. The reduced dimensionality of the interspace even increases performance compared to $\#\mathcal{F} \geq 9$. A reason for this might be the increased information flow induced by a denser structure of the interspace. The best result for $p = 0.995$, with test accuracy 88.9%, is achieved with $\#\mathcal{F} = 5$. In comparison, SP-SNIP has 10.0% test accuracy for the same number of non-zero parameters.

Pruning rate	Size in kB	$\frac{\text{sparse size}}{\text{dense size}} \cdot 100\%$	Sparse & mask
Dense training	53,256	—	—
0.2	60,765	114.1	82.3
0.35	49,345	92.7	67.9
0.6	30,446	57.2	43.0
0.85	11,461	21.5	16.9
0.995	410	0.8	0.6

Table A1. Compression for the PaI method IP-SNIP after training a VGG16 on CIFAR-10. All stored network parameters are in full precision, *i.e.* 32bit floating points. Sparse networks are stored in the CSR format whereas the dense one is stored raw. Moreover, dense and sparse networks are compressed by using `numpy.savez_compressed`. *Sparse and mask* denotes the percentage of the theoretically needed memory if only sparse parameters are stored together with the entropy encoded pruning mask.

B.7. Freezing coefficients

FreezeNet [A.30] is closely related to pruning before training via SNIP [A.16]. FreezeNet trains the same parameters as SNIP but *freezes* the un-trained coefficients during training instead of pruning them. By using pseudo random initializations for the network, the frozen coefficients do not have to be stored after training but can be recovered with the used random seed. By always guaranteeing a strong gradient signal, FreezeNet outperforms SNIP significantly for low numbers of trained parameters as shown in Fig. A5(b). The opposite is true if more parameters are trained.

We further compare freezing of spatial coefficients, *standard FreezeNet*, and freezing interspace coefficients, *IP-FreezeNet*. Using adaptive FBs instead of freezing the spatial coefficients again significantly improves performance. Thus, improvements induced by interspace representations are not limited to pruning but also hold for other dimensionality reductions like freezing parts of a CNN during training.

C. Storing unstructured sparse networks

Storing sparse parameters in formats such as the *compressed sparse row format* (CSR) [A.27] creates additional overhead. The CSR format stores all non-zero elements of a matrix together with an array that contains the column indices and an additional array with the number of elements in each row. Therefore, additional parameters have to be stored for each non-zero element to determine the corresponding column- and row index. However, the two additional arrays do not need to be stored in 32bit full precision, but only as integers. The CSR format can be used for efficiently computing sparse matrix vector products which we also used for determining the sparse speed up for IP and SP, see Sec. D.4.

We empirically tested the overhead for real memory costs

of sparse networks stored in the CSR format, see Tab. A1. Note, IP or SP pruned networks have, up to some insignificant differences, equal memory costs in practice and theory. Therefore, we report IP pruned networks in Tab. A1. Training 0.5% of all parameters compressed the network to 0.8% of the dense network's size for IP-SNIP with a VGG16 [A.24] trained on CIFAR-10. Of course, for such a small number of non-zero elements, the overhead of the CSR format is also quite small. For pruning 85% of the parameters, 21.8% of the dense memory is needed. As can be seen, additional index memory for sparse row formats increases with a decreasing pruning rate. Thus, for $p \leq 0.5$ CSR will not lead to good compression results and finally even lead to a higher memory requirement than storing the network in a dense format. As shown in Fig. A6(a), using the CSR format for such low pruning rates does not significantly speed up the network inference.

Therefore, other formats for storing the sparse network can be used for lower pruning rates. By storing the pruning mask via entropy encoding, *e.g.* [A.31], at most 1bit is needed for each mask parameter. To be exact, storing the network's pruning mask for a pruning rate $p \in (0, 1)$ ideally needs

$$1 \geq S = -p \cdot \log_2 p - (1-p) \cdot \log_2(1-p) \text{ bits} \quad \text{(A.2)}$$

for each element in the mask. If the mask is known, only the non-zero parameters have to be stored in the right order and in full precision. Thus, storing the sparse network of total size d with pruning rate p needs, in the ideal case, $d \cdot (S + (1-p) \cdot 32)$ bits, compared to $d \cdot 32$ bits for the dense network. In total, using entropy encoding for the pruning mask compresses the sparse network to $S/32 + (1-p)$ of its original size. As shown in Tab. A1, storing the pruning mask together with the non-zero coefficients is cheaper than CSR for all pruning rates.

D. Comparing computational costs for convolutions with spatial and interspace representations

For simplicity we will do the analysis with a FB $\mathcal{F}$ consisting of K^2 elements in the following. But it is straight forward to do similar computations with an arbitrary FD $\mathcal{F}$ of size N.[A.1] As a results, all computational costs for the interspace setting are multiplied by a factor N/K^2 to get the costs for the arbitrary FD case.[A.2] This shows that computations for IP are more expensive if an overcomplete FD with $\#\mathcal{F} > K^2$ is used. On the other hand, by reducing the size of a FD, the computations can be sped up.

[A.1] Summing from 1 to N instead of K^2 or doing needed computations N times instead of K^2 times.

[A.2] Except the costs for computing $\frac{\partial \mathcal{L}}{\partial h}$ needed to update $\mathcal{F}$ which are equal for all sizes of $\mathcal{F}$.

In this Section, we determine the number of FLOPs needed to evaluate a standard 2D convolutional layer and a FB 2D convolutional layer. We use FLOPs as a measure since they are easy to determine and replicable in a mathematical framework but can also be measured in real time applications. A FLOP corresponds to either a multiplication or a summation.

For the forward pass, we show that the number of required FLOPs is increased by a small, constant amount for FB-CNNs compared to standard CNNs for all pruning rates. Since the FB formulation can easily be converted to a standard representation, dense FB-CNNs therefore could be transformed into standard CNNs after training. If a CNN is pruned, this transformation is not advisable since it usually destroys the sparsity of the network.

In the backward pass, similar results hold. Moreover, we need to compute the gradient of the FB which of course requires additional resources in the IP setting.

In the following, we will assume the convolutions to have quadratic $K \times K$ kernels as well as no zero padding, stride 1×1 and dilation 1×1.

D.1. Computations in the forward pass

Standard convolution.

Let $h = (h^{(\alpha,\beta)})_{\alpha,\beta} \in \mathbb{R}^{c_{out} \times c_{in} \times K \times K}$ denote a convolutional layer of a CNN. Furthermore, let $X = (X^{(\beta)})_\beta \in \mathbb{R}^{c_{in} \times h \times w}$ be the input feature map of the corresponding layer. In the following, we determine the number of FLOPs needed to evaluate this layer. In order to do so, we first analyze the costs for one cross-correlation $\star$, used in practice to compute 2D convolutional layer [A.1, A.21], *i.e.*

$$h^{(\alpha,\beta)} \star X^{(\beta)} = \left((h^{(\alpha,\beta)} \star X^{(\beta)})_{i,j} \right)_{i,j} \quad \text{(A.3)}$$

$$= \left(\sum_{m,n=1}^{K} h^{(\alpha,\beta)}_{m,n} \cdot X^{(\beta)}_{m+i,n+j} \right)_{i,j} \in \mathbb{R}^{d_1 \times d_2}, \quad \text{(A.4)}$$

with $d_1 := h + 1 - K$ and $d_2 := w + 1 - K$, the dimensions of the output. Equation (A.4) shows that the cost for one cross-correlation is given by $2 \cdot K^2 \cdot d_1 \cdot d_2$ FLOPs. The output of a 2D convolutional layer is given by

$$Y = \left(Y^{(\alpha)} \right)_\alpha = \left(\sum_{\beta=1}^{c_{in}} h^{(\alpha,\beta)} \star X^{(\beta)} \right)_\alpha \in \mathbb{R}^{c_{out} \times d_1 \times d_2}, \quad \text{(A.5)}$$

which finally leads to $c_{out} \cdot c_{in}$ times the costs to compute a single cross-correlation Eq. (A.4). Therefore, $2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot d_1 \cdot d_2$ FLOPs are needed in total to compute a standard 2D convolutional layer.

FB convolution

Let $h = (h^{(\alpha,\beta)})_{\alpha,\beta} = (\sum_n \lambda_n^{(\alpha,\beta)} \cdot g^{(n)})_{\alpha,\beta} \in \mathbb{R}^{c_{out}\times c_{in}\times K\times K}$ be the interspace representation of h, where the FB is given by $\mathcal{F} = \{g^{(1)}, \dots, g^{(K^2)}\} \subset \mathbb{R}^{K\times K}$. The 2D convolution of this layer with input $X \in \mathbb{R}^{c_{in}\times h\times w}$ can be computed via

$$Y = \left(Y^{(\alpha)}\right)_\alpha = \left(\sum_{\beta=1}^{c_{in}} \sum_{n=1}^{K^2} \lambda_n^{(\alpha,\beta)} \cdot \left(g^{(n)} \star X^{(\beta)}\right)\right)_\alpha . \tag{A.6}$$

Using the last equation in Eq. (A.6), we see that $g^{(n)} \star X^{(\beta)}$ has to be computed once for each combination of β and n, *i.e.* $c_{in} \cdot K^2$ many times. The costs for computing all $g^{(n)} \star X^{(\beta)}$ is therefore given by $2 \cdot c_{in} \cdot K^4 \cdot d_1 \cdot d_2$ FLOPs. For each combination of α, β and n, $g^{(n)} \star X^{(\beta)}$ has to be multiplied by the scalar $\lambda_n^{(\alpha,\beta)}$. These are $d_1 \cdot d_2$ many FLOPs for each α, β and n. Summing over β and n yields another $c_{out} \cdot c_{in} \cdot K^2 \cdot d_1 \cdot d_2$ FLOPs in total. Thus, the total costs for computing a FB 2D convolutional layer is given by $2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot d_1 \cdot d_2 + 2 \cdot c_{in} \cdot K^4 \cdot d_1 \cdot d_2$ FLOPs.

By using FB convolutions, the numbers of needed FLOPs is therefore slightly increased by $2 \cdot c_{in} \cdot K^4 \cdot d_1 \cdot d_2$. Which is a relative increase of $K^2/c_{out} \cdot 100\%$ compared to the standard case.

Pruned networks

In the following we assume the convolutional layer $h \in \mathbb{R}^{c_{out}\times c_{in}\times K\times K}$ to be pruned with a pruning rate of $p \in [0, 1]$.[A.3] We suppose all zero coefficients to be known. Thus, the corresponding multiplications do not have to be computed in Eqs. (A.4) and (A.6).

The required number of computations for a standard 2D convolutional layer with pruning rate p is therefore given by

$$2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot d_1 \cdot d_2 \cdot (1 - p) \text{ FLOPs} . \tag{A.7}$$

For a pruned FB 2D convolutional layer,

$$2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot d_1 \cdot d_2 \cdot (1-p) + 2 \cdot c_{in} \cdot K^4 \cdot d_1 \cdot d_2 \text{ FLOPs} \tag{A.8}$$

are needed for evaluation.

The number of FLOPs for IP is increased for all pruning rates by $2 \cdot c_{in} \cdot K^4 \cdot d_1 \cdot d_2$ compared to SP. These are exactly the costs for computing all combinations of $g^{(n)} \star X^{(\beta)}$, needed for the forward pass for FB 2D Convolutions. These costs are independent of the pruning rate and therefore a constant overhead of IP compared to SP. Thus the additional costs for IP in the forward pass compared to SP are K^2/c_{out} times the costs of the dense forward pass.

[A.3] For simplicity, we assume the number of non-zero coefficients for IP and SP to be equal here. Due to extra FB parameters, the number of non-zero interspace coefficients is always slightly smaller than for the standard case in our experiments.

D.2. Backward Pass

Up to now, we have computed additional FLOP costs for IP compared to SP in the forward pass. Now we want to have a closer look at the backward pass. We note that $\frac{\partial \mathcal{L}}{\partial Y}$ always has the same cost for the standard- and the FB 2D convolution layer. This holds since $\hat{X} = \sigma(Y)$ for some activation function σ and consequently $\frac{\partial \mathcal{L}}{\partial Y} = \frac{\partial \mathcal{L}}{\partial \hat{X}} \odot \sigma'(Y)$.

Computing the gradient for X.

Furthermore, it is known that

$$\frac{\partial \mathcal{L}}{\partial X^{(\beta)}} = \sum_{\alpha=1}^{c_{out}} h^{(\alpha,\beta)} \hat{\star} \frac{\partial \mathcal{L}}{Y^{(\alpha)}} \tag{A.9}$$

with a strided convolution $\hat{\star}$ that corresponds to the forward pass and which needs $2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot h \cdot w$ FLOPs. By representing $h^{(\alpha,\beta)} = \sum_{n=1}^{K^2} \lambda_n^{(\alpha,\beta)} \cdot g^{(n)}$ and using the linearity of $\hat{\star}$, we now get the computational overhead of $2 \cdot c_{out} \cdot K^4 \cdot h \cdot w$ FLOPs which are the costs for computing

$$g^{(n)} \hat{\star} \frac{\partial \mathcal{L}}{Y^{(\alpha)}} \tag{A.10}$$

for all $n \in \{1, \dots, K^2\}$ and $\alpha \in \{1, \dots, c_{out}\}$. This results in an overhead of K^2/c_{in} compared to the costs of the standard, dense network.

In the sparse case, again the FLOP costs for SP are decreased by a factor $(1-p)$. Furthermore, the overhead K^2/c_{in} is constant since the $g^{(n)}$ are not pruned. Similar formulas to Eqs. (A.7) and (A.8) hold also in the backpropagation case which results in a constant overhead of IP compared to SP for computing $\frac{\partial \mathcal{L}}{\partial X}$ equal to K^2/c_{in} times the costs of the dense computation of $\frac{\partial \mathcal{L}}{\partial X}$.

Gradients for coefficients.

The backpropagation formulas for the spatial and FB coefficients are given by

$$\frac{\partial \mathcal{L}}{\partial h_{i,j}^{(\alpha,\beta)}} = \left(\frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}} \star X^{(\beta)}\right)_{i,j} \tag{A.11}$$

and

$$\frac{\partial \mathcal{L}}{\partial \lambda_n^{(\alpha,\beta)}} = \left\langle \frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}}, g^{(n)} \star X^{(\beta)} \right\rangle , \tag{A.12}$$

respectively. Since $g^{(n)} \star X^{(\beta)}$ is already computed in the forward pass, both computations for the standard case and the FB representation have equal FLOP costs. In total, this equals to $2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot d_1 \cdot d_2$ FLOPs for computing $\frac{\partial \mathcal{L}}{\partial h}$ or $\frac{\partial \mathcal{L}}{\partial \lambda}$. If pruning is applied, this reduces to $2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot d_1 \cdot d_2 \cdot (1 - p)$ FLOPs for IP and SP, since gradients for pruned coefficients do not need to be computed.

Note, if the size $d_1 \cdot d_2$ of $Y \in \mathbb{R}^{c_{out} \times d_1 \times d_2}$ is bigger than the kernel size K^2 it is even cheaper to compute the gradient of $\lambda_n^{(\alpha,\beta)}$ via

$$\frac{\partial \mathcal{L}}{\partial \lambda_n^{(\alpha,\beta)}} = \left\langle g^{(n)}, \frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}} \star X^{(\beta)} \right\rangle . \tag{A.13}$$

In Eq. (A.12) there are $2 \cdot d_1 \cdot d_2$ FLOPs needed (if $g^{(n)} \star X^{(\beta)}$ is known which we can assume due to the forward pass) whereas Eq. (A.13) needs $2 \cdot K^2$ FLOPs *if* $\frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}} \star X^{(\beta)}$ is known. As we will see in the following, $g^{(n)}$ needs $\frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}} \star X^{(\beta)}$ to be computed for all α, β and consequently we can assume them to be known. In summary we can say that the computation of the interspace coefficients λ requires the same number of FLOPs, or even less, compared to the spatial coefficients.

Gradient for the filter base.

The computation of the gradients $\frac{\partial \mathcal{L}}{\partial g^{(n)}}$ also generates extra costs for the backward pass of training interspace representations. It holds

$$\frac{\partial \mathcal{L}}{\partial g^{(n)}} = \sum_{\alpha=1}^{c_{out}} \sum_{\beta=1}^{c_{in}} \lambda_n^{(\alpha,\beta)} \cdot \left(\frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}} \star X^{(\beta)} \right) . \tag{A.14}$$

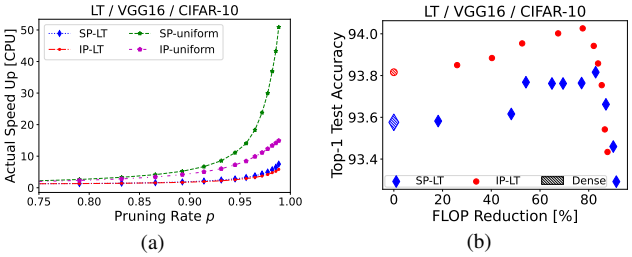
As shown in Eq. (A.14), $\frac{\partial \mathcal{L}}{\partial g^{(n)}}$ first needs to compute all $\frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}} \star X^{(\beta)} = \frac{\partial \mathcal{L}}{\partial h^{(\alpha,\beta)}}$. This is exactly the cost for computing the dense gradient $\frac{\partial \mathcal{L}}{\partial h}$ which needs $2 \cdot c_{in} \cdot c_{out} \cdot K^2 \cdot d_1 \cdot d_2$ FLOPs. The scaling and summation in the sum Eq. (A.14) requires $2 \cdot c_{in} \cdot c_{out} \cdot K^2$ FLOPs in total. If pruning is applied, this reduces to $2 \cdot c_{in} \cdot c_{out} \cdot K^2 \cdot (1-p)$. Altogether, computing the gradients of $g^{(1)}, \ldots, g^{(K^2)}$ needs $2 \cdot c_{out} \cdot c_{in} \cdot K^2 \cdot (d_1 \cdot d_2 + (1-p) \cdot K^2)$ FLOPs. In simple terms, the total computation of $\frac{\partial \mathcal{L}}{\partial g}$ lies in $\mathcal{O}(costs(\frac{\partial \mathcal{L}}{\partial h}))$.

Summary for the backward pass.

In summary, the computation of $\frac{\partial \mathcal{L}}{\partial X}$ of IP induces a constant overhead compared to IP. This corresponds to K^2/c_{in} times the costs of computing the dense gradient of $\frac{\partial \mathcal{L}}{\partial X}$ by using spatial coefficients. On top of that, IP also needs to compute the gradient for the FB $\mathcal{F}$ which is in $\mathcal{O}(costs(\frac{\partial \mathcal{L}}{\partial h}))$.

D.3. Upper bounds for gradients.

As Eq. (A.13) and Eq. (A.14) show, jointly optimizing $\mathcal{F}$ and λ leads to non trivial correlations between them. With a slight abuse of notation we assume for the following discussion $\mathcal{F}$ to be the $K^2 \times K^2$ matrix containing all flattened $g^{(n)}$. Further, let $h, \lambda \in \mathbb{R}^{K^2 \times c_{out} c_{in}}$ contain all spatial and interspace coefficients of the layer, respectively. Therefore, it holds $h = \mathcal{F} \cdot \lambda$. By using $\frac{\partial \mathcal{L}}{\partial h^{(\alpha,\beta)}} = \frac{\partial \mathcal{L}}{\partial Y^{(\alpha)}} \star X^{(\beta)}$ and



(a) (b)

Figure A6. (a) Speed up on CPU for varying p for SP and IP for uniform sparsity and LT's sparsity. (b) Top-1 test accuracy over FLOPs reduction for IP- and SP-LT.

the Cauchy-Schwartz inequality, the gradients for $\mathcal{F}$ and λ are bounded by

$$\left\| \frac{\partial \mathcal{L}}{\partial \lambda} \right\|_F \leq \|\mathcal{F}\|_F \left\| \frac{\partial \mathcal{L}}{\partial h} \right\|_F \quad \text{and} \quad \left\| \frac{\partial \mathcal{L}}{\partial \mathcal{F}} \right\|_F \leq \|\lambda\|_F \left\| \frac{\partial \mathcal{L}}{\partial h} \right\|_F . \tag{A.15}$$

This shows that upper bounds for $\frac{\partial \mathcal{L}}{\partial \mathcal{F}}$ and $\frac{\partial \mathcal{L}}{\partial \lambda}$ are determined by the spatial gradient $\frac{\partial \mathcal{L}}{\partial h}$. This boundedness of the gradients leads to stable convergence for both, $\mathcal{F}$ and λ, while the convergence behavior of λ is similar to the standard coefficients h, see Fig. 5b. Moreover, Fig. A3(a) even shows that adaptive FBs help to overcome vanishing gradients for SNIP by becoming spatially dense. IP-SNIP can use that to recover during training from a complete, PaI induced information loss, while SP-SNIP is stuck with zero gradient flow.

D.4. Real runtime measurements

To measure and compare the real runtime accelerations of IP and SP for inference, we used `scipy`'s `sparse` package. To be precise, we used `scipy.sparse.csr_matrix`, see online documentation. As discussed in Sec. D.1, sparse FB convolutions can be computed by first convolving all $X^{(\beta)}$ with all $g^{(n)}$. Afterwards, sparse matrix multiplications can be used to compute the actual output Y. To rule out runtime differences induced by mismatches between sparse implementations of matrix multiplications and convolutions, we simulated sparse convolutions with sparse matrix multiplications of matching dimensions. Therefore, a sparse convolution $h \star X$ with $h \in \mathbb{R}^{c_{out} \times c_{in} \times K \times K}$ and $X \in \mathbb{R}^{c_{in} \times H \times W}$ corresponds to a matrix-vector multiplication $\hat{h} \cdot \hat{X}$ with $\hat{h} \in \mathbb{R}^{c_{out} \cdot H \cdot W \times c_{in} \cdot K^2}$ and $\hat{X} \in \mathbb{R}^{c_{in} \cdot K^2}$.

We measured the runtime of a VGG16 on input images $X \in \mathbb{R}^{3 \times 32 \times 32}$ (*i.e.* CIFAR-10) with two different sparsity configurations, the sparsity distribution found by pruning with LT, see Sec. 5.3, and uniform sparsity for each layer. For simplicity, we omit the batch normalization layers and non-linearities. Runtime is measured on one core of an Intel XEON E5-2680 v4 2.4 GHz CPU where we used batch size 1 and the mean runtime of 25 runs.

Figure A6(a) shows the comparison between model spar-

sity and the actual runtime speed up on a CPU. Since the used CSR [A.27] format for sparse coefficients adds additional overhead to the actually executed computations, runtime is sped up significantly only for $p \geq 0.75$. Note, different sparsity distributions can lead to varying accelerations for a similar global pruning rate p. IP indeed has a longer runtime due to the mentioned extra computations. But by boosting performance of sparse models, IP reaches similar results than dense training with 5.2 times speed up and better results than SP for similar runtime, as shown in Fig. 5a.

For a comparison between actual and theoretical accelerations of IP and SP, the top-1 accuracy over the theoretical FLOPs reduction for the LT experiment is given in Fig. A6(b). Results for actual sparse speed ups can be seen in Fig. 5a in the main part of the paper.

E. Transformation rules in the interspace

Since the interspace representation is obtained by a linear transformation of the standard, spatial representation, we will derive formulas for this transformation. By knowing them, it will be straight forward to also determine transformation rules for the corresponding gradients and higher derivatives. Those transformation rules might be useful if pruning methods that need first or second order information are used. We test three methods in our work that need the information of the gradient, SNIP [A.16], GraSP [A.29] and SynFlow [A.26]. Moreover, GraSP needs second order information as well. Since all these methods are applied at initialization and we use an initialization equivalent to the standard network in the main part of this work, the gradient and Hessian are equivalent at that time. Still, if such pruning methods are applied with different initializations, the knowledge of these transformation rules might be helpful to overcome scaling problems. Furthermore, we believe the transformation rules to be fruitful for analyzing the information flow in FB-CNNs which we think is an interesting direction for future work.

Again we will assume the special case considered in the paper, *i.e.* $\mathcal{F}$ forming a basis.

E.1. Transformation rules for filters

For a given layer in a CNN, let α denote the output channel, β the corresponding input channel and $K \times K$ be the kernel size of a filter $h^{(\alpha,\beta)}$. For the layer's FB $\mathcal{F} = \{g^{(1)}, \dots, g^{(K^2)}\}$, the filter's interspace representation is given by

$$h^{(\alpha,\beta)} = \sum_{n=1}^{K^2} \lambda_n^{(\alpha,\beta)} \cdot g^{(n)} \ . \tag{A.16}$$

Here, the FB coefficients of $h^{(\alpha,\beta)}$ are given by $\lambda^{(\alpha,\beta)} = (\lambda_n^{(\alpha,\beta)})_n \in \mathbb{R}^{K^2}$. Let $\mathcal{B}$ be the standard basis for $\mathbb{R}^{K\times K}$. Then, the spatial representation of filter $h^{(\alpha,\beta)}$ is given by

$$h^{(\alpha,\beta)} = \sum_{n=1}^{K^2} h_{i_n,j_n}^{(\alpha,\beta)} \cdot e^{(n)} = \sum_{n=1}^{K^2} \varphi_n^{(\alpha,\beta)} \cdot e^{(n)} \ , \tag{A.17}$$

with spatial coefficients $\varphi^{(\alpha,\beta)} := (\varphi_n^{(\alpha,\beta)})_n = (\langle h^{(\alpha,\beta)}, e^{(n)} \rangle)_n \in \mathbb{R}^{K^2}$ and standard basis $\mathcal{B}$ given by $\mathcal{B} = \{e^{(1)}, \dots, e^{(K^2)}\}$ and

$$e_{i,j}^{(n)} = \delta_{i,i_n} \cdot \delta_{j,j_n} \ , (i_n, j_n) \in \{1, \dots, K\}^2 \ , \\ (i_n, j_n) \neq (i_m, j_m) \text{ for } n \neq m \ . \tag{A.18}$$

Consequently,

$$\varphi^{(\alpha,\beta)} = \Psi \cdot \lambda^{(\alpha,\beta)} \ , \ \Psi = \left(\langle g^{(m)}, e^{(n)} \rangle \right)_{n,m} \in \mathbb{R}^{K^2 \times K^2} \tag{A.19}$$

holds. Note, since FBs are shared for at least one layer, the basis transformation matrix Ψ is not labeled with the input- and output channels α and β, respectively. But of course, formulas can be adapted to the case of more than one FB per layer. Since we assume $\mathcal{F}$ to form a basis, the reverse is given by

$$\lambda^{(\alpha,\beta)} = \Psi^{-1} \cdot \varphi^{(\alpha,\beta)} \ . \tag{A.20}$$

Note, if we use $\mathcal{F}$ as a general dictionary, and not a basis anymore, a reverse can still be computed by the Moore-Penrose pseudo inverse $\Psi^\dagger = \Psi^T \cdot (\Psi \cdot \Psi^T)^{-1}$ if $\mathcal{F}$ forms a *generating system* for $\mathbb{R}^{K\times K}$. If $\mathcal{F}$ forms a linear independent, undercomplete dictionary, we can express $\lambda^{(\alpha,\beta)} = \hat{\Psi} \varphi^{(\alpha,\beta)}$ for a suitable $\hat{\Psi} \in \mathbb{R}^{\#\mathcal{F} \times K^2}$. A reverse of this is given by $\varphi^{(\alpha,\beta)} = \hat{\Psi}^\dagger \lambda^{(\alpha,\beta)}$, where again $\hat{\Psi}^\dagger = \hat{\Psi}^T \cdot (\hat{\Psi} \cdot \hat{\Psi}^T)^{-1}$ forms the Moore-Penrose pseudo inverse.

E.2. Transformation rules for gradients

Let $\mathcal{L}$ denote the loss function used to train the CNN. Assuming $h^{(\alpha,\beta)}$ to be given in the interspace representation Eq. (A.16),

$$\frac{\partial \mathcal{L}}{\partial \lambda^{(\alpha,\beta)}} = \left(\frac{\partial \varphi^{(\alpha,\beta)}}{\partial \lambda^{(\alpha,\beta)}} \right)^T \cdot \frac{\partial \mathcal{L}}{\partial \varphi^{(\alpha,\beta)}} = \Psi^T \cdot \frac{\partial \mathcal{L}}{\partial \varphi^{(\alpha,\beta)}} \tag{A.21}$$

holds by the chain rule and Eq. (A.19). Consequently,

$$\frac{\partial \mathcal{L}}{\partial \varphi^{(\alpha,\beta)}} = (\Psi^{-1})^T \cdot \frac{\partial \mathcal{L}}{\partial \lambda^{(\alpha,\beta)}} \tag{A.22}$$

is true for the gradient as well.

By comparing Eq. (A.20) with Eq. (A.21) (or Eq. (A.19) with Eq. (A.22)) we see that the coefficients and their corresponding gradients transform complementary to each other if $\Psi^T \neq \Psi^{-1}$. Note, $\Psi^{-1} = \Psi^T$ if and only if Ψ is orthogonal which is equivalent to $\mathcal{F}$ forming an orthonormal basis (ONB).

E.3. Transformation rules for Hessian

In order to compute the Hessian of the loss function, we have to index all possible filters in a CNN. Let $\lambda^{(\alpha,\beta;l)}$ and $\varphi^{(\alpha,\beta;l)}$ denote the interspace and spatial coefficients of a filter in layer l corresponding to input channel β and output channel α. Here, the basis transformation in layer l is given by $\Psi^{(l)}$. If a FB is shared for layers l and $l+1$, then $\Psi^{(l)} = \Psi^{(l+1)}$ would hold. Furthermore, $K_{(l)}$ is the filter size in layer l and $c_{out}^{(l)}$ and $c_{in}^{(l)}$ denote the number of output and input channels, respectively. We assume the CNN to have L_c convolutional layers in total. Let $H^{\mathcal{B}}$ be the Hessian matrix of $\mathcal{L}$ w.r.t. to coefficients of $\mathcal{B}$. The corresponding values of the Hessian are given by

$$H^{\mathcal{B}}((\alpha,\beta,n;l),(\alpha',\beta',n';l')) = \frac{\partial^2 \mathcal{L}}{\partial \varphi_n^{(\alpha,\beta;l)} \partial \varphi_{n'}^{(\alpha',\beta';l')}} \,. \tag{A.23}$$

Equivalently, the Hessian w.r.t. $\mathcal{F}$ is given by $H^{\mathcal{F}}$ with values

$$H^{\mathcal{F}}((\alpha,\beta,n;l),(\alpha',\beta',n';l')) = \frac{\partial^2 \mathcal{L}}{\partial \lambda_n^{(\alpha,\beta;l)} \partial \lambda_{n'}^{(\alpha',\beta';l')}} \,. \tag{A.24}$$

Using multi-index notation, we can describe the transformation of the Hessian matrix compactly. For $\mathbf{m} := (\alpha,\beta,n;l)$ and $\mathbf{m}' := (\alpha',\beta',n';l')$, define

$$\varphi_{\mathbf{m}} := \varphi_n^{(\alpha,\beta;l)} \,,\ \varphi := (\varphi_{\mathbf{m}})_{\mathbf{m}\in\mathcal{M}} \tag{A.25}$$

$$\lambda_{\mathbf{m}} := \lambda_n^{(\alpha,\beta;l)} \,,\ \lambda := (\lambda_{\mathbf{m}})_{\mathbf{m}\in\mathcal{M}} \tag{A.26}$$

$$\Psi_{\mathbf{m},\mathbf{m}'} := \delta_{\alpha,\alpha'} \cdot \delta_{\beta,\beta'} \cdot \delta_{l,l'} \cdot \Psi_{n,n'}^{(l)} \,, \tag{A.27}$$

$$\Psi := (\Psi_{\mathbf{m},\mathbf{m}'})_{\mathbf{m},\mathbf{m}'\in\mathcal{M}} \,, \tag{A.28}$$

where all possible multi-indices are given by

$$\mathcal{M} := \bigcup_{l=1}^{L_c} \bigcup_{\alpha=1}^{c_{out}^{(l)}} \bigcup_{\beta=1}^{c_{in}^{(l)}} \bigcup_{n=1}^{K_{(l)}^2} \{(\alpha,\beta,n;l)\} \,. \tag{A.29}$$

Let $d := \#\mathcal{M}$ be the dimension of the CNN. For all matrices/vectors $A \in \mathbb{R}^{d_1 \times d}$ and $B \in \mathbb{R}^{d \times d_2}$ with $d_1, d_2 \in \{1, d\}$, indexed with multi-indices, we define multi-index multiplication via

$$(A \cdot B)_{\mathbf{m},\mathbf{m}'} := \sum_{\mathbf{m}''\in\mathcal{M}} A_{\mathbf{m},\mathbf{m}''} \cdot B_{\mathbf{m}'',\mathbf{m}'} \,. \tag{A.30}$$

Using the multi-index notation, together with Eqs. (A.25), (A.26) and (A.28), leads to simple transformations of coefficients and their gradients for the whole CNN, given by

$$\varphi = \Psi \cdot \lambda \,, \qquad \frac{\partial \mathcal{L}}{\partial \varphi} = (\Psi^{-1})^T \cdot \frac{\partial \mathcal{L}}{\partial \lambda} \tag{A.31}$$

with the transpose of a multi-index matrix A defined via

$$A^T_{\mathbf{m},\mathbf{m}'} := A_{\mathbf{m}',\mathbf{m}} \tag{A.32}$$

and

$$\Psi^{-1} := (\Psi^{-1}_{\mathbf{m},\mathbf{m}'})_{\mathbf{m},\mathbf{m}'\in\mathcal{M}} \tag{A.33}$$

$$\Psi^{-1}_{\mathbf{m},\mathbf{m}'} := \delta_{\alpha,\alpha'} \cdot \delta_{\beta,\beta'} \cdot \delta_{l,l'} \cdot (\Psi^{(l)})^{-1}_{n,n'} \,. \tag{A.34}$$

It holds

$$H^{\mathcal{B}}_{\mathbf{m},\mathbf{m}'} = \frac{\partial^2 \mathcal{L}}{\partial \varphi_{\mathbf{m}} \partial \varphi_{\mathbf{m}'}} \tag{A.35}$$

$$= \frac{\partial}{\partial \varphi_{\mathbf{m}}} \left(\sum_{\mathbf{m}''\in\mathcal{M}} \Psi^{-1}_{\mathbf{m}'',\mathbf{m}'} \cdot \frac{\partial \mathcal{L}}{\partial \lambda_{\mathbf{m}''}} \right) \tag{A.36}$$

$$= \sum_{\mathbf{m}'',\mathbf{m}'''\in\mathcal{M}} \Psi^{-1}_{\mathbf{m}''',\mathbf{m}} \cdot H^{\mathcal{F}}_{\mathbf{m}''',\mathbf{m}''} \cdot \Psi^{-1}_{\mathbf{m}'',\mathbf{m}'} \tag{A.37}$$

$$= \left((\Psi^{-1})^T \cdot H^{\mathcal{F}} \cdot \Psi^{-1} \right)_{\mathbf{m},\mathbf{m}'} \,. \tag{A.38}$$

F. Computation of pruning scores

In the following, we will derive the computations of the pruning scores used in the experimental evaluation in Sec. 5 in the main paper. We will present the original scores for SP and their corresponding IP version. In this Section, we assume all FBs $\mathcal{F}$ to form bases and fully connected layers to be described by 1×1 convolutions.

F.1. Pruning scores in general

In Secs. F.2 - F.6, five different methods for computing a pruning score vector $S \in \mathbb{R}^d$ are presented. These are the pruning scores used in our experimental evaluation.

A global pruning score means that the whole network is pruned altogether based on this score vector. Here, d denotes the number of all prunable parameters – for simplicity pooled in a big vector $\lambda \in \mathbb{R}^d$. For each parameter λ_j, there exists exactly one corresponding pruning score S_j. The higher a pruning score, the more important the corresponding parameter is. Thus, for pruning a network with prunable parameters λ to pruning rate $p \in [0,1]$, only the k biggest entries in S are not pruned, where

$$k := \lfloor (1-p) \cdot d \rfloor \,. \tag{A.39}$$

Consequently, the global pruning mask $\mu \in \{0,1\}^d$ is defined via

$$\mu_j = \begin{cases} 1 \,, & S_j \text{ belongs to the } k \text{ biggest entries in } S \\ 0 \,, & \text{else} \end{cases} \,. \tag{A.40}$$

After the pruning mask is computed, the network's prunable parameters are masked with the pruning mask via $\lambda \odot \mu$.

Note, for DST methods, we use *layerwise* pruning. The pruning score is computed equivalent to the global case but each layer is pruned with an individual pruning rate. Consequently, the number of pruned parameters is computed for each layer individually.

F.2. Random

Random pruning scores are easy to obtain. For each coefficient $\varphi_n^{(\alpha,\beta;l)}$ in the SP case or $\lambda_n^{(\alpha,\beta;l)}$ in the IP case, a corresponding random number is drawn i.i.d. from a $\mathcal{N}(0,1)$ distribution.

F.3. Magnitude

Magnitudes are used as pruning criterion for LTs [A.7], the DST methods SET [A.18] and RigL [A.6], FT [A.22] and GMP [A.8]. Using magnitudes as pruning criterion assumes that big coefficients are more likely to significantly influence the network's output than smaller ones. The corresponding formula is straight forward and given by

$$S_{Mag}(\alpha,\beta,n;l,\mathcal{B}) := \left|\varphi_n^{(\alpha,\beta;l)}\right| . \tag{A.41}$$

The corresponding formula for the FB representation is given by

$$S_{Mag}(\alpha,\beta,n;l,\mathcal{F}) := \left|\lambda_n^{(\alpha,\beta;l)}\right| . \tag{A.42}$$

By the transformation rules for the coefficients Eq. (A.19), it holds

$$\left\|\varphi^{(\alpha,\beta;l)}\right\| = \left\|\Psi^{(l)}\lambda^{(\alpha,\beta;l)}\right\| . \tag{A.43}$$

Consequently, $K \times K$ filters $h^{(\alpha,\beta;l)}$ do not need to have the same total pruning score in the spatial and FB representation, but might be scaled differently. We used magnitude pruning during or after training, where usually $\varphi^{(\alpha,\beta;l)} \neq \lambda^{(\alpha,\beta;l)}$. Even though magnitude pruning is normally used for the spatial representation, we did not have any scaling issues in the IP setting. This again indicates that jointly optimizing the FBs and their coefficients is stable.

F.4. SynFlow

SynFlow [A.26] is a pruning score, calculating the contribution of a parameter to the CNN's overall information flow. This is done, by differentiating the so called L_1 *path norm* of the network. The formula is given by

$$S_{SynFlow}(\alpha,\beta,n;l,\mathcal{B}) := \frac{\partial \mathcal{R}}{\partial \varphi_n^{(\alpha,\beta;l)}} \cdot \varphi_n^{(\alpha,\beta;l)} , \tag{A.44}$$

with

$$\mathcal{R} := \sum_{p\in\mathcal{P}} \prod_{\varphi_n^{(\alpha,\beta;l)}\in p} |\varphi_n^{(\alpha,\beta;l)}| . \tag{A.45}$$

Here,

$$\mathcal{P} = \{\{\varphi_{n_1}^{(\alpha_1,\beta;1)}, \varphi_{n_2}^{(\alpha_2,\alpha_1;2)}, \dots, \varphi_{n_{L_c}}^{(\alpha_{L_c},\alpha_{L_c-1};L_c)}\}\} \tag{A.46}$$

describes all existing paths in a CNN which start in the input layer and end in the output layer.

Consequently, the corresponding SynFlow score w.r.t. coefficients for the FBs is given by

$$S_{SynFlow}(\alpha,\beta,n;l,\mathcal{F}) := \frac{\partial \mathcal{R}}{\partial \lambda_n^{(\alpha,\beta;l)}} \cdot \lambda_n^{(\alpha,\beta;l)} . \tag{A.47}$$

The basis transformation between $\mathcal{B}$ and $\mathcal{F}$ does not change the total pruning score of a filter $h^{(\alpha,\beta;l)}$. This can be seen by

$$\begin{aligned}
&\sum_{n=1}^{K^2} S_{SynFlow}(\alpha,\beta,n;l,\mathcal{B}) && \text{(A.48)}\\
&= \left\langle \frac{\partial \mathcal{R}}{\partial \varphi^{(\alpha,\beta;l)}}, \varphi^{(\alpha,\beta;l)} \right\rangle && \text{(A.49)}\\
&= \left\langle \left(\Psi^{(l)^{-1}}\right)^T \frac{\partial \mathcal{R}}{\partial \lambda^{(\alpha,\beta;l)}}, \Psi^{(l)}\lambda^{(\alpha,\beta;l)} \right\rangle && \text{(A.50)}\\
&= \left\langle \frac{\partial \mathcal{R}}{\partial \lambda^{(\alpha,\beta;l)}}, \Psi^{(l)^{-1}}\Psi^{(l)}\lambda^{(\alpha,\beta;l)} \right\rangle && \text{(A.51)}\\
&= \left\langle \frac{\partial \mathcal{R}}{\partial \lambda^{(\alpha,\beta;l)}}, \lambda^{(\alpha,\beta;l)} \right\rangle && \text{(A.52)}\\
&= \sum_{n=1}^{K^2} S_{SynFlow}(\alpha,\beta,n;l,\mathcal{F}) . && \text{(A.53)}
\end{aligned}$$

The second equality is induced by the transformation formulas Eqs. (A.19) and (A.22). By having the same total pruning score for a filter for coefficients w.r.t. $\mathcal{F}$ and $\mathcal{B}$, we do not need to worry about possible scaling issues for the SynFlow score.

F.5. SNIP

SNIP [A.16] computes a so called *saliency score* for each parameter of a CNN before training. The idea is to measure the effect of changing the activation of a coefficient on the loss function. If this effect is big, the corresponding coefficient is trained, otherwise it is pruned. Let $\varphi^{(\alpha,\beta;l)} \in \mathbb{R}^{K_{(l)}^2}$ be the vector consisting of all spatial coefficients in the l-th layer of a CNN with input channel β and output channel α. Its saliency score is then computed as

$$\begin{aligned}
S_{SNIP}(\alpha,\beta,n;l,\mathcal{B}) &:= \left| \frac{\partial \mathcal{L}(m\cdot\varphi_n^{(\alpha,\beta;l)})}{\partial m}\bigg|_{m=1} \right| && \text{(A.54)}\\
&= \left| \frac{\partial \mathcal{L}}{\partial \varphi_n^{(\alpha,\beta;l)}} \cdot \varphi_n^{(\alpha,\beta;l)} \right| , && \text{(A.55)}
\end{aligned}$$

where $m \in \mathbb{R}$ models the activation of the filter value and $\mathcal{L}$ is the used loss function. The second equality is induced by using the chain rule [A.29].

The corresponding SNIP score w.r.t. $\mathcal{F}$ is given by

$$S_{SNIP}(\alpha,\beta,n;l,\mathcal{F}) := \left|\frac{\partial \mathcal{L}(m\cdot\lambda_n^{(\alpha,\beta;l)})}{\partial m}\right|_{m=1} \tag{A.56}$$

$$= \left|\frac{\partial \mathcal{L}}{\partial \lambda_n^{(\alpha,\beta;l)}}\cdot\lambda_n^{(\alpha,\beta;l)}\right| . \tag{A.57}$$

By inserting the transformation formulas (A.19) and (A.22) into Eq. (A.55), we get the relationship for the SNIP score of a filter $h^{(\alpha,\beta;l)}$ as

$$S_{SNIP}(\alpha,\beta;l,\mathcal{B}) \tag{A.58}$$

$$:= (S_{SNIP}(\alpha,\beta,n;l,\mathcal{B}))_{n=1}^{K^2} \tag{A.59}$$

$$= \left|(\Psi^{(l)^{-1}})^T\frac{\partial \mathcal{L}}{\partial \lambda^{(\alpha,\beta;l)}}\right| \odot \left|\Psi^{(l)}\lambda^{(\alpha,\beta;l)}\right| . \tag{A.60}$$

By comparing Eqs. (A.57) and (A.60), we see that changing the basis from $\mathcal{B}$ to $\mathcal{F}$ leads to different transformations of the gradient and the basis coefficient for a non-orthonormal FB $\mathcal{F}$.

In our experiments in the main body of the work, we computed the SNIP score with $\mathcal{F}=\mathcal{B}$, thus spatial and FB SNIP scores are equivalent. But, if arbitrary FBs are used, the scaling Eq. (A.60) might cause problems and should be taken into account.

F.6. GraSP

The GraSP score [A.29] approximates the influence of the removal of a spatial coefficient onto the network's gradient flow before training starts, the so called *importance score*. Using multi-index notation, it is computed as

$$S_{GraSP}(\mathbf{m};\mathcal{B}) := -\left(H^{\mathcal{B}}\cdot\frac{\partial \mathcal{L}}{\partial \varphi}\right)_{\mathbf{m}}\cdot\varphi_{\mathbf{m}} . \tag{A.61}$$

The corresponding score w.r.t. to basis coefficients $\mathcal{F}$ is given by

$$S_{GraSP}(\mathbf{m};\mathcal{F}) := -\left(H^{\mathcal{F}}\cdot\frac{\partial \mathcal{L}}{\partial \lambda}\right)_{\mathbf{m}}\cdot\lambda_{\mathbf{m}} . \tag{A.62}$$

By inserting the transformation rules for the coefficient, gradient and Hessian matrix, we derive

$$-\left(H^{\mathcal{B}}\cdot\frac{\partial \mathcal{L}}{\partial \varphi}\right)_{\mathbf{m}}\cdot\varphi_{\mathbf{m}} \tag{A.63}$$

$$= -\left((\Psi^{-1})^T\cdot H^{\mathcal{F}}\cdot\Psi^{-1}\cdot(\Psi^{-1})^T\cdot\frac{\partial \mathcal{L}}{\partial \lambda}\right)_{\mathbf{m}}\cdot(\Psi\cdot\lambda)_{\mathbf{m}} . \tag{A.64}$$

Therefore, the GraSP score is scaled differently for varying layers. Similar to the SNIP score, scaling issues might needed to be handled if FBs do not form ONBs.

Algorithm A1 Standard interspace initialization for a FB 2D convolutional layer

Require: Filter size $K\times K$, number of output channels c_{out}, number of input channels c_{in}

1: $\mu_h \leftarrow 0$ mean of spatial coefficients
2: $\sigma_h \leftarrow \sqrt{\frac{2}{c_{in}\cdot K^2}}$ variance of spatial coefficients
3: Initialize $g^{(1)},\dots,g^{(K^2)}$ via $g^{(n)}=e^{(n)}$ for all $n=1,\dots,K^2$
4: Initialize $\lambda_n^{(\alpha,\beta)} \sim \mathcal{N}(\mu_h,\sigma_h^2)$ i.i.d. for all $\alpha\in\{1,\dots,c_{out}\},\beta\in\{1,\dots,c_{in}\},n\in\{1,\dots,K^2\}$
5: **return** FB $\mathcal{F}=\{g^{(1)},\dots,g^{(K^2)}\}$, FB coefficients $\lambda=(\lambda_n^{(\alpha,\beta)})_{\alpha,\beta,n}$

On the other hand, if all FBs form ONBs, Eq. (A.64) reduces to

$$S_{GraSP}(\mathbf{m};\mathcal{B}) = -\left(\Psi\cdot H^{\mathcal{F}}\cdot\frac{\partial \mathcal{L}}{\partial \lambda}\right)_{\mathbf{m}}\cdot(\Psi\cdot\lambda)_{\mathbf{m}}. \tag{A.65}$$

Similar to SynFlow, it therefore holds

$$\sum_{\mathbf{m}\in\mathcal{M}_{\alpha,\beta;l}} S_{GraSP}(\mathbf{m};\mathcal{B}) = \sum_{\mathbf{m}\in\mathcal{M}_{\alpha,\beta;l}} S_{GraSP}(\mathbf{m};\mathcal{F}) \tag{A.66}$$

for $\mathcal{M}_{\alpha,\beta;l} := \{(\alpha,\beta,n;l) : n=1,\dots,K_{(l)}^2\}$, the multi-indices corresponding to an arbitrary filter $h^{(\alpha,\beta;l)}$. In this case, the total pruning score of a filter $h^{(\alpha,\beta;l)}$ does not depend on the representation.

G. Pruning methods and initialization of the interspace

In our experiments, we used the so called `kaiming normal` initialization [A.11] for the standard CNNs. Meaning that $h_{i,j}^{(\alpha,\beta)} \sim \mathcal{N}(\mu_h,\sigma_h^2)$ i.i.d. with

$$\mu_h = 0 \text{ and } \sigma_h = \sqrt{\frac{2}{c_{in}\cdot K^2}} . \tag{A.67}$$

We initialized all FB-CNNs such that their *spatial representations* follow a `kaiming normal` initialization, see Algs. A1 - A3. For simplicity, we propose the initialization of FB coefficients together with the FB. Of course, if a FB is shared for more than one layer, it has to be initialized just once.

Derivation of rescaling in Algorithm A3

In Alg. A3, $\mathcal{F}$ may contain $N\neq K^2$ elements. Thus, obtaining an equivalent initialization to the spatial `kaiming normal` initialization can not always be obtained by a simple basis transformation. Consequently, we rescale $\mathcal{F}$

Algorithm A2 Random `ONB` interspace initialization for a FB 2D convolutional layer

Require: Filter size $K \times K$, number of output channels c_{out}, number of input channels c_{in}

$\mu_h \leftarrow 0$ mean of spatial coefficients
$\sigma_h \leftarrow \sqrt{\frac{2}{c_{in} \cdot K^2}}$ variance of spatial coefficients
Initialize $\tilde{g}^{(1)}, \dots, \tilde{g}^{(K^2)} \in \mathbb{R}^{K \times K}$ with $\tilde{g}_{i,j}^{(n)} \sim \mathcal{N}(0,1)$ i.i.d. $\triangleright \{\tilde{g}^{(1)}, \dots, \tilde{g}^{(K^2)}\}$ with $\mathbb{P} = 1$ lin. independent
Apply Gram-Schmidt on $\{\tilde{g}^{(1)}, \dots, \tilde{g}^{(K^2)}\}$ to obtain ONB $\mathcal{F} = \{g^{(1)}, \dots, g^{(K^2)}\}$
Initialize spatial coefficients $\varphi_n^{(\alpha,\beta)} \sim \mathcal{N}(\mu_h, \sigma_h^2)$ i.i.d. for all $\alpha \in \{1, \dots, c_{out}\}, \beta \in \{1, \dots, c_{in}\}, n \in \{1, \dots, K^2\}$
Compute basis transformation matrix Ψ according to Eq. (A.19)
$\lambda^{(\alpha,\beta)} \leftarrow \Psi^T \cdot \varphi^{(\alpha,\beta)}$ FB coefficients
return FB $\mathcal{F} = \{g^{(1)}, \dots, g^{(K^2)}\}$, FB coefficients $\lambda = (\lambda_n^{(\alpha,\beta)})_{\alpha,\beta,n}$

Algorithm A3 `Random` interspace initialization for a FD 2D convolutional layer with $\#\mathcal{F} = N$ arbitrary

Require: Size of filter dictionary N, filter size $K \times K$, number of output channels c_{out}, number of input channels c_{in}

$\mu_h \leftarrow 0$ mean of spatial coefficients
$\sigma_h \leftarrow \sqrt{\frac{2}{c_{in} \cdot K^2}}$ variance of spatial coefficients
Initialize $\tilde{g}^{(1)}, \dots, \tilde{g}^{(N)} \in \mathbb{R}^{K \times K}$ with $\tilde{g}_{i,j}^{(n)} \sim \mathcal{N}(0,1)$ i.i.d. $\triangleright \{\tilde{g}^{(i_1)}, \dots, \tilde{g}^{(i_m)}\}$ with $i_1 \neq \dots \neq i_m$ and $m \leq K^2$ with $\mathbb{P} = 1$ lin. independent
Compute pixelwise sample mean $\tilde{\mu}_{i,j}$ and sample variance $\tilde{\sigma}_{i,j}$ according to Eq. (A.68)
$g_{i,j}^{(n)} \leftarrow \sqrt{\frac{1}{N} - \frac{1}{N^2}} \cdot \frac{\tilde{g}_{i,j}^{(n)} - \tilde{\mu}_{i,j}}{\tilde{\sigma}_{i,j}} + \frac{1}{N}$ $\triangleright$ rescale FD
Initialize $\lambda_n^{(\alpha,\beta)} \sim \mathcal{N}(\mu_h, \sigma_h^2)$ i.i.d. for all $\alpha \in \{1, \dots, c_{out}\}, \beta \in \{1, \dots, c_{in}\}, n \in \{1, \dots, N\}$
return FD $\mathcal{F} = \{g^{(1)}, \dots, g^{(N)}\}$, FD coefficients $\lambda = (\lambda_n^{(\alpha,\beta)})_{\alpha,\beta,n}$

in order to mimic a `kaiming normal` initialization of spatial coefficients if FB coefficients are initialized with a `kaiming normal` initialization as well.

Let $\mu_{i,j}$ and $\sigma_{i,j}^2$ be the pixel wise sample mean and sample variance of the FD $\mathcal{F}$ with arbitrary size $N \geq 1$, *i.e.*

$$\mu_{i,j} := \frac{1}{N} \sum_{n=1}^{N} g_{i,j}^{(n)} \text{ and } \sigma_{i,j}^2 := \frac{1}{N} \sum_{n=1}^{N} (g_{i,j}^{(n)} - \mu_{i,j})^2 \,. \tag{A.68}$$

By using Eq. (A.68) it holds for an arbitrary i.i.d. initialization of λ with mean μ_λ and variance σ_λ^2

$$\mathbb{E}[h_{i,j}] = \mathbb{E}[\sum_{n=1}^{N} \lambda_n g_{i,j}^{(n)}] = \sum_{n=1}^{N} \mathbb{E}[\lambda_n] g_{i,j}^{(n)} = \mu_\lambda N \mu_{i,j} \tag{A.69}$$

and

$$\mathbb{E}[h_{i,j}^2] = \sum_{n,m} \mathbb{E}[\lambda_n \lambda_m] g_{i,j}^{(n)} g_{i,j}^{(m)} \tag{A.70}$$

$$= \sum_{n} \mathbb{E}[\lambda_n^2] {g_{i,j}^{(n)}}^2 + \sum_{n} \sum_{n \neq m} \mathbb{E}[\lambda_n]^2 g_{i,j}^{(n)} g_{i,j}^{(m)} \tag{A.71}$$

$$= (\sigma_\lambda^2 + \mu_\lambda^2) \sum_{n} {g_{i,j}^{(n)}}^2 + \mu_\lambda^2 \sum_{n} \sum_{n \neq m} g_{i,j}^{(n)} g_{i,j}^{(m)} \tag{A.72}$$

$$= \sigma_\lambda^2 \sum_{n} {g_{i,j}^{(n)}}^2 + \mu_\lambda^2 \sum_{n,m} g_{i,j}^{(n)} g_{i,j}^{(m)} \tag{A.73}$$

$$= N \sigma_\lambda^2 (\sigma_{i,j}^2 + \mu_{i,j}^2) + \mu_\lambda^2 N^2 \mu_{i,j}^2 \,. \tag{A.74}$$

We now want to determine $\mu_{i,j}$ and $\sigma_{i,j}$ such that $\mathbb{E}[h_{i,j}] = \mu_\lambda$ and $\mathbb{E}[h_{i,j}^2] = \sigma_\lambda^2 + \mu_\lambda^2$ holds, *i.e.* the distribution of λ and h have the same mean and variance. By Eq. (A.69), setting $\mu_{i,j} = 1/N$ guarantees $\mathbb{E}[h_{i,j}] = \mu_\lambda$. By inserting $\mu_{i,j} = 1/N$ into Eq. (A.74), we see that $\sigma_{i,j}^2 = 1/N - 1/N^2$ implies $\mathbb{E}[h_{i,j}^2] = \sigma_\lambda^2 + \mu_\lambda^2$. Consequently, FDs are rescaled in Alg. A3 to have pixelwise sample mean $\mu_{i,j} = 1/N$ and sample variance $\sigma_{i,j}^2 = 1/N - 1/N^2$.

G.1. General setup

For SP and the dense baselines, we initialized the standard CNN with the `kaiming normal` initialization. For IP, we initialize networks according to Alg. A1 for all experiments in the main body of the text. In Sec. B, also Algs. A2 and A3 are used as initializations for the interspace.

Pruning masks are computed according to the formulas described in Sec. F for SP and IP. In the following we will describe the used pruning methods in detail.

G.2. Lottery tickets with resetting coefficients

LTs with resetting coefficients to an early training iteration [A.7] are obtained as follows. We first train the network to epoch $t_0 = 500$ and store the corresponding model, optimizer, *etc*. Afterwards, the network is trained to convergence. Then, 20% of the coefficients are pruned, based

on their magnitudes. All non-zero parameters are reset to their values at training time $t_0 = 500$ and pruned ones are fixed at zero from now on. Thus, contrarily to DST, a pruned coefficient will never be able to recover. The training schedule parameters, like learning rate or moving averages for batch normalization and SGD with momentum, are reset to their corresponding value at t_0 as well. For IP, also the FBs are reset to step t_0. Then, the network is again trained to convergence, 20% of the non-zero coefficients are pruned and the remaining non-zero parameters are reset again. This is done, until the desired pruning rate is reached. Note, for the last pruning step also $< 20\%$ of the non-zero parameters might be pruned to exactly match the desired pruning rate. If the final pruning rates is reached, the network with desired sparsity is trained for a last, final time. All in all,

$$k = 1 + \left\lceil \frac{\log(1-p)}{\log 4 - \log 5} \right\rceil \tag{A.75}$$

trainings are needed to obtain and train a network with sparsity p using this iterative approach.

Following [A.7], we do not prune the fully connected layer for LTs but keep it dense.

G.3. Dynamic sparse training

Dynamic sparse training methods adapt the network's pruning mask during training [A.2,A.4,A.6,A.17,A.18,A.19]. In this work we use SET [A.18] which is based on estimating the importance of coefficients via magnitude pruning and regrowing coefficients due to a random selection. RigL [A.6] improves this approach by regrowing coefficients which have the biggest gradient magnitudes.

Before training, the networks are pruned randomly. It was shown in [A.6] that using layer wise sparsity corresponding to an Erdős-Rényi-kernel leads to good results. This means that each layer has sparsity depending on its size, *i.e.*

$$1 - \varepsilon \cdot \frac{c_{out}^{(l)} + c_{in}^{(l)} + 2 \cdot K_{(l)}}{c_{out}^{(l)} \cdot c_{in}^{(l)} \cdot K_{(l)}^2}\,, \tag{A.76}$$

and ε is a global parameter, tuned such that a global sparsity of p is obtained.

During training, the pruning mask is frequently updated. For this, parameters are pruned for each layer with rate p_t. To be precise, all *non-zero* parameters in a layer are pruned with the rate p_t. The pruning rate p_t depends on the training step t and decays with a cosine schedule in order to improve convergence [A.6]. It holds

$$p_t = p_{min} + \frac{1}{2}\left(p_{init} - p_{min}\right) \cdot \left(1 + \cos\left(\frac{t\pi}{T}\right)\right) \tag{A.77}$$

where T is the number of total training steps, $p_{min} = 0.005$ the minimal pruning rate and $p_{init} = 0.5$ the initial rate used for updating the pruning mask. Of course, in each layer an equal number of non trained coefficients are regrown after pruning. Following [A.18] and [A.6], regrown coefficients are initialized with value 0 but are updated via SGD from this moment on.

SET and RigL work optimal for different pruning mask update frequencies [A.17]. According to [A.17], we update pruning masks each $1,500$ training steps for SET and each $4,000$ steps for RigL.

G.4. Pruning at initialization

We test PaI methods, SNIP [A.16], GraSP [A.29] and SynFlow [A.26] together with random PaI. In contrast to LTs and DST, PaI is quite simple. For SNIP and GraSP we compute the pruning scores described in Secs. F.5 and F.6 with the help of 100 batches of training data for the CIFAR-10 experiments and 15 for ImageNet. As proposed by [A.26], we compute the pruning scores for SNIP and GraSP with all batch normalization layers [A.13] set to PyTorch's `train` mode. Afterwards, all coefficients are pruned one-shot.

The GraSP scores for SP, Eq. (A.61), and IP, Eq. (A.62), require the computation of a Hessian vector product $H \cdot g$. Fortunately, not the whole Hessian H needs to be computed to evaluate such products. For this, we use the linearity of the derivative. For $H \in \mathbb{R}^{d\times d}$ and an arbitrary vector $v \in \mathbb{R}^d$, it holds

$$(H \cdot v)_i = \sum_j \frac{\partial^2 \mathcal{L}}{\partial \lambda_i \partial \lambda_j} \cdot v_j \tag{A.78}$$

$$= \frac{\partial}{\partial \lambda_i}\left(\sum_j \frac{\partial \mathcal{L}}{\partial \lambda_j} v_j\right) \tag{A.79}$$

$$= \frac{\partial}{\partial \lambda_i}\left\langle \frac{\partial \mathcal{L}}{\partial \lambda}, v \right\rangle\,, \tag{A.80}$$

and consequently $H \cdot v = \frac{\partial \langle \frac{\partial \mathcal{L}}{\partial \lambda}, v\rangle}{\partial \lambda}$. By using v as the fixed gradient $g_v = \frac{\partial \mathcal{L}}{\partial \lambda}$, we can compute $H \cdot g_v$ with only three backward passes. The first one is needed to compute the fixed gradient g_v. The second and third are required to compute $H \cdot g_v = \frac{\partial \langle \frac{\partial \mathcal{L}}{\partial \lambda}, g_v\rangle}{\partial \lambda}$. An implementation of this can be found in the official code base for GraSP, see this link (MIT license).

In contrast to these one-shot methods, pruning masks for SynFlow are computed in an iterative fashion. For this purpose, we compute the pruning score proposed in Sec. F.4 with one forward- and backward pass, prune a small fraction of elements and repeat it 100 times. The pruning rate grows with an exponential schedule [A.26] which gives the pruning rate

$$p_k = 1 - (1-p)^{k/100} \tag{A.81}$$

after the k^{th} pruning iteration. For computing the SynFlow score, we set all batch normalization layers to `eval` mode,

as suggested by [A.26].

G.5. Gradual Magnitude Pruning

Following [A.8] we gradually sparsify the model based on the coefficients' magnitudes. After each N training iterations, the pruning rate is increased and new weights are pruned. The pruning rate at iteration $k \cdot N$ is given by

$$p(k \cdot N) = \begin{cases} 0, & k \cdot N < t_0 \\ p \cdot \left(1 - \left(1 - \frac{k \cdot N - t_0}{t_1 - t_0}\right)^3\right), & k \cdot N \in [t_0, t_1] \\ 1, & k \cdot N > t_1 \end{cases} . \tag{A.82}$$

Here, p denotes the final pruning rate and t_0, t_1 are the training iterations where the gradual pruning begins and ends, respectively. For each iteration $k \cdot N$ with $k \in \mathbb{N}$, the $p(k \cdot N) \cdot d$ weights with smallest magnitude are pruned. Since pruned weights are frozen at 0, those weights will be pruned again and they will therefore never recover.

We follow the suggestions in [A.8] for choosing t_0, t_1 and N for the final pruning rates $p \in \{0.8, 0.9\}$ for the ResNet50 on ImageNet (summarized in this table). Since we use a different batch size than [A.8] in our experiments (256 compared to $1,024$), we adapt their choices for t_0, t_1 and N to our batch size by multiplying them by 4. However, these choices are not optimized for batch size 256 and we therefore report slightly worse results than [A.8]. For our experiments, we use $N = 8,000, t_0 = 160,000$ and $t_1 = 400,000$ for $p = 0.8$. For $p = 0.9$ we set $N = 8,000, t_0 = 160,000$ and $t_1 = 304,000$.

G.6. Fine tuning

We also compare IP and SP for magnitude pruning applied on a pre-trained, dense network while the sparse network is fine-tuned afterwards. As suggested by [A.22], we do not use a classical fine-tuning setup beginning with a small learning rate, but use the setup of the dense pre-training also for fine-tuning. Thus, the sparse fine-tuning starts with a high learning rate. Naturally, for IP we use the pre-trained FBs as starting point for the fine-tuning. For FT, we use the same training setup as for RigL.

H. Experimental setup

In this Section, we describe the setup for the experiments discussed in the main body of the paper and in Sec. B. Our experiments were conducted on an internal cluster with CentOS Linux release 7.9.2009 (Core). We used Python 3.8 with the deep learning framework PyTorch1.9 [A.21] (BSD license) together with cudatoolkit 10.2 [A.20]. As hardware we had an Intel XEON E5-2680 v4 2.4 GHz CPU and n NVIDIA GeForce 1080ti GPUs, where $n = 1$ for the CIFAR-10 experiments, $n \in \{2, 4\}$ for ResNet50 on ImageNet and $n = 8$ for ResNet18 on ImageNet. Further details on the training time for one epoch, the number of used CPU cores, used RAM and GPU memory are given in Tab. A2.

Since we compare and adapt different pruning methods, we used the publicly available codes for reproducing the results and modifying them for the IP setting. These are:

- Link to code for SynFlow [A.26] (unknown license),
- Link to code for GraSP [A.29] (MIT license),
- PyTorch adaption of link to code for SNIP [A.16] (MIT license),
- Link to code as the official PyTorch version for Lottery Tickets [A.7] (MIT license),
- Link to code as a base for DST methods which is the official code for [A.17] (unknown license). In [A.17], training schedules for SET [A.18] and RigL [A.6] are improved. This code base is also used for FT [A.22] and GMP [A.8].

For the IP versions of these pruning methods, we additionally updated the code to use Alg. 1 as a 2D convolution.

The initializations of the used CNNs and FB-CNNs are described in Sec. G. Used hyperparameters are summarized in Tab. A3. We also trained the dense standard networks with the same training schedules as the pruned ones. Results for the dense models can be found in Tab. A2.

Training schedule. As common in the literature of sparse training, see for example [A.26, A.16, A.29], no hyperparameter tuning is done in this work. We want to highlight, that all used training setups are chosen from one of the adapted pruning methods. Especially, FBs and FB coefficients are trained with the standard learning rates, optimized for training spatial coefficients.

For PaI on CIFAR-10, we used the setup from [A.16], whereas SET uses the training schedule from [A.17]. The CIFAR-10 experiment for LTs is equal to the one used in [A.7]. For ImageNet on ResNet18, we used the standard PyTorch ImageNet training (see this link) (BSD 3-clause license) as baseline for our training – the same as used in [A.29]. Results can be further improved by adding learning rate warm-up for 5 epochs [A.10] and label smoothing [A.25] with smoothing parameter $\varepsilon = 0.1$. This improvement is inherited from [A.17] and applied to train RigL on ResNet50. For FT, we use the same training hyperparameters as for RigL whereas we use the suggested one from the original paper [A.8] for GMP.

CIFAR-10. CIFAR-10 [A.14] consists of $60,000$ 32×32 RGB images. CIFAR-10 is a publicly available dataset with, best to our knowledge, no existing licenses. Authors are allowed to use the datasets for publications if the tech report [A.14] is referenced. CIFAR-10 has 10 classes with $6,000$ images per class. The data is split into $50,000$ training and $10,000$ test images. For each training, we randomly split the

Dataset/Model	Mean ± Std	# Params	# FB Params `fine` sharing	Time 1 epoch [s]	#GPUs	GPU memory	#CPU cores	RAM
CIFAR-10/VGG16	$93.41 \pm 0.07\%$	15.3mio	$1,053$	23.7	1	11 GB	1	12 GB
CIFAR-10/VGG16-LT	$93.58 \pm 0.10\%$	14.7mio	$1,053$	21.9	1	11 GB	1	12 GB
ImageNet/ResNet18	$69.77 \pm 0.06\%$	11.7mio	$1,296$	$2,770.5$	8	8×11 GB	8	8×12 GB
ImageNet/ResNet50 (RigL & FT)	$77.15 \pm 0.04\%$	25.6mio	$3,697$	$4,354.2$	2	2×11 GB	4	4×12 GB
ImageNet/ResNet50 (GMP)	$76.64 \pm 0.06\%$	25.6mio	$3,697$	$2,622.8$	4	4×11 GB	8	8×12 GB

Table A2. Top-1 test accuracies for densely trained models with additional information, including the hardware setup. Note, # FB parameters for `fine` sharing is the *most* number of extra parameters induced by the interspace representation. We suggest to use at least 2 times the number of CPU cores than GPUs, otherwise data loading becomes a bottleneck (compare ResNet18 and ResNet50 for GMP which have approximately the same runtime despite different network sizes and # GPUs).

Experiment	CIFAR-10 (no LTs)	CIFAR-10 (LTs)	ImageNet (PaI)	ImageNet (RigL & FT)	ImageNet (GMP)
Network	VGG16	VGG16-LT	ResNet18	ResNet50	ResNet50
# Trainings	5	5	3	3	3
# Epochs	250	160	90	100	100
Batch Size	128	128	512	128	256
# GPUs	1	1	8	2	4
Optimizer: SGD-	Momentum	Momentum	Momentum	Momentum	Momentum
Momentum	0.9	0.9	0.9	0.9	0.9
Learning Rate	0.1	0.1	0.1	0.1	0.1
LR Decay	$\times 0.1$ every 30k iterations	$\times 0.1$ epochs $80/120$	$\times 0.1$ epochs $30/60$	$\times 0.1$ epochs $30/60/90$	$\times 0.1$ epochs $30/60/80$
LR Warm-up	✗	✗	✗	5 epochs, linear	5 epochs, linear
Weight Decay	$5 \cdot 10^{-4}$	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Label smoothing	✗	✗	✗	$\varepsilon = 0.1$	$\varepsilon = 0.1$

Table A3. Setups for experiments in the main body of the paper and Sec. B. Each run of the # trainings was executed with a different random seed.

training images into two parts, $45,000$ images for training and $5,000$ images for validation. All images for training, validation and testing are normalized by their channel wise mean and standard deviation. Furthermore, we additionally use the standard data augmentation for CIFAR on the training images. This is given by cropping and random horizontal flipping of the images. Test results are reported for the early stopping epoch, the epoch with the highest validation accuracy. Used hyperparameters are summarized in Tab. A3.

ImageNet. The ImageNet ILSVRC2012 [A.23] dataset is an image classification dataset, containing approximately 1.2 million RGB images for training and $150,000$ RGB images for validation, divided into $1,000$ classes. ImageNet has a custom license allowing non-commercial research. To be allowed to use ImageNet for non-commercial research, access to the image database has to be requested – which we did. Full terms for the usage of the ImageNet database can be found in this link.

Again, all images are normalized for each channel. Training images are randomly cropped to size 224×224 and randomly flipped in the horizontal direction. The validation images are resized to size 256×256 and their central 224×224 pixels are used for validation. For the ResNet50 experiment, we also add label smoothing on the training loss with a smoothing factor 0.1. As common in the literature, we report results on the validation set, since labels for the test set are publicly not available. Hyperparameters are provided in Tab. A3.

I. Network architectures

In this Section we provide the used network architectures VGG16 [A.24] and the adapted version VGG16-LT for CIFAR-10, as well as ResNet18 and ResNet50 [A.12] for ImageNet.

The architectures are shown in Tabs. A4 to A6. Note, we use two different versions of a VGG16, a small one for the LT experiments and a bigger one for the remaining experiments. A graphical description of the residual block and the bottleneck block used for ResNets, is shown in Figs. A7 and A8, respectively. Furthermore, all architectures are used in their standard form or with FB convolutions. Therefore, (FB) always indicates, that a FB version of the filter is used for the FB-CNN. Not all convolutional layers are marked with a (FB), since we only apply the FB formulation on

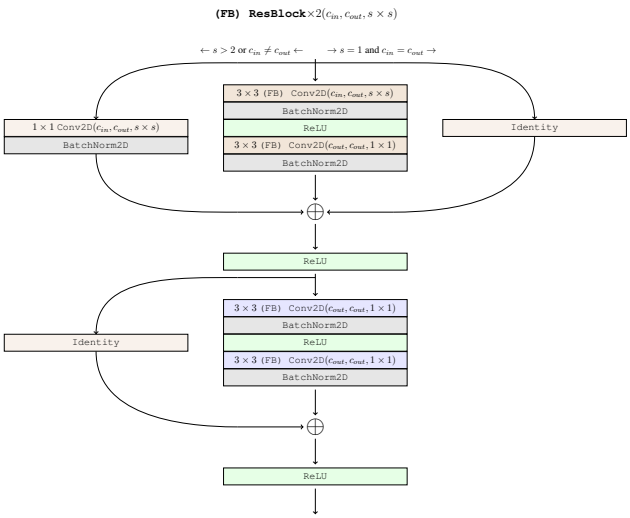


Figure A7. Architecture of a `(FB) ResBlock`×2 with c_{in} input channels, c_{out} output channels and stride $s \times s$ for the first (FB) convolution. The first residual connection is is a standard 1×1 `2D Convolution` followed by a `BatchNorm2D` layer if $s > 1$ or $c_{in} \neq c_{out}$. However, if $c_{in} = c_{out}$ and $s = 1$, the first residual connection is simply an identity mapping. The second residual connection is always an identity mapping. All `(FB) Conv2D` layers do not have biases.

convolutional layers with kernel size $K > 1$. Additionally, we indicate the layers which share one FB for `coarse`, `medium` and `fine` FB sharing in Tabs. A4 to A6.

All tensor dimensions of standard 2D convolutional filters are given as $c_{out} \times c_{in} \times K \times K$, where c_{in} equals the number of input channels, c_{out} the number of output channels and $K \times K$ the size of each convolutional kernel. FB convolutions have coefficients represented by a tensor of size $c_{out} \times c_{in} \times K^2$. For pooling layers, $K \times K$ denotes the tiling size. In the case of linear layers, the tensor size is given as $c_{out} \times c_{in}$, where c_{in} is the number of incoming neurons and c_{out} the number of outgoing neurons. For training the networks, we used the cross entropy loss function.

J. Proof of Theorem 1

SDL optimizes a dictionary $\mathbf{F} \in \mathbb{R}^{m \times M}$ jointly with its coefficients $R \in \mathbb{R}^{M \times n}$ w.r.t. the non-convex problem

$$\inf_{\mathbf{F},R} \|U - \mathbf{F} \cdot R\|_F \;\text{ s.t. } \|R\|_0 \leq s \;, \tag{A.83}$$

for a target $U \in \mathbb{R}^{m \times n}$ and sparsity constraint s. In our context U corresponds to a convolutional layer, the dictionary $\mathbf{F}$ to the layer's FB (FD) $\mathcal{F}$ and R to the FB (FD) coefficients. Standard magnitude pruning can be seen as a special case of SDL where the dictionary $\mathbf{F}$ is fixed to form the standard basis, *i.e.* $\mathbf{F} = \mathrm{id}_{\mathbb{R}^m}$. Accordingly,

$$\inf_{\Phi} \|U - \Phi\|_F \;\text{ s.t. } \|R\|_0 \leq s \tag{A.84}$$

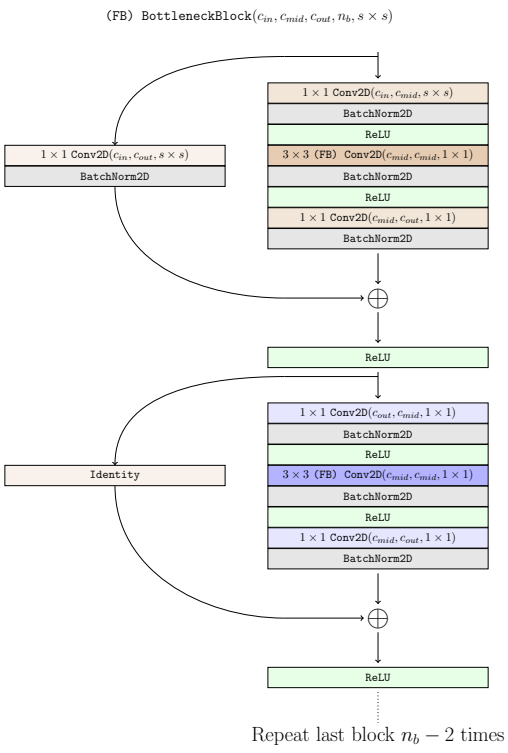


Figure A8. Architecture of a `(FB) BottleneckBlock` with c_{in} input channels, c_{mid} middle channels and c_{out} output channels and stride $s \times s$ for the first 1×1 (FB) convolution. The number of small blocks forming the `(FB) BottleneckBlock` is given by n_b. The first residual connection is is a standard 1×1 `2D Convolution` followed by a `BatchNorm2D` layer. The following residual connections are always identity mappings. All `(FB) Conv2D` layers do not have biases.

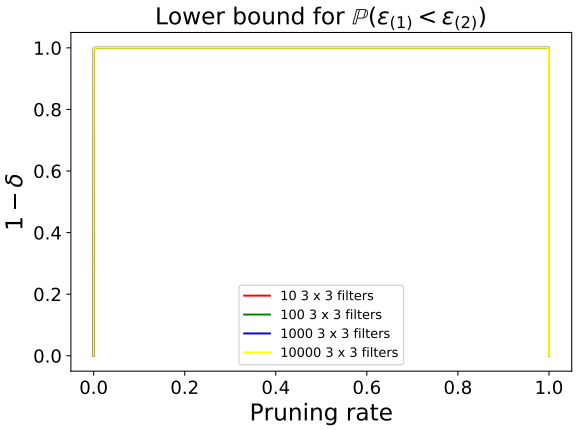


Figure A9. Lower bound $1 - \delta$ for $\mathbb{P}(\varepsilon_{(1)} < \varepsilon_{(2)})$, *i.e.* the optimization problem (A.83) having a *smaller* solution than the problem (A.84). Computed for varying numbers n of 3×3 filters in a layer and varying pruning rates $p \in [0, 1]$.

Module	Output Size	c_{in}	c_{out}	K	Repeat	Stride	Padding	Bias	BatchNorm	ReLU	Coarse	Medium	Fine
(FB) Conv2D	32×32	3	64	3×3	$\times 1$	1×1	1×1	✓	✓	✓			
(FB) Conv2D	32×32	64	64	3×3	$\times 1$	1×1	1×1	✓	✓	✓			
MaxPool2D	16×16	64	64	2×2	$\times 1$	2×2	0×0	✗	✗	✗			
(FB) Conv2D	16×16	64	128	3×3	$\times 1$	1×1	1×1	✓	✓	✓			
(FB) Conv2D	16×16	128	128	3×3	$\times 1$	1×1	1×1	✓	✓	✓			
MaxPool2D	8×8	128	128	2×2	$\times 1$	2×2	0×0	✗	✗	✗			
(FB) Conv2D	8×8	128	256	3×3	$\times 1$	1×1	1×1	✓	✓	✓			
(FB) Conv2D	8×8	256	256	3×3	$\times 2$	1×1	1×1	✓	✓	✓			×2
MaxPool2D	4×4	256	256	2×2	$\times 1$	2×2	0×0	✗	✗	✗			
(FB) Conv2D	4×4	256	512	3×3	$\times 1$	1×1	1×1	✓	✓	✓			
(FB) Conv2D	4×4	512	512	3×3	$\times 2$	1×1	1×1	✓	✓	✓			×2
MaxPool2D	2×2	512	512	2×2	$\times 1$	2×2	0×0	✗	✗	✗			
(FB) Conv2D	2×2	512	512	3×3	$\times 3$	1×1	1×1	✓	✓	✓			×3
MaxPool2D	1×1	512	512	2×2	$\times 1$	2×2	0×0	✗	✗	✗			
Linear	512	512	512	—	$\times 2$	—	—	✓	✓	✓	**Removed for VGG16-LT**		
Linear	10	512	10	—	$\times 1$	—	—	✓	✗	✗			

Table A4. VGG16 and VGG16-LT for CIFAR-10 with `coarse`, `medium` and `fine` FB sharing schemes specified in the last three columns. Each ▮, connected by either another ▮ or |, corresponds to exactly one FB $\mathcal{F}$ shared for all filters in the corresponding layers. The thin connection | corresponds to `MaxPool2D` layers which do not use the FBs themselves. Note, for VGG16-LT, the first and second `Linear` layers are removed.

Module	Output Size	c_{in}	c_{out}	K	Stride	Padding	Bias	BatchNorm	ReLU	Coarse	Medium	Fine
Conv2D	112×112	3	64	7×7	2×2	3×3	✗	✓	✓			
MaxPool2D	56×56	64	64	3×3	2×2	1×1	✗	✗	✗			
(FB) ResBlock×2	56×56	64	64	3×3	1×1	1×1	✗	✓	✓			×4
(FB) ResBlock×2	28×28	64	128	3×3	2×2	1×1	✗	✓	✓			×4
(FB) ResBlock×2	14×14	128	256	3×3	2×2	1×1	✗	✓	✓			×4
(FB) ResBlock×2	7×7	256	512	3×3	2×2	1×1	✗	✓	✓			×4
AvgPool2D	1×1	512	512	7×7	0×0	0×0	✗	✗	✗			
Linear	$1,000$	512	$1,000$	—	—	—	✓	✗	✗			

Table A5. ResNet18 for ImageNet with `(FB) ResBlock×2`, shown in Fig. A7. Last three columns declare `coarse`, `medium` and `fine` FB sharing schemes. Each ▮ connected by another ▮ corresponds to exactly one FB $\mathcal{F}$ shared for all filters in the corresponding layers. Note, the first convolutional layer has kernel size 7×7 and we do not use a FB formulation for this layer.

is minimized.

Theorem 1. *Let $1 < m \leq M$, $0 < s < m \cdot n$ and $U_{i,j} \sim \mathcal{N}(0,1)$ i.i.d. Further assume that $\varepsilon_{(1)}$ is the infimum of Eq. (A.83) and $\varepsilon_{(2)}$ the minimum of Eq. (A.84). Assume Φ^* to be the minimizer for Eq. (A.84). Then $\varepsilon_{(1)} < \varepsilon_{(2)}$ holds with probability $\mathbb{P} = 1$.*

If furthermore supp R *for Eq. (A.83) is fixed to be equal to* supp Φ^**, then $\varepsilon_{(1)} \leq \varepsilon_{(2)}$ and strict inequality holds with $\mathbb{P} \geq 1 - \delta$, where*

$$\delta = \begin{cases} 0 & , \text{ if } s \not\equiv 0 (mod\ m) \\ \frac{\binom{n}{s}}{\binom{m \cdot n}{s}} & , \text{ if } s \equiv 0 (mod\ m) \end{cases} . \tag{A.85}$$

Figure A9 shows the probability of the solution to Eq. (A.84) being strictly bigger than the solution of Eq. (A.83) if supp R is restricted to be supp Φ^*. Precisely, it shows $1 - \delta$ for varying pruning rates p. It can be seen that, except for the trivial case of a network being completely pruned or being not pruned at all, δ is numerically equal to zero, even for a network with only 100 filters.[A.4] Thus, despite $\delta > 0$, numerically the chance of $\varepsilon_{(1)} = \varepsilon_{(2)}$ *is* equal to zero.

The proof of Thm. 1 is split in several parts.

- Lemma 1 shows that Eq. (A.84) always has a minimum and constructs the minimizing Φ^*.
- Lemma 2 will show that for each feasible point Φ_0 for Eq. (A.84), there exists an equivalent feasible point $(\mathbf{F}_0, R_0)$ for Eq. (A.83) with the same sparsity

[A.4] The minimum δ we computed for non-trivial pruning rates $p \in (0,1)$ for a network with $n = 10$ filters of size 3×3 was given by $\delta \leq 10^{-10}$. For $n \geq 100$, numerically $\delta = 0$ for all $p \in (0,1)$.

Module	Output Size	c_{in}	c_{mid}	c_{out}	n_b	K	Stride	Padding	Bias	BN	ReLU	Coarse adapt	Medium	Fine
(FB) Conv2D	112×112	3	—	64	—	7×7	2×2	3×3	✗	✓	✓	■	■	■×1
MaxPool2D	56×56	64	—	64	—	3×3	2×2	1×1	✗	✗	✗			
(FB) BottleneckBlock	56×56	64	64	256	3	3×3	1×1	1×1	✗	✓	✓	■	■	■×3
(FB) BottleneckBlock	28×28	256	128	512	4	3×3	2×2	1×1	✗	✓	✓	■	■	■×4
(FB) BottleneckBlock	14×14	512	256	1,024	6	3×3	2×2	1×1	✗	✓	✓	■	■	■×6
(FB) BottleneckBlock	7×7	1,024	512	2,048	3	3×3	2×2	1×1	✗	✓	✓	■	■	■×3
AvgPool2D	1×1	512	—	512	—	7×7	0×0	0×0	✗	✗	✗			
Linear	1,000	512	—	1,000	—	—	—	—	✓	✗	✗			

Table A6. ResNet50 for ImageNet with (FB) BottleneckBlock, shown in Fig. A8. Last three columns declare the adapted coarse, medium and fine FB sharing schemes. Each ■ connected by another ■ corresponds to exactly one FB $\mathcal{F}$ shared for all filters in the corresponding layers.

$\|R_0\|_0 = \|\Phi_0\|_0$ and distance $\|U - \mathbf{F}_0 \cdot R_0\|_F = \|U - \Phi_0\|_F$.

- Consequently, Corollary 3 shows that the solution to Eq. (A.83) is always smaller or equal to the solution of Eq. (A.84).
- The first part of the proof of Thm. 1 shows that the solution obtained by Eq. (A.84) can only with a small chance $\delta_0 \leq \delta$ be the optimum of Eq. (A.83). This is based on two facts, first we construct the equivalent point $(\mathbf{F}_0, R_0)$ to Φ^* according to Lemma 2. Then, we show that with a probability of at most δ, $(\mathbf{F}_0, R_0)$ fulfills a necessary condition for solving Eq. (A.83). This condition is given by $\mathbf{F}_0$ yielding a local optimum[A.5] of the smooth, convex function

$$f : \mathbb{R}^{m \times M} \to \mathbb{R} \ , \ \mathbf{F} \mapsto \|U - \mathbf{F} \cdot R_0\|_F^2 \ , \quad \text{(A.86)}$$

  which is evaluated by looking at the probability of $\mathbf{F}_0$ being a root of $\frac{\partial f}{\partial \mathbf{F}}$.
- The second part of Thm. 1 adapts $(\mathbf{F}_0, R_0)$ *if* $\operatorname{supp} R_0$ is not fixed to be equal to $\operatorname{supp} \Phi^*$. By setting one column of Φ^* as a new basis element, the number of coefficients needed to match $\Phi^* = \mathbf{F}^* \cdot R^*$ with the adapted $(\mathbf{F}^*, R^*)$ is reduced. This of course provides new unused coefficients which are used to *better approximate* the target U.

Lemma 1. The optimization problem (A.84) always has a solution $\Phi^* \in \mathbb{R}^{m \times n}$ obtained by

$$\Phi^* = (\Phi^*_{i,j})_{i,j} \quad \text{with}$$
$$\Phi^*_{i,j} = \begin{cases} U_{i,j} \ , & \text{if } (i,j) \in \operatorname{TOP}_s(U) \\ 0 \ , & \text{else} \end{cases} \ . \quad \text{(A.87)}$$

Here,

$$\operatorname{TOP}_s(U) := \{(i_0, j_0) \in \{1, \dots, m\} \times \{1, \dots, n\} : U_{i_0,j_0} \text{ belongs to the top } s \text{ magnitudes of } \{U_{i,j}\}\} \quad \text{(A.88)}$$

defines the indices corresponding to the s highest magnitudes of U.

Proof of Lemma 1. To solve Eq. (A.84), we rewrite the optimization problem into its equivalent, squared form

$$\inf_{\Phi \in \mathbb{R}^{m \times n}} \|U - \Phi\|_F^2 \ \text{ s.t. } \|\Phi\|_0 \leq s \ . \quad \text{(A.89)}$$

The problem (A.89) is equivalent to

$$\inf_{\Phi \in \bigcup_{k=1}^r \mathcal{S}_k} \|U - \Phi\|_F^2 = \min_{k \in \{1, \dots, r\}} \inf_{\Phi \in \mathcal{S}_k} \|U - \Phi\|_F^2 \quad \text{(A.90)}$$

with $r = \binom{n \cdot m}{s}$,

$$\mathcal{S}_k = \{A \in \mathbb{R}^{m \times n} : \operatorname{supp} A \subset S_k\}, \quad S_k \subset \{1, \dots, m\} \times \{1, \dots, n\} \ , \#S_k = s \quad \text{(A.91)}$$

satisfying $S_k \neq S_j$ for $k \neq j$ and $\bigcup_{k=1}^r \mathcal{S}_k = \{A \in \mathbb{R}^{m \times n} : \|A\|_0 \leq s\}$.

In order to solve Eq. (A.90), we minimize for each k

$$\inf_{\Phi \in \mathcal{S}_k} \|U - \Phi\|_F^2 = \inf_{\Phi \in \mathcal{S}_k} \sum_{i,j} (U_{i,j} - \Phi_{i,j})^2 \quad \text{(A.92)}$$

individually. The problem (A.92) is minimized by $\Phi^{(k)*}$ with

$$\Phi^{(k)*}_{i,j} = \begin{cases} U_{i,j} \ , & \text{if } (i,j) \in S_k \\ 0 \ , & \text{else} \end{cases} \ . \quad \text{(A.93)}$$

Thus, the minimum of Eq. (A.92) for a $k \in \{1, \dots, r\}$ is given by

$$\|U - \Phi^{(k)*}\|_F^2 = \sum_{(i,j) \notin S_k} U_{i,j}^2 \quad \text{(A.94)}$$

Equation (A.94) leads to the solution of Eq. (A.90), given by

$$\min_{k \in \{1, \dots, r\}} \sum_{(i,j) \notin S_k} U_{i,j}^2 = \max_{k \in \{1, \dots, r\}} \sum_{(i,j) \in S_k} U_{i,j}^2 \quad \text{(A.95)}$$

which is reached by choosing k such that $S_k = \operatorname{TOP}_s(U)$. □

[A.5] By convexity of f it is therefore a global minimum.

Lemma 2. Let $m \leq M$, then for each $\Phi \in \mathbb{R}^{m\times n}$ there exists a $\mathbf{F} \in \mathbb{R}^{m\times M}$ and a $R \in \mathbb{R}^{M\times n}$ with $\|R\|_0 = \|\Phi\|_0$ and $\mathbf{F}\cdot R = \Phi$.

Proof of Lemma 2. Let $\Phi = (\Phi_{i,j})_{i,j} \in \mathbb{R}^{m\times n}$ be given. Now, we define $R \in \mathbb{R}^{M\times n}$ and $\mathbf{F} \in \mathbb{R}^{m\times M}$ via

$$R_{i,j} = \begin{cases} \Phi_{i,j} & \text{, if } i \leq m \\ 0 & \text{, else} \end{cases} \tag{A.96}$$

and

$$\mathbf{F}_{i,j} = \begin{cases} 1 & \text{, if } i = j \text{ and } j \leq m \\ 0 & \text{, else} \end{cases} . \tag{A.97}$$

By construction of R and $\mathbf{F}$, it holds $\Phi = \mathbf{F}\cdot R$ and $\|R\|_0 = \|\Phi\|_0$. □

Corollary 3. Let $m \leq M$, $\varepsilon_{(1)}$ be the infimum of Eq. (A.83) and $\varepsilon_{(2)}$ be the minimum of Eq. (A.84), respectively. Then it holds $\varepsilon_{(1)} \leq \varepsilon_{(2)}$.

Proof of Corollary 3. By Lemma 1, Eq. (A.84) is always minimized by a $\Phi^* \in \mathbb{R}^{m\times n}$. By using Lemma 2, since $m \leq M$, there exists $\mathbf{F}_0 \in \mathbb{R}^{m\times M}$ and $R_0 \in \mathbb{R}^{M\times n}$ with $\Phi^* = \mathbf{F}_0 \cdot R_0$ and $\|R_0\|_0 = \|\Phi^*\|_0 = s$. Thus, $(\mathbf{F}_0, R_0)$ is feasible for Eq. (A.83) and consequently $\varepsilon_{(1)} \leq \varepsilon_{(2)}$. □

Proof of Theorem 1. **First part of proof with** $\operatorname{supp} R = \operatorname{supp} \Phi^*$**.** W.l.o.g. we assume $\varepsilon_{(1)} = \varepsilon_{(2)}$. By Corollary 3, $\varepsilon_{(1)} \leq \varepsilon_{(2)}$ always holds. If $\varepsilon_{(1)} < \varepsilon_{(2)}$, we would be finished with the proof. Furthermore, we assume w.l.o.g. $m = M$, since otherwise we just fill the corresponding entries in $R_{i,j}$ and $\mathbf{F}_{k,i}$ for index values $m < i \leq M$ and arbitrary j, k with zeros.

Therefore, let $\varepsilon_{(1)} = \varepsilon_{(2)}$. Let $\Phi^* \in \mathbb{R}^{m\times n}$ solve Eq. (A.84). Then, there exists an equivalent feasible point $(\mathbf{F}_0, R_0) \in \mathbb{R}^{m\times m} \times \mathbb{R}^{m\times n}$ for Eq. (A.83), constructed according to Eqs. (A.96) and (A.97) in the proof of Lemma 2. *I.e.*, $\Phi^* = \mathbf{F}_0 \cdot R_0$ and $\|\Phi^*\|_0 = \|R_0\|_0$. By the assumption $\varepsilon_{(1)} = \varepsilon_{(2)}$, $(\mathbf{F}_0, R_0)$ also solves Eq. (A.83). Especially, $\mathbf{F}_0$ defines a *global* minimum of the smooth, convex function

$$f : \mathbb{R}^{m\times m} \to \mathbb{R}, \mathbf{F} \mapsto \|U - \mathbf{F}\cdot R_0\|_F^2 . \tag{A.98}$$

Note, minimizing f is, contrarily to Eq. (A.83), a convex problem.

A necessary, and by convexity of f even sufficient, condition for $\mathbf{F}_0$ to minimize f is given by

$$\left.\frac{\partial f}{\partial \mathbf{F}}\right|_{\mathbf{F}=\mathbf{F}_0} = 0 \in \mathbb{R}^{m\times m} . \tag{A.99}$$

It holds

$$\frac{\partial f}{\partial \mathbf{F}} = \frac{\partial}{\partial \mathbf{F}}\|U - \mathbf{F}\cdot R_0\|_F^2 = 2\cdot(\mathbf{F}\cdot R_0 - U)\cdot R_0^T . \tag{A.100}$$

By combining Eqs. (A.99) and (A.100), we get a necessary condition for $\mathbf{F}_0$ yielding a minimum for f, given by

$$(U - \mathbf{F}_0 \cdot R_0)\cdot R_0^T = 0 . \tag{A.101}$$

Consequently, Eq. (A.101) is a necessary condition for $(\mathbf{F}_0, R_0)$ to define the minimum for Eq. (A.83). From the construction of $\mathbf{F}_0$ and R_0 we know that $\mathbf{F}_0 \cdot R_0 = \Phi^*$, $\mathbf{F}_0 = \mathrm{id}_{\mathbb{R}^m}$ and $R_0 = \Phi^*$. By Lemma 1, Φ^* is given by $\Phi^*_{i,j} = \chi_{\{(i,j)\in \mathrm{TOP}_s(U)\}} \cdot U_{i,j}$ with the *characteristic function* $\chi_{\{\cdot\}}$. Combining this with Eq. (A.101) leads to the necessary condition

$$\hat{U}\cdot\check{U}^T = 0 \tag{A.102}$$

with

$$\hat{U}_{i,j} = \begin{cases} U_{i,j}, & \text{if } (i,j) \notin \mathrm{TOP}_s(U) \\ 0, & \text{if } (i,j) \in \mathrm{TOP}_s(U) \end{cases} \tag{A.103}$$

and

$$\check{U}_{i,j} = \begin{cases} U_{i,j}, & \text{if } (i,j) \in \mathrm{TOP}_s(U) \\ 0, & \text{if } (i,j) \notin \mathrm{TOP}_s(U) . \end{cases} \tag{A.104}$$

In the following we will compute an upper bound δ for the probability $\mathbb{P}(\hat{U}\cdot\check{U}^T = 0)$. By using the fact that $\hat{U}\cdot\check{U}^T = 0$ is a necessary condition for $(\mathbf{F}_0, R_0)$ being a minimizer to Eq. (A.83), which is equivalent to $\varepsilon_{(1)} = \varepsilon_{(2)}$, we finally get

$$\begin{aligned} \mathbb{P}(\varepsilon_{(1)} < \varepsilon_{(2)}) &= 1 - \mathbb{P}(\varepsilon_{(1)} \geq \varepsilon_{(2)}) && \text{(A.105)} \\ &= 1 - \mathbb{P}(\varepsilon_{(1)} = \varepsilon_{(2)}) && \text{(A.106)} \\ &\geq 1 - \mathbb{P}(\hat{U}\cdot\check{U}^T = 0) . && \text{(A.107)} \end{aligned}$$

Thus, the last step is to find an upper bound $\delta \geq \mathbb{P}(\hat{U}\cdot\check{U}^T = 0)$. In order to compute δ, we have a closer look on $\hat{U}\cdot\check{U}^T$. It holds

$$\begin{aligned} (\hat{U}\cdot\check{U}^T)_{i,j} &= \sum_{k=1}^{n} \hat{U}_{i,k}\check{U}_{j,k} && \text{(A.108)} \\ &= \begin{cases} \sum_{k\in\mathcal{T}_{i,j}} U_{i,k}U_{j,k}, & \text{if } \mathcal{T}_{i,j} \neq \emptyset \\ 0, & \text{else} \end{cases} , && \text{(A.109)} \end{aligned}$$

where for each $(i,j) \in \{1,\dots,m\}^2$,

$$\begin{aligned} \mathcal{T}_{i,j} := \{k \in \{1,\dots,n\} :& (i,k) \notin \mathrm{TOP}_s(U) \text{ and} \\ & (j,k) \in \mathrm{TOP}_s(U)\} . \end{aligned} \tag{A.110}$$

Now assume $S \subset (\{1,\dots,m\}\times\{1,\dots,n\})^2$ with $S \neq \emptyset$ to be given, then

$$\mathbb{P}\Big(\sum_{(i_1,j_1),(i_2,j_2)\in S} U_{i_1,j_1}\cdot U_{i_2,j_2} = 0\Big) = 0 . \tag{A.111}$$

This equality holds since for each $S \subset (\{1,\dots,m\} \times \{1,\dots,n\})^2$ with $S \neq \emptyset$, $\sum_{(i_1,j_1),(i_2,j_2)\in S} U_{i_1,j_1} \cdot U_{i_2,j_2}$ follows a continuous probability distribution.

Consequently,

$$\begin{aligned}
& \mathbb{P}(\hat{U} \cdot \check{U}^T = 0) && \text{(A.112)}\\
= \; & \mathbb{P}(\forall (i,j) : \mathcal{T}_{i,j} = \emptyset \\
& \quad \vee (\mathcal{T}_{i,j} \neq \emptyset \wedge \sum_{k \in \mathcal{T}_{i,j}} U_{i,k} \cdot U_{j,k} = 0)) && \text{(A.113)}\\
\leq \; & \mathbb{P}((\forall (i,j) : \mathcal{T}_{i,j} = \emptyset) \\
& \quad \vee (\exists S \subset (\{1,\dots,m\} \times \{1,\dots,n\})^2 \setminus \emptyset : \\
& \quad \sum_{(i_1,j_1),(i_2,j_2)\in S} U_{i_1,j_1} \cdot U_{i_2,j_2} = 0)) && \text{(A.114)}\\
\leq \; & \mathbb{P}(\forall (i,j) : \mathcal{T}_{i,j} = \emptyset) \\
& \quad + \left(\sum_{\substack{S \subset (\{1,\dots,m\} \times \{1,\dots,n\})^2 \\ S \neq \emptyset}} \right. \\
& \quad \left. \mathbb{P}(\sum_{(i_1,j_1),(i_2,j_2)\in S} U_{i_1,j_1} \cdot U_{i_2,j_2} = 0) \right) && \text{(A.115)}\\
= \; & \mathbb{P}(\forall (i,j) : \mathcal{T}_{i,j} = \emptyset) \, , && \text{(A.116)}
\end{aligned}$$

where the inequality (A.115) uses the subadditivity of probability measures and the final equality (A.116) is achieved by using Eq. (A.111). By looking at the definition of $\mathcal{T}_{i,j}$, we see that $\forall (i,j) : \mathcal{T}_{i,j} = \emptyset$ only happens if for each $k \in \{1,\dots,n\}$ either

$$\forall i \in \{1,\dots,m\} : (i,k) \in \mathrm{TOP}_s(U) \qquad \text{(A.117)}$$

or

$$\forall i \in \{1,\dots,m\} : (i,k) \notin \mathrm{TOP}_s(U) \qquad \text{(A.118)}$$

holds true. Otherwise, if $k \in \{1,\dots,n\}$, $i,j \in \{1,\dots,m\}$ exist with $(i,k) \notin \mathrm{TOP}_s(U)$ and $(j,k) \in \mathrm{TOP}_s(U)$, obviously $\mathcal{T}_{i,j} \neq \emptyset$. This shows, that $\varepsilon_{(1)} = \varepsilon_{(2)}$ is only possible in the trivial case, where each of the n filters (with m coefficients) is either completely pruned or not pruned at all.

Therefore, we need to compute the probability

$$\begin{aligned}
\mathbb{P}(\forall k : \quad & (\forall i : (i,k) \in \mathrm{TOP}_s(U)) \\
\vee \quad & (\forall i : (i,k) \notin \mathrm{TOP}_s(U))) \, .
\end{aligned} \qquad \text{(A.119)}$$

Due to the i.i.d. assumption of the $U_{i,k}$, all (i,k) have the same probability of being in $\mathrm{TOP}_s(U)$. Thus, deciding $(i,k) \in \mathrm{TOP}_s(U)$ or $(i,k) \notin \mathrm{TOP}_s(U)$ for all $(i,k) \in \{1,\dots,m\} \times \{1,\dots,n\}$ together can equivalently be modeled with choosing a subset of size s from a set of size $m \cdot n$, where each subset has the same probability of being sampled, *i.e.* with probability $\frac{1}{\binom{m\cdot n}{s}}$.

Furthermore, Eq. (A.119) is only possible *if* $s = \alpha \cdot m$ for some $\alpha \in \mathbb{N}$. Otherwise, there needs to exists at least one k,i,j such that $(i,k) \notin \mathrm{TOP}_s(U)$ and $(j,k) \in \mathrm{TOP}_s(U)$. Assuming $s = \alpha \cdot m$ for some α, there exist exactly $\binom{n}{\alpha}$ different choices to find α many k that satisfy $\forall i : (i,k) \in \mathrm{TOP}_s(U)$ which, by the discussion above, all have similar probability.

Altogether, the probability Eq. (A.119) is given by

$$\delta = \begin{cases} 0 & , \text{ if } s \not\equiv 0 (\mathrm{mod}\ m) \\ \frac{\binom{n}{\frac{s}{m}}}{\binom{m\cdot n}{s}} & , \text{ if } s \equiv 0 (\mathrm{mod}\ m) \end{cases} . \qquad \text{(A.120)}$$

Finally,

$$\begin{aligned}
\mathbb{P}(\hat{U} \cdot \check{U}^T = 0) \leq \; & \mathbb{P}(\forall (i,j) : \mathcal{T}_{i,j} = \emptyset) && \text{(A.121)}\\
= \; & \mathbb{P}(\forall k : (\forall i : (i,k) \in \mathrm{TOP}_s(U)) \\
& \quad \vee (\forall i : (i,k) \notin \mathrm{TOP}_s(U))) && \text{(A.122)}\\
\leq \; & \delta \, . && \text{(A.123)}
\end{aligned}$$

Using the estimation in Eq. (A.107), we finally get

$$\mathbb{P}(\varepsilon_{(1)} < \varepsilon_{(2)}) \geq 1 - \mathbb{P}(\hat{U} \cdot \check{U}^T) \geq 1 - \delta \, , \qquad \text{(A.124)}$$

which finishes the first part of the proof where $\mathrm{supp}\, R$ is fixed to be equal to $\mathrm{supp}\, \Phi^*$.

Second part of proof with arbitrary $\mathrm{supp}\, R$**.** As shown in the first part of the proof, Eq. (A.119) is a necessary condition for $\varepsilon_{(1)} = \varepsilon_{(2)}$. This means that all columns of Φ^* are either $\Phi^*_{:,k} = 0 \in \mathbb{R}^m$ or $\|\Phi^*_{:,k}\|_0 = m$ which we therefore will assume from now on.[A.6]

By assumption, $0 < s$ and therefore, there exists a k with $\|\Phi^*_{:,k}\|_0 = m$. Now, set $\hat{\mathbf{F}}_{:,1} = \Phi^*_{:,k}$ and $\hat{\mathbf{F}}_{:,j} = (\mathbf{F}_0)_{:,j}$ for all other j. Then, with $\mathbb{P} = 1$, $\hat{\mathbf{F}}$ still forms a basis.

Setting $\hat{R}_{i,k} = \delta_{i,1}$ for all $i \in \{1,\dots,m\}$ yields $(\hat{\mathbf{F}} \cdot \hat{R})_{:,k} = U_{:,k} = \Phi^*_{:,k}$. For all other $j \neq k$ with $\|\Phi^*_{:,j}\|_0 = m$ there exists a $\hat{R}_{:,j}$ with $(\hat{\mathbf{F}} \cdot \hat{R})_{:,j} = \Phi^*_{:,j} = U_{:,j}$ since $\hat{\mathbf{F}}$ forms a basis.

Setting the remaining $\hat{R}_{:,j} = 0$ leads to $\hat{\mathbf{F}} \cdot \hat{R} = \Phi^*$ and $\|\hat{R}\|_0 \leq \|\Phi^*\|_0 - (m-1) < \|\Phi^*\|_0$. The last inequality holds, since $m > 1$ is assumed.

Finally, one of the (at least) remaining $m-1$ coefficients which were not spend up to now can be used to better approximate one column of U which is completely zeroed in Φ^*. Such a column j_0 must fulfill $U_{:,j_0} \neq 0$ and $\Phi^*_{:,j_0} = 0$. Since $s < m \cdot n$ and $U_{i,j}$ i.i.d. $\mathcal{N}(0,1)$, such a column j_0 exists with $\mathbb{P} = 1$. Since $\hat{\mathbf{F}}$ forms a basis, we can find some l, λ_l such that

$$\|U_{:,j_0} - \lambda_l \hat{\mathbf{F}}_{:,l}\|_2 < \|U_{:,j_0}\|_2 \, . \qquad \text{(A.125)}$$

[A.6] For a matrix $A \in \mathbb{R}^{d_1 \times d_2}$, the j^{th} column $A_{:,j}$ is given by $(A_{i,j})_i \in \mathbb{R}^{d_1}$.

Setting $\hat{R}_{l,j_0} = \lambda_l$ leads to

$$\|U - \hat{\mathbf{F}} \cdot \hat{R}\|_F^2 = \|U - \Phi^*\|_F^2 \tag{A.126}$$

$$- (\|U_{:,j_0}\|_2^2 - \|U_{:,j_0} - \lambda_l \hat{\mathbf{F}}_{:,l}\|_2^2) \tag{A.127}$$

$$< \|U - \Phi^*\|_F^2 \,, \tag{A.128}$$

which finishes the proof. □

References in the Appendix

[A.1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. iv

[A.2] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. xii

[A.3] Albert Cohen, Wolfgang Dahmen, and Ronald Devore. Compressed sensing and best k -term approximation. *Journal of the American Mathematical Society*, 22(1):211–231, 2009. iii

[A.4] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *CoRR*, abs/1907.04840, 2019. xii

[A.5] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006. iii

[A.6] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. ix, xii, xiii

[A.7] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. ix, xi, xii, xiii

[A.8] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. ix, xiii

[A.9] Stuart Geman, Elie Bienenstock, and Rene Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):158, 1992. i

[A.10] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. xiii

[A.11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, 2015. x

[A.12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. xiv

[A.13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015. xii

[A.14] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012. http://www.cs.toronto.edu/~kriz/cifar.html. xiii

[A.15] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems 4*. 1992. i

[A.16] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H.S. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019. iii, vii, ix, xii, xiii

[A.17] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? In-time over-parameterization in sparse training. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. xii, xiii

[A.18] Decebal Mocanu, Elena Mocanu, Peter Stone, Phuong Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9, 2018. ix, xii, xiii

[A.19] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. xii

[A.20] NVIDIA, Pter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. xiii

[A.21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. 2019. iv, xiii

[A.22] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020. ix, xiii

[A.23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. xiv

[A.24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. iv, xiv

[A.25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. xiii

[A.26] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems 33*, 2020. vii, ix, xii, xiii

[A.27] W.F. Tinney and J.W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55(11):1801–1809, 1967. iii, vii

[A.28] M. Unser and T. Blu. Mathematical properties of the jpeg2000 wavelet filters. *IEEE Transactions on Image Processing*, pages 1080–1090, 2003. iii

[A.29] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020. vii, ix, x, xii, xiii

[A.30] Paul Wimmer, Jens Mehnert, and Alexandru Paul Condurache. FreezeNet: Full performance by reduced storage costs. In *Proceedings of the Asian Conference on Computer Vision*, 2020. iii

[A.31] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520540, 1987. iv