S1 Supplementary Material for FreezeNet: Full Performance by Reduced Storage Costs

S1.1 Training Hyperparameters for MNIST and CIFAR-10

Our results in the main body of the text are benchmarked against SNIP [S.2], since we share its feature selection and it is also a one-shot method, applied before training. For the comparison we use their training schedule and do not tune any hyperparameters. They take a batch size of 100 (MNIST) or 128 (CIFAR) and SGD with momentum parameter 0.9 [S.3] as optimizer. The initial learning rate is set to 0.1 and divided by ten at every 25k/30k optimization steps for MNIST and CIFAR, respectively. For regularization, weight decay with coefficient $5 \cdot 10^{-4}$ is applied. The overall number of training epochs is 250.

S1.2 Tiny ImageNet Experiment

In addition to the MNIST and CIFAR experiments, we also tested FreezeNet on a ResNet34 [S.1] on the Tiny ImageNet classification task [S.1]. The ResNet34 architecture is shown in Table S5. The Tiny ImageNet dataset consists of 64×64 RGB images with 200 different classes. Those classes have 500 training images, 50 validation images and 50 test images, each. For training, we use the same data augmentation as for CIFAR, i.e. random horizontal flipping and translations up to 4 pixels. We train the network for 100 epochs with an initial learning rate of 0.01. The learning rate is decayed by a factor 10 after epoch 30, 60 and 80. As optimizer, again SGD with momentum parameter 0.9 is used. Also, weight decay with coefficient $5 \cdot 10^{-4}$ is applied. The networks are initialized with a Kaiming-uniform initialization.

[S.1] https://tiny-imagenet.herokuapp.com/ with training schedule adapted from https://github.com/ganguli-lab/Synaptic-Flow.

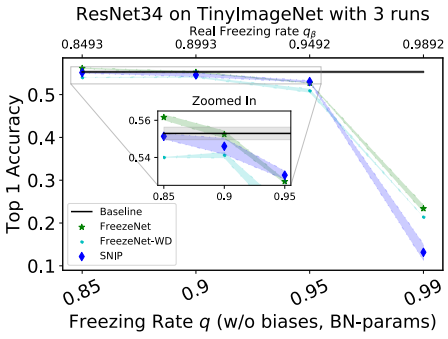

Fig. S1. Comparison of FreezeNet, FreezeNet-WD and SNIP for a ResNet34 trained on Tiny ImageNet.

In Figure S1, the results for FreezeNet, FreezeNet-WD and SNIP are shown. The mean and standard deviation for every freezing rate and method are each calculated for three runs with different random seeds. We see, that FreezeNet can be used to train a ResNets successfully. FreezeNet has the same accuracy than the baseline method while training only 10% of its weights. For all rates, FreezeNet achieves better or equal results than SNIP. Also, FreezeNet's results are more stable, shown by the slimmer standard deviation bands. Contrarily to the CIFAR-10 experiment, FreezeNet-WD shows worse results than SNIP for lower freezing rates. Reaching good results with a FreezeNet on ResNet34 needs a higher rate of trainable parameters than on VGG16, see Table 3.

S1.3 GraSP

The GraSP criterion [S.4] is based on the idea of preserving, or even increasing, the pruned network's gradient flow compared to the unpruned counterpart. Thus,

$$\Delta L(W) := \left\| \frac{\partial L}{\partial W} \right\|_2^2 \tag{S.1}$$

is tried to be maximized with the masked weights $m \odot W$. For that purpose,

$$\Delta L(m \odot W) \approx \Delta L(W) - W^T \cdot H_L \cdot \frac{\partial L}{\partial W} + (m \odot W)^T \cdot H_L \cdot \frac{\partial L}{\partial W} \tag{S.2}$$

is approximated via the first order Taylor series, with the Hessian $H_L := \frac{\partial^2 L}{\partial W^2} \in \mathbb{R}^{D \times D}$. As $\Delta L(W) - W^T \cdot H_L \cdot \frac{\partial L}{\partial W}$ is fix, the *importance score* for all weights $W \in \mathbb{R}^D$ is computed as

$$S(W) := W \odot \left(H_L \cdot \frac{\partial L}{\partial W} \right) \tag{S.3}$$

and the weights with the $1 - q$ highest importance scores are trained, the other ones are pruned. Contrarily to the saliency score (1), the importance score (S.3) takes the sign into account. Pruning with the GraSP criterion is applied, as SNIP and FreezeNet, once before the training starts [S.4]. Thus, GraSP pruned networks also have sparse gradients during training.

Training Setup for Result in Table 4 Different freezing rates are displayed in Table 4, as we first tried to find results in the corresponding papers with freezing rates approximately $q = 0.99$. If no result for such a freezing rate was given, we report the result with the closest accuracy to FreezeNet's performance for $q = 0.99$.

We used the experimental setup described in Section 4 with hyperparameters from Section S1.1. The official method to calculate the GraSP score was used.[S.2] Moreover, learning rates $2^n \cdot 0.1$ with $n \in \{-4, \dots, 4\}$ were tested for a split of

[S.2] https://github.com/alecwangcq/GraSP.

Algorithm S1 FreezeNet with Reinitialization after Computation of Freezing Mask

Require: Freezing rate q, initial parametrization $\Theta_0 = W_0 \cup B_0$, reinitialized parameters $\Theta_1 = W_1 \cup B_1$, network f, loss function L

1: Calculate saliency score $g \in \mathbb{R}^{|W_0|}$ according to equation (1) with f_{Θ_0} for one training batch
2: Define freezing mask $m \in \mathbb{R}^{|W_0|}$
3: Calculate freezing threshold ε, where ε denotes the $\lfloor (1-q) \cdot |W_0| \rfloor$-highest magnitude of g
4: Set $m_k = 0$ if $|g_k| < \varepsilon$ else $m_k = 1$
5: Set networks parameters to Θ_1, i.e. use f_{Θ_1} for training
6: Start training with forward propagation as usual but backpropagate gradient $m \odot \frac{\partial L}{\partial W_1}$ for weights and $\frac{\partial L}{\partial B_1}$ for biases

training/validation images of 9/1, 19/1 and 3/1. Each of these 27 combinations was run 5 times with a different initialization. For each combination, the network with the best validation accuracy was tested at its early stopping time. The best result was reported for the learning rate $\lambda = 0.1$ with a split of training/validation images of 19/1 — the same setup as used for the best FreezeNet result. The reported test accuracy for the GraSP pruned network is given by 98.9%.

S1.4 Reinitialization Tests

Up to now it is unanswered how the reinitialization of a network's weights after the calculation of the saliency scores affects the trainability of this network. To check this, we modify Algorithm 1 by adding a reinitialization step after the computation of the saliency score. FreezeNets with reinitializations are introduced in Algorithm S1. We have tested a LeNet-5-Caffe baseline architecture for MNIST on Algorithm S1. Again we follow the training setup from Sections 4 together with S1.1.

First we tested combinations of initializing and reinitializing a FreezeNet with the Xavier-normal and the Kaiming-uniform initialization schemes. We name the network without reinitialized weights FreezeNet-K for a Kaiming-uniform initialized network, or FreezeNet-X for a Xavier-normal initialized one. The reinitialized networks are denoted as FreezeNet-A-B for A,B $\in$ {K,X} where A stands for the initialization used for finding the freezing mask and B for the reinitialization, applied before training. The left plot in Figure S2 compares FreezeNet-X with FreezeNet-X-X and FreezeNet-K-X. This graph shows, that reinitializations do not significantly change the performance of FreezeNets. It also does not seem to make a difference if the probability distribution used for the reinitialization differs from the one used to calculate the freezing mask, examined through FreezeNet-K-X and FreezeNet-X-X. Similar results are reported for the comparison of FreezeNet-K with FreezeNet-K-K and FreezeNet-X-K, shown in the left part of Figure S3.

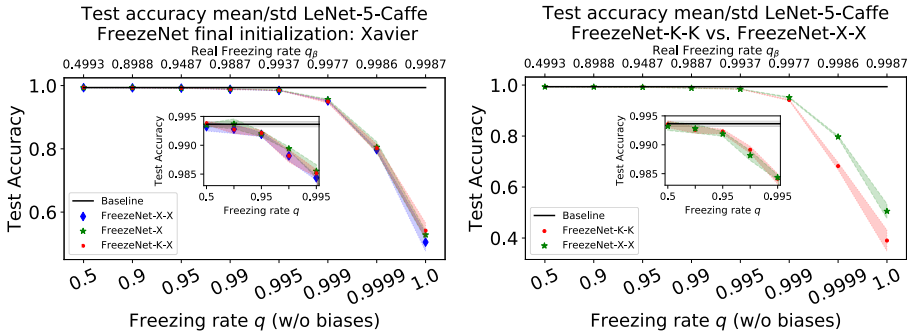

Fig. S2. LeNet-5-Caffe baseline architecture compared to a FreezeNet-X, a FreezeNet-X-X and a FreezeNet-K-X in the left plot. On the right side we compare a FreezeNet-X-X with a FreezeNet-K-K. Inserted: Zoomed in versions of the plots.

In Figure S3, right plot, various other reinitialization methods are tested on a Xavier-normal initialized FreezeNet. We can see, that the variance scaling reinitialization methods Xavier-normal, Kaiming-uniform and $\texttt{pm}_\sigma$ lead to the same results. The $\texttt{pm}_\sigma$ initialization scheme is introduced in Section 4.5 in the main body of the text. Using other, non-variance scaling methods as constant reinitialization (either with all values 1 or the layers variance σ) or drawing weights i.i.d. from $\mathcal{N}(0, 1)$ generates networks which cannot be trained at all.

The right plot in Figure S2 shows that the initialization, used after the freezing mask is computed, is important to solve the MNIST task successfully for high freezing rates. The Kaiming init- and reinitialization, FreezeNet-K-K, performs slightly better for the lower freezing rates and is outperformed by FreezeNet-X-X for the higher ones. Without reinitialization, a similar behaviour for low and high freezing rates can be seen in Figure 3 — right plot.

Summarized, using an appropriate initialization scheme for the weights, after the freezing mask is computed, is essential for achieving good results with FreezeNets. Based on our experiments we suggest initializing (and reinitializing, if wanted) FreezeNets with variance scaling methods.

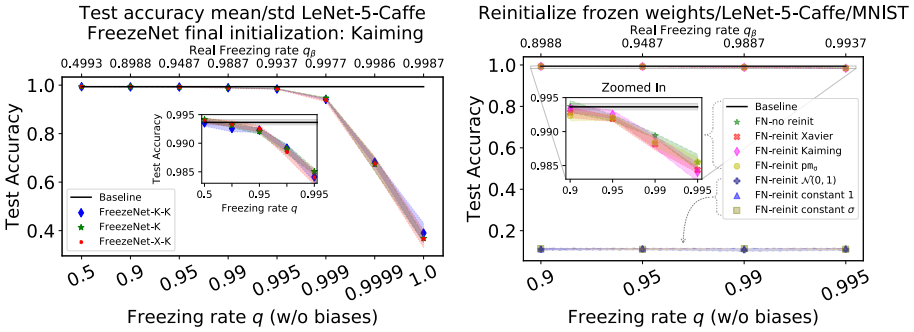

Fig. S3. Left: LeNet-5-Caffe baseline architecture with a corresponding FreezeNet-K, a FreezeNet-K-K and a FreezeNet-X-K. Right: Comparison of different reinitializations for a FreezeNet (FN) with LeNet-5-Caffe baseline architecture with Xavier-normal initialization. Inserted: Zoomed in versions of the plots.

S1.5 Learning Rate Experiments

SGD is used in our experiments, thus we also tested a broad range of learning rates for different freezing rates. This test was done with the same setup as described in Sections 4 and S1.1. The results for a LeNet-5-Caffe baseline architecture on the MNIST classification task are shown in Figures S4 and S5. In order to cover a broad range of learning rates, we used $2^n \cdot \lambda_0, n \in \{-4, -3, \ldots, 4\}$ and $\lambda_0 = 0.1$ as learning rates. All learning and freezing rate combinations were trained with three different random initializations. The learning rates with the best results are shown in the left part of Figure S4. For almost any freezing rate, the learning rate 0.1 works best. Thus, FreezeNets do not need expensive learning rate searches but can use a learning rate performing well on the baseline architecture. But optimizing the learning rate can improve FreeNets' performance beyond question. Another conclusion we want to highlight is that higher learning rates can be applied for higher freezing rates. Even if they do not work well for smaller freezing rates, as the example of $\lambda = 0.2$ in the left part of Figure S4 shows. The same holds for learning rates bigger than 0.2, which require even higher freezing rates to lead to satisfying results, as shown in the right part of Figure S5. For high freezing rates, using higher learning rates can lead to better and faster training, as shown in Figure S4, right side.

The results for the learning rate search for the CIFAR10 task with a VGG16-D are shown in Figure S6. Again, $\lambda_0 = 0.1$ performs best for most of the freezing rates and is only slightly improved for some freezing rates by $\lambda = 0.2$ and $\lambda = 0.05$.

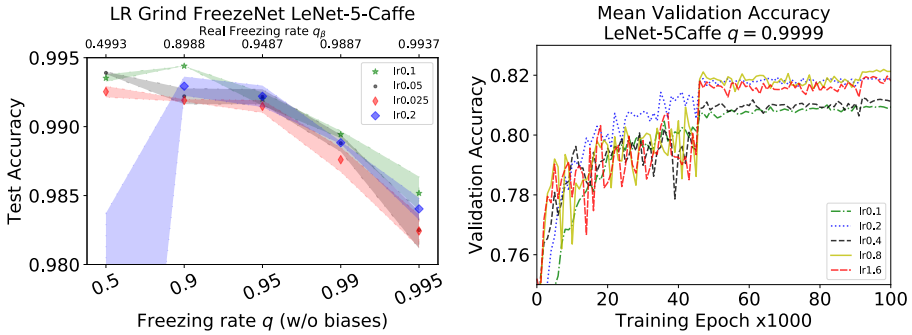

Fig. S4. Learning rate tests for a FreezeNet with a LeNet-5-Caffe baseline. The left part shows the mean test accuracy for the best performing learning rates. The right plot shows the mean validation accuracy for three training runs recorded over the first 100k epochs for different learning rates.

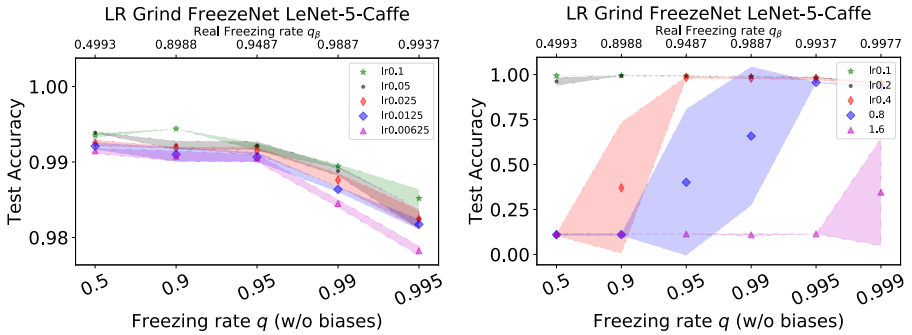

Fig. S5. Learning rate tests for a FreezeNet with a LeNet-5-Caffe baseline. The left part shows the mean test accuracy for the learning rates $\lambda \leq 0.1$. The right plot shows the mean validation accuracy for the learning rates $\lambda \geq 0.1$.

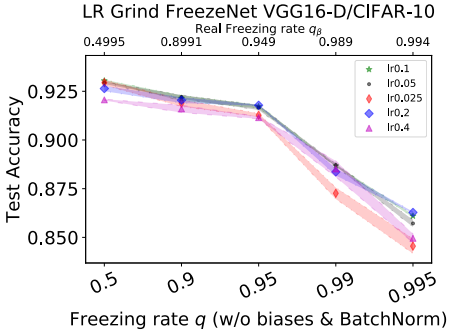

Fig. S6. Best performing learning rates for a FreezeNet with a VGG16-D baseline on the CIFAR-10 classification task.

S1.6 Figures for LeNet-5-Caffe on MNIST

Figure S7 shows the comparison of FreezeNet and SNIP over a broad range of freezing rates, discussed in Section 4.1. Here, the networks are trained and evaluated as described in Section 4 with hyperparameters from Section S1.1.

The training progress of FreezeNet's result, reported in Table 4, is shown in Figure S8.

S1.7 Network Architectures

Figures S9, S10 and S11 visualize the used LeNet-300-100, LeNet-5-Caffe and VGG16-D network architectures, respectively. The ResNet34 architecture can be looked up in Table S5 together with Figure S12.

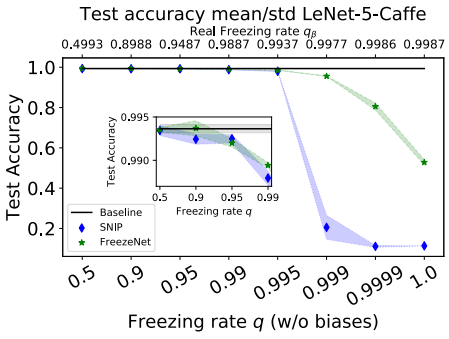

Fig. S7. Comparison SNIP and FreezeNet for the MNIST classification task and a LeNet-5-Caffe baseline architecture. The inserted plot is a zoomed version.

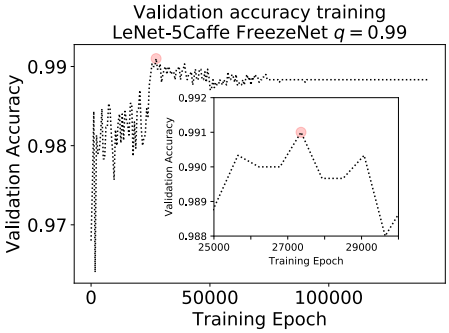

Fig. S8. Training of FreezeNet for freezing rate $q = 0.99$ and baseline architecture LeNet-5-Caffe. Training procedure is done as described in Section S1.1 with hyperparameters from Section S1.1 but with a split 19/1 of training and validation images. This plot shows the run with the best validation accuracy out of five tries. The corresponding test accuracy equals 99.1%, calculated with the weights stored in the early stop epoch, as reported in Table 4. The red circle highlights the epoch where early stopping occurs. The inserted plot is a zoomed version showing the early stopping epoch.

LeNet-300-100 Architecture

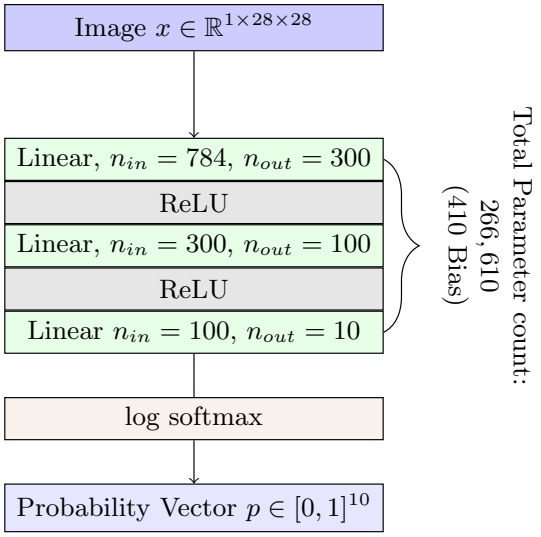

Fig. S9. Architecture of the used LeNet-300-100. In front of the first layer, the feature map $x \in \mathbb{R}^{1\times28\times28}$ is flattened to $\hat{x} \in \mathbb{R}^{784}$. For linear layers, n_{in} and n_{out} denote the number of incoming and outgoing neurons, respectively.

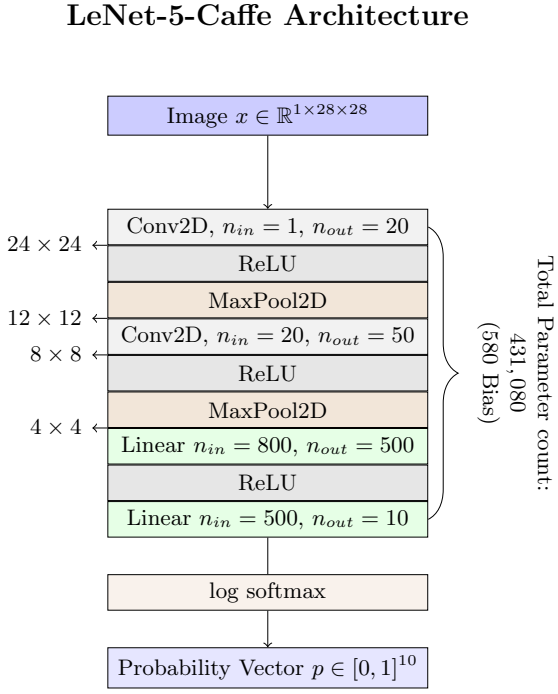

Fig. S10. Architecture of the used LeNet-5-Caffe. All 2D-convolutional layers have kernel size 5×5, 1×1 stride and no zero padding. A max-pooling layer has kernel size 2×2, stride 2×2 and dilation 1×1. Before entering the first linear layer, the feature map $\hat{x} \in \mathbb{R}^{50\times4\times4}$ is flattened to $\hat{\hat{x}} \in \mathbb{R}^{800}$. The resolutions left to the blocks denote the resolution of the feature maps, processed by the corresponding layers. For convolutional layers, n_{in} and n_{out} denote the number of incoming and outgoing channels, respectively. For linear layers, n_{in} and n_{out} denote the number of incoming and outgoing neurons, respectively.

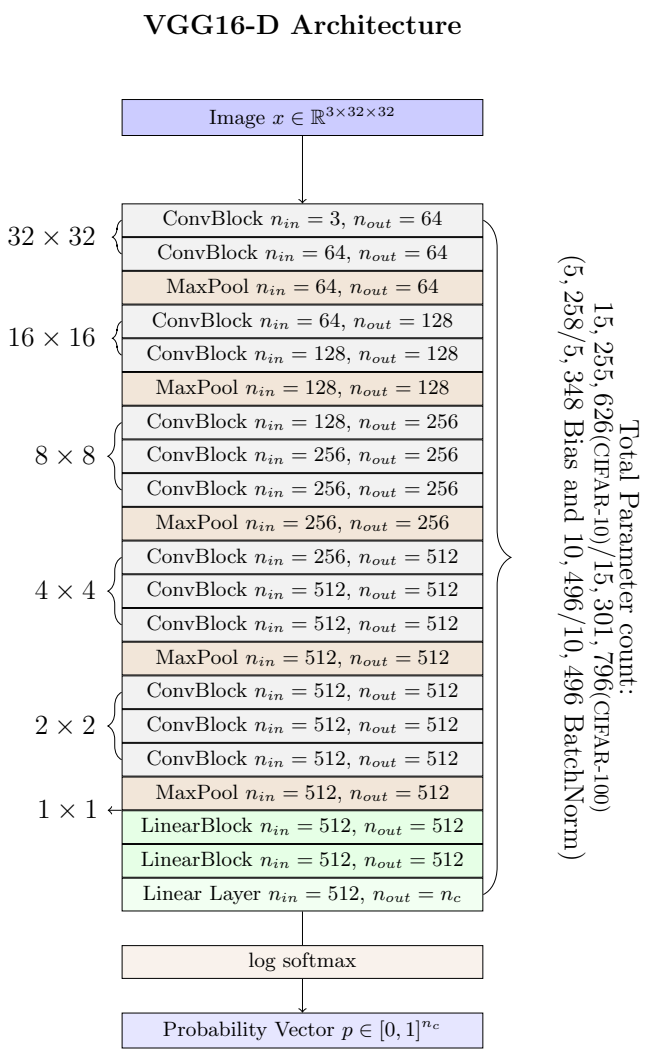

Fig. S11. Architecture of VGG16-D. Here, $n_c \in \{10, 100\}$ equals the numbers of classes for the given classification task. For CIFAR-10, $n_c = 10$ and for CIFAR-100 we have $n_c = 100$. A ConvBlock consists of a 2D-convolutional layer with kernel size 3×3, and 1×1 stride and padding. Each convolution is followed by a 2D-Batch Normalization Layer and a ReLU activation function. The max-pooling layer has kernel size 2×2, stride 2×2 and dilation 1×1. Before entering the first LinearBlock, the feature map $\hat{x} \in \mathbb{R}^{512\times1\times1}$ is flattened to $\hat{\hat{x}} \in \mathbb{R}^{512}$. A LinearBlock consist of a fully connected layer followed by a 1D-Batch Normalization Layer and a ReLU activation function. The resolution left to the blocks denotes the resolution of the feature maps, processed by the corresponding blocks. For Convolutional blocks, n_{in} and n_{out} denote the number of incoming and outgoing channels, respectively. For linear blocks and layers, n_{in} and n_{out} denote the number of incoming and outgoing neurons, respectively.

Table S5. ResNet34 with `ResBlocks`, shown in Figure S12. The kernel size is given by k. For Convolutional layers and `ResBlocks`, n_{in} and n_{out} denote the number of incoming and outgoing channels, respectively. For the linear layer, n_{in} and n_{out} denote the number of incoming and outgoing neurons, respectively. Before entering the linear layer, the feature map $x \in \mathbb{R}^{512\times1\times1}$ is flattened to $\hat{x} \in \mathbb{R}^{512}$. A ResNet34 consists of $21,383,816$ parameters in total. Thereof, $21,366,464$ weights, 200 biases and $17,152$ BatchNorm parameters.

Module	Output Size	Repeat	n_{in}	n_{out}	k	Stride	Padding	Bias	BatchNorm	ReLU
Conv2D	64×64	$\times1$	3	64	3	1	1	✗	✓	✓
`ResBlock`	32×32	$\times1$	64	64	3	2	1	✗	✓	✓
`ResBlock`	32×32	$\times2$	64	64	3	1	1	✗	✓	✓
`ResBlock`	16×16	$\times1$	64	128	3	2	1	✗	✓	✓
`ResBlock`	16×16	$\times3$	128	128	3	1	1	✗	✓	✓
`ResBlock`	8×8	$\times1$	128	256	3	2	1	✗	✓	✓
`ResBlock`	8×8	$\times5$	256	256	3	1	1	✗	✓	✓
`ResBlock`	4×4	$\times1$	256	512	3	2	1	✗	✓	✓
`ResBlock`	4×4	$\times2$	512	512	3	1	1	✗	✓	✓
AvgPool2D	1×1	$\times1$	512	512	4	0	0	✗	✗	✗
Linear	200	$\times1$	512	200	—	—	—	✓	✗	✗
log softmax	10	$\times1$	200	200	—	—	—	✗	✗	✗

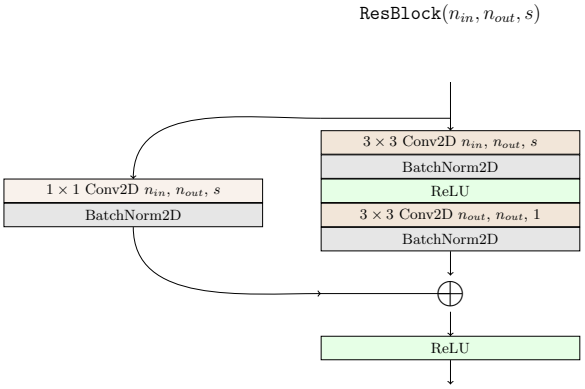

Fig. S12. Architecture of a `ResBlock` with n_{in} input channels, n_{out} output channels and stride s for the first convolution. The second 3×3 conolution has stride 1 and c_{out} in- and output channels. All residual connections are a 1×1 2D Convolution with c_{in} input and c_{out} output channels and stride s followed by a BatchNorm2D layer. All Conv2D layers are initialized without biases.

References Supplementary Material

1. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR **abs/1502.01852** (2015)
2. Lee, N., Ajanthan, T., Torr, P.: SNIP: Single-shot network pruning based on connection sensitivity. In: International Conference on Learning Representations (2019)
3. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning. pp. 1139–1147 (2013)
4. Wang, C., Zhang, G., Grosse, R.: Picking winning tickets before training by preserving gradient flow. In: International Conference on Learning Representations (2020)