FreezeNet: Full Performance by Reduced Storage Costs

Paul Wimmer[1,2*], Jens Mehnert[1*], and Alexandru Condurache[1,2]

[1] Robert Bosch GmbH, Daimlerstrasse 6, 71229 Leonberg, Germany
{Paul.Wimmer,JensEricMarkus.Mehnert,AlexandruPaul.Condurache}@de.bosch.com
[2] University of Luebeck, Ratzeburger Allee 160, 23562 Luebeck, Germany

Abstract. Pruning generates sparse networks by setting parameters to zero. In this work we improve one-shot pruning methods, applied before training, without adding any additional storage costs while preserving the sparse gradient computations. The main difference to pruning is that we do not sparsify the network's weights but learn just a few key parameters and keep the other ones fixed at their random initialized value. This mechanism is called *freezing the parameters*. Those frozen weights can be stored efficiently with a single 32bit random seed number. The parameters to be frozen are determined one-shot by a single for- and backward pass applied before training starts. We call the introduced method *FreezeNet*. In our experiments we show that FreezeNets achieve good results, especially for extreme freezing rates. Freezing weights preserves the gradient flow throughout the network and consequently, FreezeNets train better and have an increased capacity compared to their pruned counterparts. On the classification tasks MNIST and CIFAR-10/100 we outperform SNIP, in this setting the best reported one-shot pruning method, applied before training. On MNIST, FreezeNet achieves 99.2% performance of the baseline LeNet-5-Caffe architecture, while compressing the number of trained and stored parameters by a factor of ×157.

Keywords: Network Pruning · Random Weights · Sparse Gradients · Preserved Gradient Flow · Backpropagation

1 Introduction

Between 2012 and 2018, computations required for deep learning research have been increased by estimated $300,000$ times which corresponds to doubling the amount of computations every few months [36]. This rate outruns by far the predicted one by Moore's Law [18]. Thus, it is important to reduce computational costs and memory requirements for deep learning while preserving or even improving the status quo regarding performance [36].

Model compression lowers storage costs, speeds up inference after training by reducing the number of computations, or accelerates the training which uses less energy. A method combining these factors is *network pruning*. To follow

[*] equal contribution

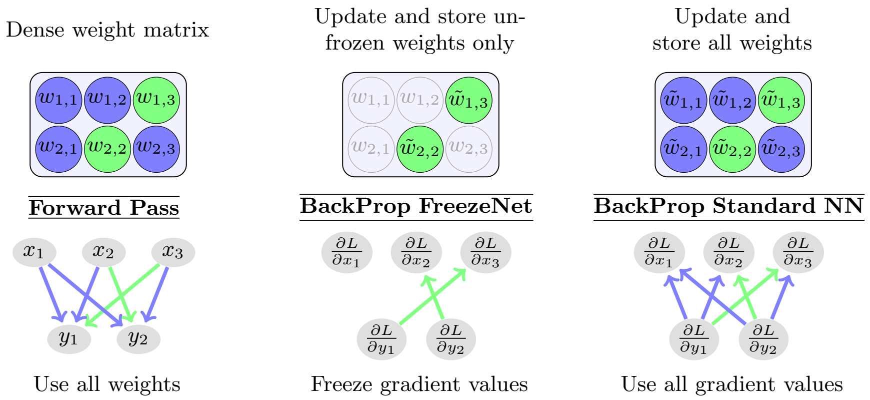
Dense weight matrix
Update and store unfrozen weights only
Update and store all weights
$w_{1,1}$ $w_{1,2}$ $w_{1,3}$ $w_{2,1}$ $w_{2,2}$ $w_{2,3}$
$\tilde{w}_{1,3}$ $\tilde{w}_{2,2}$
$\tilde{w}_{1,1}$ $\tilde{w}_{1,2}$ $\tilde{w}_{1,3}$ $\tilde{w}_{2,1}$ $\tilde{w}_{2,2}$ $\tilde{w}_{2,3}$
Forward Pass
BackProp FreezeNet
BackProp Standard NN
x_1 x_2 x_3 y_1 y_2
$\frac{\partial L}{\partial x_1}$ $\frac{\partial L}{\partial x_2}$ $\frac{\partial L}{\partial x_3}$ $\frac{\partial L}{\partial y_1}$ $\frac{\partial L}{\partial y_2}$
Use all weights
Freeze gradient values
Use all gradient values

Fig. 1. Graphical illustration of FreezeNet and comparison with a standard neural network (NN) for a fully connected NN with neurons x_1, x_2, x_3 and y_1, y_2, and corresponding weights and gradient values. Best viewed in colour.

the call for more sustainability and efficiency in deep learning we improve the best reported pruning method applied before training, SNIP ([29] Single-shot Network Pruning based on Connection Sensitivity), by freezing the parameters instead of setting them to zero.

SNIP finds the most dominant weights in a neural network with a single for- and backward pass, performed once before training starts and immediately prunes the other, less important weights. Hence it is a one-shot pruning method, applied before training. By one-shot pruning we mean pruning in a single step, not iteratively. This leads to sparse gradient computations during training. But if too many parameters are pruned, SNIP networks are not able to train well due to a weak flow of the gradient through the network [39]. In this work we use a SNIP related method for finding the most influential weights in a deep neural network (DNN). We do not follow the common pruning procedure of setting weights to zero, but keep the remaining parameters fixed as initialized which we call *freezing*, schematically shown in Figure 1. A proper gradient flow throughout the network can be ensured with help of the frozen parameters, even for a small number of trained parameters. The frozen weights also increase the network's expressiveness, without adding any gradient computations — compared to pruned networks. All frozen weights can be stored with a single random seed number. We call these partly frozen DNNs *FreezeNets*.

1.1 Contributions of this Paper and Applications

In this work we introduce FreezeNets, which can be applied to any baseline neural network. The key contributions of FreezeNets are:

- Smaller trainable parameter count than one-shot pruning (SNIP), but better results.

- Preservation of gradient flow, even for a small number of trained parameters.
- Efficient way to store frozen weights with a single random seed number.
- More efficient training than the baseline architecture since the same number of gradients as for pruned networks has to be computed.

Theoretically and empirically, we show that a faithful gradient flow, even for a few trainable parameters, can be preserved by using frozen weights. Whereas pruning weights eventually leads to vanishing gradients. By applying weight decay also on the frozen parameters, we modify FreezeNets to generate sparse networks at the end of training. For freezing rates on which SNIP performs well, this modified training generates networks with the same number of non-zero weights as SNIP while reaching better performances.

Due to their sparse gradient computations, FreezeNets are perfectly suitable for applications with a train-inference ratio biased towards training. Especially for research, where networks are trained for a long time and often validated exactly once, FreezeNets provide a good trade-off between reducing training resources and keeping performance. Other applications for FreezeNets are networks that have to be retrained many times due to changing data, as online learning or transfer learning. Since FreezeNets can reduce the number of stored parameters drastically, they are networks cheap to transfer. This could be of interest for autonomous vehicle fleets or internet services. For a given hardware, FreezeNets can be used to increase the size of the largest trainable network since less storage and computations are needed for applying gradient descent.

2 Related Work

2.1 Network Pruning

Pruning methods are used to reduce the amount of parameters in a network [19, 28, 31]. At the same time, the pruned network should perform equally well, or even better, than the reference network. Speeding up training can be achieved by pruning the network at the beginning of training [29] or at early training steps [12, 13]. There are several approaches to prune neural networks. One is penalizing non-zero weights [3, 4, 20] and thus achieving sparse networks. Nowadays, a more common way is given by using magnitude based pruning [12, 13, 17, 19], leading to pruning early on in training [12, 13], on-the-fly during training [17] or at the end of it [19]. These pruned networks have to be fine-tuned afterwards. For high pruning rates, magnitude based pruning works better if this procedure is done iteratively [12, 13], therefore leading to many *train-prune(-retrain)* cycles. Pruning can also be achieved by using *neural architecture search* [10, 22] or adding computationally cheap branches to predict sparse locations in feature maps [9]. The final pruning strategy we want to present is saliency based pruning. In saliency based pruning, the significance of weights is measured with the Hessian of the loss [28], or the sensitivity of the loss with respect to inclusion/exclusion of each weight [25]. This idea of measuring the effect of inclusion/exclusion of weights was resumed in [29], where a differentiable approximation of this criterion was introduced, the SNIP method. Since SNIP's pruning step is applicable with a single for- and backward pass one-shot before training, its computational overload is negligible. The GraSP (Gradient Signal Preservation) [39] method is also a pruning mechanism, applied one-shot before training. Contrarily to SNIP, they keep the weights possessing the best gradient flow at initialization. For high pruning rates, they achieve better results than SNIP but are outperformed by SNIP for moderate ones.

Dynamic sparse training is a strategy to train pruned networks, but give the sparse architecture a chance to change dynamically during training. Therefore, pruning- and regrowing steps have to be done during the whole training process. The weights to be regrown are determined by random processes [2, 30, 41], their magnitude [40] or saliency [7, 8]. An example of the latter strategy is *Sparse Momentum* [7], measuring saliencies via exponentially smoothed gradients. *Global Sparse Momentum* [8] uses a related idea to FreezeNet by not pruning the untrained weights. But the trainable weights can change and the untrained weights are not updated via gradient descent, but with a momentum parameter based on earlier updates. Whereas FreezeNet freezes weights and uses a fixed architecture, thus needs to gauge the best sparse network for all phases of training.

In pruning, the untrained parameters are set to 0 which is not done for FreezeNets, where these parameters are frozen and used to increase the descriptive power of the network. This clearly separates our freezing approach from pruning methods.

2.2 CNNs with Random Weights

The idea of fixing randomly initialized weights in Convolutional Neural Networks (CNNs) was researched in [24], where the authors showed that randomly initialized convolutional filters act orientation selective. In [35] it was shown that randomly initialized CNNs with pooling layers can act inherently frequency selective. Ramanujan et al. [33] showed that in a large randomly initialized base network ResNet50 [21] a smaller, untrained subnetwork is hidden that matches the performance of a ResNet34 [21] trained on ImageNet [6]. Recently, Frankle et al. [14] published an investigation of CNNs with only Batch Normalization [23] parameters trainable. In contrast to their work, we also train biases and chosen weight parameters and reach competitive results with FreezeNets.

A follow-up work of the *Lottery Ticket Hypothesis* [12] deals with the question of why iterative magnitude based pruning works so well [42]. Among others, they also investigate resetting pruned weights to their initial values and keeping them fix. The unpruned parameters are reset to their initial values as well and trained again. This train-prune-retrain cycle is continued until the target rate of fixed parameters is reached. In their experiments they show that this procedure mostly leads to worse results than standard iterative pruning and just outperforms it for extremely high pruning rates.

3 FreezeNets

General Setup Let $f_\Theta : \mathbb{R}^{d_0} \to [0,1]^c$ be a DNN with parameters $\Theta \subset \mathbb{R}$, used for an image classification task with c classes. We assume a train set (X, Z) with images $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^{d_0}$ and corresponding labels $Z = \{z_1, \dots, z_N\} \subset \{0, 1, \dots, c-1\}$, a test set (X_0, Z_0) and a smooth loss function L to be given. As common for DNNs, the test error is minimized by training the network with help of the training data via stochastic gradient based (SGD) optimization [34] while preventing the network to overfit on the training data.

We define the rate $q := 1 - p$ as the networks *freezing rate*, where p is the rate of trainable weights. A high freezing rate corresponds to few trainable parameters and therefore sparse gradients, whereas a low freezing rate corresponds to many trainable parameters. Freezing is compared to pruning in Section 4. For simplicity, a freezing rate q for pruning a network means exactly that $q \cdot 100\%$ of its weights are set to zero. In this work we split the model's parameters into weights W and biases B, only freeze parts of the weights W and keep all biases trainable.

3.1 SNIP

Since pruned networks are constraint on using only parts of their weights, those weights should be chosen as the most influential ones for the given task. Let $\Theta = W \cup B$ [1] be the network's parameters and $m \in \{0,1\}^{|W|}$ be a mask that shows if a weight is activated or not. Therefore, the weights that actually contribute to the network's performance are given by $m \odot W$. Here $\odot$ denotes the Hadamard product [5]. The trick used in [29] is to look at the *saliency score*

$$g := \left. \frac{\partial L(m \odot W; B, X, Z)}{\partial m} \right|_{m=1} = \frac{\partial L(W; B, X, Z)}{\partial W} \odot W \ , \qquad (1)$$

which calculates componentwise the influence of the loss function's change by a small variation of the associated weight's activation.[2] If those changes are big, keeping the corresponding weight is likely to have a greater effect in minimizing the loss function than keeping a weight with a small score. The gradient g can be approximated with just a single forward and backward pass of one training batch before the beginning of training.

3.2 Backpropagation in Neural Networks

To simplify the backpropagation formulas, we will deal with a feed-forward, fully connected neural network. Similar equations hold for convolutional layers [21]. Let the input of the network be given by $x^{(0)} \in \mathbb{R}^{d_0}$, the weight matrices are

[1] By an abuse of notation, we also use W and B as the vectors containing all elements of the set of all weights and biases, respectively.

[2] To obtain differentiability in equation (1), the mask is assumed to be continuous, i.e. $m \in \mathbb{R}^{|W|}$.

Algorithm 1 FreezeNet

```
Require: Freezing rate $q$, initial parametrization $\Theta_0 = W_0 \cup B_0$, corresponding network $f_{\Theta_0}$, loss function $L$
Compute saliency score $g \in \mathbb{R}^{|W_0|}$ for one training batch, according to equation (1)
Define freezing mask $m \in \mathbb{R}^{|W_0|}$
Use freezing threshold $\varepsilon$ as the $\lfloor (1-q) \cdot |W_0| \rfloor$-highest magnitude of $g$
Set $m_k = 0$ if $|g_k| < \varepsilon$ else $m_k = 1$
Start training with forward propagation as usual but backpropagate gradient $m \odot \frac{\partial L}{\partial W_0}$ for weights and $\frac{\partial L}{\partial B_0}$ for biases
```

given by $W^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$, $k \in \{1, \dots, K\}$ and the forward propagation rules are inductively defined as

- $y^{(k)} := W^{(k)} x^{(k-1)} + b^{(k)}$ for the layers bias $b^{(k)} \in \mathbb{R}^{d_k}$,
- $x^{(k)} := \Phi_{(k)}(y^{(k)})$ for the layers non-linearity $\Phi_{(k)} : \mathbb{R} \to \mathbb{R}$, applied componentwise.

This leads to the partial derivatives used for the backward pass, written compactly in vector or matrix form:

$$\begin{array}{ll} \frac{\partial L}{\partial y^{(k)}} = \Phi'_{(k)}\left(y^{(k)}\right) \odot \frac{\partial L}{\partial x^{(k)}} \ , & \frac{\partial L}{\partial x^{(k)}} = \left(W^{(k+1)}\right)^T \cdot \frac{\partial L}{\partial y^{(k+1)}} \ , \\ \frac{\partial L}{\partial W^{(k)}} = \frac{\partial L}{\partial y^{(k)}} \cdot \left(x^{(k-1)}\right)^T \ , & \frac{\partial L}{\partial b^{(k)}} = \frac{\partial L}{\partial y^{(k)}} \end{array} . \tag{2}$$

Here, we define $W^{(K+1)} := \mathrm{id} \in \mathbb{R}^{d_K \times d_K}$ and $\frac{\partial L}{\partial y^{(K+1)}} := \frac{\partial L}{\partial x^{(K)}}$. For sparse weight matrices $W^{(k+1)}$, equations (2) can lead to small $\frac{\partial L}{\partial x^{(k)}}$ and consequently small weight gradients $\frac{\partial L}{\partial W^{(k)}}$. In the extreme case of $\frac{\partial L}{\partial y^{(k)}} = 0$ for a layer k,all overlying layers will have $\frac{\partial L}{\partial W^{(l)}} = 0$, $l \leq k$. Overcoming the gradient's drying up for sparse weight matrices in the backward pass motivated us to freeze weights instead of pruning them.

3.3 FreezeNet

In Algorithm 1 the FreezeNet method is introduced. First, the saliency score g is calculated according to equation (1). Then, the freezing threshold ε is defined as the $\lfloor (1-q) \cdot |W_0| \rfloor$-highest magnitude of g. If a saliency score is smaller than the freezing threshold, the corresponding entry in the freezing mask $m \in \mathbb{R}^{|W_0|}$ is set to 0. Otherwise, the entry in m is set to 1. However, we do not delete the nonchosen parameters as done for SNIP pruning, but leave them as initialized. This is achieved with the masked gradient. For computational and storage capacity reasons, it is more efficient to not calculate the partial derivative for the weights with mask value 0, than masking the gradient after its computation.

The amount of memory needed to store a FreezeNet is the same as for standard pruning. With the help of pseudo random number generators, as provided

Table 1. Comparison of standard training, pruning before training and a FreezeNet.

Method	# Total Weights	# Weights to Store	Sparse Gradients	Sparse Tensor Computations	Faithful Gradient Flow
Standard	D	D	✗	✗	✓
Pruned	$D \cdot (1-q)$	$D \cdot (1-q)$	✓	✓	✗
FreezeNet	D	$D \cdot (1-q)$	✓	✗	✓

by PyTorch [32] or TensorFlow [1], just the seed used for generating the initial parametrization has to be stored, which is usually an integer and therefore its memory requirement is neglectable. The used pruning/freezing mask together with the trained weights have to be saved for both, pruning and FreezeNets. The masks can be stored efficiently via entropy encoding [11].

In this work, we only freeze weights and keep all biases learnable, as done in the pruning literature [12, 13, 29, 39]. Therefore, we compute the freezing rate as $q = 1 - \frac{\|m\|_0}{|W|}$, where m is the freezing mask calculated for the network's weights W. Here, the pseudo norm $\|\cdot\|_0$ computes the number of non-zero elements in m. Since we deal with extremely high freezing rates, $q > 0.99$, the bias parameters have an effect on the percentage of all trained parameters. Thus, we define the real freezing rate $q_\beta = 1 - \frac{\|m\|_0 + |B|}{|W| + |B|}$ and label the x-axes in our plots with both rates.

Pruned networks use masked weight tensors $m \odot W$ in the for- and backward pass. In theory, the number of computations needed for a pruned network can approximately be reduced by a factor of q_β in the forward pass. The frozen networks do not decrease the number of calculations in the forward pass. But without the usage of specialized soft- and hardware, the number of computations performed by a pruned network is not reduced, thus frozen and pruned networks have the same speed in this setting.

In the backward pass, the weight tensor needed to compute $\frac{\partial L}{\partial x^{(k-1)}}$ is given by $m^{(k)} \odot W^{(k)}$ for a pruned network, according to the backpropagation equations (2). Frozen networks compute $\frac{\partial L}{\partial x^{(k-1)}}$ with a dense matrix $W^{(k)}$. On the other hand, not all weight gradients are needed, as only $m^{(k)} \odot \frac{\partial L}{\partial W^{(k)}}$ is required for updating the network's unfrozen weights. Therefore, the computation time in the backward pass is not reduced drastically by FreezeNets, although the number of gradients to be stored. Again, the reduction in memory is approximately given by the rate q_β. The calculation of $\frac{\partial L}{\partial x^{(k-1)}}$ with a dense matrix $W^{(k)}$ helps to preserve a faithful gradient throughout the whole network, even for extremely high freezing rates, as shown in Section 4.3. The comparison of training a pruned and a frozen network is summarized in Table 1.

4 Experiments and Discussions

In the following, we present results on the MNIST [27] and CIFAR-10/100 [26] classification tasks achieved by FreezeNet. Freezing networks is compared with training sparse networks, exemplified through SNIP [29]. We further analyse how freezing weights retains the trainability of networks with sparse gradient updates by preserving a faithful gradient. We use three different network architectures, the fully connected LeNet-300-100 [27] along with the CNN's LeNet-5-Caffe [27] and VGG16-D [37]. A more detailed description of the used network architectures can be found in the Supplementary Material. Additionally, we show in the Supplementary Material that FreezeNets based on a ResNet34 perform well on Tiny ImageNet.

For our experiments we used PyTorch 1.4.0 [32] and a single Nvidia GeForce 1080ti GPU. In order to achieve a fair comparison regarding hard- and software settings we recreated SNIP.[3] To prevent a complete loss of information flow we randomly flag one weight trainable per layer if all weights of this layer are frozen or pruned for both, SNIP and FreezeNet. This adds at most 3 trainable parameters for LeNet-300-100, 4 for LeNet-5-Caffe and 16 for VGG16-D. If not mentioned otherwise, we use Xavier-normal initializations [16] for SNIP and FreezeNets and apply weight decay on the trainable parameters only. Except where indicated, the experiments were run five times with different random seeds, resulting in different network initializations, data orders and additionally for CIFAR experiments in different data augmentations. In our plots we show the mean test accuracy together with one standard deviation. A split of 9/1 between training examples and validation examples is used for early stopping in training. All other hyperparameters applied in training are listed in the Supplementary Material.

SGD with momentum [38] is used as optimizer, thus we provide a learning rate search for FreezeNets in the Supplementary Material. Because $\lambda = 0.1$ works best for almost all freezing rates, we did not include it in the main body of the text and use $\lambda = 0.1$ with momentum 0.9 for all presented results. Altogether, we use the same setup as SNIP in [29] for both, FreezeNets and SNIP pruned networks.

4.1 MNIST

LeNet-300-100 A common baseline to examine pruning mechanisms on fully connected networks is given by testing the LeNet-300-100 [27] network on the MNIST classification task [27], left part of Figure 2. The trained baseline architecture yields a mean test accuracy of 98.57%. If the freezing rate is lower than 0.95, both methods perform equally well and also match the performance of the baseline. For higher freezing rates, the advantage of using free, additional parameters can be seen. FreezeNets also suffer from the loss of trainable weights, but they are able to compensate it better than SNIP pruned networks do.

LeNet-5-Caffe For moderate freezing rates $q \in [0.5, 0.95]$, again FreezeNet and SNIP reach equal results and match the baseline's performance as shown in the Supplementary Material. In the right part of Figure 2, we show the progression of

[3] Based on the official implementation https://github.com/namhoonlee/snip-public.

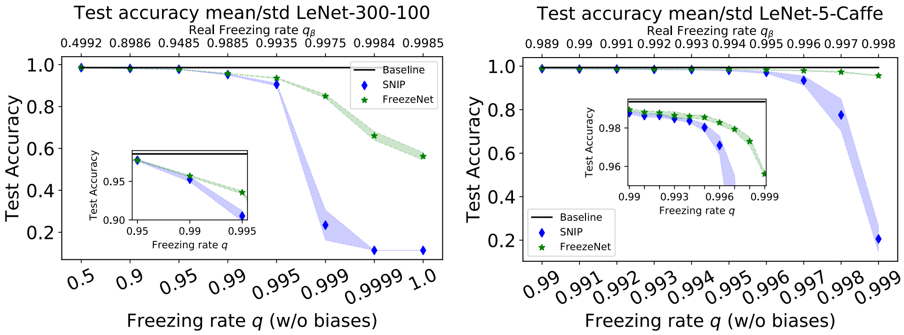

Fig. 2. Left: Test accuracy for SNIP, FreezeNet and the baseline LeNet-300-100. Right: Test accuracy for SNIP and FreezeNet for a LeNet-5-Caffe baseline. The small inserted plots are zoomed in versions for both plots.

Table 2. Comparison of FreezeNet, SNIP and the LeNet-5-Caffe baseline. Results for different *freezing rates* q with corresponding *real freezing rates* q_β are displayed. The *network's size* is calculated without compression. Thus, all weights are stored as 32bit floats. *Compress. Factor FN* is the compression factor gained by FreezeNet for the corresponding freezing rate, calculated via the ratio of the *network sizes* of the baseline and the frozen network.

q	q_β	Network Size	Compress. Factor FN	Test Acc. SNIP	Test Acc. FreezeNet	$\frac{\text{FreezeNet Acc.}}{\text{Baseline Acc.}}$
0 (Baseline)		$1,683.9$kB	1	99.36%		1.000
0.9	0.899	170.7kB	**9.9**	99.24%	**99.37**%	1.000
0.99	0.989	19.1kB	**88.2**	98.80%	**98.94**%	0.996
0.995	0.994	10.7kB	**157.4**	98.02%	**98.55**%	0.992
0.999	0.998	3.9kB	**431.8**	20.57%	**95.61**%	0.962

SNIP and FreezeNet for more extreme freezing rates $q \in \{0.99, 0.991, \dots, 0.999\}$. Until $q = 0.994$ SNIP and FreezeNet perform almost equally, however FreezeNet reaches slightly better results. For higher freezing rates, SNIP's performance drops steeply whereas FreezeNet is able to slow this drop. As Table 2 and Figure 2 show, a FreezeNet saves parameters with respect to both, the baseline architecture and a SNIP pruned network.

In order to overfit the training data maximally, we change the training setup by training the networks without the usage of weight decay and early stopping. In the left part of Figure 3, the training accuracies of FreezeNet and SNIP are reported for the last training epoch. Unsurprisingly, too many frozen parameters reduce the model's capacity, as the model is not able to perfectly memorize the training data for rates higher than $q_* = 0.992$. On the other hand, FreezeNets increase the networks capacity compared to SNIP if the same, high freezing rate is used.

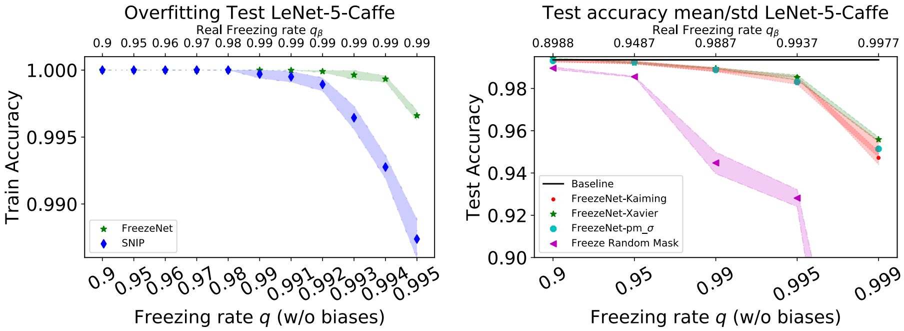

Fig. 3. Left: Final training accuracies for FreezeNet and SNIP, both trained without weight decay. Right: Different initializations for FreezeNets together with a Xavier-normal initialized FreezeNet with randomly generated freezing mask. Both plots are reported for the MNIST classification task with a LeNet-5-Caffe baseline architecture.

4.2 Testing FreezeNets for CIFAR-10/100 on VGG16-D

To test FreezeNets on bigger architectures, we use the VGG16-D architecture [37] and the CIFAR-10/100 datasets. Now, weight decay is applied to all parameters, including the frozen ones, denoted with FreezeNet-WD. As before, weight decay is also used on the unfrozen parameters only, which we again call FreezeNet. We follow the settings in [29] and exchange Dropout layers with Batch Normalization [23] layers. Including the Batch Normalization parameters, the VGG16-D network consists of 15.3 million parameters in total. We train all Batch Normalization parameters and omit them in the freezing rate q. Additionally, we augment the training data by random horizontal flipping and translations up to 4 pixels. For CIFAR-100 we report results for networks initialized with a Kaiming-uniform initialization [21]. The results are summarized in Table 3.

CIFAR-10 If more parameters are trainable, $q \leq 0.95$, SNIP performs slightly worse than the baseline but better than FreezeNet. However, using frozen weights can achieve similar results as the baseline architecture while outperforming SNIP if weight decay is applied to them as well, as shown with FreezeNet-WD. Applying weight decay also on the frozen parameters solely shrinks them to zero. For all occasions where FreezeNet-WD reaches the best results, the frozen weights can safely be pruned at the early stopping time, as they are all shrunk to zero at this point in training. For these freezing rates, FreezeNet-WD can be considered as a pruning mechanism outperforming SNIP without adding any gradient computations For higher freezing rates $q \geq 0.99$, FreezeNet still reaches reasonable results whereas FreezeNet-WD massively drops performance and SNIP even results in random guessing.

Table 3. Comparison of results for the CIFAR-10/100 tasks with a VGG16-D baseline.

			CIFAR-10	**CIFAR-100**
Method	Freezing Rate	Trained Parameters	Mean ± Std	Mean ± Std
Baseline	0	15.3mio	**93.0 ± 0.1%**	**71.6 ± 0.6%**
SNIP	0.9	1.5mio	92.9 ± 0.1%	53.9 ± 1.7%
	0.95	780k	92.5 ± 0.1%	48.6 ± 6.6%
	0.99	169k	10.0 ± 0.0%	1.0 ± 0.0%
	0.995	92k	10.0 ± 0.0%	1.0 ± 0.0%
FreezeNet	0.9	1.5mio	92.2 ± 0.1%	**70.7 ± 0.3%**
	0.95	780k	91.7 ± 0.1%	**69.0 ± 0.2%**
	0.99	169k	**88.6 ± 0.1%**	**59.8 ± 0.3%**
	0.995	92k	**86.0 ± 0.1%**	**53.4 ± 0.1%**
FreezeNet-WD	0.9	1.5mio	**93.2 ± 0.2%**	53.1 ± 1.8%
	0.95	780k	**92.8 ± 0.2%**	44.5 ± 5.4%
	0.99	169k	76.1 ± 1.0%	13.1 ± 1.8%
	0.995	92k	74.6 ± 1.1%	11.9 ± 1.4%

CIFAR-100 CIFAR-100 is more complex to solve than CIFAR-10. As the right part of Table 3 shows, SNIP is outperformed by FreezeNet for all freezing rates. Frozen parameters seem to be even more helpful for a sophisticated task. For CIFAR-100, more complex information flows backwards during training, compared to CIFAR-10. Thus, using dense weight matrices in the backward pass helps to provide enough information for the gradients to train successfully Additionally we hypothesize, that random features generated by frozen parameters can help to improve the network's performance, as more and often closely related classes have to be distinguished. Using small, randomly generated differences between data samples of different, but consimilar classes may help to separate them.

Discussion Deleting the frozen weights reduces the network's capacity — as shown for the MNIST task, Figure 3 left. But for small freezing rates, the pruned network still has enough capacity in the forward- and backward propagation. In these cases, the pruned network has a higher generalization capability than the FreezeNet, according to the bias-variance trade-off [15]. Continuously decreasing the network's capacity during training, instead of one-shot, seems to improve the generalization capacity even more, as done with FreezeNet-WD. But for higher freezing rates, unshrunken and frozen parameters improve the performance significantly. For these rates, FreezeNet is still able to learn throughout the whole training process. Whereas FreezeNet-WD reaches a point in training, where the frozen weights are almost zero. Therefore, the gradient does not flow properly through the network, since the pruned SNIP network has zero gradient flow for

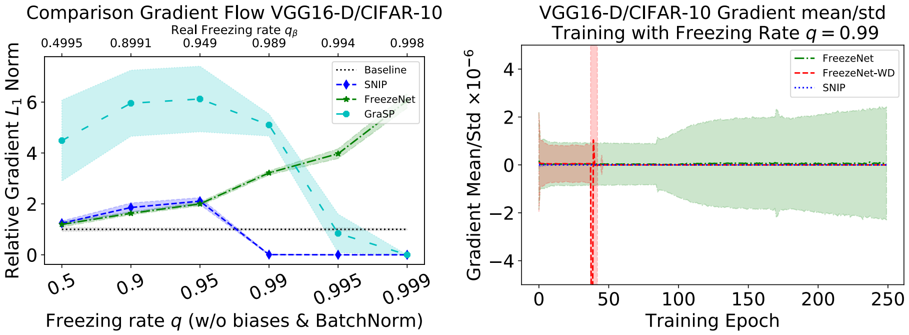

Fig. 4. Left: Shows the relative gradient norm for FreezeNet, SNIP and GraSP networks with respect to the VGG16-D baseline network on the CIFAR-10 dataset. Right: Gradient mean and std, computed over the training data, recorded for one training run with a VGG16-D architecture on the CIFAR-10 task for a freezing rate $q = 0.99$.

these rates, Figure 4 left. This change of FreezeNet-WD's gradient's behaviour is shown in Figure 4 right for $q = 0.99$. It should be mentioned that in these cases, FreezeNet-WD will have an early stopping time before all frozen weights are shrunk to zero and FreezeNet-WD can not be pruned without loss in performance.

4.3 Gradient Flow

As theoretically discussed in Section 3.2, FreezeNets help to preserve a strong gradient, even for high freezing rates. To check this, we also pruned networks with the GraSP criterion [39] to compare FreezeNets with pruned networks generated to preserve the gradient flow. A detailed description of the GraSP criterion can be found in the Supplementary Material. For this test, 10 different networks were initialized for every freezing rate and three copies of each network were frozen (FreezeNet) or pruned (SNIP and GraSP), respectively. The L_1 norm of the gradient, accumulated over the whole training set, is divided by the number of trainable parameters. As reference, the mean norm of the baseline VGG16-D's gradient is measured as well. These gradient norms, computed for CIFAR-10, are compared in the left part of Figure 4. For smaller freezing rates, all three methods have bigger gradient values than the baseline, on average. For rates $q \geq 0.95$, decreasing the number of trainable parameters leads to a reduced gradient flow for the pruned networks. Even if the pruning mask is chosen to guarantee the highest possible gradient flow, as approximately done by GraSP. Finally, the gradient vanishes, since the weight tensors become sparse for high pruning rates, as already discussed in Section 3.2. FreezeNet's gradient on the other hand is not hindered since its weight tensors are dense. The saliency score (1) is biased towards choosing weights with a high partial derivative. Therefore, FreezeNet's non-zero gradient values even become larger as the number of trainable parameters decreases. For high freezing rates, FreezeNet's gradient is able to flow through the network during the whole training process, whereas SNIP's gradient remains zero all the time — right part of Figure 4. The right part of Figure 4 also shows the mutation of FreezeNet-WD's gradient flow during training. First, FreezeNet-WD has similar gradients as FreezeNet since the frozen weights are still big enough. The red peak indicates the point where too many frozen weights are shrunken close to zero, leading to temporarily chaotic gradients and resulting in zero gradient flow.

4.4 Comparison to Pruning Methods

Especially for extreme freezing rates, we see that FreezeNets perform remarkably better than SNIP, which often degenerates to random guessing. In Table 4, we compare our result for LeNet-5-Caffe with *Sparse-Momentum* [7], SNIP, GraSP and three other pruning methods *Connection-Learning* [19], *Dynamic-Network-Surgery* [17] and *Learning-Compression* [3]. Up to now, all results are reported without any change in hyperparameters. To compare FreezeNet with other pruning methods, we change the training setup slightly by using a split of 19/1 for train and validation images for FreezeNet, but keep the remaining hyperparameters fixed. We also recreated results for GraSP [39]. The training setup and the probed hyperparameters for GraSP can be found in the Supplementary Material. All other results are reported from the corresponding papers. As shown in Table 4, the highest accuracy of 99.2% is achieved by the methods *Connection-Learning* and *Sparse-Momentum.* With an accuracy of 99.1% our FreezeNet algorithm performs only slightly worse, however Connection-Learning trains 8.3% of its weights — whereas FreezeNet achieves 99.37% accuracy with 10% trained weights, see Table 2. Sparse-Momentum trains with sparse weights, but updates the gradients of all weights during training and redistributes the learnable weights after each epoch. Thus, their training procedure does neither provide sparse gradient computations nor one-shot pruning and is hence more expensive than FreezeNet. Apart from that, FreezeNet achieves similar results to *Dynamic-Network-Surgery* and better results than *Learning-Compression*, GraSP and SNIP, while not adding any training costs over GraSP and SNIP and even reducing them for *Dynamic-Network-Surgery* and *Learning-Compression.*

4.5 Further Investigations

The right part of Figure 3 shows on the one hand, that FreezeNet reaches better and more stable results than freezing networks with a randomly generated freezing mask. This accentuates the importance of choosing the freezing mask consciously, for FreezeNets done with the saliency score (1).

On the other hand, different variance scaling initialization schemes are compared for FreezeNets in the right part of Figure 3. Those initializations help to obtain a satisfactory gradient flow at the beginning of the training [16, 21]. Results for the Xavier-normal initialization [16], the Kaiming-uniform [21] and the $\mathtt{pm}_\sigma$-initialization are shown. All of these initializations lead to approximately the same results. Considering all freezing rates, the Xavier-initialization yields the

Table 4. Comparison of different pruning methods with FreezeNet on LeNet-5-Caffe.

Method	Sparse Gradients in Training	Additional Hyperparameters	Percent of trainable parameters	Test Accuracy
Baseline [27]	–	–	100%	99.4%
SNIP [29]	✓	✗	1.0%	98.9%
GraSP [39]	✓	✗	1.0%	98.9%
Connection-Learning [19]	✗	✗	8.3%	99.2%
Dynamic-Network-Surgery [17]	✗	✗	0.9%	99.1%
Learning-Compression [3]	✗	✓	1.0%	98.9%
Sparse-Momentum [7]	✗	✓	1.0%	99.2%
FreezeNet (ours)	✓	✗	1.0%	99.1%

best results. The $\mathtt{pm}_\sigma$-initialization is a variance scaling initialization, using zero mean and a variance of $\sigma^2 = \frac{2}{fan_{in}+fan_{out}}$, layerwise. All weights are set to $+\sigma$ with probability $\frac{1}{2}$ and $-\sigma$ otherwise. Using the $\mathtt{pm}_\sigma$-initialization shows, that even the simplest variance scaling method leads to good results for FreezeNets.

In the Supplementary Material we exhibit that FreezeNets are robust against reinitializations of their weights after the freezing mask is computed and before the actual training starts. The probability distribution can even be changed between initialization and reinitialization while still leading to the same performance.

5 Conclusions

With FreezeNet we have introduced a pruning related mechanism that is able to reduce the number of trained parameters in a neural network significantly while preserving a high performance. FreezeNets match state-of-the-art pruning algorithms without using their sophisticated and costly training methods, as Table 4 demonstrates. We showed that frozen parameters help to overcome the vanishing gradient occurring in the training of sparse neural networks by preserving a strong gradient signal. They also enhance the expressiveness of a network with few trainable parameters, especially for more complex tasks. With the help of frozen weights, the number of trained parameters can be reduced compared to the related pruning method SNIP. This saves storage space and thus reduces transfer costs for trained networks. For smaller freezing rates, it might be better to weaken the frozen parameters' influence, for example by applying weight decay to them. Advantageously, using weight decay on frozen weights contracts them to zero, leading to sparse neural networks. But for high freezing rates, weight decay in its basic form might not be the best regularization mechanism to apply to FreezeNets, since only shrinking the frozen parameters robs them of a big part of their expressiveness in the forward and backward pass.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. CoRR **abs/1603.04467** (2016)
2. Bellec, G., Kappel, D., Maass, W., Legenstein, R.: Deep rewiring: Training very sparse deep networks. In: International Conference on Learning Representations (2018)
3. Carreira-Perpinan, M.A., Idelbayev, Y.: ”Learning-compression” algorithms for neural net pruning. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8532–8541 (2018)
4. Chauvin, Y.: A back-propagation algorithm with optimal use of hidden units. In: Advances in Neural Information Processing Systems 1 (1989)
5. Davis, C.: The norm of the schur product operation. Numerische Mathematik **4**(1), 343–344 (1962)
6. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255 (2009)
7. Dettmers, T., Zettlemoyer, L.: Sparse networks from scratch: Faster training without losing performance. CoRR **abs/1907.04840** (2019)
8. Ding, X., Ding, G., Zhou, X., Guo, Y., Han, J., Liu, J.: Global sparse momentum sgd for pruning very deep neural networks. In: Advances in Neural Information Processing Systems 32, pp. 6382–6394 (2019)
9. Dong, X., Huang, J., Yang, Y., Yan, S.: More is less: A more complicated network with less inference complexity. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
10. Dong, X., Yang, Y.: Network pruning via transformable architecture search. In: Advances in Neural Information Processing Systems 32, pp. 760–771 (2019)
11. Duda, J., Tahboub, K., Gadgil, N.J., Delp, E.J.: The use of asymmetric numeral systems as an accurate replacement for huffman coding. In: Picture Coding Symposium. pp. 65–69 (2015)
12. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2018)
13. Frankle, J., Dziugaite, G.K., Roy, D.M., Carbin, M.: Stabilizing the lottery ticket hypothesis. CoRR **abs/1903.01611** (2019)
14. Frankle, J., Schwab, D.J., Morcos, A.S.: Training batchnorm and only batchnorm: On the expressive power of random features in cnns. CoRR **abs/2003.00152** (2020)
15. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Comput. **4**(1), 1–58 (1992)
16. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. pp. 249–256 (2010)
17. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient dnns. In: Advances in Neural Information Processing Systems 29, pp. 1379–1387 (2016)
18. Gustafson, J.L.: Moore’s law. In: Encyclopedia of Parallel Computing. pp. 1177–1184 (2011)
19. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems 28 (2015)

20. Hanson, S.J., Pratt, L.Y.: Comparing biases for minimal network construction with back-propagation. In: Advances in Neural Information Processing Systems 1 (1989)
21. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR **abs/1502.01852** (2015)
22. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. Lecture Notes in Computer Science p. 815–832 (2018)
23. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning. pp. 448–456 (2015)
24. Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multi-stage architecture for object recognition? In: ICCV. pp. 2146–2153 (2009)
25. Karnin, E.D.: A simple procedure for pruning back-propagation trained neural networks. IEEE Transactions on Neural Networks **1**(2), 239–242 (1990)
26. Krizhevsky, A.: Learning multiple layers of features from tiny images. University of Toronto (2012), http://www.cs.toronto.edu/ kriz/cifar.html
27. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
28. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems 2 (1990)
29. Lee, N., Ajanthan, T., Torr, P.: SNIP: Single-shot network pruning based on connection sensitivity. In: International Conference on Learning Representations (2019)
30. Mocanu, D., Mocanu, E., Stone, P., Nguyen, P., Gibescu, M., Liotta, A.: Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nature Communications **9** (2018)
31. Mozer, M.C., Smolensky, P.: Skeletonization: A technique for trimming the fat from a network via relevance assessment. In: Advances in Neural Information Processing Systems 1 (1989)
32. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035 (2019)
33. Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., Rastegari, M.: What's hidden in a randomly weighted neural network? In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
34. Robbins, H., Monro, S.: A stochastic approximation method. The Annals of Mathematical Statistics **22**(3), 400–407 (1951)
35. Saxe, A., Koh, P.W., Chen, Z., Bhand, M., Suresh, B., Ng, A.: On random weights and unsupervised feature learning. In: Proceedings of the 28th International Conference on Machine Learning. pp. 1089–1096 (2011)
36. Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green AI. CoRR **abs/1907.10597** (2019)
37. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
38. 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)
39. Wang, C., Zhang, G., Grosse, R.: Picking winning tickets before training by preserving gradient flow. In: International Conference on Learning Representations (2020)
40. Wortsman, M., Farhadi, A., Rastegari, M.: Discovering neural wirings. In: Advances in Neural Information Processing Systems 32, pp. 2684–2694 (2019)

41. Xie, S., Kirillov, A., Girshick, R., He, K.: Exploring randomly wired neural networks for image recognition. 2019 IEEE/CVF International Conference on Computer Vision (ICCV) (2019)
42. Zhou, H., Lan, J., Liu, R., Yosinski, J.: Deconstructing lottery tickets: Zeros, signs, and the supermask. In: Advances in Neural Information Processing Systems 32, pp. 3597–3607 (2019)