Parameterized Structured Pruning for Deep Neural Networks

Günther Schindler[1], Wolfgang Roth[2], Franz Pernkopf[2], and Holger Fröning[1]

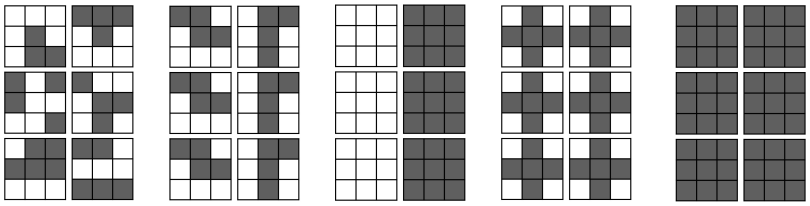
[1] Institute of Computer Engineering,
Ruprecht Karls University, Heidelberg, Germany
{guenther.schindler, holger.froening}@ziti.uni-heidelberg.de
[2] Signal Processing and Speech Communication Laboratory,
Graz University of Technology, Austria
{roth, pernkopf}@tugraz.at

Abstract. As a result of the growing size of Deep Neural Networks (DNNs), the gap to hardware capabilities in terms of memory and compute increases. To effectively compress DNNs, quantization and pruning are usually considered. However, unconstrained pruning usually leads to unstructured parallelism, which maps poorly to massively parallel processors, and substantially reduces the efficiency of general-purpose processors. Similar applies to quantization, which often requires dedicated hardware.
We propose Parameterized Structured Pruning (PSP), a novel technique to dynamically learn the shape of DNNs through structured sparsity. PSP parameterizes structures (e.g. channel- or layer-wise) in a weight tensor and leverages weight decay to learn a clear distinction between important and unimportant structures. As a result, PSP maintains prediction performance, creates a substantial amount of sparsity that is structured and, thus, easy and efficient to map to a variety of massively parallel processors, which are mandatory for utmost compute power and energy efficiency.

1 Introduction

Deep Neural Networks (DNNs) are widely used for many applications including object recognition, speech recognition and robotics. The ability of modern DNNs to excellently fit training data is suspected to be due to heavy over-parameterization, i.e., using more parameters than the total number of training samples, since there always exists parameter choices that achieve a training error of zero. While over-parameterization is essential for the learning ability of neural networks, it results in extreme memory and compute requirements for training (development) as well as inference (deployment). Recent research showed that training can be scaled to up to 1024 accelerators operating in parallel, resulting in a development phase not exceeding a couple of minutes, even for large-scale image classification. However, the deployment has usually much harder constraints than the development, as energy, space and monetary resources are scarce in mobile devices.

Model compression techniques are targeting this issue by training an over-parameterized model and compressing it for deployment. Popular compression techniques are pruning, quantization, knowledge distillation, and low-rank factorization, with the first two being most popular due to their extreme efficiency. Pruning connections [3] achieves impressive theoretical compression rates through fine-grained sparsity (Fig. 1a) without sacrificing prediction performance, but has several practical drawbacks such as indexing overhead, load imbalance and random memory accesses: (i) Compression rates are typically reported without considering the space requirement of additional data structures to represent non-zero weights. For instance, using indices, a model with 8-bit weights, 8-bit indices and 75% sparsity saves only 50% of the space, while a model with 50% sparsity does not save memory at all. (ii) It is a well-known problem that massively parallel processors show notoriously poor performance when the load is not well balanced. Unfortunately, since the end of Dennard CMOS scaling, massive parallelization is mandatory for a continued performance scaling. (iii) Sparse models increase the amount of randomness in memory access patterns, preventing caching techniques, which rely on predictable strides, from being effective. As a result, the amount of cache misses increases the average memory access latency and the energy consumption, as off-chip accesses are 10-100 time higher in terms of latency, respectively 100-1000 times higher in terms of energy consumption. Quantization has recently received plenty of attention and reduces the computational workload, as the complexity of additions and multiplications scales approximately linearly and quadratically with the number of bits, respectively. However, in comparison, pruning avoids a computation completely.



(a) Weights (b) Columns (c) Channels (d) Shapes (e) Layers

Fig. 1: Illustration of fine-grained (Fig. 1a) and several structured forms of sparsity (Fig. 1b-1d) for a 4-dimensional convolution tensor. The large squares represent the kernels, and the corresponding horizontal and vertical dimensions represent the number of input feature and output feature maps, respectively. The computation of all structured forms of sparsity can be lowered to matrix multiplications (independent of stride and padding).

Structured pruning methods can prevent these drawbacks by inducing sparsity in a hardware-friendly way: Fig. 1b-1e illustrate exemplary a 4-dimensional convolution tensor (see [2] for details on convolution lowering), where hardware-

friendly sparsity structures are shown as channels, layers, etc. However, pruning whole structures in a neural network is not as trivial as pruning individual connections and usually causes high accuracy degradation under mediocre compression constraints.

Structured pruning methods can be roughly clustered into two categories: re-training-based and regularization-based methods (see Sec. 4 for details). Re-training-based methods aim to remove structures by minimizing the pruning error in terms of changes in weight, activation, or loss, respectively, between the pruned and the pre-trained model. Regularization-based methods train a randomly initialized model and apply regularization, usually an ℓ_1 penalty, in order to force structures to zero.

This work introduces a new regularization-based method leveraging learned parameters for structured sparsity without substantial increase in training time. Our approach differs from previous methods, as we explicitly parameterize certain structures of weight tensors and regularize them with weight decay, enabling a clear distinction between important and unimportant structures. Combined with threshold-based magnitude pruning and a straight-through gradient estimator (STE) [1], we can remove a substantial amount of structure while maintaining the classification accuracy. We evaluate the proposed method based on state-of-the-art Convolutional Neural Networks (CNNs) like ResNet [4] and DenseNet [8], and popular datasets like CIFAR-10/100 and ILSVRC2012.

The remainder of this work is structured as follows: In Sec. 2 we introduce the parameterization and regularization approach together with the pruning method. We present experimental results in Sec. 3. Related work is summarized in Sec. 4, before we conclude in Sec. 5.

2 Parameterized Pruning

DNNs are constructed by layers of stacked processing units, where each unit computes an activation function of the form $z = g(\mathbf{W} \oplus \mathbf{x})$, where $\mathbf{W}$ is a weight tensor, $\mathbf{x}$ is an input tensor, $\oplus$ denotes a linear operation, e.g., a convolution, and $g(\cdot)$ is a non-linear function. Modern neural networks have very large numbers of these stacked compute units, resulting in huge memory requirements for the weight tensors $\mathbf{W}$, and compute requirements for the linear operations $\mathbf{W} \oplus \mathbf{x}$. In this work, we aim to learn a structured sparse substitute $\mathbf{Q}$ for the weight tensor $\mathbf{W}$, so that there is only minimal overhead for representing the sparsity pattern in $\mathbf{Q}$ while retaining computational efficiency using dense tensor operations. For instance, by setting all weights at certain indices of the tensor to zero, it suffices to store the indices of non-zero elements only once for the entire tensor $\mathbf{Q}$ and not for each individual dimension separately. By setting all weights connected to an input feature map to zero, the corresponding feature map can effectively be removed without the need to store any indices at all.

2.1 Parameterization

Identifying the importance of certain structures in neural networks is vital for the prediction performance of structured-pruning methods. Our approach is to train the importance of structures by parameterizing and optimizing them together with the weights using backpropagation. Therefore, we divide the tensor $\mathbf{W}$ into subtensors $\{\mathbf{w}_i\}$ so that each $\mathbf{w}_i = (w_{i,j})_{j=1}^{m}$ constitutes the m weights of structure i. During forward propagation, we substitute $\mathbf{w}_i$ by the structured sparse tensor $\mathbf{q}_i$ as

$$\mathbf{q}_i = \mathbf{w}_i \alpha_i \tag{1}$$

where α_i is the structure parameter associated with structure i. Following the chain rule, the gradient of the structure parameter α_i is:

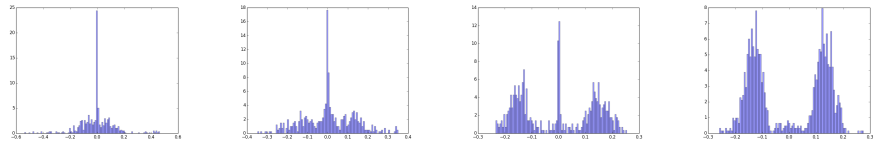
$$\frac{\partial E}{\partial \alpha_i} = \sum_{j=1}^{m} \frac{\partial E}{\partial w_{i,j}} \ , \tag{2}$$

where E represents the objective function. Thus, the dense structure parameters α_i descend towards the predominant direction of the structure weights. As a result, the structure parameters are optimized together with the weights of structure i but can be regularized and pruned independent to the weights of structure i. Training the structures introduces additional parameters, however, during inference they are folded into the weight tensors, resulting in no extra memory or compute costs.

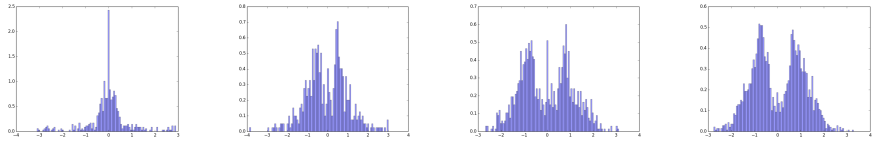
2.2 Regularization

Reducing the complexity of a neural network by limiting the growth of parameters is highly advantageous for their generalization abilities. It can be realized by adding a term to the cost function that penalizes parameters. Most commonly the ℓ_1 or ℓ_2 norm are used as penalty term, extending the cost function to $E_{\ell_1}(\alpha_i) = E(\alpha_i) + \lambda|\alpha_i|$ or $E_{\ell_2}(\alpha_i) = E(\alpha_i) + \frac{\lambda}{2}\alpha_i^2$, respectively. Applying ℓ_1 regularization to the structure parameters α_i changes the update rule as: $\Delta\alpha_i(t+1) = -\eta\frac{\partial E}{\partial \alpha_i} - \lambda\eta\,\mathrm{sign}(\alpha_i)$, where η is the learning rate and λ is the regularization strength. For gradient-based optimizations, ℓ_1 regularization only considers the sign of the parameters while the gradient in zero is undefined. Hence, it acts as a feature selector since certain weights are reduced to zero, resulting in sparse structure parameters.

For ℓ_2 regularization (weight decay), the update rule of the structure parameters is: $\Delta\alpha_i(t+1) = -\eta\frac{\partial E}{\partial \alpha} - \lambda\eta\alpha_i$. While ℓ_1 regularization only considers the direction of the parameters, weight decay also takes the magnitude of the parameters into account. This makes weight decay the standard regularization method since it significantly improves the learning capabilities of SGD based neural networks, resulting in faster convergence and better generalization. The benefits of weight decay can be best visualized using the distributions of the structure parameters α_i (corresponding to different layers) in Fig. 2.



(a) ℓ_2 reg.; group 0. (b) ℓ_2 reg.; group 1. (c) ℓ_2 reg.; group 2. (d) ℓ_2 reg.; group 3.



(e) ℓ_1 reg.; group 0. (f) ℓ_1 reg.; group 1. (g) ℓ_1 reg.; group 2. (h) ℓ_1 reg.; group 3.

Fig. 2: Different distributions of column-wise structure parameters with weight decay (ℓ_2) and ℓ_1 regularization of a fully trained ResNet with 18 layers on ImageNet. The distributions correspond to the first convolution in the first block in the respective group (g0-g3). Note that peaks visually close to zero are not exactly zero.

Parameterizing structures and regularization ultimately shrink the complexity (variance of the layers) of a neural network in a structured way. We observe that weight decay produces unimodal, bimodal and trimodal distributions (Fig. 2a-2d), indicating different complexities, with a clear distinction between important and unimportant structure parameters. In contrast, ℓ_1 regularization (Fig. 2e-Fig. 2h) lacks the ability to form this clear distinction. Second, ℓ_1 regularized structure parameters are roughly one order of magnitude larger than parameters trained with weight decay, making them more sensitive to small noise in the input data and reducing the effective learning rate. Third, even though ℓ_1 implicitly produces sparse models, weight decay reduces more parameters close to zero and therefore achieves better pruning potential. Consequently, we use weight decay for regularizing the structure parameters and perform pruning explicitly.

2.3 Pruning

The explicit pruning can be performed by a simple threshold-based magnitude pruning method. Let ν_i be the regularized *dense structure parameter* associated with structure i, then the *sparse structure parameter* α_i are obtained as:

$$\alpha_i(\nu_i) = \begin{cases} 0 & |\nu_i| < \epsilon \\ \nu_i & |\nu_i| \geq \epsilon \end{cases}, \tag{3}$$

where ϵ is a tuneable pruning threshold. As the threshold function is not differentiable at $\pm\epsilon$ and the gradient is zero in $[-\epsilon, \epsilon]$, we approximate the gradient of ν_i by defining an STE as $\frac{\partial E}{\partial \nu_i} = \frac{\partial E}{\partial \alpha_i}$. We use the sparse parameters α_i for forward and backward propagation and update the respective dense parameters ν_i

based on the gradients of α_i. Updating the dense structure parameters ν_i instead of the sparse parameters α_i is beneficial because improperly pruned structures can reappear if ν_i moves out of the pruning interval $[-\epsilon, \epsilon]$, resulting in faster convergence to a better performance.

2.4 Hardware-friendly structures in CNNs

We consider CNNs with $R \times S$ filter kernels, C input and K output feature maps. Different granularities of structured sparsity yield different flexibilities when mapped to hardware. In this work, we consider only coarse-grained structures such as layer, channel and column pruning, that can be implemented using off-the-shelf libraries on general-purpose hardware or shape pruning for direct convolutions on re-configurable hardware.

Layer pruning simply removes unimportant layers and ultimately shrinks the depth of a network (Fig. 1e), making the hardware mapping extremely efficient on every processor type. Note that layer pruning is only applicable to multi-branch architectures (e.g. DenseNet). **Channel pruning** refers to removing input or output channels in a convolutional layer and the respective input or output feature maps (Fig. 1c). Hence, it shrinks the width of a network and, similar to layer pruning, is applicable to every processor type. **Shape pruning** targets to prune filter kernels per layer equally (Fig. 1d), which can be mapped onto re-configurable hardware when direct convolution operations are in use. Convolutions are usually lowered onto matrix multiplications in order to explore data locality and the massive amounts of parallelism in general-purpose GPUs, CPUs or specialized processors like TPUs. This lowering is performed using the *im2col* approach, where discrete input blocks (depending on filter size and stride) are duplicated and reshaped into columns of a two dimensional matrix. The reader may refer to the work of [2] for a detailed explanation. Although layer, channel and shape pruning can be easily mapped to matrix multiplication, the potential sparsity is higher when a finer granularity is used. Thus, **column pruning** sparsifies weight tensors in a way that a whole column of the flattened weight tensor and the respective row of the input data can be removed (Fig. 1b), achieving finer granularity while allowing the efficient mapping to matrix multiplication.

The structure parameters for individual structures and their corresponding gradients are shown in Table 1. PSP is not restricted to these forms of granularities; arbitrary structures and combinations of different structures are possible as well as other layer types such as recurrent or dense layers. For instance, blocks can be defined and pruned in order to enable efficient processor vectorization or tiling techniques for cache and register blocking. Or layer and channel pruning can be combined when a simple hardware mapping is targeted.

3 Experiments

We use the CIFAR10/100 and the ILSVRC 2012 (ImageNet) datasets on ResNet [4] and DenseNet [8] architectures. Both networks can apply 1×1 convolutions as

Table 1: Representation of the dense structure parameters and the gradient calculation.

Pruning method	Structure parameter	Gradient
Layer pruning	$\alpha \in \mathbb{R}$	$\partial E/\partial \alpha = \sum_{k=1}^{K} \sum_{c=1}^{C} \sum_{r=1}^{R} \sum_{s=1}^{S} \partial E/\partial W_{k,c,r,s}$
Channel pruning	$\alpha \in \mathbb{R}^C$	$\partial E/\partial \alpha_c = \sum_{k=1}^{K} \sum_{r=1}^{R} \sum_{s=1}^{S} \partial E/\partial W_{k,c,r,s}$
Shape pruning	$\alpha \in \mathbb{R}^{R \times S}$	$\partial E/\partial \alpha_{r,s} = \sum_{k=1}^{K} \sum_{c=1}^{C} \partial E/\partial W_{k,c,r,s}$
Column pruning	$\alpha \in \mathbb{R}^{R \times S \times C}$	$\partial E/\partial \alpha_{r,s,c} = \sum_{k=1}^{K} \partial E/\partial W_{k,c,r,s}$

bottleneck layers before the 3×3 convolutions to improve compute and memory efficiency. DenseNet further improves model compactness by reducing the number of feature maps at transition layers. If bottleneck and transition compression is used, the models are labeled as *ResNet-B* and *DenseNet-BC*, respectively. Removing the bottleneck layers in combination with our compression approach has the advantage of reducing both, memory/compute requirements and the depth of the networks. We apply PSP to all convolutional layers except the sensitive input, output, transition and shortcut layers, which have negligible impact on overall memory and compute costs.

We use a weight decay of 10^{-4} and a momentum of 0.9 for weights and structure parameters throughout this work. We use the initialization introduced by [5] for the weights and initialize the structure parameters randomly using a zero-mean Gaussian with standard deviation 0.1.

3.1 Pruning different structures

We compare the performance of the different structure granularities using DenseNet on CIFAR10 (Table 2, with 40 layers, a growth rate of $k = 12$ and a pruning threshold of $\epsilon = 0.1$). We report the required layers, parameters and Multiply-Accumulate (MAC) operations.

While all structure granularities show a good prediction performance, with slight deviations compared to the baseline error, column- and channel-pruning achieve the highest compression ratios. Shape pruning results in the best accuracy but only at a small compression rate, indicating that a higher pruning threshold is more appropriate. It is worth noticing that PSP is able to automatically remove structures, which can be seen best when comparing layer pruning and a combination of layer and channel pruning: layer pruning removes 12 layers from the network but still requires 0.55M parameters and 0.14G MACs, while the combination of layer and channel pruning removes only 7 layers but requires only 0.48M parameters and 0.12G MACs.

3.2 CIFAR10/100 and ImageNet

To validate the effectiveness of PSP, we now discuss results from ResNet and DenseNet on CIFAR10/100 and ImageNet. We use column pruning throughout

Table 2: Layer-, channel, shape- and column-pruning using PSP, validated on DenseNet40 ($k = 12$) on the CIFAR10 dataset. M and G represents 10^6 and 10^9, respectively.

Model	Layers	Params.	MACs	Error
Baseline	40	1.02M	0.27G	5.80%
Layer pruning	28	0.55M	0.14G	6.46%
Channel pruning	40	0.35M	0.09G	5.61%
Layer+channel	33	0.48M	0.12G	6.39%
Shape pruning	40	0.92M	0.23G	5.40%
Column pruning	40	0.22M	0.05G	5.76%

this section, as it offers the highest compression rates while preserving classification performance.

Table 3 reports results for CIFAR10/100. As can be seen, PSP maintains classification performance for a variety of networks and datasets. This is due to the ability of self-adapting the pruned structures during training, which can be best seen when changing the network topology or dataset: for instance, when we use the same models on CIFAR10 and the more complex CIFAR100 task, we can see that PSP is able to automatically adapt as it removes less structure from the network trained on CIFAR100. Furthermore, if we increase the number of layers by 2.5× from 40 to 100, we also increase the over-parameterization of the network and PSP automatically removes 2.4× more structure.

Table 3: ResNet and DenseNet on CIFAR10/100 using column pruning for PSP.

	Layers	Param. [10^6]	MAC [10^9]	Error [%]
CIFAR10				
ResNet	56	0.85 (1.0×)	0.13 (1.0×)	**6.35 (+0.00)**
ResNet-PSP	56	**0.21 (4.0×)**	**0.03 (4.3×)**	6.55 (+0.20)
DenseNet	40	1.02 (1.0×)	0.27 (1.0×)	5.80 (+0.00)
DenseNet-PSP	40	**0.22 (4.6×)**	**0.05 (5.3×)**	5.76 (-0.03)
DenseNet	100	6.98 (1.0×)	1.77 (1.0×)	**4.67 (+0.00)**
DenseNet-PSP	100	**0.99 (7.1×)**	**0.22 (8.0×)**	4.87 (+0.20)
CIFAR100				
ResNet	56	0.86 (1.0×)	0.13 (1.0×)	27.79 (+0.00)
ResNet-PSP	56	**0.45 (1.9×)**	**0.07 (1.9×)**	**27.15 (-0.64)**
DenseNet	40	1.06 (1.0×)	0.27 (1.0×)	26.43 (+0.00)
DenseNet-PSP	40	**0.37 (2.9×)**	**0.08 (3.4×)**	26.30 **(-0.13)**
DenseNet	100	7.09 (1.0×)	1.77 (1.0×)	**22.83 (+0.00)**
DenseNet-PSP	100	**1.17 (6.1×)**	**0.24 (7.4×)**	23.42 (+0.41)

The same tendencies can be observed on the large-scale ImageNet task as shown in Table 4; when applying PSP, classification accuracy can be maintained (with some negligible degradation) and a considerable amount of structure can be removed from the networks (e.g. 2.6× from ResNet18 or 1.8× from DenseNet121). Furthermore, PSP obliterates the need for 1×1 bottleneck layers, effectively reducing network depth and MACs. For instance, removing the bottleneck layers from the DenseNet121 network in combination with PSP removes 2.6× parameters, 4.9× MACs and 1.9× layers, while only sacrificing 2.28% top-5 accuracy.

Table 4: ResNet and DenseNet on ImageNet using column pruning.

Model	Layers	Param. [10^6]	MAC [10^9]	Top-1 [%]	Top-5 [%]
ResNet-B	18	11.85 (1.0×)	1.82 (1.0×)	**29.60 (+0.00)**	**10.52 (+0.00)**
ResNet-B-PSP	18	**5.65 (2.1×)**	**0.82 (2.2×)**	30.37 (+0.67)	11.10 (+0.58)
ResNet-B	50	25.61 (1.0×)	4.09 (1.0×)	**23.68 (+0.00)**	6.85 (+0.00)
ResNet-B-PSP	50	**15.08 (1.7×)**	**2.26 (1.8×**	24.07 (+0.39)	**6.69 (-0.16)**
DenseNet-BC	121	7.91 (1.0×)	2.84 (1.0×)	**25.65 (+0.00)**	8.34 (+0.00)
DenseNet-BC-PSP	121	**4.38 (1.8×)**	**1.38 (2.1×)**	25.95 (+0.30)	**8.29 (-0.05)**
DenseNet-C	63	10.80 (1.0×)	3.05 (1.0×)	**28.87 (+0.00)**	**10.02 (+0.00)**
DenseNet-C-PSP	63	**3.03 (3.6×)**	**0.58 (5.3×)**	29.66 (+0.79)	10.62 (+0.60)
DenseNet-C	87	23.66 (1.0×)	5.23 (1.0×)	**26.31 (+0.00)**	**8.55 (+0.00)**
DenseNet-C-PSP	87	**4.87 (4.9×)**	**0.82 (6.4×)**	27.46 (+1.15)	9.15 (+0.40)

3.3 Ablation experiments

We end the experiments with an ablation experiment to validate methods and statements made in this work. This experiment is evaluated on the ResNet architecture, using column pruning, with 56 layers using the CIFAR10 dataset (Fig. 3). We report the validation error for varying sparsity constraints, and with the baseline error set to the original unpruned network, with some latitude to filter out fluctuations: $6.35\% \pm 0.25$. The dashed vertical lines indicate the maximum amount of sparsity while maintaining the baseline error.

A common way [13] to estimate the importance of structures is the ℓ_1 norm of the targeted structure in a weight tensor $A_{norm} = ||\mathrm{W}||_1$, which is followed by pruning the structures with the smallest norm. We use this rather simple approach as a baseline, denoted as ℓ_1 *norm*, to show the differences to the proposed parameterized structure pruning. The parameterization in its most basic form is denoted as *PSP (fixed sparsity)*, where we do not apply regularization ($\lambda = 0$) and simply prune the parameters with the lowest magnitude. As can be seen, the parameterization achieves about 10% more sparsity compared to the

baseline (ℓ_1 norm) approach, or 1.8% better accuracy under a sparsity constraint of 80%.

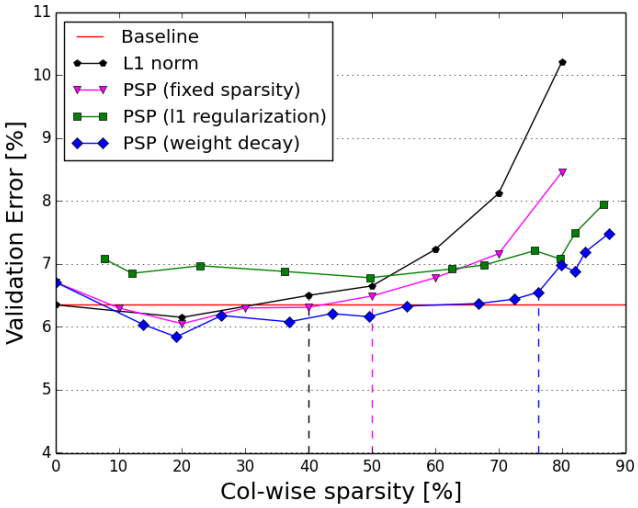


Fig. 3: ResNet network with 56 layers on CIFAR10 and column pruning.

Furthermore, we observe that regularized dense structure parameters are able to learn a clear distinction between important and unimportant structures (Sec. 2.2). Thus, it seems appropriate to use a simple threshold heuristic (Eq. 3) rather than pruning all layers equally (as compared to *PSP (fixed sparsity)*).

We also show the impact of the threshold heuristic in combination with ℓ_1 regularization and weight decay in Fig. 3. These methods are denoted as *PSP (ℓ_1 regularization)* and *PSP (weight decay)*, respectively. We vary the regularization strength for ℓ_1 regularization, since it induces sparsity implicitly, while we vary the threshold parameter for weight decay: for *PSP (ℓ_1 regularization)*, we set the threshold $\epsilon = 10^{-3}$ and the initial regularization strength $\lambda = 10^{-10}$, which is changed by an order of magnitude ($\times 10$) to show various sparsity levels. For *PSP*, we set the regularization strength $\lambda = 10^{-4}$ and the initial threshold $\epsilon = 0.0$ and increase ϵ by $2 \cdot 10^{-2}$ for each sparsity level. Both methods show higher accuracy for high sparsity constraints (sparsity $\geq 80\%$), but only weight decay achieves baseline accuracy.

4 Related Work

Re-training-based methods: [10] evaluate the importance of filters by calculating its absolute weight sum. [13] prune structures with the lowest ℓ_1 norm. Channel Pruning (CP) [7] uses an iterative two-step algorithm to prune each

layer by a LASSO regression based channel selection and least square reconstruction. Structured Probabilistic Pruning (SPP) [14] introduces a pruning probability for each weight where pruning is guided by sampling from the pruning probabilities. Soft Filter Pruning (SFP) [6] enables pruned filters to be updated when training the model after pruning, which results in larger model capacity and less dependency on the pre-trained model. ThiNet [12] shows that pruning filters based on statistical information calculated from the following layer is more accurate than using statistics of the current layer. Discrimination-aware Channel Pruning (DCP) [17] selects channels based on their discriminative power.

Regularization-based methods: Group Lasso [16] allows predefined groups in a model to be selected together. Adding an ℓ_1 penalty to each group is a heavily used approach for inducing structured sparsity in CNNs [9, 15]. Network Slimming [11] applies ℓ_1 regularization on coefficients of batch-normalization layers in order to create sparsity in a structured way.

In contrast to Group Lasso, the explicit parameterization allows to optimize and regularize certain structures independently to the weights of a tensor. Different to the regularization on coefficients of batch-normalization layers, the parameterization allows arbitrary structure selection within a tensor and, therefore, more efficient hardware mappings and structure granularities. Previous works leverage ℓ_1 regularization in order to enforce sparsity on structures although weight decay has superior convergence and generalization abilities for neural networks. The parameterization in combination with threshold-based magnitude pruning and straight-through estimation allows PSP to leverage the superior weight decay as regularizer.

5 Conclusion

We have presented PSP, a novel approach for compressing DNNs through structured pruning, which reduces memory and compute requirements while creating a form of sparsity that is inline with massively parallel processors. Our approach exhibits parameterization of arbitrary structures (e.g. channels or layers) in a weight tensor and uses weight decay to force certain structures towards zero, while clearly discriminating between important and unimportant structures. Combined with threshold-based magnitude pruning and backward approximation, we can remove a large amount of structure while maintaining prediction performance. Experiments using state-of-the-art DNN architectures on real-world tasks show the effectiveness of our approach. As a result, the gap between DNN-based application demand and capabilities of resource-constrained devices is reduced, while this method is applicable to a wide range of processors.

Acknowledgments

We gratefully acknowledge funding by the German Research Foundation (DFG) under the project number FR3273/1-1 and the Austrian Science Fund (FWF) under the project number I2706-N31.

References

1. Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. Technical Report, Universite de Montreal (2013)
2. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cudnn: Efficient primitives for deep learning. ArXiv (2014)
3. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. Advances in Neural Information Processing Systems (NIPS) (2015)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
5. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. International Conference on Computer Vision (ICCV) (2016)
6. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. nternational Joint Conference on Artificial Intelligence (IJCAI) (2018)
7. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. International Conference on Computer Vision (ICCV) (2017)
8. Huang, G., Liu, Z., Weinberger, K.Q.: Densely connected convolutional networks. Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
9. Lebedev, V., Lempitsky, V.S.: Fast convnets using group-wise brain damage. Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
10. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. International Conference on Learning Representations (ICLR) (2017)
11. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. International Conference on Computer Vision (ICCV) (2017)
12. Luo, J., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. International Conference on Computer Vision (ICCV) (2017)
13. Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., Dally, W.J.: Exploring the regularity of sparse structure in convolutional neural networks. Advances in Neural Information Processing Systems (NIPS) (2017)
14. Wang, H., Zhang, Q., Wang, Y., Hu, R.: Structured probabilistic pruning for deep convolutional neural network acceleration. British Machine Vision Conference (BMVC) (2018)
15. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. Advances in Neural Information Processing Systems (NIPS) (2016)
16. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society (2006)
17. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-aware channel pruning for deep neural networks. Advances in Neural Information Processing Systems (NIPS) (2018)