Fast ConvNets Using Group-wise Brain Damage

Vadim Lebedev[1,2] Victor Lempitsky[1]

[1] Skolkovo Institute of Science and Technology (Skoltech)
[2] Yandex
{vadim.lebedev, lempitsky}@skoltech.ru

Abstract

We revisit the idea of brain damage, i.e. the pruning of the coefficients of a neural network, and suggest how brain damage can be modified and used to speedup convolutional layers in ConvNets. The approach uses the fact that many efficient implementations reduce generalized convolutions to matrix multiplications. The suggested brain damage process prunes the convolutional kernel tensor in a group-wise fashion. After such pruning, convolutions can be reduced to multiplications of thinned dense matrices, which leads to speedup. We investigate different ways to add group-wise prunning to the learning process, and show that several-fold speedups of convolutional layers can be attained using group-sparsity regularizers. Our approach can adjust the shapes of the receptive fields in the convolutional layers, and even prune excessive feature maps from ConvNets, all in data-driven way.

1. Introduction

In the original *Optimal Brain Damage* work [32] of 25 years ago, LeCun et al. observed that a carefully designed "brain-damage" process can sparsify the coefficients of a multi-layer neural network very significantly while incurring minimal or no loss of the prediction accuracy. Such process resembles the biological learning processes in mammals, in whose brains the number of synapses peak during early childhood and is then reduced substantially in the process of *synaptic pruning* [8]. The optimal brain damage algorithm and its variants, however, impose sparsity in an unstructured way. As a result, while a large number of parameters can be pruned, the attained level of sparsity in the network is usually insufficient to achieve substantial computational speedup on modern architectures.

These days, due to the overwhelming success of very big convolutional neural networks (ConvNets) [30] on a variety of machine learning problems, the task of speeding up ConvNets has become a topic of active research and engineering. *Generalized convolution*, i.e. the operation of convolving a 4D *kernel tensor* with the stack of input maps in order to produce the stack of output maps, is at the core of ConvNets and also represents their speed bottleneck. Here, we present a simple approach that modifies the standard generalized convolution process by imposing *structured* "brain-damage" on the kernel tensor. We demonstrate that considerable speed-up of ConvNets can be obtained for a certain structure.

This structure is motivated by the observation that the majority of current implementations of generalized convolutions (including the most efficient one at the time of submission) [9, 16, 26, 12, 46, 37] compute generalized convolutions by reducing them to matrix multiplications (this reduction is also referred to as *lowering*, *unrolling*, or the `im2col` operation). While unstructured brain damage in a convolutional layer, i.e. shrinking some of the coefficients of the convolutional kernel tensor to zero, will make one of the factor matrices (*the filter matrix*) sparse, it will not make the overall multiplication run faster. Our idea therefore is to group together the entries of the convolutional tensor in a certain fashion and to shrink such groups to zero in a coordinated way. By doing this, we can eliminate rows and columns from both factor matrices that are multiplied when convolution is reduced to matrix multiplication. Repeated elimination of rows and columns makes both factor matrices thinner (but still dense) and results in faster matrix multiplication.

We demonstrate that conventional group sparsity regularizer [48] embedded into stochastic gradient descent minimization is able to accomplish group-wise brain damage efficiently. The use of group sparsity thus allows us to optimize receptive fields in the convolutional network. Our approach therefore makes the case for the natural idea of using structured sparsity as a simple way to optimize connectivity in deep architectures. In the experiments, our

speed-up factors exceed those obtained by recent tensor-factorization based methods. For example, we show that group-wise brain damage can accelerate the bottleneck layers of AlexNet ('conv2' and 'conv3') by a factor of 8.5x simultaneously, while incurring only modest (1%) loss of the prediction accuracy.

2. Related work

As ConvNets are growing in size and are spreading towards real-time and large-scale computer vision systems, a lot of attention is attracted to the problem of speeding up convolutional layers (e.g. through the use of Fourier transforms [36, 45]). Several recent works investigate various kinds of tensor factorization in order to break generalized convolution into a sequence of smaller convolutions with fewer parameters [21, 15, 29]. Using inexact low-rank factorizations within such approaches allows to obtain considerable speedup when low enough decomposition rank is used. Our approach is related to tensor-factorization approaches as we also seek to replace full convolution tensor with a tensor that has fewer parameters. Our approach however does not perform any sort of decomposition/factorization for the kernel tensor.

Another more distantly related approach is represented by a group of methods [2, 20, 41] that compress the initial large ConvNet into a smaller network with different architecture while trying to match the outputs of the two networks. Our approach is also related to methods that use structured sparsity [48, 42, 24] to discover optimal architectures of certain machine learners, e.g. to discover the optimal structure of a graphical model [22] or the optimal receptive fields in the two-layered image classifier [25]. On the other hand, since our approach effectively learns receptive fields within a ConvNet, it can be related to other receptive field learning approaches, e.g. [13, 35].

The combination of sparsity and deep learning has been investigated within several unsupervised approaches such as sparse autoencoders [5, 7] and sparse deep belief networks [33]. We also note two reports that use some form of sparsification of deep feedforward networks and appeared in the recent months as we were developing our approach. Similarly to [32], the work [14] uses sparsification to reduce the number of parameters in the memory-bound scenario. Their goal is thus to save memory rather than to attain acceleration. In the report of [17], the output of the convolution is computed at a sparsified set of locations with the gaps being filled by interpolation. This approach does not sparsify the convolutional kernel and is therefore different from the group-wise brain damage approach we suggest here.

Our work focuses on the task of speeding up convolutional layers (as they represent the speed bottleneck) and is therefore complementary to approaches that focus on the reduction of size/memory footprint of fully-connected layers [11, 18, 39, 43, 47].

Finally, we note that the use of sparse matrices in order to speed-up convolutional neural networks was vrey recently explored in [34]. Learning with regularizations (including group lasso) was used there to obtain sparse weights, which were applied to data via efficiently implemented sparse matrix multiplication. In contrast to their work, we aim to keep matrix multiplication dense (by imposig structure during sparsification), as such operation is much more efficient on modern architectures.

3. Group-Sparse Convolutions

Below, we discuss the reduction from generalized convolution to matrix multiplication [9] and introduce the notation along the way. We then explain the group-sparse convolution idea. Generalized convolution within a convolutional layer transforms an input stack of S maps of size $W' \times H'$, which can be treated as a three-dimensional tensor (array) U_{whs}, into an output stack of T maps of size $W'' \times H''$ which form a three-dimensional tensor V_{wht}. The exact relation between W', H' and W'', H'' depends on the padding and stride settings within the layer, and our approach can handle any padding/striding settings seamlessly. The transformation is defined by the following formula:

$$V(x,y,t) = \sum_{s=1}^{S} \sum_{\substack{i=1..d \\ j=1..d}} K(i,j,s,t) \cdot U(x+i-\tfrac{d+1}{2}, y+j-\tfrac{d+1}{2}, s) \quad (1)$$

Here, K is a four-dimensional *kernel tensor* of size $d \times d \times S \times T$ with the first two dimensions corresponding to the spatial dimensions, the third dimension corresponding to input maps, the fourth dimension corresponding to output maps. The spatial width and height of the kernel are denoted as d (for simplicity, we assume square shaped kernels and odd d).

The implementation of (1) constitutes the speed bottleneck for ConvNets. In [10], it was suggested to reduce the computation of all entries of V to the multiplication of two large and dense matrices. The reduction allows to use highly optimized implementations of dense matrix multiplications (e.g. variants of BLAS [6] libraries) that have been developed over many years for all possible computing architectures. The reduction proceeds as follows:

- The kernel tensor K is reshaped into the *filter matrix* F of size $T \times d^2 S$, where the t-th row corresponds to a sequence of S 2D filters $K(:,:,s,t)$ reshaped in a row-wise fashion into row vectors.
- The input map stack U is reshaped into the *patch matrix* P of size $d^2 S \times W'' H''$, where the l-th column

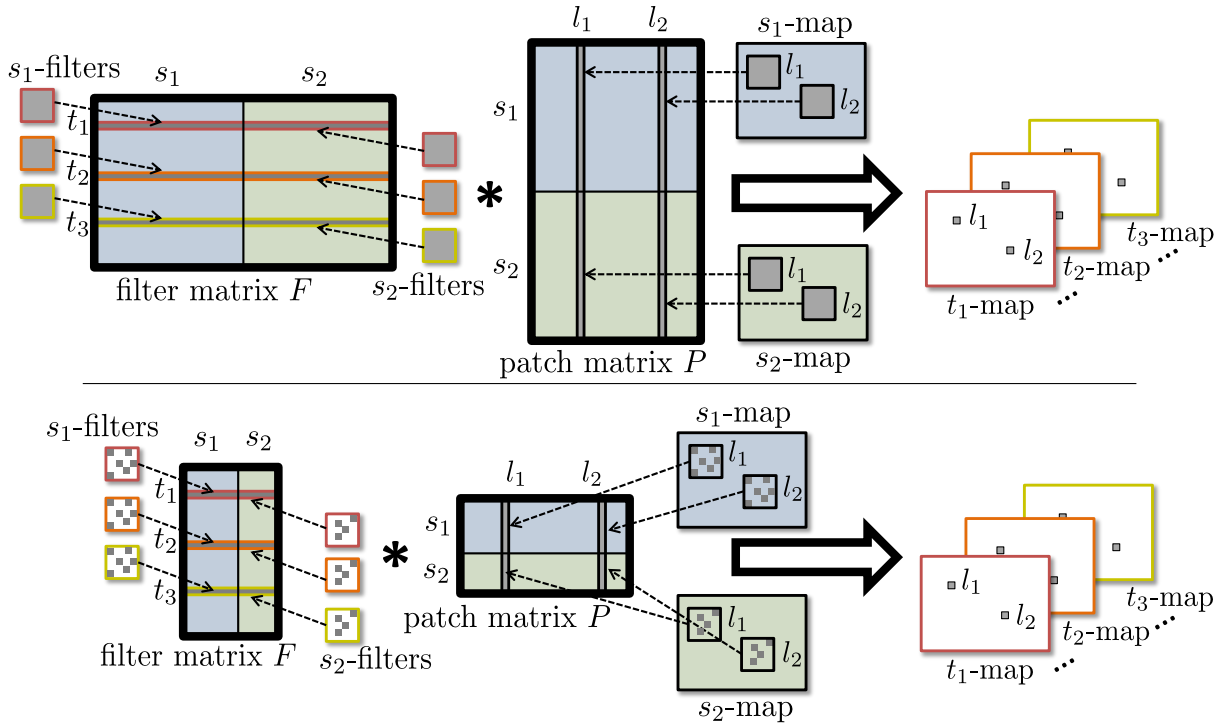


Figure 1: Standard Generalized Convolution (top) vs. Generalized Convolution after Group-wise Brain Damage (bottom). In both cases, we show the diagram for two input maps ($S = 2$, blue-green color coding). We highlight three output maps t_1, t_2, t_3 color-coded red-orange-yellow, and we also highlight two spatial locations l_1 and l_2. In both cases, the output map stack is obtained by reshaping the product of the filter matrix and the patch matrix. In the standard case, the filters and the patches sampled during the formation of the patch matrix are dense. After group-wise brain damage, both the filters and the patch sampling patterns are group-sparse (one sparsity pattern per input map), which results in much thinner filter and patch matrices and thus leads to much faster matrix multiplication/convolution.

corresponds to a certain output location $l = (x, y)$ and is stacked from the S patches extracted from S input maps, all centered at this location and reshaped in a row-wise fashion into column vectors.

- The filter matrix F is multiplied by the patch matrix P resulting in a matrix $\tilde{V}$ of size $T \times W''H''$ that contains all the elements of V (each column corresponds to a certain location and contains the values of this location in the T output maps). The multiplication implements (1) exactly, as each row-by-column product within the multiplication corresponds to one instance of the computation (1) for certain (x, y, t). The output tensor (map stack) V can be obtained from $\tilde{V}$ by reshaping.

The construction discussed above has proven to be highly successful and is used in the majority of modern ConvNets "backends", e.g. [10, 16, 26, 12, 46, 37]. Our key idea is to train ConvNets with sparse convolutional kernels that are consistent with this construction.

Such consistency can be achieved if the sparsity patterns are aligned in a certain way. Formally, group-wise brain damage introduces a *sparsity pattern* Q_s for every input map $s \in 1 \dots S$. The sparsity pattern is defined as a subset of the full spatial d-by-d grid, i.e. $Q_s \subset \{1 \dots d\} \otimes \{1 \dots d\}$. The convolutional operation then becomes a slight modification of (1):

$$V(x, y, t) = \sum_{s=1}^{S} \sum_{(i,j) \in Q_s} K(i, j, s, t) \cdot U(x+i-\tfrac{d+1}{2}, y+j-\tfrac{d+1}{2}, s) \quad (2)$$

The reduction of (2) is an almost straightforward replication of the procedure [10]. The only modifications are (Figure 1):

- When the filter matrix is assembled, each 2D filter $K(:, :, s, t)$ is reshaped into a row-vector of length $|Q_s|$ by including only non-zero elements. The filter matrix thus becomes of size $T \times \sum_{s=1}^{S} |Q_s|$.

- When the patch matrix is assembled, each 2D patch at

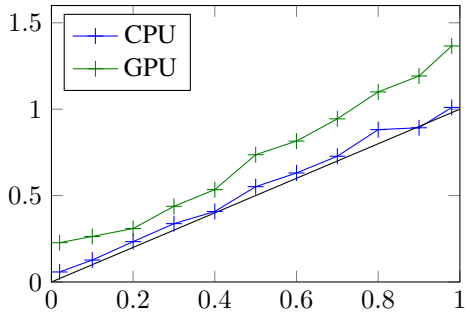


Figure 2: The blue curve shows relative speed-up versus density level τ, measured for forward propagation in the second convolutional layer of AlexNet on CPU and GPU. Importantly, the observed speedup is almost linear in the sparsity level (diagonal). Green curve shows the 'im2col' layer speedup on a GPU. While a certain constant overhead can be seen, we believe that (part of) it can be eliminated via more elaborated GPU implementation.

location $l = (x, y)$ in map S is reshaped into a column vector of size $|Q_s|$ by sampling the input map $U(:, :, s)$ sparsely at locations $(x+i-\frac{d+1}{2}, y+j-\frac{d+1}{2})$, where $(i, j) \in Q_s$. The patch matrix thus becomes of size $\sum_{s=1}^{S} |Q_s| \times W''H''$.

As a result of this modification, the multiplication of two dense matrices of sizes $T \times d^2S$ and $d^2S \times W''H''$ is replaced by the multiplication of two dense matrices of sizes $T \times \sum_{s=1}^{S} |Q_s|$ and $\sum_{s=1}^{S} |Q_s| \times W''H''$, which results in the $d^2S / \sum_{s=1}^{S} |Q_s|$-times reduction in the number of scalar operations. In our experiments with the reference implementation of [27] the wall-clock reduction in the convolution time between the original implementation and the group-sparse convolution was almost matching the "theoretical" speed-up factor (Figure 2).

4. Fast ConvNets with Group-sparse convolutions

We consider two different scenarios that obtain fast ConvNets with group-sparse convolutions. First, we consider training such networks from scratch, and secondly we consider obtaining such networks by modification of pretrained architectures (i.e. performing "brain damage").

4.1. Training from scratch

Predefined group-sparsity pattern. The simplest solution that we consider is to choose the sparsity patterns Ω_S in advance in a data-independent manner, and enforce these patterns during the learning of the network. One particular case of this approach is simply reducing the spatial size of filters to a minimum, e.g. three-by-three, or even smaller rectangular pattern all the way to one-by-one (this is in line with a recent work of [19] where they consider 2×2 filters for some of their architectures). Note, that with our approach we are free to choose non-rectangular filters, and in the experiments we found this very useful.

One of the downsides of this approach is that when designing an architecture with multiple convolution layers, there are no clear design principles that can guide the choice of the filter shapes. In contrast, the methods discussed below can start with larger filters and then shrink their sizes towards optimally-shaped small filters.

Training with group-sparsity regularizer. Rather than fixing the group-sparsity pattern in advance, it is possible to find it as a part of learning process while the network is trained. A classical way to achieve this is through the use of group-sparsity regularization [48, 42, 24]. Thus, we consider a regularizer based on $l_{2,1}$-norm:

$$\Omega_{2,1}(K) = \lambda \sum_{i,j,s} \|\Gamma_{ijs}\| = \lambda \sum_{i,j,s} \sqrt{\sum_{t=1}^{T} K(i,j,s,t)^2}\,, \tag{3}$$

where the vector Γ_{ijs} denotes the group of kernel tensor entries $K(i, j, s, :)$. The effect of the regularizer (3) is in shrinking some of such groups to zero in a coordinated fashion. When an entire group Γ_{ijs} is set to zero, one can set the pixel (i, j) in the sparsity pattern Ω_s to zero, thus increasing the group-sparsity.

For a convolutional layer that is being sparsified, the gradient of (3), i.e.:

$$\frac{\partial \Omega_{2,1}(K)}{\partial K(i,j,s,t)} = \lambda \frac{K(i,j,s,t)}{\sqrt{\sum_{z=1}^{T} K(i,j,s,z)^2}} \tag{4}$$

can simply be added to the gradient of the learning loss while performing stochastic gradient updates in the course of learning. The coefficient λ in (3) and (4) controls the strength of the regularization w.r.t. the main learning loss.

Generally, using the regularizer (3) will result in a group-sparsified kernel tensor with some of Γ_{ijs} having only near-zero entries. Because of the stochastic nature of SGD and non-differentiability of l_{12} norm near zero, the entries in these groups will not be exactly zero, and further postprocessing is needed to nullify the near zero groups and to set the sparsity patterns Ω_S accordingly.

4.2. Sparsifying with Group-wise Brain Damage

While it is possible to train ConvNets with group-sparse convolutions from scratch, the main focus of our paper is developing algorithms that can speed-up existing pretrained networks that often take excessive time for training. Towards this end, we have developed two approaches that can accelerate pretrained networks by inflicting group-wise

brain damage in a way that the drop in the prediction accuracy is kept small. In both cases, we assume that we have access to the training dataset D, the model was trained on.

Group-wise sparsification with fine-tuning. Our first implementation is also based on the group-sparse regularizer (3). We start with the input ConvNet and run the learning process on the dataset D with the added regularizer (3). After a certain amount of iterations, a predefined number of groups Γ_{ijs} with the smallest $l2$-norm is set to zero. For a desired density level $\tau \in [0, 1]$ and respective speedup $1/\tau$, we set $d^2 S\,(1 - \tau)$ groups to zero, making the respective Q_S sparse.

We have found two complications with this approach. Firstly, for a given density τ it was generally hard to set appropriate regularization strength λ in advance without trying several values. Secondly, for small τ (large speedup) the appropriate regularization strength λ typically leads to an excessive regularization, as many groups end up being biased towards zero but not close to zero. Because of that, the prediction accuracy for such λ experienced significant drop in the process of learning as compared to the input ConvNet.

Fortunately, one can recover from most of this drop by the subsequent *fine-tuning* of the network, that follows after the brain-damage process. For the fine-tuning, we fix the sparsity patterns Q_S and restart learning without group-sparse regularization. We then train for an excessive number of epochs. As a result of such fine-tuning, the network adapts to the imposed sparsity patterns, while the prediction accuracy goes up and recovers most of the drop.

Gradual group-wise sparsification. To avoid the two complications discussed above we developed an alternative approach that essentially combines the brain-damage and the fine-tuning processes, and furthermore avoids most of the need for manual search for good meta-parameter values. The approach also often leads to considerably better results.

In this approach, we consider the *truncated* l_{12} regularizer:

$$\Omega^T_{2,1}(K) = \lambda \sum_{i,j,s} \min(\|\Gamma_{ijs}\|, \theta) \qquad (5)$$

The gradient of (5) equals (4) when $\|\Gamma_{ijs}\| < \theta$ and is zero otherwise. Informally speaking, the value of θ controls which groups are considered "promising" and are being shrinked towards zero, and which groups are considered to be too far from zero and therefore stay unaffected by the regularizer (5).

To perform brain-damage, we then create a validation set on which we monitor the performance of the network. We choose the maximum drop δ of the prediction accuracy on the validation set that we are willing to tolerate. We then start with an input ConvNet and perform learning with the regularizer (5) while varying θ. Specifically, after each epoch we monitor the performance of the network on a hold-out set and increase θ (intensifying brain damage) if the accuracy drop is less than δ and decrease θ, thus relieving certain groups from the effect of the regularizer, if the drop is greater than δ.

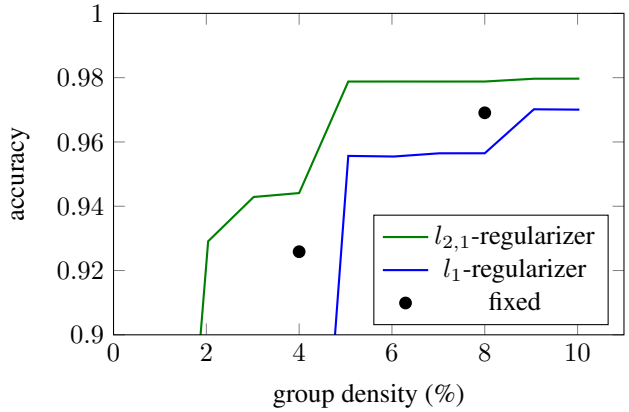


Figure 3: Accuracy vs. density level on MNIST dataset (LeNet architecture) for various ConvNets with group-sparse convolutions. We compare the results obtained by training with $l_{2,1}$ and l_1 regularizations followed by sparsifications, as well as training with predefined sparsity patterns Ω_S (black dots). Overall, training with $l_{2,1}$ regularizer obtains the best result that can be further improved by fine-tuning without regularization.

To perform the actual sparsification, we also introduce an additional threshold $\epsilon \ll \delta$. In the process of learning, when the norm of a certain group falls below the threshold (i.e. $\|\Gamma_{ijs}\| < \epsilon$) the group is greedily fixed to zero and eliminated from the tensor. The sparsity thus monotonically increases through the process, and we carry on training until the sparsification process stalls, i.e. the system keeps training with Γ and performance drop oscillating, while no new groups have their lengths fall under ϵ for a number of epochs. In our experiments, all increments and decrements of θ was based on five-percent quantiles of the groups. I.e. every time θ is adjusted, we set θ to bring 5% of groups Γ_{ijs} in or out of the $\|\Gamma_{ijs}\| < \theta$ "territory".

Overall, we found the whole procedure to be rather insensitive to the choices of λ and ϵ, and overall to be more practical and lead to higher group-sparsity and speed-ups than those attainable by the sparsification with fine-tuning approach. Most importantly, we could use same λ and ϵ, as well as same shared value of θ when sparsifiying multiple layers simultaneously.

5. Experiments

Implementation details. Our implementation is based on `Caffe` [27] and modifies their original convolution, which is implemented as two subsequent transformations

method	density	speed-up	accuracy drop
Accelerating the second convolutional layer of AlexNet			
Denton et al. [15]: Tensor decomposition + Fine-tuning		2.7x	$\sim 1\%$
Lebedev et al. [29]: CP-decomposition + Fine-tuning		4.5x	$\sim 1\%$
Jaderberg et al. [21]: Tensor decomposition + Fine-tuning		6.6x	$\sim 1\%$
Training with fixed sparsity patterns	0.12	8.33	0.82%
Training with fixed sparsity patterns	0.2	5x	0.16%
Group-wise sparsification + Fine-tuning	0.1	10x	1.13%
Group-wise sparsification + Fine-tuning	0.2	5x	0.43%
Group-wise sparsification + Fine-tuning	0.3	3.33x	0.11%
Group-wise sparsification + Fine-tuning	0.4	2.5x	-0.09%
Gradual group-wise sparsification	0.11	9.0x	0.28%
Gradual group-wise sparsification	0.05	20x	1.07%
Accelerating the second and the third convolutional layers of AlexNet			
Training with fixed sparsity patterns	0.12	8.7x	1.54%
Training with fixed sparsity patterns	0.35	2.9x	0.36%
Training with fixed sparsity patterns	0.54	1.9x	-0.53%
Group-wise sparsification + Fine-tuning	0.2	5x	1.50%
Group-wise sparsification + Fine-tuning	0.3	3.33x	1.17%
Group-wise sparsification + Fine-tuning	0.5	2x	0.57%
Gradual group-wise sparsification	0.12	8.5x	1.04%
Accelerating all five convolutional layers of AlexNet			
Training with fixed sparsity patterns	0.34	3.0x	1.34%
Gradual group-wise sparsification	0.31	3.2x	1.43%

Table 1: **Accelerating convolutional layers of the pretrained AlexNet architecture**: results of the two variants of our method for various sparsity levels alongside tensor-decomposition based methods (note: the results for [21] are reproduced from [29]).

(`im2col` and matrix multiplication). To implement the group-sparse convolution we focused on the forward propagation step and CPU computation. Most of our methods can be extended for backprop step and for GPUs, however making such extensions efficient is non-trivial. For our purpose, we only needed to modify the `im2col` function, so that it can fill in the patch matrix while following certain sparsity patterns.

Datasets. We perform the following experiments. Firstly, we consider a small-scale setting, and compare training ConvNets with group-wise brain damage from scratch with baselines. We use MNIST dataset [31] for these small-scale experiments. We then consider a large-scale problem, namely ImageNet (ILSVRC) image classification and the task of accelerating of a pretrained architecture, namely the Caffe version of AlexNet [28]. We also give preliminary results for one of the VGGNet networks [44].

5.1. MNIST experiments

We trained the LeNet architecture on the MNIST dataset from random initialization while adding the group-sparse regularization (section 4.1) while varying the regularization strength λ and picking the optimal one for each sparsity level. The sparsification affects both convolutional layers of LeNet, and the same density level τ is enforced in both layers. We also consider a number of baselines:

- A simple baseline that trains the network without regularization and then simply eliminates (sets to zero) a certain number of groups Γ_{ijs} with the smallest l2-norms. The performance of this baseline was clearly below all other methods and it is not reported.

- Picking sparsity patterns Q_S in advance. We consider filters with only one central non-zero entry and filters with two adjacent central non-zero elements. These options correspond to the density of 4% and 8% respectively. The former is essentially equivalent to a non-convolutional network, where almost all processing happens in the fully-connected layers.

- We also consider a simpler non-group-wise sparsification by training with l1-norm regularizer (with varying λ) but then nullifying groups $|\Gamma_{ijs}$ based on their norms.

The results of the proposed method and the baselines are shown in Figure 3. The rightmost plot shows the comparison of the l_1-envelope, $l_{2,1}$-envelope, and the performance of the group-wise brain damage applied to the network trained without sparsity-inducing regularizer. The use of group-sparsity regularization boosts the performance of group-wise brain damage very considerably. Twenty-fold acceleration of convolutional layers can be obtained while keeping the error low (2.1%, reduced to 1.71% after fine-tuning). Using l_1-regularizer followed by optimal brain damage works worse than $l_{2,1}$-regularizer. Pre-fixing sparsity patterns achieves good results, which are still worse than training with group-sparsity regularizer. Note also that all methods except the baseline with the pre-fixed patterns can be improved via fine-tuning.

5.2. ILSVRC experiments

We first consider the AlexNet (Caffe reimplementation) architecture that has five convolutional layers. We consider the following subtasks: (i) accelerating the second convolutional layer (which is the slowest of all layers), (ii) accelerating the second and the third layers (which are the two slowest layers), (iii) accelerating all five convolutional layers (which together take the vast majority of the forward-propagation time). When reporting the final density in subtasks (ii) and (iii), we weigh the densities in different layers by the forward propagation times.

We focused on accelerating the existing network from Caffe zoo (Table 1). As an additional baseline, we evaluate the variant of our method that trims the network according to some predefined sparsity pattern, and then learns the network while keeping the same fixed pattern. Namely we consider the following symmetric centered patterns: vertical or horizontal block 1×3, the 3×3 cross pattern, 3×3 square or diamond shape inside 5×5 filter.

For the first two subtasks, we evaluated the variant of our method with sparsity-inducing regularizer for various sparsity levels. For several desired density levels τ we searched for optimal λ through a large range with ten-fold increments. For each τ we pick λ that results in the minimal accuracy drop after sparsification before fine-tuning. After picking the optimal λ, we perform fine-tuning (with fixed sparsity patterns). Figure 4 demonstrate sparsity patterns Ω_S obtained for different sparsity levels.

Finally, for all three subtasks we evaluated the most advanced of our methods, namely gradual group-wise sparsification. We set the parameters λ and ϵ to 0.01 and 0.1 respectively. We split the test set of ILSVRC randomly into two halves and use one of the halves solely to estimate the drop of the classification accuracy in the dynamical adjustment of θ. We then report the performance drop on the other half of the test set. We set the acceptable performance drop to be 1% of top-1 accuracy.

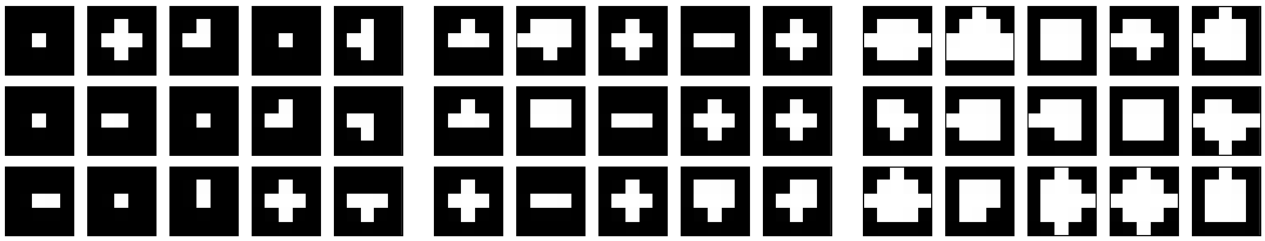
As shown in Table 1, the results of gradual sparsification outperform the tensor factorization methods as well as sparsification with fine-tuning considerably, achieving higher group-sparsification/speed-up for similar prediction accuracy drop. Notably, the proposed approach is more successful in speeding-up AlexNet than a number of approaches based on tensor decomposition. Figure 5 further visualizes the process of the simultaneous gradual brain damage inflicted on all five layers of AlexNet.

"External" computer vision task. Convolutional layers of large networks pretrained on large annotated training sets such as ILSVRC can be used as universal spatially localized features in a variety of ways [34, 1], which is particularly valuable for problems with considerably smaller training sets. Recently, [3] showed that descriptors obtained by sum-pooling of the features that emerge in the last convolutional layer of a pretrained network can be used as state-of-the-art holistic descriptors for image retrieval. We followed their approach (that includes PCA whitening and normalization as postprocessing) to assess the effect of group-sparsification on an external task. Comparing AlexNet as a base model, and the network with the simultaneous group-sparisification of all convolutional layers from Table 1 with $3.2x$ speedup, we have found a negligible drop in performance for the INRIA holidays dataset [23] from 0.783 mAP to 0.780 mAP, and a reasonably small drop for the Oxford Building dataset [40] from 0.45 to 0.41.

VGGNet results. We have also applied the gradual group-wise sparsification to the slowest convolutional layer of VGGNet (the deeper 19 layer version of [44], starting from its Caffe Zoo version. The sparsification obtained the density $\tau = 0.13$ with only 0.2% top-1 accuracy drop. Interestingly, unlike the experiments with AlexNet where we rarely observed empty sparsity patterns Ω_S ("dead feature maps"), in this example such all-zero patterns were present (29 out of 64), suggesting that this manually designed architecture contains excessive number of feature maps in this layer. This result also suggest that our approach is suitable even for networks with very small initial filter sizes in convolutional layers (3×3 for VGGNet). When applied to all convolutional layers of the VGG network, our method obtains different densities for different layer. It can pose a problem if some layers are reaching density close to zero while others are mostly dense, which proved to be the case with VGGNet. We've implemented rescaling described in [38] to level group norms of different layers and reached density $\tau = 0.45$ with this approach, which corresponds to more than $2\times$ speedup with 0.7% accuracy drop

6. Discussion

We have presented an approach to speeding up ConvNets that uses the group-wise brain damage process that sparsifies convolution operations. The approach takes into ac-



(a) sparsity $1 - \tau = 0.9$ (b) sparsity $1 - \tau = 0.8$ (c) sparsity $1 - \tau = 0.6$

Figure 4: The sparsity patterns obtained by group-wise brain damage on the second convolutional layer of AlexNet for different sparsity levels. Nonzero weights are shown in white. In general, group-wise brain damage shrinks the receptive fields towards the center and tends to make them circular.

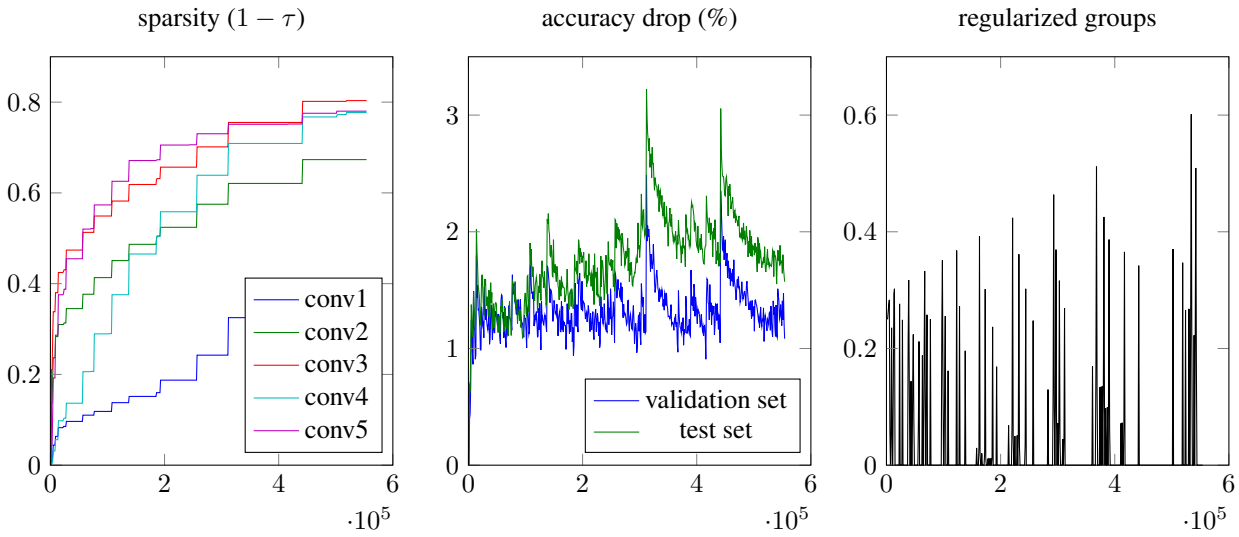


Figure 5: The process of sparsification of all five layers in AlexNet. The left plot shows the monotonic growth of the sparsity levels of the five convolutional layers as the iterations progress. The middle plot shows the relative prediction accuracy drop for the current system for the validation part and for the hold-out test set. Finally, the right part visualizes the process of the adjustment of θ threshold in the truncated $l_{2,1}$ regularization. This plot shows the percentile of groups Γ_{ijs} with the l_2-norm less than θ. θ is increased or decreased dependent on whether the performance drop on the validation set is greater or smaller than 1.2%.

count the way generalized convolutions are reduced to matrix multiplications, and prune the entries of the convolution kernel in a groupwise fashion. The exact sparsity patterns can be learned from data using group-sparsity regularization. When applied after learning with such regularization and followed by fine-tuning, group-wise brain damage obtains state-of-the-art performance for speeding up ConvNets.

Aside from the practical value, the proposed approach also makes the case for the use of sparse learning for automated discovery of optimal network architectures, which is arguably one of the main unsolved problems in deep learning. In our case, group-sparse regularizer allows the model to discover optimal receptive fields (Figure 4). It is interesting to see that the optimization process decided to shrink the receptive fields towards the center compared to the full version (which is consistent with the findings in [44, 19]). Perhaps, even more interesting is to see that in general, the learning process decided to make the receptive fields roughly circular. Also, the process treated AlexNet and VGGNet differently, eliminating entire feature maps by assigning their sparsity patterns Ω_S to zero maps in the latter case. Note that such elimination brings additional speedup (since the entire map needs not be computed in the previous layer). Such elimination can be explicitly encouraged within our approach using hierarchical group-sparsity regularizers [4, 24].

References

[1] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson. From generic to specific deep representations for visual recognition. *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops, Boston, MA, USA, June 7-12, 2015*, pp. 36–45, 2015.

[2] J. Ba and R. Caruana. Do deep nets really need to be deep? *Advances in Neural Information Processing Systems (NIPS)*, pp. 2654–2662, 2014.

[3] A. Babenko and V. Lempitsky. Aggregating local deep features for image retrieval. *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[4] F. R. Bach. Exploring large feature spaces with hierarchical multiple kernel learning. *Advances in neural information processing systems (NIPS)*, pp. 105–112, 2009.

[5] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems (NIPS)*, 19:153, 2007.

[6] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, and G. Henry. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[7] Y.-l. Boureau and Y. LeCun. Sparse feature learning for deep belief networks. *Advances in neural information processing systems (NIPS)*, pp. 1185–1192, 2008.

[8] G. Chechik, I. Meilijson, and E. Ruppin. Synaptic pruning in development: a computational account. *Neural computation*, 10(7):1759–1777, 1998.

[9] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[10] K. Chellapilla, S. Puri, and P. Simard. High Performance Convolutional Neural Networks for Document Processing. In G. Lorette, editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), 2006. Université de Rennes 1, Suvisoft. http://www.suvisoft.com.

[11] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 2285–2294, 2015.

[12] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[13] A. Coates and A. Y. Ng. Selecting receptive fields in deep networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, pp. 2528–2536. Curran Associates, Inc., 2011.

[14] M. D. Collins and P. Kohli. Memory bounded deep convolutional networks. *CoRR*, abs/1412.1442, 2014.

[15] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Inernational Conference on Learning Representations (ICLR)*, 2014.

[16] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. *Proceedings of The 31st International Conference on Machine Learning*, pp. 647–655, 2014.

[17] M. Figurnov, D. Vetrov, and P. Kohli. Perforated CNNs: Acceleration through elimination of redundant convolutions. *CoRR*, abs/1504.08362, 2015.

[18] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 1737–1746, 2015.

[19] K. He and J. Sun. Convolutional neural networks at constrained time cost. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[20] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS 2014 Deep Learning Workshop*, 2014.

[21] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *Proceedings of the British Machine Vision Conference (BMVC)*, 2014.

[22] A. Jalali, P. D. Ravikumar, V. Vasuki, and S. Sanghavi. On learning discrete graphical models using group-sparse regularization. *International Conference on Artificial Intelligence and Statistics*, pp. 378–387, 2011.

[23] H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. *European Conference on Computer Vision*, 2008.

[24] R. Jenatton, J.-Y. Audibert, and F. Bach. Structured variable selection with sparsity-inducing norms. *The Journal of Machine Learning Research*, 12:2777–2824, 2011.

[25] Y. Jia, C. Huang, and T. Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3370–3377. IEEE, 2012.

[26] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678. ACM, 2014.

[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems (NIPS)*, pp. 1097–1105, 2012.

[29] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *Inernational Conference on Learning Representations (ICLR)*, 2015.

[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. *Advances in Neural Information Processing Systems (NIPS)*, pp. 598–605. Morgan Kaufmann, 1990.

[33] H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief net model for visual area v2. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems (NIPS)*, pp. 873–880. Curran Associates, Inc., 2008.

[34] L. Liu, C. Shen, and A. van den Hengel. The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[35] M. Malinowski and M. Fritz. Learning smooth pooling regions for visual recognition. *24th British Machine Vision Conference*, pp. 1–11. BMVA Press, 2013.

[36] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through FFTs. *arXiv preprint arXiv:1312.5851*, 2013.

[37] NervanaSystems. NervanaGPU. `https://github.com/NervanaSystems/nervanagpu`, 2015.

[38] B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro. Data-dependent path normalization in neural networks. *CoRR*, abs/1511.06747, 2015.

[39] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. Tensorizing neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[40] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[41] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *Inernational Conference on Learning Representations (ICLR)*, 2015.

[42] V. Roth and B. Fischer. The group-lasso for generalized linear models: uniqueness of solutions and efficient algorithms. *Proceedings of the 25th international conference on Machine learning*, pp. 848–855. ACM, 2008.

[43] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pp. 6655–6659, 2013.

[44] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *Inernational Conference on Learning Representations (ICLR)*, 2015.

[45] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast convolutional nets with fbfft: A GPU performance evaluation. *Inernational Conference on Learning Representations (ICLR)*, 2015.

[46] A. Vedaldi and K. Lenc. MatConvNet – convolutional neural networks for matlab. *CoRR*, abs/1412.4564, 2014.

[47] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pp. 2365–2369, 2013.

[48] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.