PRUNE YOUR NEURONS BLINDLY: NEURAL NETWORK COMPRESSION THROUGH STRUCTURED CLASS-BLIND PRUNING

Abdullah Salama, Oleksiy Ostapenko, Tassilo Klein, Moin Nabi

SAP Machine Learning Research

ABSTRACT

High performance of deep learning models typically comes at cost of considerable model size and computation time. These factors limit applicability for deployment on memory and battery constrained devices such as mobile phones or embedded systems. In this work, we propose a novel pruning technique that eliminates entire filters and neurons according to their relative L1-norm as compared to the rest of the network, yielding more compression and decreased parameters' redundancy. The resulting network is non-sparse, however, much more compact and requires no special infrastructure for deployment. We prove the viability of our method by achieving 97.4%, 86.1%, 47.8% and 53% compression of LeNet-5, VGG-16, ResNet-56 and ResNet-110 respectively, exceeding state-of-the-art compression results reported on VGG-16 and ResNet without losing any performance compared to the baseline. Our approach does not only exhibit good performance but is also easy to implement.

Index Terms— Model Compression, Pruning, Efficient Deep Learning, Neural Networks

1. INTRODUCTION

While deep learning models have become the method of choice for a multitude of applications, their training entails optimizing a large number of parameters at extensive computational costs (energy, memory footprint, inference time).

This effectively limits their deployment on storage and battery constrained devices, such as mobile phones and embedded systems. To study their parameterization behavior, in [1], significant parameters' redundancy was shown in several deep learning models. To reduce redundancy and compress deep learning models without loss in accuracy, previous work proposed pruning weights by optimizing network complexity using second-order derivative information [2, 3]. Due to the high computational overhead of second order derivative, low-rank approximations to reduce the size of the weight tensors were explored in [4, 5] .

Another line of work [6, 7], proposed to prune individual layer weights with the lowest absolute value (non-structural sparsification of layer weights). The same strategy was followed in [8] while incorporating quantization and Huffman coding to further boost compression. While the aforementioned methods considered every layer independently, [9] pruned the network weights in a class-blind manner, e.g. individual layer weights are pruned according to their magnitude as compared to all weights in the network.

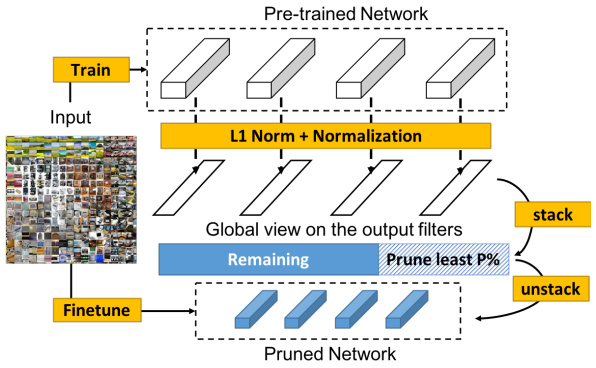

Fig. 1: Overview. We calculate the L1-norm of each filter in each layer, then we normalize each filter norm according to its number of kernel weights, followed by stacking all normalized norms of all filters from all layers, finally, sort then prune the filters corresponding to the least $p\%$ of normalized norms.

Noteworthy, all non-structured pruning approaches, generally result in high sparsity models that require special hardware and software. Structured pruning alleviates this by removing whole filters or neurons, producing a non-sparse compressed model. In this regard, channel-wise pruning according to the L1-norm of the corresponding filter was proposed in [10]. [11] learned a compact model based on learning structured sparsity of different parameters. An algorithm was implemented to remove redundant neurons iteratively on fully connected layers in [12]. In [13], connections leading to weak activations were pruned. Finally, [14] pruned the least important neurons after measuring their importance with respect to the penultimate layer. Generally, in structured pruning,each layer's importance/sensitivity to pruning was evaluated separately and each layer was pruned accordingly.

This work features two key components: a) **Blindness**: all layers are considered simultaneously; blind pruning was first introduced by [9] to prune individual weights; b) **Structured Pruning**: removal of entire filters instead of individual

weights. To the best of our knowledge, we are the first to use these two components together to consider structured pruning across all layers simultaneously. This is achieved by pruning filters based on their relative L1-norm compared to the sum of all filters' L1-norms across the network, instead of pruning filters according to their L1-norm within the layer [10], inducing a global importance score for each filter (Fig 1). Most importantly, due to the global importance score constructed, we do not impose any restrictions on which layers or filters to be pruned. his is in contrast to the limitations of [10], which propose pre-calculation of layers' sensitivity to pruning and consequently avoid pruning sensitive layers completely, assuming that a layer containing some high sensitive filters is inherently very sensitive. Such assumption is not accurate, as each layer can contain different filter sensitivities and subsequently least sensitive filters can be pruned

The contribution of this paper is two-fold: **i)** Proposing a structured class-blind pruning technique to compress the network by removing whole filters and neurons, which results in a compact non-sparse network with the same baseline performance. **ii)** Introducing a visualization of global filter importance to devise the pruning percentage of each layer.

As a result, the proposed approach achieves higher compression gains with higher accuracy compared to the state-of-the-art results reported on VGG-16, ResNet-56 and ResNet-110 on the CIFAR10 dataset [15].

2. BACKGROUND

Several work studied compressing deep learning models while maintaining the same baseline performance. TThe most related works to our method can be categorized as following:

Weight Pruning: Recently, [6] proposed pruning multiple deep learning models up to 92% by zeroing out the least percentage of weights per layer based on a layer weight standard of deviation. A follow-up work by [8] incorporated pruning by quantization and Huffman coding to limit the non-sparse weight representation, and compressing the resulting weight representation, respectively. Moreover, [7] proposed binary masks that are learned with the weights simultaneously during training. Thereby, the weights are multiplied by the masks, resulting in a sparse parameter representation. While all these methods produce high sparsity models, their hyperparameter optimization is very complex. Most importantly, in order to benefit from high sparsity models, special hardware or software is required.

Structured Pruning. To address the limitation of sparsity induced pruning, structured pruning was introduced. The underlying idea is to produce structured sparsity, e.g. remove parts of the structure, equivalent to filters/neurons in CNNs. [10] pruned filters having the lowest weights in terms of L1-norm within each layer, eventually removing filters of a trained network, followed by retraining the network to regain accuracy. [11] proposed a method that learns a compact model based on learning structured sparsity of filters, channels, filter shapes, and layer depth of the base model. Moreover, [12] implemented a data-free algorithm to remove redundant neuron by neuron iteratively on fully connected layers. Also, [13] identified redundant connections based on analyzing weak neurons on a validation set according to their weak activations. Thus, pruning the connections leading to the weak neurons iteratively until a compact model is obtained. In [16], a sparsity regularizer is imposed on outputs of various network entities such as neurons, groups or residual blocks after introducing a scale factor to scale the structures' output. By minimizing the scaling factors, speed-up on some models was achieved as well as limited compression results. Finally, [14] pruned neurons by measuring the neuron importance with respect to the last layer before the classification layer. To the best of our knowledge, they achieved state-of-the-art structured pruning compression results.

Similar to [10], our proposed approach prunes filters employing the L1-norm. However, instead of choosing which filter to prune according to its relative L1-norm within its layer, we prune according to the relative norm with respect to the all layers.

3. STRUCTURED CLASS-BLIND PRUNING

Consider a network with a convolutional (conv) layer and a fully connected (fc) layer. We denote each filter $Filter_i$, where $i \in [1, F]$, and F is the total number of filters in the conv layer. Each filter is a 3D kernel space consisting of channels, where each channel is associated with 2D kernel weights. For the fc layer, we denote W_m, a 1-D feature space containing all the weights connected to certain neuron $Neuron_m$, with $m \in [1, N]$ and N denoting the number of neurons. It should be noted that we do not prune the classification layer.

Each pruning iteration in our approach is structured as presented in Algorithm 1. Moreover, we present important elements in our method as below:

Importance calculation: Although pre-calculation of filters or layers' sensitivity to be pruned is not needed in our method, it can be visualized as part of the pruning criteria. In our algorithm, blindness implies constructing a hidden importance score, which corresponds to the relative normalized L1-norm. For instance, the relevant importance for a certain filter in a conv layer w.r.t. all other filters in all layers is the ratio between the filter's normalized norm and the sum of all filters' normalized norms across the network.

Normalization: As each layer's filters have different number of kernel weights, we normalize filters' L1-norms by dividing each over the number of kernel weights corresponding to the filter (Line 3 and 6 as indicated in Algorithm 1). Alternatively, without compensating for the number of weights, filters with more kernel weights would have higher probabilities of higher L1-norms, hence lower probability to get pruned.

Retraining process: Pruning without further adaption,

Algorithm 1 Pruning procedure

for $i \leftarrow 1$ to F **do** ▷ loop over filters of a conv layer
  $L1\_conv(i) \leftarrow sum(|Filter_i|)$ ▷ calculate L1-norm of all channels' kernel weights of each filter
  $norm\_conv(i) \leftarrow L1\_conv(i)/size(Filter_i)$ ▷ normalize by filter weights count
for $m \leftarrow 1$ to N **do** ▷ loop over Neurons of a fc layer
  $L1\_fc(m) \leftarrow sum(|W_m|)$ ▷ for each Neuron, calculate L1-norm of incoming weights
  $norm\_fc(m) \leftarrow L1\_fc(m)/size(W_m)$ ▷ normalize by number of weights connected
$norms \leftarrow stack(norm\_conv, norm\_fc)$ ▷ stack all normalized norms from all layers
$sorted \leftarrow sort(norms)$ ▷ sort ascendingly
$threshold \leftarrow perc(sorted, p)$ ▷ threshold based on a percentage p of sorted norms values
for $i \leftarrow 1$ to F **do**
  if $norm\_conv_(i) < threshold$ **then**
    $prune(Filter_i)$ ▷ remove filter if its normalized norm is less than threshold
for $m \leftarrow 1$ to N **do**
  if $norm\_fc(m) < threshold$ **then**
    $prune(Neuron_m)$ ▷ remove neuron if its normalized norm is less than threshold

results in accuracy loss. Therefore, in order to regain base performance, it is necessary for the model to be retrained. To this end, we apply an iterative pruning schedule that alternates between pruning and retraining. This is conducted until a maximum compression is reached without losing the base accuracy.

4. EXPERIMENT

In order to assess the efficacy of the proposed method, the performance of our technique is evaluated on a set of different networks: LeNet-5 on MNIST [17], and a version of VGG-16 [10], ResNet-56 and ResNet-110 ([18]) on CIFAR-10 [15]. Also, we conduct an ablation study of our method on LeNet-5. Finally, we analyze some of the resulted pruning patterns.

We use identical training settings for ResNet and VGG-16, as [18], except after pruning where we retrain for 50 epochs and with an initial learning rate of 0.005. Moreover, when a filter is pruned, the corresponding batch-normalization weight and bias applied to that filter are pruned accordingly.

We report compression results on the existing benchmark [10, 14]. As shown in Table 1, we outperform the state-of-the-art compression results reported by [14] on both ResNet-56 and ResNet-110 and on VGG-16 as reported by [10], both with less error compared to the baseline.

In Table 3, while using one-shot pruning, the influence of our method's different components; structured pruning and blindness, is analyzed by removing a component each test, resulting in: **i)** Non-Structured - pruning applied on weights separately. **ii)** Non-Blind - every layer is pruned individually. Then, the effect of the pruning strategy on the method with all its components is analyzed by comparing: **i)** Ours-Oneshot - using one-shot pruning and **ii)** Ours - using iterative pruning.

By comparing the previous versions that use one-shot pruning, our method has fewer parameters; (Non-Structured and Non-Blind). Also, applying pruning iteratively is superior to one-shot pruning (Table 3).

We also show in Table 2 that our method performs better than previously mentioned non-structured weight pruning techniques [6, 7]. Proposed structured class-blind pruning achieves comparable performance as [8], without requiring customized hardware and software to realize the full advantage of the method's compression.

Finally, we explore the patterns of layers pruned and capture the behavior of class-blind pruning on realizing the most sensitive layers. To be able to compare with the other structured pruning methods, we plot the pruning percentage of each layer according to percentage of filters/neurons pruned from the layer. In Fig. 2, a comparison between our method and [6] in terms of the resulted sparsity of the weights and a comparison between our method and [10] in terms of filters/neurons pruning pattern are done. We observe similar pruning pattern as both methods, where the first layer is the least pruned and the third layer is the most pruned for LeNet, while for VGG-16, similar but much smoother pruning pattern can be observed.

In Fig. 3 the effect of pruning in our method vs. the non-structured version of our method is studied by iteratively pruning 94% of the network using each method. It is obvious after retraining that our method has a higher percentage of very weak neurons (mostly pruned) and a higher percentage of strong neurons, while the non-structured version has the highest percentage of neurons having intermediate values.

5. DISCUSSION

Our method exhibits superior performance in terms of compression and error score as shown on the tests on VGG-16 and ResNet (Table 1). Moreover, studying the components of the method in Table (3) shows that every component in the method has a significant impact on increasing compression.

Network	Model	Error%	Params%
VGG-16	Baseline	6.75	
	Li et al.-A [10]	6.60	64.00
	Ours	**6.59**	**86.10**
ResNet-56	Baseline	6.96	
	Li et al.-A [10]	6.90	9.40
	Li et al.-B [10]	6.94	13.70
	NISP-56 [14]	6.99	42.60
	Ours	**6.88**	**47.86**
ResNet-110	Baseline	6.47	
	Li et al.-A [10]	6.45	2.30
	Li et al.-B [10]	6.70	32.40
	NISP-110 [14]	6.65	43.25
	Ours	**6.44**	**53.06**

Table 1: Benchmark results. Error.% denotes the error percentage; Param% denotes the percentage of pruned parameters

Method	Error%	Params%	Eff. Params%
Baseline	0.80		
Han et al. [6]	0.77	92.00	84.00
Srinivas et al. [7]	0.81	95.84	91.68
Han et al. [8]	**0.74**	**97.45**	-
Ours	0.75	97.40	**97.40**

Table 2: Results on LeNet-5. Params.% is the parameters' pruning percentage; Eff.Params%. is the effective parameters' pruning percentage with taking into account the extra indices storage for non-structured pruning as studied by [19]

Method	Error%	Params%	Eff. Params%
Baseline	0.80		
Non-Structured	0.77	93.04	86.08
Non-Blind	0.76	89.80	89.80
Ours-Oneshot	0.80	96.06	96.06
Ours	**0.75**	**97.40**	**97.40**

Table 3: Study of different components of our method. Params.% is the parameters' pruning percentage; Eff.Params%. is the effective parameters' pruning percentage with taking into account the extra indices storage for non-structured pruning as studied by [19].

Also, the pruning pattern study shows that the pruning pattern resulted from our method is similar to [6] on LeNet-5 and similar to [10] using VGG-16, except we do not avoid sensitive layers as [10] and hence our pattern is much smoother. Additionally, through the experiment on ResNet, we noticed that layers that are sensitive to pruning include those that lie at residual blocks close to the layers where the number of feature maps changes (less pruned), which is similar to the findings of [10]. However, in contrast to the same work, we find that earlier layers are less pruned than the deeper layers. We suggest that our evaluation of layer's sensitivity to pruning is more accurate because of our global importance score for

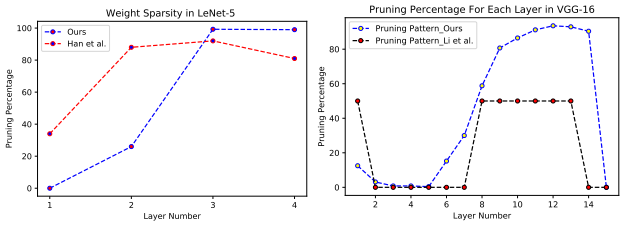

Fig. 2: Left: Weights sparsity for LeNet-5 after pruning vs Han et al. [6]. Right: Filters/Neurons sparsity for VGG-16 vs Li et al. [10].

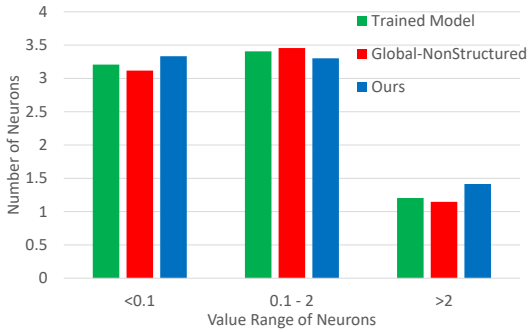

Fig. 3: Effect of different methods on the neurons' distribution; the x-axis shows the magnitude value range of neurons, while the y-axis the number of neuron in log-scale.

each filter. Finally, it can be deduced from the results of Table 3 that although our method prunes a lot of weak neurons, it encourages more activated neurons when compared to the unstructured counterpart. By pruning neurons, the unpruned neurons becomes more active to compensate for their pruned counterparts.

6. CONCLUSION

We presented a novel structured pruning method to compress neural networks without losing accuracy. By pruning layers simultaneously instead of looking at each layer individually, our method combines all filters and output features of all layers and prunes them according to a global threshold. We have surpassed state-of-the-art compression results in our results that included VGG-16, ResNet-56 and ResNet-110 on CIFAR-10. Also, we showed that only 11K parameters are sufficient to exceed the baseline performance on LeNet-5, compressing more than 97%. Additionally, thanks to the hidden importance score, our method succeeded to extract similar pruning patterns to other methods which used heuristics. To realize the advantages of our method, no dedicated hardware or library is needed. For future work, we are dedicated to proving the applicability of our method for further architectures and datasets. Hence, we plan to experiment using multiple models such as ResNet on ImageNet and/or other comparable architectures. Additionally, we plan to further investigate structured pruning and coupling it with an information theoretic view.

7. REFERENCES

[1] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas, "Predicting parameters in deep learning," *CoRR*, vol. abs/1306.0543, 2013.

[2] Yann Le Cun, John S. Denker, and Sara A. Solla, "Advances in neural information processing systems 2," chapter Optimal Brain Damage, pp. 598–605. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[3] Babak Hassibi and David G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, San Francisco, CA, USA, 1993, pp. 164–171, Morgan Kaufmann Publishers Inc.

[4] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *CoRR*, vol. abs/1511.06530, 2015.

[5] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *CoRR*, vol. abs/1412.6553, 2014.

[6] Song Han, Jeff Pool, John Tran, and William J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015.

[7] Suraj Srinivas, Akshayvarun Subramanya, and R. Venkatesh Babu, "Training sparse neural networks," *CoRR*, vol. abs/1611.06694, 2016.

[8] Song Han, Huizi Mao, and William J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[9] Abigail See, Minh-Thang Luong, and Christopher D. Manning, "Compression of neural machine translation models via pruning," *CoRR*, vol. abs/1606.09274, 2016.

[10] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2017.

[11] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li, "Learning structured sparsity in deep neural networks," *CoRR*, vol. abs/1608.03665, 2016.

[12] Suraj Srinivas and R. Venkatesh Babu, "Data-free parameter pruning for deep neural networks," *CoRR*, vol. abs/1507.06149, 2015.

[13] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *CoRR*, vol. abs/1607.03250, 2016.

[14] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis, "NISP: pruning networks using neuron importance score propagation," *CoRR*, vol. abs/1711.05908, 2018.

[15] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[16] Zehao Huang and Naiyan Wang, "Data-driven sparse structure selection for deep neural networks," *CoRR*, vol. abs/1707.01213, 2017.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[19] Maxwell D. Collins and Pushmeet Kohli, "Memory bounded deep convolutional networks," *CoRR*, vol. abs/1412.1442, 2014.