Pruning at a Glance: A Structured Class-Blind Pruning Technique for Model Compression

Anonymous Author(s)
Affiliation
Address
email

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 constraint 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 redundancy in the parameters. The resulting network is non-sparse, however, much more compact and requires no special infrastructure for its deployment. We prove the viability of our method by achieving 97.4%, 47.8% and 53% compression of LeNet-5, ResNet-56 and ResNet-110 respectively, exceeding state-of-the-art compression results reported on ResNet without losing any performance compared to the baseline. Our approach does not only exhibit good performance, but is also easy to implement on many architectures.

1 Introduction

While deep learning models have become the method of choice for a multitude of applications, their training requires a large number of parameters and extensive computational costs (energy, memory footprint, inference time). This limits their deployment on storage and battery constraint devices, such as mobile phones and embedded systems. To compress deep learning models without loss in accuracy, previous work proposed pruning weights by optimizing network’s complexity using second order derivative information \[2, 5\]. While second order derivative introduces a high computational overhead, \[8, 10\] explored low rank approximations to reduce the size of the weight tensors. Another line of work \[4, 15\], proposed to prune individual layer weights with the lowest absolute value (nonstructural sparsification of layer weights). \[3\] followed the same strategy while incorporating quantization and Huffman coding to further boost compression. While the aforementioned methods considered every layer independently, \[13\] proposed to prune 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.

Noteworthy, all approaches that prune weights non-structurally, generally result in high sparsity models that require dedicated hardware and software. Structured pruning alleviates this by removing whole filters or neurons, producing a non-sparse compressed model. In this regard, \[12\] proposed channel-wise pruning according to the L1-norm of the corresponding filter. \[16\] learned a compact model based on learning structured sparsity of different parameters. A data-free algorithm was implemented to remove redundant neurons iteratively on fully connected layers in \[14\]. In \[7\], connections leading to weak activations were pruned. Finally, \[17\] pruned neurons by measuring their importance with respect to the penultimate layer. Generally, in structured pruning, each layer is pruned separately, which requires calculation of layer importance before training.

This work follows the notion of structured pruning, with the key difference of considering all parameters in the network in a class-blind manner instead of each layer independently. Such a strategy makes the layer importance calculation unnecessary, as stacking all filters and output features in the network provides a global importance insight.

The contribution of this paper includes: i) Proposing a structured class-blind pruning technique to compress the network by structurally removing whole filters and neurons, which results in a compact non-sparse network with the same baseline performance. ii) Introducing a global importance parameter as a part of the pruning technique, to devise the pruning percentage of each layer. iii) Exceeding state-of-the-art results reported on ResNet-56 and ResNet-110 on the CIFAR10 dataset [9].

2 Structured class-blind pruning

Consider a network with a convolutional (conv.) layer and a fully connected (fc.) layer. We denote all conv. channels $C_i$ belonging to a certain filter $i$, where $i \in [1, F]$, and $F$ is the total number of filters in the conv. layer. Filter channels $C_i$ is a 3D filter, each channel containing 2D kernel weights. For the fc layer, we denote $W_m$, a 1-D feature space containing all the weights connected to certain output feature $m$, where $m \in [1, out]$ where $out$ is the number of output features. We don’t prune the classification layer.

Each pruning iteration in our algorithm is structured as follows:

Algorithm 1 Pruning procedure

1: for $i \leq F$; $i = i + 1$ do \hspace{1cm} ▷ loop over filters of a conv layer
2: \hspace{1cm} $L_{1\text{conv}}(i) \leftarrow \text{sum}(|C_i|)$ \hspace{1cm} ▷ calculate L1-norm of all channels’ kernel weights
3: \hspace{1cm} $\text{norm}_\text{conv}(i) \leftarrow L_{1\text{conv}}(i)/\text{nonzero}(C_i)$ \hspace{1cm} ▷ normalize by non-zero weights count
4: for $m \leq out$; $m = m + 1$ do \hspace{1cm} ▷ loop over output features of a fc layer
5: \hspace{2cm} $L_{1\text{fc}}(m) \leftarrow \text{sum}(|W_m|)$ \hspace{1cm} ▷ for output features, calculate L1 of incoming weights
6: \hspace{2cm} $\text{norm}_\text{fc}(m) \leftarrow L_{1\text{fc}}(m)/\text{nonzero}(W_m)$ \hspace{1cm} ▷ normalize by non-zero weights connected
7: \hspace{1cm} norms $\leftarrow \text{stack}($norm\_conv$,$norm\_fc$)$ \hspace{1cm} ▷ stack all normalized norms from all layers
8: \hspace{1cm} sorted $\leftarrow \text{sort}(\text{norms})$ \hspace{1cm} ▷ sort ascendingly
9: \hspace{1cm} threshold $\leftarrow \text{perc}(\text{sorted}, p)$ \hspace{1cm} ▷ threshold based on a percentage $p$ of sorted norms values
10: for $i \leq F$; $i = i + 1$ do \hspace{1cm} ▷ same for fc, remove neuron
11: \hspace{2cm} if norm\_conv($i$) $<$ threshold then
12: \hspace{3cm} prune(Filter$_i$) \hspace{1cm} ▷ remove filter if its normalized norm is less than threshold
13: \hspace{2cm} for $m \leq out$; $m = m + 1$ do
14: \hspace{3cm} if norm\_fc($m$) $<$ threshold then
15: \hspace{4cm} prune(Neuron$_m$) \hspace{1cm} ▷ same for fc, remove neuron

Importance calculation. The importance and sensitivity calculation of each layer is part of the pruning method. Pruning blindly implies constructing a hidden importance score, which corresponds to the normalized L1-norm. For example, the relevant importance for a certain filter of a conv layer w.r.t. all other layers is the ratio between its normalized norm and the sum of all normalized norms across the network.

Normalization. As we don’t copy the non-pruned parameters to a new compact model between pruning iterations, but only once after the maximum pruning percentage is reached, normalization becomes very important. As we sum over the weights, normalization over the nonzero elements is needed, otherwise, the most pruned layer will contain the least norms and will have an unfair advantage to get much more pruned than the non-pruned layers.

Retraining process. Pruning without further adaption, results in performance 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.
ResNet-56
Baseline 6.96
Li et al. [12]-A 6.90 9.40
Li et al. [12]-B 6.94 13.70
NISP [17] 6.99 42.60
Ours 6.88 47.86

ResNet-110
Baseline 6.47
Li et al. [12]-A 6.45 2.30
Li et al. [12]-B 6.70 32.40
NISP [17] 6.65 43.25
Ours 6.44 53.06

Table 1: Compression Benchmark Results. Error% percentage for different percentage of parameters pruned (Par.%)

Table 2: Results on LeNet-5. "Par" is the number of parameters after pruning; "E.Par" is the effective number of parameters needed to store parameters and indices for non-structured pruning as studied by [1].

3 Experiment

In order to assess the efficacy of the proposed method, we evaluate the performance of our technique on a set of different networks: first, LeNet-5 on MNIST [11]; second, ResNet-56 and ResNet-110 (6]) on CIFAR-10 [9]. We use identical training settings as [6], after pruning we retrain with learning rate of 0.05.

For ResNet, when a filter is pruned, the corresponding batch-normalization weight and bias applied on that filter are pruned accordingly. After all pruning iterations are finished, a new model with the remaining number of parameters is created.

We report compression results on the existing benchmark [12, 17]. As shown in Table 1, we outperform the state-of-the-art compression results reported by [17] on both ResNet-56 and ResNet-110 with a lower classification error even compared to the baseline.

In Table 2, 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) "BlindOneshot" - using one-shot pruning and ii) "BlindIterative" - using iterative pruning.

By comparing the previous versions that are using one-shot pruning, our method has less number of parameters compared to the other versions; ("Non-Structured" and "Non-Blind").

Finally, applying pruning iteratively is superior to one-shot pruning. We also show that our method performs better than previously mentioned non-structured weight pruning techniques [4, 15]. Proposed structured class-blind pruning offers comparable performance as [3], without requiring dedicated hardware and software to realize compression.

4 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 in our results that included ResNet-56 and ResNet-110 on CIFAR-10, compressing more than 47% and 53% respectively. Also, we showed that only 11K parameters are sufficient to exceed the baseline performance on LeNet-5, compressing more than 97%. To realize the advantages of our method, no dedicated hardware or libraries are needed. It is worth to say that due to removing whole filters and neurons, the pruning percentage reflects the effective model compression percentage. For the future work, we are dedicated to proving the applicability of our method on several different architectures and datasets. Hence, we plan to experiment on VGG-16, ResNet on ImageNet and/or other comparable architectures.
References


