PruNet: Class-Blind Pruning Method For Deep Neural Networks

PruNet: Class-Blind Pruning Method for Deep Neural Networks

***Abstract*— Deep Neural Networks are very memory and computationally intensive. They are unfeasible to deploy in real time or mobile applications, where power and memory are constrained. Introducing sparsity in the network is a way to reduce those requirements. We propose a methodology that applies iteratively a magnitude-based pruning, called Class-Blind, to compress a Neural Network and obtain a sparse model. It can be applied to any kind of DNN, from a shallow one to a Deep Convolutional Network. We demonstrate that retraining after pruning is essential to restore accuracy. With our methodology we are able to reduce by around two orders of magnitude the model size, without affecting the accuracy. Our method requires many iterations of pruning and retraining, but can achieve up to 190X Memory Saving Ratio (LeNet on MNIST) with respect to the baseline model. Similar results are obtained also for more complex tasks (e.g., 91X for VGG-16 on CIFAR100). If we combine this work with an efficient coding for sparse networks, like Compressed Sparse Column (CSC) or Compressed Sparse Row (CSR), we can obtain a reduced memory footprint. Our methodology can be complemented by other compression techniques, like weight sharing, quantization or fixed-point conversion, that allows to further reduce memory and computations.**

I. Introduction

Recent developments have allowed to train very Deep Neural Networks (DNN) and achieve high level accuracy in computer vision task [15], [19], [28], [30]. Even though they can beat humans in image classification task [35], they require a huge amount of storage and memory accesses. In real applications, DNN inference is not easy to deploy because it usually exceeds the available resources on mobile platforms and does not meet real-time constraints, because of GPU limited bandwidth. For this reason, many researchers in deep learning community have focused on DNN compression and efficiency improvements. There are numerous variants of compression techniques, that include different pruning methods and reduced precision representation.

Our contribution. We propose a magnitude-based pruning method, called Class-Blind, as described by See et al. [27]. Our methodology requires many iterations of pruning and retraining, but we are able to achieve a significant compression with respect to the baseline model. As a side effect, pruning has a regularizing property, because during the first stages of pruning, the sparse networks outperforms the original one in terms of accuracy. The process of iteratively pruning and retraining is effective, because it allows to progressively adjust the parameters and maintain a good level of accuracy. Our methodology has been tested both on a simple task (digit recognition on MNIST dataset, with quite simple networks, like LeNet-5 and LeNet-300-100 [20]) and on more complex configurations (VGG-16 net [28] on CIFAR-10, VGG-16 [28], AlexNet [19] and GoogleNet [30] on CIFAR-100). If we combine this work with an efficient coding for sparse networks, like CSC or CSR, we can achieve similar benefits as reported in [12]. We do not discuss such coding methods in our work.

An overview of our work is shown in Figure 1. The rest of the paper is organized as follows: in Section II we recall and summarize the work by other researchers on the same field subject; Section III explains what are the motivations of our research and the reasons why we are choosing to work on this topic; in Section IV we present our methodology, that represents the fundamental aspect of our contribution; Section V describes an example, which provides us useful intuitions; the experiment section (Section VI) reports the most significant results; Section VII concludes the article.

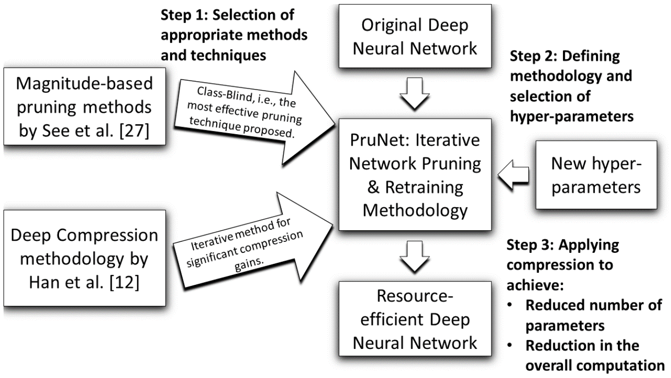

Step 1: Selection of appropriate methods and techniques
Original Deep Neural Network
Step 2: Defining methodology and selection of hyper-parameters
Magnitude-based pruning methods by See et al. [27]
Class-Blind, i.e., the most effective pruning technique proposed.
PruNet: Iterative Network Pruning & Retraining Methodology
New hyper-parameters
Deep Compression methodology by Han et al. [12]
Iterative method for significant compression gains.
Step 3: Applying compression to achieve:
• Reduced number of parameters
• Reduction in the overall computation
Resource-efficient Deep Neural Network

Fig. 1: General overview of our contribution.

II. Related work

Making DNN inference manageable on mobile applications has become a very attractive topic and a large variety of ideas have been proposed. Vanhoucke et al. [32] propose a fixed-point implementation, achieving a considerable speedup. Other works include low-rank approximations for weight matrices (Denton et al. [7], Jaderberg et al. [18]), weight sharing (Chen et al. [3], Han et al. [12]), reduced precision (Lin et al. [23], Courbariaux et al. [6], Gupta et al. [11], Lin et al. Venkatesh et al. [33]) or even binary weights (Lin et al. [22], Courbariaux et al. [5], Courbariaux and Bengio [4]). Hinton et al. [17] proposed the 'distillation' technique, that leads to the teacher-student approach. These techniques are orthogonal to connection pruning, as explained by Han et al. [12], where a good compression rate have been obtained. There exist several varieties of pruning methods. Structured or channel pruning (Anwar et al. [1], Li et al. [21], Wen et al.

[34], He et al. [16]) allows to achieve a significant inference speedup. Recently, Changpinyo et al. [2] demonstrated also other benefits of channel-wise sparsity, but the accuracy drop limits its deployment in practice. Based on Optimal Brain Surgeon (Hassibi et al. [14]), Dong et al. proposed a layer-wise pruning approach [8] that leads to a significant compression ratio, but it requires the computation of reverse Hessian matrix. Molchanov et al. [25] introduced a new way to introduce sparsity in DNNs, called "variational dropout" The original dropout technique, introduced by Srivastava et al. [29] is modified in a way that some connections are permanently removed from the network. Inspired by this work, Louizos et al. [24] applied successfully variational dropout, obtaining an efficient coding scheme. Recently, Federici et al. [9] combined this work with Soft Weight Sharing (Ullrich et al. [31]), obtaining state-of-the-art values of compression rates.

Another widely used pruning procedure is the so called magnitude-based method: Han et al. [13] obtained a good compression ratio without sacrificing the accuracy and demonstrated in [12] that pruning can be successfully combined with other compression techniques, like weight sharing, quantization and Huffman coding. Zhu and Gupta [36] showed that, having the same memory footprint, a large sparse model (obtained by pruning and retraining) outperforms the small dense one in terms of accuracy. The main drawback of this approach is the increase of time and computation during training. In order to overcome this effect, Narang et al. [26] proposed a way to prune the network during training, with the purpose of limiting the overall computational effort during the training phase. However, this method leads to an accuracy loss and the final sparsity which can be obtained is far from the ones achieved with other approaches. Another important result has been obtained by Guo et al. [10]: they were able to significantly compress the model without loss in accuracy. Their approach is more complex than the other ones, because they exploit two operations: pruning and splicing. The splicing operation is effective because it allows to restore some important connections that were erroneously pruned in the previous step. A deeper analysis on magnitude-based methods was made by See et al. [27]. They showed that there are different pruning schemes leading to different results, while in general, despite their simplicity, magnitude-based pruning methods are effective also on Neural Machine Translation applications. A more detailed explanation of these methods is reported in Section III-B.

III. Motivations

DNN pruning has become a hot topic nowadays because of its efficiency and it is becoming widely used in everyday applications for improving the performance efficiency of inference in embedded platforms. Our contribution is to propose a simple and efficient method to reduce the memory requirements during inference in mobile applications, where the memory occupied by the model and the power consumption are an issue, as well as real-time constraints. For this reason, the overhead introduced by an iterative approach of retraining, as explained in the work of Han et al. [13], has been considered less relevant. Iteratively pruning and retraining the network implies a much more intensive computational effort with respect to the standard training. However, this process can be performed in large data-centers, provided with powerful GPUs and optimized accelerators for Deep Learning, in such a way that time and power consumed are still acceptable.

Since the number of parameters in a network is directly proportional to the number of computations at the inference stage, an effective and commonly used parameter to evaluate pruning methods is the Compression Ratio (CR: the ratio between the number of parameters in the original model and in the sparse model, respectively), as described in Equation (1).

$$CR = \frac{\#\ parameters_{original\ model}}{\#\ parameters_{sparse\ model}} \tag{1}$$

Consequently, once we have obtained a sparse structure, we can use the same storing methods proposed by Han et al. [12], like Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC). Moreover, pruning represents only the first step of DNN compression: works made by Han et al. [12] and Federici et al. [9] showed that other compression techniques are orthogonal to pruning and can be applied in further stages.

Our contribution focuses on an efficient way to apply pruning, based on Class-Blind method, that outperforms the results obtained by Han et al. [13], without a relevant accuracy loss. A detailed description of the Class-Blind method, compared to other magnitude-based methods, is reported in Section III-B.

A. *Efficiency of magnitude-based pruning*

Among all DNN compression methods analyzed in Section II, we choose to use a magnitude-based pruning method. Despite the effectiveness in terms of CR of methods based on Variational Dropout by Molchanov et al. [25] and Bayesian Compression by Louizos et al. [24], the complexity introduced by considering weights as a distribution makes these approaches difficult to deploy in everyday applications.

On the contrary, the results obtained by Han et al. [12], [13] on AlexNet [19] and VGG-16 [28] trained on ImageNet dataset look very attractive for what concerns accuracy. However, some details of the pruning process (threshold value, number of retraining iterations) performed by Han et al. have not been revealed in their publications. This is another important motivation for our work: trying to obtain result comparable (or better) than Han et al. [13], using a clearer methodology. Moreover, See et al. [27] showed as a side effect that magnitude-based pruning introduces a regularizing effect. That is the reason why low pruned models outperform their respective baseline models. As a consequence, our idea is to further prune and retrain the network (in an iterative way) beyond the level where the accuracy increases, in order to maximize the compression ratio, while keeping the accuracy loss acceptable.

B. *Efficiency of Class-Blind method*

See et al. [27] gave a great contribution to our work, because they analyzed three different magnitude-based pruning schemes. Despite they applied those schemes on Neural Machine Translation application only, their concepts are the key for our work, because they can be applied also to other applications, like image classification. They based the differences on the concept of ”class”, which means a group of neurons performing similar operations. We revisited that concept for Feed-Forward Neural Networks and we assume that each class can be seen as a weighted layer of the network. In the work by See et al. [27], the concept of class was a little bit different, because it is related to Recurrent Neural Networks for Neural Machine Translation applications. In the followings, we refer to class and layer as synonyms.

The three pruning schemes that were presented by See et al. [27] are the following:

1) Class-Distribution (CD): select threshold T, common for every layer, and compute the standard deviation σ of each layer. Then, for each layer, prune parameters below σT. This method is used by Han et al. [13]. The threshold T is equal across each layer, but the product σT can vary. *Finding the optimal value of T can be tricky.*
2) Class-Uniform (CU): select a certain percentage x and, for each layer, prune the smallest x% parameters. Easier to implement than CD. In this way pruning is equally distributed among all the layers. However, *the accuracy drop is not minimized.*
3) **Class-Blind (CB):** select a certain percentage x and prune the smallest x% parameters, regardless of which layer they belong to. In this way some layers are pruned more than others. This method is adopted by See et al. [27], showing that *it is the most efficient among these three.*

Based on the results obtained by See et al. [27], we decided to use the Class-Blind method, not only because of its simplicity, but also because it outperforms the other schemes.

IV. Methodology

We now propose a general methodology, called ”PruNet”, that can be applied to sparsify a neural network. We do not describe the baseline training procedure, since it is out of the scope of our work. Then, we start from a pre-trained model of the network, i.e., already trained over the same dataset that will be applied for the pruning phase.

Figure 2 summarizes our methodology in a schematic way. First of all, we set the hyper-parameters. Some intuitions about how to choose these values are explained in Section V-A. We reduce the initial learning rate with respect to the one used for the baseline training (e.g., if the baseline initial learning rate is 0.01, the initial one in each retraining phase could be 0.003). If the network contains some dropout layers, the dropout rate must be changed according to the rule proposed by Han et al. [13]:

$$D_r = D_o\sqrt{\frac{C_{ir}}{C_{io}}} \tag{2}$$

where D_r is the dropout rate during retraining, D_o the original dropout of the baseline training, C_{io} and C_{ir} are the number of connections of layer i for the original network and the sparse one, respectively. The number of training epochs is scaled by a factor 2X. While Han et al. [13] retrain convolutional layers and fully-connected layers separately, we perform retraining on the whole network at the same time.

Another group of hyper-parameters to set is composed by the ones introduced for our purpose: pruning percentage, maximum acceptable accuracy loss and pruning iterations. We have to fix only the first two values, since the number of pruning iterations depends on the other two. We choose to set the pruning percentage to a mid-range value, according to the trade-off reasons explained in Section V-A. In this way, at each pruning iteration, the number of weights is reduced. The choice of the maximum accuracy loss is delicate, because it depends on the application. However, as shown in Figure 3, the curves become very steep toward high MSR. As a consequence, the most appropriate value lies around the focal point of an exponential fitting curve (black line of Figure 3).

Then, the iterative process starts. For each iteration we perform the following operations:

1) Sort the weights of the whole network in ascending absolute value order and mark the X% lowest ones according to the pruning percentage.
2) Apply a mask for each layer: a 0 corresponds to a pruned weight, while a 1 stands for a not pruned weight.
3) Retrain the (sparse) network, then compute the accuracy loss.

Finally, if the accuracy loss is still below the threshold, loop back to the next iteration, otherwise the process is terminated. Note that the final model is not the one at the last iteration, but the previous one. Since we exit from the loop when the accuracy is not acceptable, we have to restore the model at the previous step, so as to comply with the constraint.

V. Case study of a simple task

This section will give some useful intuitions that have been essential to formulate the more generic methodology described in Section IV, as well as a comparison between our approach and the Class-Uniform method. For this purpose, we train and prune LeNet-5 (developed by LeCun et al. [20]) on MNIST dataset. The network is quite shallow, because it has only two convolutional layers and two fully-connected layers, but it has been referenced as example throughout the literature very frequently. MNIST dataset is a collection of handwritten digits, grouped in 10 categories. It contains 60000 training images and 10000 test images.

We first trained the dense network, like described by LeCun et al. [20], obtaining an accuracy of 99.13 %. Hardware and software setup is reported in Section VI-A. Afterwards, several pruning experiments, varying some parameters, have been performed.

A. Intuitions

The pruning process is delicate, because removing connections from the network implies an accuracy loss. However, as successfully explained by Han et al. [13], the retraining

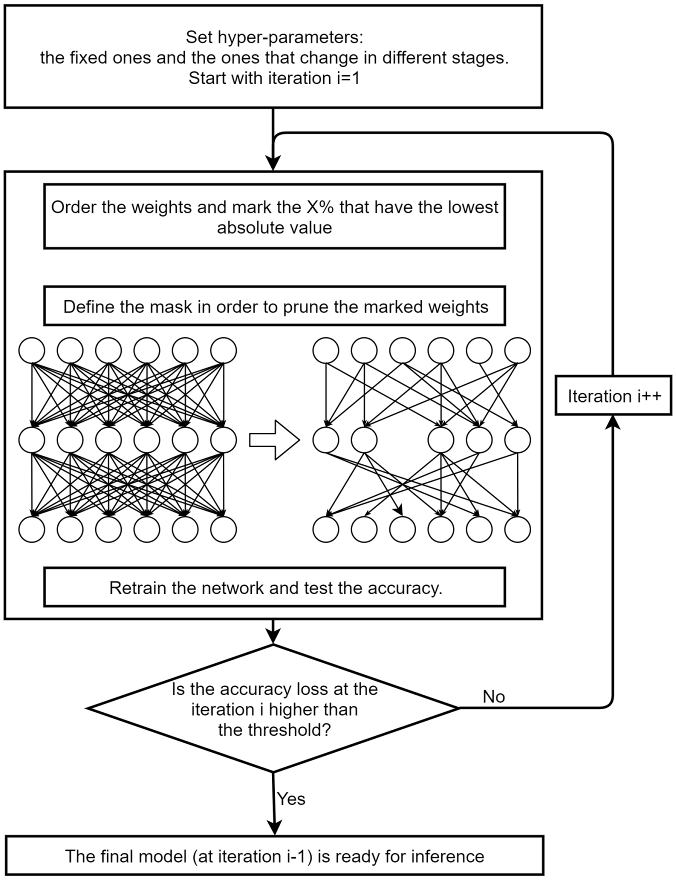

Set hyper-parameters:
the fixed ones and the ones that change in different stages.
Start with iteration i=1
Order the weights and mark the X% that have the lowest absolute value
Define the mask in order to prune the marked weights
Iteration i++
Retrain the network and test the accuracy.
Is the accuracy loss at the iteration i higher than the threshold?
No
Yes
The final model (at iteration i-1) is ready for inference

Fig. 2: Summary scheme of methodology.

process is fundamental to recover from the errors. They suggest an adjustment for the dropout rate, since the sparsity introduces itself another form of regularization. Indeed, the dropout rate should be changed according to that.

Another important hyper-parameter to consider is the learning rate. Usually, it is not constant during the whole training process, but it decreases after a certain amount of training epochs. At the first training step, all the weights and biases of the network are initialized as their default value. The pruning process, however, does not change the remaining parameter values. That is equivalent to retrain the sparse network from a pretrained model. As a consequence, the starting learning rate during retraining can be lower that the respective value set for the first train. As a demonstration of this intuition, See et al. [27] propose to divide by 2 the starting learning rate of the original model. Selecting that value is not an easy task and cannot be generalized for every setup, because it depends on the network and the dataset. Moreover, at each retraining iteration it is possible to further tune this value. Since the retraining process has some similarities to transfer learning (i.e., the network is not trained from scratch), the total number of epochs during retraining can be reduced with respect to the ones adopted for the baseline training.

Using Class-Blind pruning method, a new hyper-parameter has been introduced: the pruning percentage. It represents how many weights are going to be pruned away permanently from the network. Han et al. [13] demonstrated that iterative pruning (and retraining) is more efficient than doing the same procedure in a single iteration. In order to avoid possible terminology misunderstanding, we refer to pruning percentage as *the amount of parameters that are going to be pruned at each iteration* (e.g., after 50% pruning at the first iteration, if we prune again with a pruning percetage equal to 50%, we obtain 25% parameters with respect to the original model). It is worth noting that pruning percentage should not be confused with the Compression Ratio (Equation (1)). If we neglect the overhead due to a sparse memory coding scheme, the CR can be equaled to the Memory Saving Ratio (MSR): the ratio between the amount of memory occupied by the original model and the amount required by the sparse model (Equation (3)).

$$MSR = \frac{mem_{original\ model}}{mem_{sparse\ model}} \tag{3}$$

Choosing this value have a huge impact on the success of our procedure, since it defines the ability of the network to recover the accuracy from the error introduced by pruning. It has implications also on retraining time / retraining epochs, while their respective relations are quite complex. Empirically, we can assume that:

- A low value of pruning percentage allows a lower retraining effort to restore the accuracy, but requires high expense in training time due to more iterations.
- A faster approach can be obtained using an high value of pruning percentage, but it implies higher damages to the network, that cannot always guarantee an optimal recovery at the retraining stage.

According to the aforementioned considerations, a middle-range value looks more attractive. At each iteration, the value could either remain constant or change. While it is simpler to keep the pruning percentage constant at every iteration, a possible design could be starting with high value in the first stages and progressively decreasing it.

Since our final goal is to maximize the compression ratio, while maintaining an acceptable accuracy, we define another parameter, the Accuracy Loss (AL, Equation (4)), which is the relative difference of accuracy between the sparse and the original model.

$$AL = \frac{Acc_{sparse\ model} - Acc_{original\ model}}{Acc_{original\ model}} \tag{4}$$

Considering acceptable, for example, an Accuracy Loss of 0.1%, we are able to set the number of pruning (and retraining) iterations such that the accuracy loss stays below that threshold. Thanks to the regularizing effect of pruning, explained by See et al. [27], for relatively low values of MSR, the Accuracy Loss reaches negative values, while at a certain point it grows exponentially. The behavior is shown explicitly in Figure 3 for our current setup (LeNet-5 on MNIST). The graph looks very promising, because it outperforms by around a factor 10 the results obtained by Han et al. [12] on a similar approach. This has been obtained by running simulations, using Class-Blind method, with different (constant) pruning percentages, from 40% to 90%, and setting the threshold of AL equal to 0.1%. Overall, the figure shows that for low MSR, the sparse network outperforms the baseline, due to the regularizing effect introduced by pruning. The accuracy loss remains quite stable, until a certain point (around a MSR of 100X) where it starts growing exponentially. In the first part the pruning percentage is not strictly relevant, since the accuracy loss lies in the range [-0.2,0]. For higher MSR, however, the difference is more significant: while for high pruning percentages the Accuracy Loss grows very quickly, a lower pruning percentage allows to better recover from that error, thus pushing the MSR to the maximum.

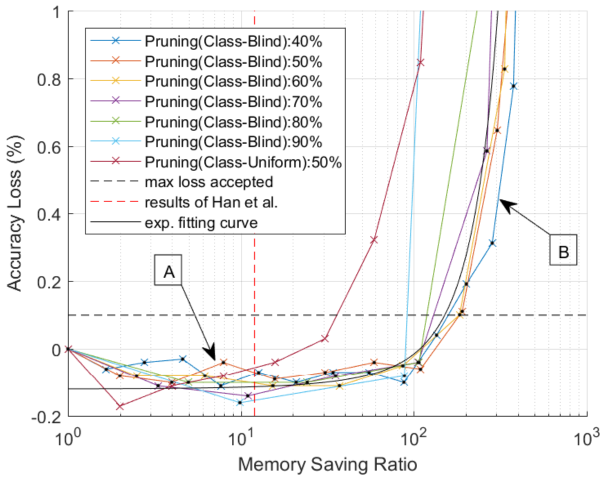


Fig. 3: Accuracy Loss with respect to Memory Saving Ratio in LeNet-5. Box A: accuracy is slightly improved because pruning+retraining has a regularizing effect. Box B: accuracy drops significantly because the number of parameters becomes too low.

Figure 3, however, shows only the general behavior of the network, while many aspects cannot be seen directly. Introducing a magnitude-based sparsity implies removing weights that have a low absolute value. For this reason, showing the weight distribution can be useful to understand the pruning process and what actually happens inside the neural network. Figures 5(a) to 5(l) show the weight distribution of our reference network, LeNet-5 (architecture reported in Figure 4), at each pruning stage. Every figure shows a separate distribution for each layer of the network, where L1 and L2 stand for the first two convolutional layers, and L3 and L4 stand for the third and fourth (fully-connected) layers. Each graph has been obtained by subdividing the weights in intervals, counting the weight values belonging to each interval and plotting histograms. Each red curve represents the fitting normal distribution.

In the first figures, until the seventh pruning and retraining stage (Figure 5(h)), the total number of parameters is progressively decreasing, but at the same time, for every layer, there is a clear peak centered at 0. Thus the large majority of weights have small absolute value. Starting from the eighth stage (Figure 5(i)), the distributions are becoming smoother. Finally, in the last stages, the pruning effect introduces a hole around zero. This result means that the network is approaching its limit, because the retraining procedure is not able to recover completely from the error generated by pruning. In other words, in the last stages, few weights are remaining. Then, also the weights with middle range value are pruned. As a drawback, the accuracy is getting lower.

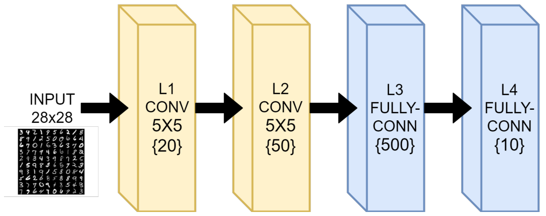


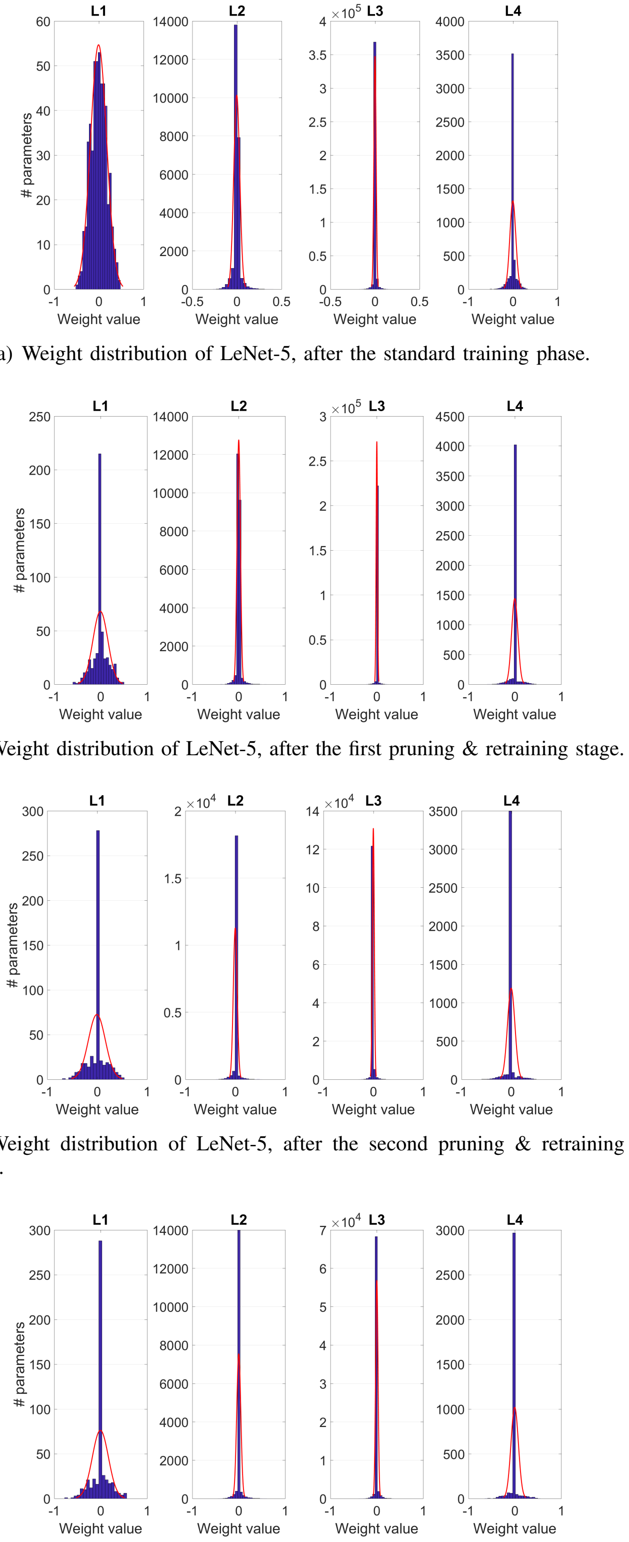
Fig. 4: LeNet-5 architecture for our experiments.

B. *Class-Uniform vs. Class-Blind*

Similar experiments, setting the pruning percentage to 50%, have been carried on using Class-Uniform method. This approach is less flexible, because it does not allow to have sparsity unbalance across layers. Figure 3 shows also a comparison between the two methods at the same conditions (pruning percentage of 50%): it is clear that Class-Blind method outperforms the Class-Uniform one, since in the latter configuration, the Accuracy Loss start growing rapidly after the 5^{th} iteration (MSR = 31), while the former one allows to reach the 8^{th} one (MSR = 191). The reason of that behavior can be explained again looking at the weight distribution graphs. Looking at Figure 6, that represents the weight distribution across each layer of LeNet-5 after the fifth iteration, it is evident that sparsity penalizes very heavily the layer 1, that contains few parameters. On the contrary, layer 3, that contains the majority of weights, could have been sparsified more, as with Class-Blind method in Figure 5(f).

VI. Experiments

We apply our methodology not only to LeNet-5 on MNIST dataset, which corresponds to the example provided in Section V, but also on other tasks: LeNet-300-100 on MNIST (Section VI-B), VGG-16 on CIFAR-10 and CIFAR-100 (Section VI-C), two other networks, AlexNet and GoogleNet, on CIFAR-100 (Section VI-D). A summary of results is reported in Table I, which shows quantitatively the Memory Saving Ratio and the Accuracy Loss due to our proposed methodology. While a shallow Convolutional Neural Network like LeNet-5 can be pruned to achieve the best result in terms of MSR, also deeper networks (VGG-16 and AlexNet) are able to achieve competitive MSR, compared to the other state-of-the-art methods reported in Section II. The complete setup, as well as the simulation environment, is described in Section VI-A.



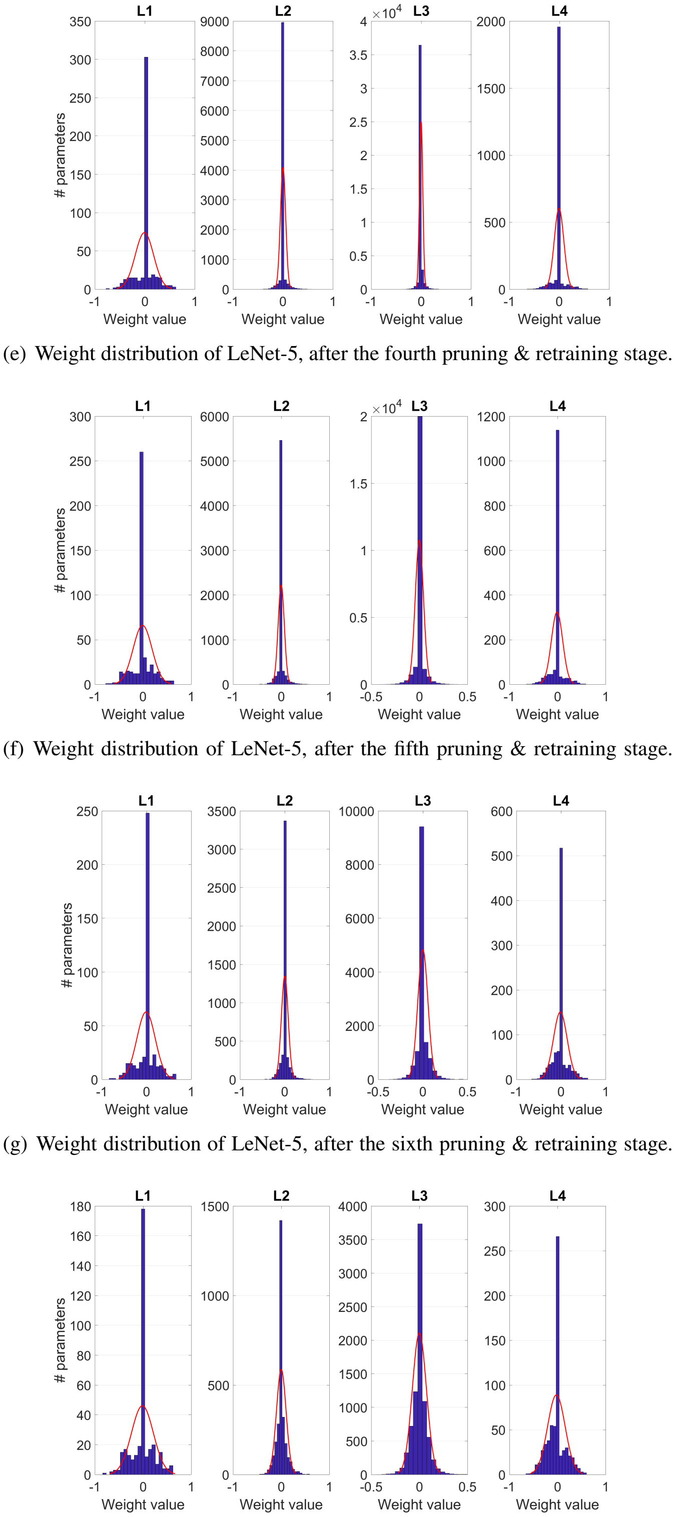
(a) Weight distribution of LeNet-5, after the standard training phase.

(b) Weight distribution of LeNet-5, after the first pruning & retraining stage.

(c) Weight distribution of LeNet-5, after the second pruning & retraining stage.

(d) Weight distribution of LeNet-5, after the third pruning & retraining stage.

Fig. 5: Weight distribution of LeNet-5, across different pruning & retraining stages.



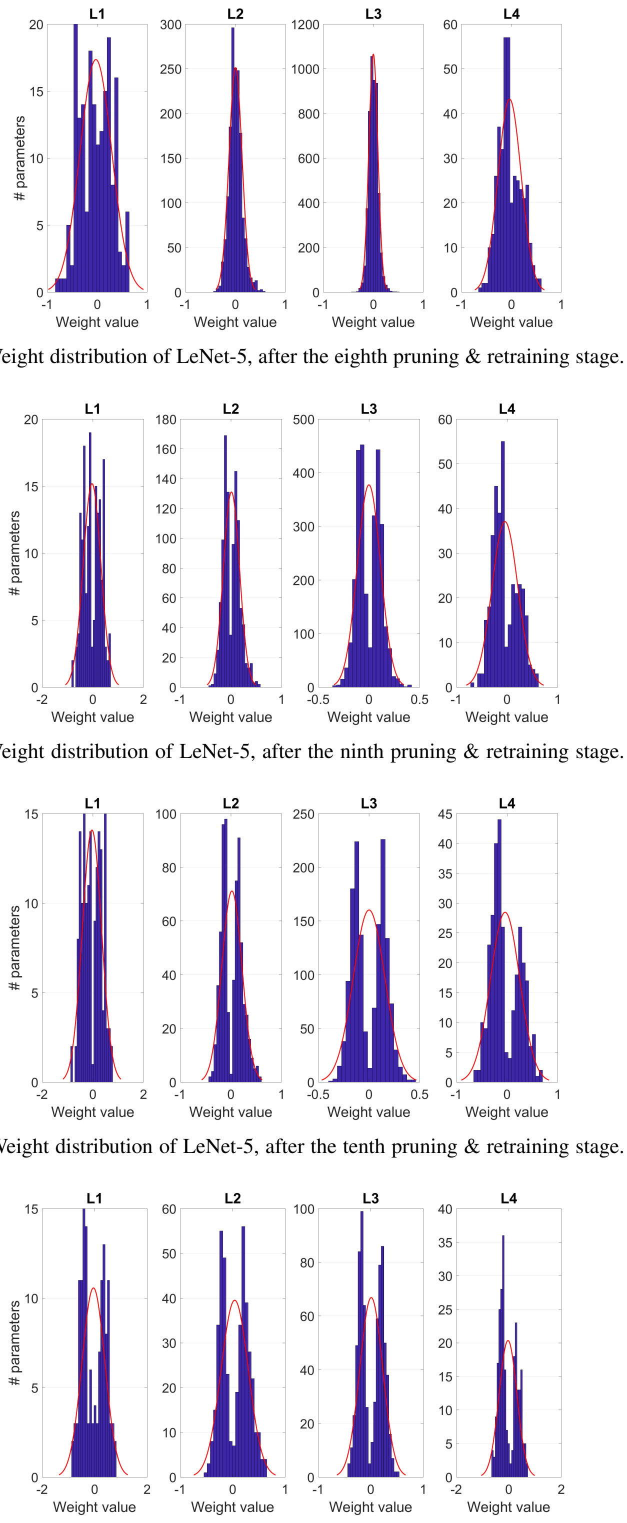
(e) Weight distribution of LeNet-5, after the fourth pruning & retraining stage.

(f) Weight distribution of LeNet-5, after the fifth pruning & retraining stage.

(g) Weight distribution of LeNet-5, after the sixth pruning & retraining stage.

(h) Weight distribution of LeNet-5, after the seventh pruning & retraining stage.

Fig. 5: (continued) Weight distribution of LeNet-5, across different pruning & retraining stages.



(i) Weight distribution of LeNet-5, after the eighth pruning & retraining stage.

(j) Weight distribution of LeNet-5, after the ninth pruning & retraining stage.

(k) Weight distribution of LeNet-5, after the tenth pruning & retraining stage.

(l) Weight distribution of LeNet-5, after the eleventh pruning & retraining stage.

Fig. 5: (continued) Weight distribution of LeNet-5, across different pruning & retraining stages.

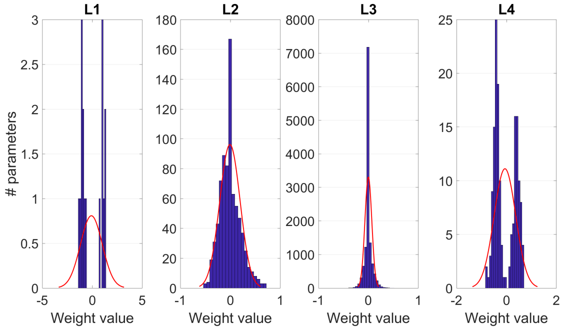


Fig. 6: Weight distribution of LeNet-5, after the fifth pruning & retraining stage, using Class-Uniform method.

Dataset	Network	AL	MSR
MNIST	LeNet-5	0.11097%	190.75X
MNIST	LeNet-300-100	0.07165%	107.072X
CIFAR-10	VGG-16	-0.2143%	115.382X
CIFAR-100	VGG-16	-0.8324%	91.462X
CIFAR-100	AlexNet	0.0772%	62.727X
CIFAR-100	GoogleNet	0.0772%	15.136X

TABLE I: Experiment results in terms of Accuracy Loss and Memory Saving Ratio.

A. *HW/SW Setup*

We use pyTorch framework [37] for our experiments. We implement sparsity as masked layers, where each mask is multiplied with the weight matrix during the retraining process. The accuracy has been computed by running inference on the validation set, while MSR has been evaluated as the ratio between the nonzero parameters of the baseline model and the nonzero ones of the sparse model. The experiments have been run on Nvidia GTX 1070 GPU, whose specs are reported in Table II. A schematic view of the process flow is reported in Figure 7.

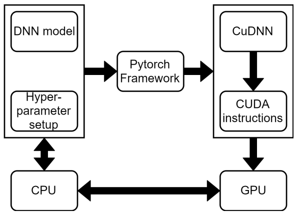


Fig. 7: SW to HW setup

NVIDIA GTX 1070 specs	
CUDA cores	1920
Memory	8 GB DDR5
Mem. interface width	256-bit
Mem. bandwidth	256 GB/s
Single precision Flops	6.5 TeraFLOPS
Power requirement	150 W

TABLE II: GPU specs

B. *LeNet-300-100 on MNIST*

LeNet-300-100 is a Neural Network consisting of three fully-connected layers, as described in [20]. We applied our methodology to this Network on MNIST dataset, obtaining a sparsity percentage of 0.93%, which corresponds to a Memory Saving Ratio of 107X. Such result has been achieved after the seventh iteration with constant pruning percentage of 50%. The initial learning rate has been reduced accordingly, from 0.01 of the train-from-scratch process, to 0.003 for retraining.

C. VGG-16 on CIFAR-10 and CIFAR-100

VGG-16 network is a quite deep neural network, with 13 convolutional layers and 3 fully-connected ones. The model used to be trained on ImageNet dataset is described by Simonyan and Zisserman [28]. We applied some minor modifications to adapt it for other datasets: CIFAR-10 and CIFAR-100. Both datasets are composed of 50000 training images and 10000 test images. While CIFAR-10 contains 10 different classes of images, CIFAR-100 has 100 classes. We applied our methodology on this network and the two datasets provide different results: we obtained a sparsity percentage of 0.87% on CIFAR-10 and 1.09% on CIFAR-100; the respective Memory Saving Ratios are 115X and 91X. For both configurations, the starting learning rate has been set to 0.003 and the pruning percentage to 50%.

D. AlexNet and GoogleNet on CIFAR-100

Again, CIFAR-100 dataset has been used for training other two DNNs, AlexNet [19] and GoogleNet [30]. Since in their respective original papers, the two networks were designed to be trained on input images of size 224x224, they have been adapted to the size of CIFAR-100 images (32x32). For AlexNet we achieved a MSR equal to 63X, while for GoogleNet 15X. Note that the MSR for GoogleNet is quite low compared to the other experiments, because this DNN is already more sparse than the others in the original form. For this reason, we are not able to compress much the network without affecting the accuracy. The starting learning rate used for AlexNet is 0.003, while the respective value for GoogleNet is 0.03. The pruning percentage set for both processes is 50%.

VII. Conclusions

In this paper we present a simple but at the same time effective methodology, based on Class-Blind pruning scheme, to compress wide and dense Neural Networks (GoogleNet is considered a less dense one, as explained in Section VI-D) from 60X to 190X, without affecting the accuracy. This result allows to reduce memory and computational requirements, making inference more feasible to deploy on mobile applications. Since our "PruNet" method is orthogonal with other compression techniques, like quantization and weight sharing, the memory savings can be improved further.

References

[1] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *arXiv preprint arXiv:1512.08571*, 2015.

[2] Soravit Changpinyo, Mark Sandler, Andrey Zhmoginov. The power of sparsity in convolutional neural networks. In *ICLR*, 2017.

[3] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[4] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or 1. *arXiv preprint arXiv:1602.02830*, 2016.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.

[6] Matthieu Courbariaux, Jean-Pierre David, and Yoshua Bengio. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[7] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[8] X. Dong, S. Chen, and S. J. Pan, Learning to prune deep neural networks via layer-wise optimal brain surgeon, *arXiv preprint arXiv:1705.07565*, 2017.

[9] Marco Federici, Karen Ullrich, Max Welling, Improved Bayesian Compression. In *NIPS*, 2017.

[10] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *NIPS*, 2016.

[11] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *CoRR*, 2015.

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2015.

[13] S. Han, J. Pool, J. Tran, and W. Dally, Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

[14] Babak Hassibi, David G Stork, et al. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 1993.

[15] K. He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

[16] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS*, 2015.

[18] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *NIPS*, 2014.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (NIPS), pages 10971105, 2012.

[20] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.

[21] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[22] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. In *ICLR*, 2016.

[23] Darryl D Lin, Sachin S Talathi, and V Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. *arXiv preprint arXiv:1511.06393*, 2015.

[24] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *NIPS*, 2017.

[25] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[26] Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. In *CoRR*, 2017.

[27] Abigail See, Minh-Thang Luong, and Christopher D. Manning. Compression of neural machine translation models via pruning. In *CoNLL*, 2016.

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[29] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. In *Journal of MachineLearning Research*, 2014.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[31] K. Ullrich, E. Meeds, and M. Welling. Soft Weight-Sharing for Neural Network Compression. *ICLR*, 2017.

[32] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *NIPS*, 2011.

[33] Ganesh Venkatesh, Eriko Nurvitadhi, and Debbie Marr. Accelerating deep convolutional networks using low-precision and sparsity. *arXiv preprint arXiv:1610.00324*, 2016.

[34] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In Advances In *NIPS*, 2016.

[35] R. Wu, S. Yan, Y. Shan, et al., Deep image: scaling up image recognition, *arXiv preprint, arXiv: 1501.02876*, 2015.

[36] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[37] pyTorch framework: https://github.com/pytorch/pytorch