Deep Neural Network Pruning with Progressive Regularizer

Yexu Zhou[1♣], Haibin Zhao[1♣], Michael Hefenbrock[2], Siyan Li[1], Yiran Huang[1], and Michael Beigl[1]
[1]*Karlsruhe Institute of Technology,* [2]*RevoAI GmbH*
[1]{yexu.zhou, haibin.zhao, siyan.li, yiran.huang, michael.beigl}@kit.edu
[2]michael.hefenbrock@revoai.de

***Abstract*—Pruning is a pivotal approach in network compression. It not only encourages lightweight deep neural networks, but also helps to mitigate overfitting. Generally, regularization is used to guide more parameters towards zero and thus reduce the overall model complexity. Unfortunately, there are two issued remaining unsolved in regularization based pruning. One is that the optimal trade-off between regularization and loss minimization, often expressed via a scaling hyperparameter, needs to be found through extensive experimentation. The other one is the importance criteria that can reflect the true relative importance. The most widely used criterion is the magnitude-based, which has been argued to be inaccurate. To these two issues, in this paper, we propose a progressive regularization scheme, in which the factor scaling the regularization term is gradually increased during training, until the target sparsity for filter pruning is reached. Compared to the previous approach, the scaling factor is no longer a hyperparameter that needs to be tuned, but is replaced with a sparsity-aware parameter that increases progressively. In this way, the value of the scaling factor can be automatically aligned with the target sparsity, avoiding the drawbacks in its tuning. Furthermore, Only parameters below a minimal and learnable soft threshold are pruned, therefore, informative parameters (even with small magnitudes) are optimally preserved. Consequently, the performance loss due to pruning is mitigated. Experiments with various models and benchmark datasets prove the effectiveness of the proposed method.**

***Index Terms*—progressive regularizer, neural network, tiny machine learning, lightweight**

I. Introduction

Due to the remarkable expressiveness and scalability [1], deep neural networks (DNNs) have achieved astonishing results in many fields, such as machine vision [2], human activity recognition [3], and natural language processing [4]. However, for large DNNs, the enormous number of parameters leads to expensive computation and memory demands [5]. Consequently, this severely affects their real-time performance and hinders their deployment on edge devices [6].

Regularization-based neural network pruning has gained significant interest among researchers due to its ability to not only shrink network size by promoting greater sparsity but also to mitigate overfitting to a certain degree. Despite extensive research on regularization functions and pruning methods, two key issues still remain to be addressed: 1) The scaling factor (a hyperparameter) for regularization requires careful tuning. Selecting a small scaling factor may not sufficiently drive parameters toward zero, while a large scaling factor, though increasing sparsity, may result in increasingly worse conditioning of the optimization problem; 2) Establishing a definitive pruning criterion, specifically, a reliable metric to assess parameter importance, is yet to be determined. Only parameters that are precisely zero are pruned, have no impact on the model performance.

In this work, we propose a progressive scheme for regularization-based pruning, which addresses the aforementioned problems. Specifically, after the training converged with a certain regularization scaling factor, the sparsity of the DNN is assessed. Here, only parameters, e.g., weights or filters, whose values below a minimal and learnable threshold, are considered. If the sparsity does not reach the target value, the scaling factor is increased and the training continues until the next convergence. This procedure is repeated until the network displays the target sparsity. To avoid large gradients resulting in ill-conditioning, we derive an update rule for the scaling factor of the regularization term. After the target sparsity is reached, the DNNs is pruned and fine-tuned. Moreover, as we use a learnable threshold for parameter pruning, only non-informative parameters will be pruned, whereas small but sensitive parameters can be optimally retained. Consequently, the accuracy loss due to pruning can be remedied.

Experiments on benchmark datasets (CIFAR-10/100, ImageNet) with structured pruning of SOTA models (ResNet, VGG) prove the effectiveness of our approach.

In summary, the contributions of this work are:

- We propose a neural network pruning based on progressive regularization, which can overcome the drawbacks of the hyperparameter tuning of the scaling factor.
- We derive reasonable step sizes of the growth of the scaling factor, by which the ill-conditioning caused by the update of the scaling factor can be mitigated.
- We introduce a learnable parameter as the soft threshold to distinguish non-informative parameters from informative parameters.
- We conduct extensive experiments with various datasets, DNN models, and regularization functions. The results show that the progressive strategy outperforms previous strategies involving direct hyperparameter tuning for the scaling factor.

The rest of this work is structured as follows: Section II gives an overview of the related work on neural network

♣Co-first authors contributed equally to this work.

pruning. Section III motivates and introduces the progressive regularization-based pruning. In Section IV, we describe our experiment and evaluate our approach by comparing with previous approaches and other methods. Section V summarizes this work.

II. Related Work

Pruning in neural networks is typically categorized into two types: *structured* and *unstructured*. *Unstructured* pruning involves the independent removal of individual parameters, as described by Han et al. [7]. This approach offers flexibility and can enhance performance, but it necessitates the use of specialized libraries for sparse matrix operations. In contrast, *structured* pruning, which targets specific filters or channels within the network [8], is more straightforward to implement in practical applications. In this work, since we mainly focus on model acceleration, so that we focused on the filter pruning scenarios.

The primary goal of pruning is to eliminate filters that contribute minimal information, thereby reducing redundancy. Various criteria have been proposed to assess the informativeness of filters. For example, the research by Li et al. [9] eliminates filters based on their variance and inter-filter similarity, identifying those that are non-informative or redundant. Other studies, such as those by Yang et al. [10] and Verdenius et al. [11], utilize sensitivity-based criteria to evaluate the impact of filter removal on model performance. However, a challenge with this approach is the difficulty to match the predetermined pruning ratio, as the training process may not be attuned to this goal. To achieve high pruning ratios, filters that still influence outcomes may be removed, potentially leading to a decline in accuracy.

Magnitude-based pruning, which estimates the importance of parameters based on their values or scaling factors in batch normalization [12], [13], is a popular method for attaining higher pruning ratios. This technique incorporates a regularization term to encourage more parameters to approach zero, thereby minimizing the impact of their removal. Various regularization functions have been proposed, with the ideal being the ℓ_0-norm [14]. However, due to its non-differentiable nature, the ℓ_1-norm is frequently employed as a practical substitute [15]. Research has explored diverse regularizers, such as $\ell_{0.5}$-norm [16], transformed ℓ_1 ($t\ell_1$) [17], capped ℓ_1 ($c\ell_1$) [18], smoothly clipped absolute deviation (SCAD) [19], minimax concave penalty (MCP) [20], log penalty [21], relaxed ℓ_0 ($r\ell_0$) [22], etc. All regularizers have been shown to outperform the ℓ_1-norm in specific cases.

Despite its advantages, magnitude-based pruning has limitations, as some parameters with low magnitudes are sensitive for network output [23]. Removing these parameters can degrade model performance. Additionally, setting the trade-off parameter in regularization-based pruning poses a challenge. A parameter that is too small may fail to drive sufficient weights to zero, hindering the achievement of the target pruning ratio. Conversely, a parameter that is too large can cause many filters to attain very small values prematurely, impeding the model's learning ability [24].

To address these challenges, some studies have proposed progressive regularization, which involves gradually increasing the strength of regularization during training. For instance, the approach in [22] progressively transforms a regularization function from a negative squared exponential to a near-ℓ_0 shape, mitigating the drawbacks of ℓ_0 regularization. Wang et al.[23] suggest progressively pushing unimportant filters to very small values through increasing regularization. However, as mentioned earlier, very small values may still impact performance. Similarly, the DPFPS method [24] proposes gradually increasing the regulation imposed on unimportant filters, setting them to zero when their norms fall below a progressively increasing threshold. Nonetheless, the pattern of this threshold increase is predetermined, and experimental results indicate that the hyperparameters involved significantly affect the outcomes. Another progressive approach PSFP [25] involves gradually increasing the pruning ratio in combination with a soft threshold to achieve the target pruning ratio. However, this method requires a predefined function to determine the variation of the pruning ratio, and the hyperparameters must also be set within this function.

In contrast, our approach offers several distinctive features:

1) We utilize a learnable soft threshold that is automatically adjusted in conjunction with the objective function, effectively pushing more weights to exact zero values.

2) Our progressive strategy focuses on the scaling factor of the regularizer, rendering it compatible with a variety of regularization functions from previous research.

3) We avoid predefined schedules for pruning ratios or regularization.

4) Unlike gradual pruning methods, which remove filters during the training process, our approach prunes DNNs only at the end of training. This prevents the premature removal of parameters that exhibit zero magnitude only temporarily, allowing such parameters the opportunity to recover.

III. Progressive Regularization

We first motivate our approach and introduce the progressive regularizer in detail. In addition, we derive the step size of the scaling factor update during training and introduce a learnable soft threshold to prune non-informative parameters while retaining sensitive parameters as far as possible. Lastly, we suggest an *invariant regularization* against increasing model size and varying functional forms of regularizers.

A. Motivation

In previous works on regularization-based pruning, attention has often been paid to the design of regularization functions to encourage higher sparsity. They thus usually follow the pruning strategy described in Algorithm 1 [8]. However, several flaws might arise in this process.

- The scaling factor here is a hyperparameter, that requires numerous experiments for tuning. Additionally, general regularizers are not invariant to the model size and

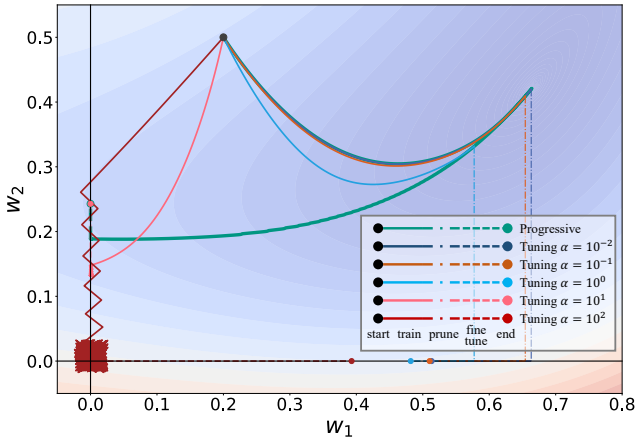

Fig. 1. Trajectories of a two-dimensional pruning example with 50% target pruning ratio. The bold green line indicates the progressive scheme, while the thin lines in different colors denote the non-progressive (hyperparameter tuning) scheme. Black point indicates the start point, solid lines refer to training, dash-and-dot lines refer to pruning, dot lines refer to fine-tuning, and points in corresponding colors denote the end position of the training.

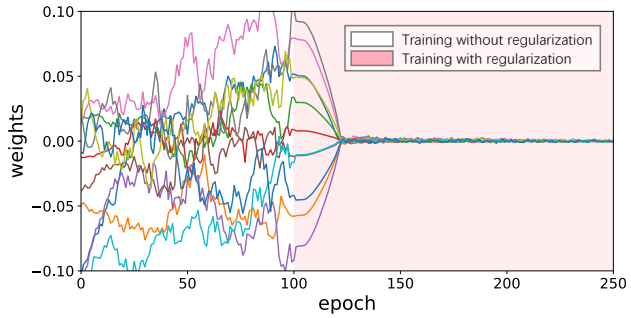

Fig. 2. Exemplary weight changes during the training process. Since the 100-th epoch, a large regularization is introduced to force weights towards zero.

regularization functional forms, i.e., the magnitude of the regularization term varies strongly across networks and regularizers (see Section III-E). Therefore, a priori knowledge of the scaling factor tuning from other works is generally not helpful. Figure 1 shows an example of pruning with different scaling factors. While small factors can lead to weak regularization, large factors can lead to unstable dynamics of the optimization process.

- The pruning criterion is not well-defined. In general, smaller parameters are pruned according to the target sparsity S (in percentage) or pruning ratio. However, if the scaling factor is small, the target sparsity S can not be reached. In this case, some small but important (sensitive) parameters, such as w_2 in Figure 1, might be pruned, which could lead to a significant drop in model performance. In contrast, if the scaling factor is large, a higher sparsity will be achieved than the target value. In this case, the pruning can be seen as somehow randomly performed among those "zero-valued" parameters[1] (see red trajectory in Figure 1 and weight values in Figure 2).

[1]The parameters are not exactly zero due to numerical computing, but fluctuate almost randomly around zero.

Algorithm 1: Baseline [8]

Data: $\mathcal{D} = \{\mathbf{x}, \mathbf{y}\}$
Input : NN, L, R, α, P, S, optimizer
Init : patience $\leftarrow 0$, $\mathcal{L}_{\text{best}} \leftarrow \infty$
while patience $< P$ **do**
  $\mathcal{L} \leftarrow L(\boldsymbol{w}, \mathcal{D}) + \alpha R(\boldsymbol{w})$
  NN $\leftarrow$ optimizer$(\mathcal{L}, \text{NN})$
  if $\mathcal{L} < \mathcal{L}_{\text{best}}$ **then**
    $\mathcal{L}_{\text{best}} \leftarrow \mathcal{L}$
    patience $\leftarrow 0$
  else
    patience $\leftarrow$ patience $+ 1$
  end
end
pruning(NN, S)
fine tune(NN)

B. Progressive Regularization

To mitigate the aforementioned drawbacks, we propose a progressive regularization strategy for neural network pruning (see Algorithm 2, the differences from the baseline are marked in red). Generally, the typical objective function $\mathcal{L}$ for training DNNs with regularization is

$$\mathcal{L} = L(\boldsymbol{w}, \mathcal{D}) + \alpha R(\boldsymbol{w}), \tag{1}$$

where L is the loss function (e.g., cross entropy in classification) of data $\mathcal{D} = \{\mathbf{x}, \mathbf{y}\}$ and model parameters $\boldsymbol{w}$, $R(\cdot)$ is the regularization function, and α is the scaling factor of the regularization term, which is usually a tuning parameter.

However, in this work, we grow α gradually (and dynamically) with consideration of the network sparsity. After each training convergences (by early-stopping with patience P), α is increased slightly, and the training will then continue from the current solution. As α grows, the regularization term progressively dominates the objective function, thus, the sparsity of the network increases continuously, until the target sparsity is reached. Then, we aim to remove only the zero-valued pruning parameters, and perform fine-tuning to account for the changes in the architecture. The bold green curve in Figure 1 shows an example of progressive regularization for pruning. As can be seen, some parameter values decrease at the start of regularization, but cannot be forced smaller further (even with large regularization penalty), such as w_2, these are sensitive parameters that are important for the DNN and should not be pruned. Therefore, our intuitive idea was to prune only zero-valued parameters, as they can be removed without any effect on the output. However, due to numerical computing, there is no ideal "zero-valued" parameter during training. The zero-valued parameters fluctuate around zero. To decide a reasonable threshold, below which parameters should be pruned, we introduce a learnable soft threshold.

C. Factor Update

Large gradient often leads to instabilities in gradient-based optimization [26][27], therefore, to avoid large gradient caused

by factor update, we derive the update size of the scaling factor.

a) Derivation of step size: Suppose the training can be converged by early-stopping criterion in the k-th update, we have

$$\nabla_{\boldsymbol{w}}\mathcal{L}^{(k)} = \nabla_{\boldsymbol{w}}L(\boldsymbol{w},\mathcal{D}) + \alpha^{(k)}\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) = \mathbf{0}. \tag{2}$$

Therefore, after the $(k+1)$-th update, the following condition holds for the warm start position

$$\begin{aligned}\nabla_{\boldsymbol{w}}\mathcal{L}^{(k+1)} &= \nabla_{\boldsymbol{w}}L(\boldsymbol{w},\mathcal{D}) + \alpha^{(k+1)}\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) &(3)\\ &= -\alpha^{(k)}\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) + +\alpha^{(k+1)}\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) &(4)\\ &= \underbrace{(\alpha^{(k+1)} - \alpha^{(k)})}_{=:\Delta_\alpha^{(k+1)}}\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) &(5)\\ &= \Delta_\alpha^{(k+1)}\nabla_{\boldsymbol{w}}R(\boldsymbol{w}). &(6)\end{aligned}$$

To avoid large gradient $\nabla_{\boldsymbol{w}}\mathcal{L}^{(k+1)}$ after factor update that may lead to instable optimization process, it should hold that

$$\nabla_{\boldsymbol{w}}\mathcal{L}^{(k+1)} = \Delta_\alpha^{(k+1)}\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) \leq \nabla_{\max}, \tag{7}$$

where $\nabla_{\max}$ indicates the maximal acceptable gradient after each update. Therefore, we observe that, for a given regularization function R and the maximum acceptable gradient norm $\nabla_{\max}$, the step size Δ_α can be determined, i.e.,

$$\alpha^{(k+1)} - \alpha^{(k)} \leq \frac{\nabla_{\max}}{\|\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\|_\infty}. \tag{8}$$

Here, $\|\nabla R\|_\infty$ can be obtained by $\arg\max_{\boldsymbol{w}} R(\boldsymbol{w})$, where $\boldsymbol{w}$ is the model parameters to be regularized, and $\nabla_{\max}$ can be chosen empirically as 1.

b) Step size for certain regularizers: In the following, we derive the update step for certain regularizers, which are used in our experiment.

ℓ_p-norm: For ℓ_p-regularization, rather than going to a strictly mathematical form, we adopted a slightly different variation, which is also widely used in other machine learning works, such as [28]–[33], i.e.,

$$R(\boldsymbol{w}) = \sum_n w_n^p, \quad \nabla_{\boldsymbol{w}}R(\boldsymbol{w}) = p\boldsymbol{w}^{p-1}. \tag{9}$$

where $\boldsymbol{w} = [w_1, w_2, w_3, ...]$ collecting all the parameters in a DNN. Therefore, for $p \geq 1$, the maximum value occurs at

$$\|\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\|_\infty = pw^{p-1}\big|_{w=w_{\max}} = pw_{\max}^{p-1}, \tag{10}$$

where $w_{\max} = \|\boldsymbol{w}\|_\infty$, i.e., the element with the maximum absolute value in $\boldsymbol{w}$ that can be found from the DNN parameters. Therefore, the update step size is limited to

$$\Delta_\alpha \leq \nabla_{\max}\frac{1}{p}w_{\max}^{1-p}. \tag{11}$$

For $0 < p < 1$, the maximal gradient is

$$\|\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\|_\infty = pw^{p-1}\big|_{w=w_{\min}} = pw_{\min}^{p-1}, \tag{12}$$

where $w_{\min}$ is the minimal absolute value in $\boldsymbol{w}$, which can be found from the DNN parameters. Thus,

$$\Delta_\alpha \leq \nabla_{\max}\frac{1}{p}w_{\min}^{1-p}. \tag{13}$$

Capped ℓ_1 ($c\ell_1$):

$$R(\boldsymbol{w}) = \sum_n \min\{c, w_n\}, \tag{14}$$

$$\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) = \mathbf{0}\cdot\mathbb{1}_{\{w_n\geq c\}} + \mathbf{1}\cdot\mathbb{1}_{\{w_n<c\}}, \tag{15}$$

where c is the hyperparameter determining the specific shape of the $c\ell_1$-regularizer and $\mathbb{1}_{\{\cdot\}}$ denotes an indicator function returning 1 if the respective condition is true, else 0. Therefore,

$$\|\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\|_\infty = 1,\ i.e.,\ \Delta_\alpha \leq \nabla_{\max} \tag{16}$$

Log-norm:

$$R(\boldsymbol{w}) = \sum_n \frac{\log(\gamma|w_n|+1)}{\log(\gamma+1)}, \tag{17}$$

$$\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) = \frac{\gamma}{\log(\gamma+1)}\frac{1}{\gamma|\boldsymbol{w}|+1}, \tag{18}$$

where $\gamma > 0$ is the hyperparameter determining the specific shape of the regularizer. Therefore,

$$\|\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\|_\infty = \frac{\gamma}{\log(\gamma+1)}\frac{1}{\gamma|w_{\min}|+1}, \tag{19}$$

thus,

$$\Delta_\alpha \leq \nabla_{\max}(\gamma|w_{\min}|+1)\frac{\log(\gamma+1)}{\gamma}. \tag{20}$$

Minimax concave penalty (MCP):

$$R(\boldsymbol{w}) = \sum_n P(w_n), \tag{21}$$

$$\text{with } P(w_n) = \begin{cases}\lambda|w_n| - \dfrac{w_n^2}{2\gamma}, & |w_n| \leq \gamma\lambda\\ \dfrac{\gamma\lambda}{2}, & |w_n| > \gamma\lambda\end{cases}, \tag{22}$$

$$\nabla_{\boldsymbol{w}}R(\boldsymbol{w}) = (\lambda - \frac{1}{\gamma}\boldsymbol{w})\cdot\mathbb{1}_{\{|w_n|\leq\gamma\lambda\}} + \mathbf{0}\cdot\mathbb{1}_{\{|w_n|>\gamma\lambda\}}, \tag{23}$$

where $\gamma > 1$ and $\lambda > 0$ are hyperparameters determining the specific shape of the regularizer, therefore,

$$\|\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\|_\infty = \lambda - \frac{1}{\gamma}w_{\min},\ thus,\ \Delta_\alpha \leq \nabla_{\max}\frac{\gamma}{\gamma\lambda - w_{\min}}. \tag{24}$$

D. Learnable Soft Threshold

During the progressive regularization, an increasing number of parameters will be gradually pushed to zero. Parameters, that exhibit zero values earlier (for smaller values of α), are generally less important than those reaching zero later (for higher values of α), as their reduction can provide more regularization gain compared to reduction in loss. This observation forms the basis of our approach.

In training, the values of the parameters usually do not reach exactly zero, but rather oscillate around it. In Figure 2, we visualize some parameters, which oscillate frequently between positive and negative values during the training process. In this work, we consider these parameters to be zero-valued parameters. However, at the same time, there are

also parameters that can be pushed to small magnitudes by the regularization, but cannot be reduced to zero, as a slight reduction in their values may cause huge change in the loss. In this work, these parameters are defined as sensitive parameters.

To distinguish these two kinds of parameters, we introduce a learnable soft threshold [34]. For this, we express the original network parameters $\boldsymbol{w}$ as

$$\boldsymbol{w} \hookleftarrow \text{sign}(\boldsymbol{w}) \odot \text{ReLU}(|\boldsymbol{w}| - t), \tag{25}$$

where "$\odot$" denotes element-wise multiplication, $\text{ReLU}(x) = \max\{x, 0\}$ is also applied in element-wise on its arguments, and $t \in \mathbb{R}^+$ is the learnable soft threshold. The positive value of t can be guaranteed by mapping it through a differentiable function with only positive range. Here we use a sigmoid function, which is kept the same as in [34].

Intuitively, t should be a value close to zero, as soft pruning of parameters below t affects the performance of the DNN, unless the parameters below t have only a subtle or even negative effect on the network performance. In this case, those "zero-valued" parameters, which are, due to the numerical computing, embodied as values fluctuating around zero, are removed through the learnable threshold. Conversely, this learnable threshold will not increase to a large magnitude, as a larger threshold would remove informative and sensitive parameters. Consequently, the soft threshold will learn to stay at a value to optimally (w.r.t. objective function) filter out non-informative parameters among low value parameters.

E. Invariant Regularization

In Equation 1, α can be interpreted also as the trade-off between task loss $L(\boldsymbol{w}, \mathcal{D})$ and the regularization term $R(\boldsymbol{w})$ corresponding to the sparsity. However, the value of $R(\boldsymbol{w})$ is usually highly related to the size of DNNs (i.e., size of $\boldsymbol{w}$) and the functional form of $R(\cdot)$. Therefore, if either the size of DNNs or the regularization function changes, the value of $R(\boldsymbol{w})$ will also change. Consequently, the magnitude of α has to be re-tuned for rebalancing $L(\boldsymbol{w}, \mathcal{D})$ and $R(\boldsymbol{w})$. However, this process re-tune process generally has no a priori knowledge. To overcome this problem, we modified all the regularizers to be invariant of the growing model size. Moreover, to balance the magnitudes among different regularization functions, we suggest to normalize the regularization values in consideration of the initial parameter distribution.

a) Size invariant regularization: To guarantee the invariance of the regularization against model size, we scaled the regularization term by $1/N^k$, i.e.,

$$\tilde{R}(\boldsymbol{w}) = \frac{R(\boldsymbol{w})}{N^k}, \tag{26}$$

where N is the size of the vector being regularized (here $\boldsymbol{w}$), and k is the order of N contributing to the growing of the regularization, i.e., $\mathcal{O}(N^k)$. Taking ℓ_1-norm, as an example, i.e.,

$$\ell_1(\boldsymbol{w}) = \sum_{n=1}^{N} |w_n|,$$

Algorithm 2: Progressive Regularization

Data: $\mathcal{D} = \{\mathbf{x}, \mathbf{y}\}$
Input : NN, L, R, Δ_α, P, $\boldsymbol{S}$, optimizer
Init : sparsity $\leftarrow 0$
while sparsity$^{(l)} < S^{(l)}$, $\forall l$ **do**
  $\mathcal{L}_{\text{best}} \leftarrow \infty$
  patience $\leftarrow 0$
  while patience $< P$ **do**
    $\mathcal{L} \leftarrow L(\boldsymbol{w}, \mathcal{D}) + \alpha R(\boldsymbol{w})$
    NN, $\boldsymbol{t} \leftarrow$ optimizer$(\mathcal{L}, \text{NN})$
    if $\mathcal{L} < \mathcal{L}_{\text{best}}$ **then**
      $\mathcal{L}_{\text{best}} \leftarrow \mathcal{L}$
      patience $\leftarrow 0$
    else
      patience $\leftarrow$ patience $+ 1$
    end
  end
  update sparsity$^{(l)}$, $\forall l$
  $\alpha^{(l)} \leftarrow \alpha^{(l)} + \Delta_\alpha^{(l)}$, $\forall$ sparsity$^{(l)} < S^{(l)}$
end
pruning (NN, $\boldsymbol{t}$)
fine tune (NN)

where w_n is the parameter following a certain distribution with expected value $\mathbb{E}\{w\}$, the expected value of $\ell_1(\boldsymbol{w})$ is

$$\mathbb{E}\{\ell_1(\boldsymbol{w})\} = \mathbb{E}\left\{\sum_{n=1}^{N} |w_n|\right\} = N\mathbb{E}\{|w|\} = \mathcal{O}(N), \tag{27}$$

i.e., the magnitude of ℓ_1-regularizer increases with the size N of $\boldsymbol{w}$ by $\mathcal{O}(N)$. Therefore, for the invariance against model size, the ℓ_1-regularizer is then modified to

$$\tilde{\ell}_1(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} |w_n|.$$

By now, the value of ℓ_1-regularizer is stable with growing size of $\boldsymbol{w}$.

b) Functional form invariant regularization: After making the regularization values invariant to model size, we also try to keep the values at the same level among different functional forms. To this end, we normalize the regularization value by the initial parameters, i.e.,

$$\mathcal{R}(\boldsymbol{w}) = \frac{\tilde{R}(\boldsymbol{w})}{\tilde{R}(\boldsymbol{w}_{\text{init}})}. \tag{28}$$

Here, $\boldsymbol{w}_{\text{init}}$ is the initial value of the parameters, e.g., whose distribution generally follows a Gaussian distribution [35].

c) Invariant regularizer for group sparsity: As mentioned earlier, we employ structured pruning on filter level. Therefore, rather than putting each parameter separately into the regularizer $\mathcal{R}(\cdot)$, we collect them in groups (also known as group sparsity [36]), i.e.,

$$\mathcal{R}(\boldsymbol{f}) = \mathcal{R}([f_1, f_2, ...]), \tag{29}$$

where $\boldsymbol{f}$ summarizes all the size-invariant Euclidean lengths of filters f_i with

$$f_i = \tilde{\ell_2}([w_1^{(f_i)}, w_2^{(f_i)}, ...]) = \frac{1}{\sqrt{N}} \left\| [w_1^{(f_i)}, w_2^{(f_i)}, ...] \right\|_2 . \quad (30)$$

Here, $[w_1^{(f_i)}, w_2^{(f_i)}, ...]$ collects all parameters within the i-th filter. We can interpret Equation 29 in the following way: We first express the filters by their scalar Euclidean lengths, and then use a regularizer to encourage higher sparsity on filter level, i.e., to encourage more filter lengths towards zero. It is worth noting that N and $\tilde{R}(\boldsymbol{w}_{\text{init}})$ for invariant regularization $\mathcal{R}(\cdot)$ in the last paragraphs are modified accordingly to the number of filters and $\tilde{R}(\boldsymbol{f}(\boldsymbol{w}_{\text{init}}))$.

TABLE I
REGULARIZERS AND TUNING PARAMETERS IN THE EXPERIMENTS WITH CIFAR DATASETS.

Regularizer	Functional form	Shape tuning
ℓ_p-norm	$\sum_n w_n^p$	$p \in \{0.1, 0.2, 0.5, 1, 2\}$
$t\ell_1$	$\sum_n \frac{(a+1)\lvert w_n \rvert}{a+\lvert w_n \rvert}$	$a \in \{e^{-2}, e^{-1}, e^0, e^1, e^2\}$
$c\ell_1$	$\sum_n \min\{\lvert w_n \rvert, c\}$	$c \in \{e^{-2}, e^{-1}, e^0, e^1, e^2\}$
log-norm	$\sum_n \frac{\log(\gamma\lvert w_n \rvert+1)}{\log(\gamma+1)}$	$\gamma \in \{e^{-4}, e^{-2}, e^0, e^2, e^4\}$
MCP	$\sum_n P(w_n)^*$	$\lambda \in \{0.25, 0.5, 1\}, \quad \gamma \in \{2, 5, 8\}$

* $P(w_n) = \lambda\lvert w_n \rvert - w_n^2/(2\gamma)$.
for $\lvert w_n \rvert \leq \gamma\lambda$ and $P(w_n) = \gamma\lambda^2/2$, for $\lvert w_n \rvert > \gamma\lambda$

F. Layer-wise Pruning

Our approach can be easily adapted to global pruning [7] or layer-wise pruning [34]. In this work, we perform uniform pruning on DNNs, the reasons are two-folds: 1) The expressiveness of DNNs is also partly determined by the architecture [37], global pruning might destroy the model architecture by removing all parameters in a certain layer. 2) Global pruning usually results in higher FLOPs than layer-wise pruning [38], [39]. A comparison of global and layer-wise pruning is reported in section IV-E.

Note that, in layer-wise pruning, either uniform or non-uniform pruning can be realized by setting the same or different target sparsity $S^{(l)}$ to the l-th layer in the DNN. Analogous, the learnable soft thresholds will also be learned independently cross layers, i.e., $t^{(l)}$ for each layer l. In this case, we denote them by $\boldsymbol{S} = \{S^{(1)}, S^{(2)}, ...\}$ and $\boldsymbol{t} = \{t^{(1)}, t^{(2)}, ...\}$ for target sparsity and learnable threshold, respectively.

G. Discussion

In this section, we propose a progressive regularization scheme to overcome the weaknesses of the previous pruning scheme. We progressively increase α to force higher sparsity in the network continuously until the target sparsity is reached. In this way, the parameters being pushed to zero might have less importance than others. For this process, we also suggest the step size for factor updating and invariant regularization. To distinguish between the informative and non-informative small parameters, we introduce a learnable soft threshold with aims at pruning them without severe effects on the objective function. Although $\nabla_{\max}$ is an adjustable parameter, it generally needs no tuning, as it essentially does not change the convergence point of the optimization. A simple experiment shows that, the performance of DNNs is also not sensitive to its value (see section IV-D). Another independent work may also support this perspective, i.e., $\nabla_{\max}$, as a hyperparameter of hyperparameter (α), tends to be more robust than the hyperparameter itself [40]. Empirically, $\nabla_{\max} = 1$ works well and usually needs no additional change.

IV. EXPERIMENT

To evaluate the proposed progressive regularization, we conduct extensive experiments on the image classification tasks based on the framework developed by Intel[2] [41]. Our code is available at www.github.io. The experiments are conducted on Nvidia A100 40G GPUs.

A. Experiment Setup

We first carry out the experiments on CIFAR-10 and CIFAR-100 [42] with ResNet56 [43] and VGG19 [44] respectively. Then to further evaluate the effect of progressive regularization on large scale dataset, we evaluate ResNet34 and ResNet50 [43] on ImageNet [45]. Regarding the pruning setup, we follow the common settings described in, e.g., [46] and [23]. Specifically, the pruning is applied to all convolution layers in VGG (except the first layer), while only applied to the first convolution layer of the normal residual blocks and the first two convolution layers of the bottleneck blocks in ResNet.

For CIFAR datasets, we first pretrain the models to have comparable accuracy to their original works, and then employ the progressive regularization. Afterwards, pruning and fine-tuning are performed. On ImageNet, due to the large scale dataset and the long training time, we directly employ our progressive scheme on the pretrained ResNet34 and ResNet50 model provided by PyTorch [47] with the MCP regularizer ($\gamma = 2$, $\lambda = 1$). All training setups (e.g., optimizer, batch size and learning rate) are kept the same as [23].

For fair comparisons, we also adopt the released pruning ratios from [23]. Namely, on CIFAR, uniform pruning is conducted with target pruning ratios ranging from 50% to 90%. While on ImageNet dataset, for comparable FLOPs drops, specified pruning ratios are applied.

In terms of our progressive regularization, we combine our approach with five regularizers, namely ℓ_p-norm, $t\ell_1$, $c\ell_1$, log-norm, and MCP. For each regularizer, we tune the scaling factor $\alpha \in \{10^{-2}, 10^{-1}, 10^0, 10^1, 10^2\}$ (as a baseline) and progressively increase α (for our approach). Moreover, as some regularizers have additional hyperparameters for modifying their shape, we additionally tune those in both approaches. The functional forms of regularizers and the tuning parameters are listed in Table I.

As evaluation metrics, three aspects are taken into account, namely, storage/memory, computing complexity, and model performance. For storage/memory, we consider the pruning

[2]https://github.com/IntelLabs/distiller

TABLE II
COMPRESSION AND TOP-1 ACCURACY (%) COMPARISON ON CIFAR-10 WITH RESNET56.

Pruning ratio			50 %	60 %	70 %	80 %	90 %
FLOPs / #Params			65.39M / 0.43M	52.88M / 0.35M	40.13M / 0.26M	27.60M / 0.18M	14.85M / 0.09M
ℓ_1 + one-shot [23]			92.97±0.15	92.31±0.23	91.88±0.09	91.03±0.14	87.34±0.21
ours	ℓ_p-norm	baseline	93.07±0.16	92.74±0.20	92.26±0.12	90.97±0.24	88.40±0.20
		progressive	93.52±0.06	93.26±0.09	92.75±0.06	91.47±0.11	89.42±0.16
	$t\ell_1$	baseline	93.13±0.14	92.79±0.11	92.32±0.21	91.32±0.23	88.73±0.13
		progressive	93.52±0.25	93.26±0.11	92.75±0.24	91.45±0.10	89.47±0.10
	$c\ell_1$	baseline	92.94±0.07	92.65±0.19	92.11±0.12	90.99±0.13	88.93±0.22
		progressive	93.64±0.13	93.01±0.08	92.80±0.05	91.43±0.14	89.66±0.25
	log-norm	baseline	93.04±0.24	92.84±0.13	92.18±0.14	91.33±0.23	88.68±0.13
		progressive	**93.69**±0.11	93.34±0.21	92.59±0.19	**91.66**±0.08	89.58±0.12
	MCP	baseline	93.22±0.19	92.92±0.20	92.23±0.11	91.37±0.19	88.33±0.12
		progressive	93.58±0.20	**93.41**±0.23	**92.81**±0.22	91.46±0.05	**89.65**±0.06
Greg [23]			93.06±0.09	92.77±0.12	92.23±0.21	91.39±0.17	89.49±0.23
Improvement over Greg			0.63	0.64	0.58	0.27	0.16

TABLE III
COMPRESSION AND TOP-1 ACCURACY (%) COMPARISON ON CIFAR-100 WITH VGG19.

Pruning ratio			50 %	60 %	70 %	80 %	90 %
FLOPs / #Params			111.29M / 5.05M	75.35M / 3.24M	45.97M / 1.83M	24.38M / 0.82M	9.35M / 0.21M
ℓ_1 + one-shot [23]			71.49±0.14	70.27±0.12	66.05±0.04	61.59±0.03	51.36±0.11
ours	ℓ_p-norm	baseline	71.32±0.16	70.14±0.14	66.51±0.11	61.94±0.10	52.39±0.22
		progressive	71.45±0.17	70.42±0.17	**67.50**±0.09	63.37±0.20	56.29±0.17
	$t\ell_1$	baseline	71.43±0.20	70.29±0.06	66.99±0.13	62.30±0.06	52.03±0.05
		progressive	71.62±0.05	70.34±0.23	67.37±0.03	63.46±0.05	56.15±0.04
	$c\ell_1$	baseline	71.32±0.17	70.04±0.06	66.21±0.22	61.72±0.09	51.85±0.21
		progressive	71.45±0.18	70.43±0.10	**67.50**±0.16	63.53±0.16	55.29±0.22
	log-norm	baseline	71.44±0.13	70.21±0.21	66.42±0.09	61.95±0.15	51.76±0.11
		progressive	71.57±0.17	**70.49**±0.04	67.29±0.09	63.39±0.22	56.43±0.12
	MCP	baseline	71.37±0.08	70.18±0.13	66.22±0.09	61.63±0.13	52.20±0.10
		progressive	**71.63**±0.12	70.42±0.18	67.39±0.13	**63.78**±0.06	**57.47**±0.06
Greg [23]			71.50±0.12	70.33±0.12	67.35±0.15	63.55±0.29	57.09±0.03
Improvement over Greg			0.13	0.16	0.15	0.23	0.38

ratio, sparsity, and number of parameters; for computing complexity, we consider the number of floating points operations (FLOPs); while for the model performance, we adopt the top-1 accuracy.

B. Result on CIFAR

The mean and standard deviation of the top-1 accuracy (in percentage) of three random runs are summarized in Table II (for CIFAR-10 with ResNet56) and Table III (for CIFAR-100 with VGG19). For each regularizer, we report the baseline (with best tuned α) and our approach (with progressive α). Moreover, we also report the results from a comparable work Greg [23] and ℓ_1 + one-shot as its baseline. For better comparison, the accuracy gain is shown in the last row of each table.

It can be seen from Table II and Table III that, pruning with progressive α can outperform tuning α in all cases of regularizers (at least for the α values evaluated). This supports our hypothesis that, through a progressively growing scaling factor α, the unimportant parameters can be pushed to zero earlier than important ones.

On CIFAR-10 with ResNet56, our progressive regularizer exhibits better accuracy in most cases compared to the SOTA Greg. For example, with $c\ell_1$ regularizer, all baselines perform worse than Greg, but with progressive scheme, the pruned models outperform Greg. This improvement is also evident with ℓ_p and log-norm regularizers. In most case with MCP and $t\ell_1$ regularizers, even our baselines can suppress Greg. For this, we speculate that, tuning of scaling factor and functional form of regularizers enhance the model performance. Nevertheless, even though these baselines are already strong, they can still be further enhanced by our progressive approach. The most significant improvement (accuracy gain) can reach 0.64%.

On CIFAR-100 with VGG19, similar phenomena can be observed with pruning ratio varying from 50% to 70%. However, at 80% and 90% pruning ratios, progressive regularization shows less superiority against Gerg. Surprisingly, however, the MCP regularizer brings in this case even a higher accuracy gain instead.

We also notice that, in most cases, pruning with MCP regularizers provide the best performances. Therefore, we take MCP regularizer with $\lambda = 1$ and $\gamma = 2$ in further experiment on the large scale ImageNet, as this function yields the best model performance (in terms of average accuracy over all experiments with CIFAR).

C. Result on ImageNet

The results on ImageNet with ResNet34 and ResNet50 are summarized in Table IV and Table V respectively. The methods for comparison include four SOTA pruning methods, namely, SFP [48], FPGM [49], Greg [23], CCEP [50]. For fair and

TABLE IV
FLOPs DROP (%) AND TOP-1 ACCURACY (%) COMPARISON ON IMAGENET WITH RESNET34.

Method	Baseline	Pruned	Acc ↓	FLOPs ↓
Greg-2	73.31	73.61	−0.30	24.24
ours	**73.29**	**73.68**	**−0.39**	**24.24**
CCEP-1	73.30	73.64	−0.34	25.44
SFP	73.92	71.83	2.09	41.10
CCEP-2	73.30	72.67	0.63	42.10
ours	**73.29**	**72.69**	**0.60**	**43.82**

TABLE V
FLOPs DROP (%) AND TOP-1 ACCURACY (%) COMPARISON ON IMAGENET WITH RESNET50.

Method	Baseline	Pruned	Acc ↓	FLOPs ↓
FPGM	76.15	74.83	1.32	54.00
CCEP-2	76.13	75.55	0.58	56.35
GReg-2	76.13	75.36	0.77	56.71
ours	**76.11**	**75.60**	**0.51**	**56.71**
CCEP-3	76.13	74.87	1.26	64.09
GReg-2	76.13	73.90	2.23	66.51
ours	**76.11**	**74.87**	**1.24**	**66.51**

easy comparison, the methods are ranked and grouped by their FLOPs drop percentage. All results are obtained directly from their original reports.

From Table IV, we can observe that, our approach outperforms Greg in terms of accuracy under same FLOPs drop on both ResNet34 and ResNet50. When compared to CCEP, although our method exhibits similar performances in terms of the top-1 accuracy, the accuracy drops due to the pruning are always lower than CCEP.

D. Experiment for $\nabla_{\max}$

We conduct a simple experiment on CIFAR-10 with ResNet56 to investigate the influence of $\nabla_{\max}$ on the accuracy of the DNNs. In the experiment, MCP regularizer with $\gamma = 2$ and $\lambda = 1$ is utilized, the model is uniformly pruned with 50% target sparsity, three $\nabla_{\max}$ values are tested, namely 0.25, 0.5, 1.0, and 2.0. The result (mean and standard deviation of top-1 accuracy) of three runs is reported in Table VI. We thus conclude that, the influence of $\nabla_{\max}$ on the result is not significant.

TABLE VI
COMPARISON FOR DIFFERENT $\nabla_{\max}$.

$\nabla_{\max}$	0.25	0.5	1.0	2.0
accuracy	$93.62_{\pm 0.17}$	$93.56_{\pm 0.21}$	$93.58_{\pm 0.20}$	$93.49_{\pm 0.24}$

E. Comparison of Global and Layer-wise Pruning

In our experiments, we also performed global pruning with independent learnable soft thresholds for each layer. Since the expressiveness of DNNs is partly determined by their architecture [37], we avoid pruning all filters of any layer by setting an upper limit of 95% pruning ratio for each layer. The pruning algorithm is described in Algorithm 3. Note that, the upper limit of pruning ratio is considered during the update of $\text{sparisty}^{(\text{global})}$.

Algorithm 3: Progressive Regularization for Global Pruning

Data: $\mathcal{D} = \{\mathbf{x}, \mathbf{y}\}$
Input : NN, L, R, P, S, optimizer
while $\text{sparsity}^{(\text{global})} < S$ **do**
  $\mathcal{L}_{\text{best}} \leftarrow \infty$
  patience $\leftarrow 0$
  while patience $< P$ **do**
    $\mathcal{L} \leftarrow L(\boldsymbol{w}, \mathcal{D}) + \alpha R(\boldsymbol{w})$
    NN, $\boldsymbol{t} \leftarrow \text{optimizer}(\mathcal{L}, \text{NN})$
    if $\mathcal{L} < \mathcal{L}_{\text{best}}$ **then**
      $\mathcal{L}_{\text{best}} \leftarrow \mathcal{L}$
      patience $\leftarrow 0$
    else
      patience $\leftarrow$ patience $+ 1$
    end
  end
  update $\text{sparsity}^{(\text{global})}$
  $\alpha \leftarrow \alpha + \Delta_\alpha$
end
pruning (NN, $\boldsymbol{t}$)
fine tune (NN)

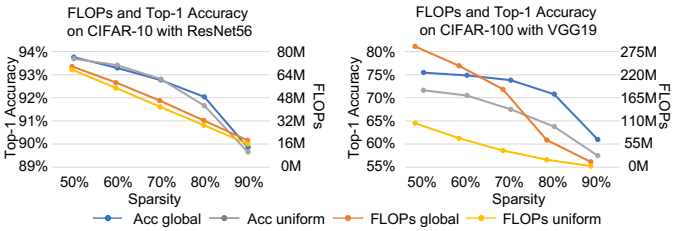

Fig. 3. FLOPs and top-1 accuracy of global and uniform pruning on ResNet56 and VGG19.

Figure 3 summarizes the FLOPs and top-1 accuracy on CIFAR-10 with ResNet56 and on CIFAR-100 with VGG19. It can be seen that, global pruning results generally both in higher FLOPs and higher accuracy.

a) VGG19: In Figure 4 we plot the FLOPs versus various filter pruning ratios of each layer in VGG19. Note that, unlike uniform pruning, the pruning ratios in global pruning is non-deterministic, therefore, we also report the standard deviation of three runs. It can be observed that, in VGG19 shallow layers tend to have higher FLOPs than deeper layers, however, with global pruning, the deeper layers are preferentially pruned, leading to much higher FLOPs than uniform pruning.

b) ResNet56: In Figure 5 we plot the FLOPs versus various filter pruning ratios of each layer in ResNet56. Different from VGG19, each layer in ResNet56 has similar FLOPs, therefore, even with global pruning, the FLOPs stay similar with that in uniform pruning.

Generally, the difference between global and layer-wise pruning is that, global pruning has much higher FLOPs given

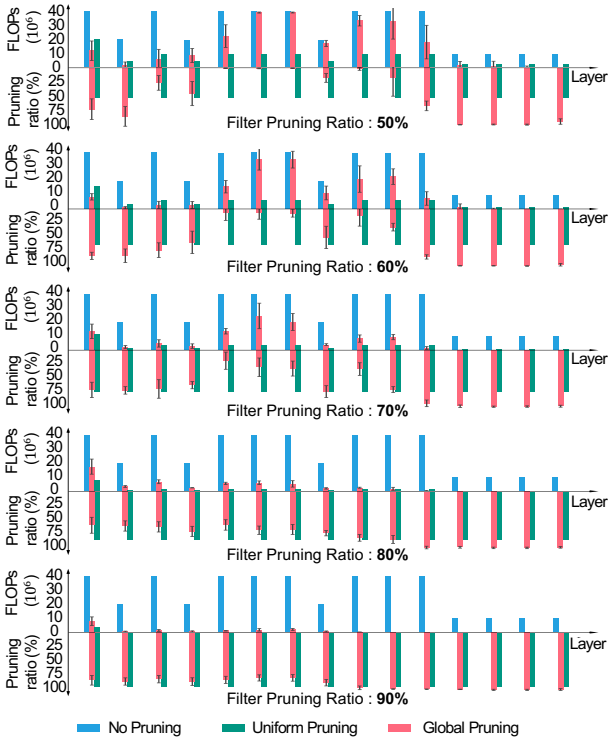

Fig. 4. Comparison of global and uniform pruning of VGG with different filter pruning ratios.

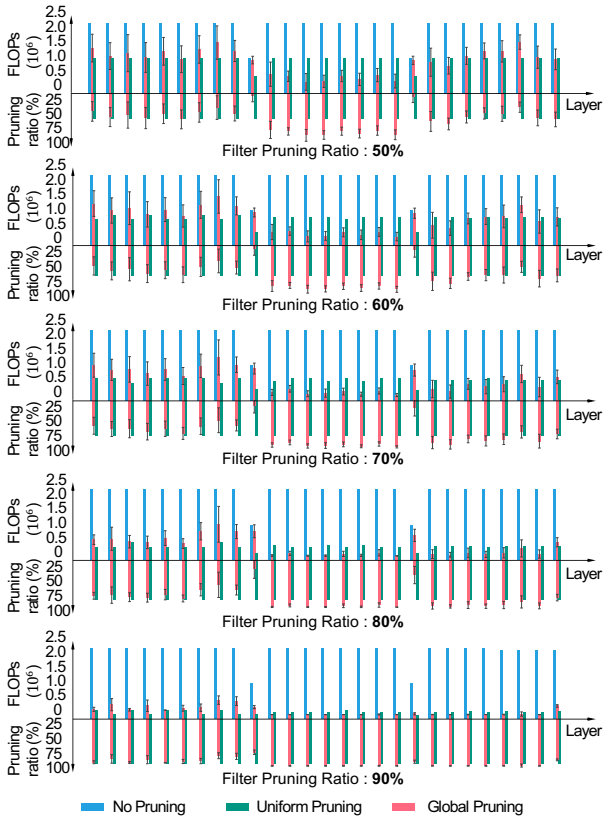

Fig. 5. Comparison of global and uniform pruning of ResNet56 with 50% filter pruning ratio.

the same sparsity. This is primarily because the parameters in the shallow layers, due to convolution with data with larger spatial size, are involved in more computations. Thus, they have a higher impact on the output than that in deep layers. Consequently, in global pruning, the parameters in deep layers are more likely to be pushed to smaller magnitudes by regularization or weight decay, and thus be pruned. In contrast, parameters in shallow layers, leading to more FLOPs, will be retained. This view was provided by the work of [39], and can be also supported by our experiments. This feature may guarantee that global pruning has a significantly better performance, but also higher computational cost at the same time. Therefore, global pruning is preferred in devices with limited storage/memory but high computational power.

V. Conclusion

In this paper, we propose a progressive regularizer for DNN pruning. The essential idea is to encourage more zero-valued parameters by a progressively increasing scaling factor of the regularizer. In this process, unimportant parameters are pushed to zero earlier (at a small scaling factor) than the important ones, and thus, can be removed with minimal impact on the output. To avoid large gradient, which may unstabilize the optimization process, due to the factor growing, we derived an update schedule for the scaling factor α. The initial plan was to prune only the zero-valued parameters, since the pruning of the zero-valued parameters has no effect on the output of the DNN. However, due to numerical computing, no parameters are exactly zero. To distinguish zero-valued parameters (which are usually small values) from small but informative parameters, we introduce a learnable soft threshold to optimally (w.r.t. the objective function) force non-informative parameters to zero. To make the regularization more stable against different model sizes and different functional forms, we suggest invariant regularizers. In this way, the balance between the regularization term and the task loss becomes more stable and robust. Experiments on multiple benchmark datasets and SOTA models prove that our progressive scheme outperforms the hyperparameter tuning scheme for all given α in the experiment. Furthermore, our method outperforms the compared SOTA methods with the same target sparsity (on CIFAR) and the same FLOPs drop (on ImageNet).

Acknowledgment

This work has been partially supported by the Carl-Zeiss-Foundation as part of "stay young with robots" (Jubot) project and the German Ministry of Research and Education as part of the SDIL (01IS19030A).

References

[1] C. Louizos, M. Welling, and D. P. Kingma, "Learning Sparse Neural Networks Through ℓ_0 Regularization," in *International Conference on Learning Representations*, 2018.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[3] Y. Zhou, H. Zhao, Y. Huang, T. Riedel, M. Hefenbrock, and M. Beigl, "Tinyhar: A Lightweight Deep Learning Model Designed for Human Activity Recognition," in *Proceedings of the 2022 ACM International Symposium on Wearable Computers*, 2022, pp. 89–93.

[4] R. Nakano, J. Hilton, S. Balaji, *et al.*, "WebGPT: Browser-assisted Question-Answering with Human Feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[5] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[6] Y. Zhou, T. King, Y. Huang, *et al.*, "Enhancing efficiency in har models: Nas meets pruning," in *22nd IEEE International Conference on Pervasive Computing and Communications (PerCom 2024)*, Institute of Electrical and Electronics Engineers (IEEE), 2024.

[7] S. Han, J. Pool, J. Tran, and W. Dally, "Learning Both Weights and Connections for Efficient Neural Network," *Advances in neural information processing systems*, vol. 28, 2015.

[8] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient Convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[9] H. Li, C. Ma, W. Xu, and X. Liu, "Feature Statistics Guided Efficient Filter Pruning," in *Proceedings of the 20th International Conference on International Joint Conferences on Artificial Intelligence (IJCAI)*, 2021, pp. 2619–2625.

[10] C. Yang and H. Liu, "Channel pruning based on convolutional neural network sensitivity," *Neurocomputing*, vol. 507, pp. 97–106, 2022.

[11] S. Verdenius, M. Stol, and P. Forré, "Pruning via iterative ranking of sensitivity statistics," *arXiv preprint arXiv:2006.00896*, 2020.

[12] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, "Layer-Adaptive Sparsity for the Magnitude-Based Pruning," in *International Conference on Learning Representations (ICLR)*, 2020.

[13] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning Efficient Convolutional Networks Through Network Slimming," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2736–2744.

[14] E. J. Candès and M. B. Wakin, "An Introduction to Compressive Sampling," *IEEE signal processing magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[15] D. L. Donoho and M. Elad, "Optimally Sparse Representation in General (Nonorthogonal) Dictionaries via ℓ_1 Minimization," *Proceedings of the National Academy of Sciences*, vol. 100, no. 5, pp. 2197–2202, 2003.

[16] Z. Xu, X. Chang, F. Xu, and H. Zhang, "$\ell_{1/2}$ Regularization: A Thresholding Representation Theory and A Fast Solver," *IEEE Transactions on neural networks and learning systems*, vol. 23, no. 7, pp. 1013–1027, 2012.

[17] R. Ma, J. Miao, L. Niu, and P. Zhang, "Transformed ℓ_1 Regularization for Learning Sparse Deep Neural Networks," *Neural Networks*, vol. 119, pp. 286–298, 2019.

[18] T. Zhang, "Multi-Stage Convex Relaxation for Learning with Sparse Regularization," *Advances in neural information processing systems*, vol. 21, 2008.

[19] J. Fan and R. Liu, "Variable Selection via Penalized Likelihood," 1999.

[20] C.-H. Zhang, "Nearly Unbiased Variable Selection Under Minimax Concave Penalty," *The Annals of statistics*, vol. 38, no. 2, pp. 894–942, 2010.

[21] R. Mazumder, J. H. Friedman, and T. Hastie, "SparseNet: Coordinate Descent with Nonconvex Penalties," *Journal of the American Statistical Association*, vol. 106, no. 495, pp. 1125–1138, 2011.

[22] S. Liu, Q. Feng, D. Eriksson, B. Letham, and E. Bakshy, "Sparse Bayesian Optimization," *arXiv preprint arXiv:2203.01900*, 2022.

[23] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Neural Pruning via Growing Regularization," in *International Conference on Learning Representations (ICLR)*, 2021.

[24] X. Ruan, Y. Liu, B. Li, C. Yuan, and W. Hu, "Dpfps: Dynamic and progressive filter pruning for compressing convolutional neural networks from scratch," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 2495–2503.

[25] Y. He, X. Dong, G. Kang, Y. Fu, and Y. Yang, "Progressive deep neural networks acceleration via soft filter pruning," *arXiv preprint arXiv:1808.07471*, vol. 1, no. 2, p. 8, 2018.

[26] H. Adeli and N.-T. Cheng, "Augmented Lagrangian Genetic Algorithm for Structural Optimization," *Journal of Aerospace Engineering*, vol. 7, no. 1, pp. 104–118, 1994.

[27] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the Exploding Gradient Problem," *CoRR, abs/1211.5063*, vol. 2, no. 417, p. 1, 2012.

[28] T. Van Laarhoven, "ℓ_2 Regularization Versus Batch and Weight Normalization," *arXiv preprint arXiv:1706.05350*, 2017.

[29] B. Bilgic, I. Chatnuntawech, A. P. Fan, *et al.*, "Fast Image Reconstruction with ℓ_2-regularization," *Journal of magnetic resonance imaging*, vol. 40, no. 1, pp. 181–191, 2014.

[30] R. C. Moore and J. DeNero, "ℓ_1 and ℓ_2 Regularization for Multiclass Hinge Loss Models," in *Symposium on Machine Learning in Speech and Natural Language Processing*, 2011.

[31] X. Ni, L. Fang, and H. Huttunen, "Adaptive ℓ_2 Regularization in Person Re-Identification," in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, 2021, pp. 9601–9607.

[32] F. Wen, L. Chu, P. Liu, and R. C. Qiu, "A Survey on Nonconvex Regularization-Based Sparse and Low-rank Recovery in Signal Processing, Statistics, and Machine Learning," *IEEE Access*, vol. 6, pp. 69 883–69 906, 2018.

[33] M. Kloft, U. Brefeld, S. Sonnenburg, and A. Zien, "ℓ_p-norm Multiple Kernel Learning," *The Journal of Machine Learning Research*, vol. 12, pp. 953–997, 2011.

[34] A. Kusupati, V. Ramanujan, R. Somani, *et al.*, "Soft Threshold Weight Reparameterization for Learnable Sparsity," in *International Conference on Machine Learning*, PMLR, 2020, pp. 5544–5555.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[36] J. Yoon and S. J. Hwang, "Combined Group and Exclusive Sparsity for Deep Neural Networks," in *International Conference on Machine Learning*, PMLR, 2017, pp. 3958–3966.

[37] D. Mittal, S. Bhardwaj, M. M. Khapra, and B. Ravindran, "Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2018, pp. 848–857.

[38] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the Lottery: Making All Tickets Winners," in *International Conference on Machine Learning*, PMLR, 2020, pp. 2943–2952.

[39] H. Mostafa and X. Wang, "Parameter Efficient Training of Deep Convolutional Neural Networks by Dynamic Sparse Reparameterization," in *International Conference on Machine Learning*, PMLR, 2019, pp. 4646–4655.

[40] K. Chandra, A. Xie, J. Ragan-Kelley, and E. Meijer, "Gradient Descent: The Ultimate Optimizer," in *Advances in Neural Information Processing Systems*.

[41] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, "Neural Network Distiller: A Python Package for DNN Compression Research," Oct. 2019.

[42] A. Krizhevsky *et al.*, "Learning Multiple Layers of Features from Tiny Images," 2009.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[44] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[46] T. Gale, E. Elsen, and S. Hooker, "The State of Sparsity in Deep Neural Networks," *arXiv preprint arXiv:1902.09574*, 2019.

[47] A. Paszke *et al.*, "Pytorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[48] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 2234–2240.

[49] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4340–4349.

[50] H. Shang, J.-L. Wu, W. Hong, and C. Qian, "Neural Network Pruning by Cooperative Coevolution," *arXiv preprint arXiv:2204.05639*, 2022.