A Proximal Algorithm for Network Slimming

Kevin Bui[1(✉)], Fanghui Xue[1], Fredrick Park[2], Yingyong Qi[1], and Jack Xin[1]

[1] University of California, Irvine 92697, CA, USA
{kevinb3,fanghuix,yqi,jack.xin}@uci.edu
[2] Whittier College, Whittier 90602, CA, USA
fpark1@whittier.edu

Abstract. As a popular channel pruning method for convolutional neural networks (CNNs), network slimming (NS) has a three-stage process: (1) it trains a CNN with ℓ_1 regularization applied to the scaling factors of the batch normalization layers; (2) it removes channels whose scaling factors are below a chosen threshold; and (3) it retrains the pruned model to recover the original accuracy. This time-consuming, three-step process is a result of using subgradient descent to train CNNs. Because subgradient descent does not exactly train CNNs towards sparse, accurate structures, the latter two steps are necessary. Moreover, subgradient descent does not have any convergence guarantee. Therefore, we develop an alternative algorithm called proximal NS. Our proposed algorithm trains CNNs towards sparse, accurate structures, so identifying a scaling factor threshold is unnecessary and fine tuning the pruned CNNs is optional. Using Kurdyka-Łojasiewicz assumptions, we establish global convergence of proximal NS. Lastly, we validate the efficacy of the proposed algorithm on VGGNet, DenseNet and ResNet on CIFAR 10/100. Our experiments demonstrate that after one round of training, proximal NS yields a CNN with competitive accuracy and compression.

Keywords: channel pruning · nonconvex optimization · convolutional neural networks · neural network compression

1 Introduction

In the past decade, convolutional neural networks (CNNs) have revolutionized computer vision in various applications, such as image classification [12,32,37] and object detection [10,16,26]. CNNs are able to internally generate diverse, various features through its multiple hidden layers, totaling millions of weight parameters to train and billions of floating point operations (FLOPs) to execute. Consequently, highly accurate CNNs are impractical to store and implement on resource-constrained devices, such as mobile smartphones.

The work was partially supported by NSF grants DMS-1854434, DMS-1952644, DMS-2151235, and a Qualcomm Faculty Award.

G. Nicosia et al. (Eds.): LOD 2023, LNCS 14506, pp. 69–83, 2024.
https://doi.org/10.1007/978-3-031-53966-4_6

To compress CNNs into lightweight models, several directions, including weight pruning [1,11], have been investigated. Channel pruning [23,33] is currently a popular direction because it can significantly reduce the number of weights needed in a CNN by removing any redundant channels. One straightforward approach to channel pruning is network slimming (NS) [23], which appends an ℓ_1 norm on the scaling factors of the batch normalization layers to the loss function being optimized. Being a sparse regularizer, the ℓ_1 norm pushes the scaling factors corresponding to the channels towards zeroes. The original optimization algorithm used for NS is subgradient descent [31], but it has theoretical and practical issues. Subgradient descent does not necessarily decrease the loss function value after each iteration, even when performed exactly with full batch of data [4]. Moreover, unless with some additional modifications, such as backtracking line search, subgradient descent may not converge to a critical point [25]. When implemented in practice, barely any of the scaling factors have values exactly at zeroes by the end of training, resulting in two issues. First, a threshold value needs to be determined in order to remove channels whose scaling factors are below it. Second, pruning channels with nonzero scaling factors can deteriorate the CNNs' accuracy since these channels are still relevant to the CNN computation. As a result, the pruned CNN needs to be retrained to recover its original accuracy. Therefore, as a suboptimal algorithm, subgradient descent leads to a time-consuming, three-step process.

In this paper, we design an alternative optimization algorithm based on proximal alternating linearized minimization (PALM) [5] for NS. The algorithm has more theoretical and practical advantages than subgradient descent. Under certain conditions, the proposed algorithm does converge to a critical point. When used in practice, the proposed algorithm enforces the scaling factors of insignificant channels to be exactly zero by the end of training. Hence, there is no need to set a scaling factor threshold to identify which channels to remove. Because the proposed algorithm trains a model towards a truly sparse structure, the model accuracy is preserved after the insignificant channels are pruned, so fine tuning is unnecessary. The only trade-off of the proposed algorithm is a slight decrease in accuracy compared to the original baseline model. Overall, the new algorithm reduces the original three-step process of NS to only one round of training with fine tuning as an optional step, thereby saving the time and hassle of obtaining a compressed, accurate CNN.

2 Related Works

Early pruning methods focus on removing redundant weight parameters in CNNs. Han *et al.*[11] proposed to remove weights if their magnitudes are below a certain threshold. Aghasi *et al.*[2] formulated a convex optimization problem to determine which weight parameters to retain while preserving model accuracy. Creating irregular sparsity patterns, weight pruning is not implementation friendly since it requires special software and hardware to accelerate inference [20,40].

An alternative to weight pruning is pruning group-wise structures in CNNs. Many works [3,8,19,24,29,33] have imposed group regularization onto various CNN structures, such as filters and channels. Li *et al.*[20] incorporated a sparsity-inducing matrix corresponding to each feature map and imposed row-wise and column-wise group regularization onto this matrix to determine which filters to remove. Lin *et al.*[21] pruned filters that generate low-rank feature maps. Hu *et al.*[13] devised network trimming that iteratively removes zero-activation neurons from the CNN and retrains the compressed CNN. Rather than regularizing the weight parameters, Liu *et al.*[23] developed NS, where they applied ℓ_1 regularization on the scaling factors in the batch normalization layers in a CNN to determine which of their corresponding channels are redundant to remove and then they retrained the pruned CNN to restore its accuracy. Bui *et al.*[6,7] investigated nonconvex regularizers as alternatives to the ℓ_1 regularizer for NS. On the other hand, Zhao *et al.*[40] applied probabilistic learning onto the scaling factors to identify which redundant channels to prune with minimal accuracy loss, making retraining unnecessary. Lin *et al.*[22] introduced an external soft mask as a set of parameters corresponding to the CNN structures (e.g., filters and channels) and regularized the mask by adversarial learning.

3 Proposed Algorithm

In this section, we develop a novel PALM algorithm [5] for NS that consists of two straightforward, general steps per epoch: stochastic gradient descent on the weight parameters, including the scaling factors of the batch normalization layers, and soft thresholding on the scaling factors.

3.1 Batch Normalization Layer

Most modern CNNs have batch normalization (BN) layers [17] because these layers speed up their convergence and improve their generalization [28]. These benefits are due to normalizing the output feature maps of the preceding convolutional layers using mini-batch statistics. Let $z \in \mathbb{R}^{B\times C\times H\times W}$ denote an output feature map, where B is the mini-batch size, C is the number of channels, and H and W are the height and width of the feature map, respectively. For each channel $i = 1, \ldots, C$, the output of a BN layer on each channel z_i is given by

$$z_i' = \gamma_i \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta_i, \tag{1}$$

where μ_B and σ_B are the mean and standard deviation of the inputs across the mini-batch B, ϵ is a small constant for numerical stability, and γ_i and β_i are trainable weight parameters that help restore the representative power of the input z_i. The weight parameter γ_i is defined to be the scaling factor of channel i. The scaling factor γ_i determines how important channel i is to the CNN computation as it is multiplied to all pixels of the same channel i within the feature map z.

3.2 Numerical Optimization

Let $\{(x_i, y_i)\}_{i=1}^N$ be a given dataset, where each x_i is a training input and y_i is its corresponding label or value. Using the dataset $\{(x_i, y_i)\}_{i=1}^N$, we train a CNN with c total channels, where each of their convolutional layers is followed by a BN layer. Let $\gamma \in \mathbb{R}^c$ be the vector of trainable scaling factors of the CNN, where for $i = 1, \ldots, c$, each entry γ_i is a scaling factor of channel i. Moreover, let $W \in \mathbb{R}^n$ be a vector of all n trainable weight parameters, excluding the scaling factors, in the CNN. NS [23] minimizes the following objective function:

$$\min_{W,\gamma} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(h(x_i, W, \gamma), y_i) + \lambda \|\gamma\|_1, \tag{2}$$

where $h(x_i, W, \gamma)$ is the output of the CNN predicted on the data point x_i; $\mathcal{L}(h(x_i, W, \gamma), y_i)$ is the loss function between the prediction $h(x_i, W, \gamma)$ and ground truth y_i, such as the cross-entropy loss function; and $\lambda > 0$ is the regularization parameter for the ℓ_1 penalty on the scaling factor vector γ. In [23], (2) is solved by a gradient descent scheme with step size δ^t for each epoch t:

$$W^{t+1} = W^t - \delta^t \nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t), \tag{3a}$$

$$\gamma^{t+1} = \gamma^t - \delta^t \left(\nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t) + \lambda \partial \|\gamma^t\|_1 \right), \tag{3b}$$

where $\tilde{\mathcal{L}}(W, \gamma) := \frac{1}{N} \sum_{i=1}^N \mathcal{L}(h(x_i, W, \gamma), y_i)$ and $\partial \|\cdot\|_1$ is the subgradient of the ℓ_1 norm.

By (3), we observe that γ is optimized by subgradient descent, which can lead to practical issues. When $\gamma_i = 0$ for some channel i, the subgradient needs to be chosen precisely. Not all subgradient vectors at a non-differentiable point decrease the value of (2) in each epoch [4], so we need to find one that does among the infinite number of choices. In the numerical implementation of NS[1], the subgradient ζ^t is selected such that $\zeta_i^t = 0$ by default when $\gamma_i^t = 0$, but such selection is not verified to decrease the value of (2) in each epoch t. Lastly, subgradient descent only pushes the scaling factors of irrelevant channels to be near zero in value but not exactly zero. For this reason, when pruning a CNN, the user needs to determine the appropriate scaling factor threshold to remove its channels where no layers have zero channels and then fine tune it to restore its original accuracy. However, if too many channels are pruned that the fine-tuned accuracy is significantly less than the original, the user may waste time and resources by iterating the process of decreasing the threshold and fine tuning until the CNN attains acceptable accuracy and compression.

To develop an alternative algorithm that does not possess the practical issues of subgradient descent, we reformulate (2) as a constrained optimization problem by introducing an auxiliary variable ξ, giving us

$$\min_{W,\gamma,\xi} \quad \tilde{\mathcal{L}}(W, \gamma) + \lambda \|\xi\|_1 \quad \text{s.t.} \quad \xi = \gamma. \tag{4}$$

[1] https://github.com/Eric-mingjie/network-slimming.

However, we relax the constraint by a quadratic penalty with parameter $\beta > 0$, leading to a new unconstrained optimization problem:

$$\min_{W,\gamma,\xi} \quad \tilde{\mathcal{L}}(W,\gamma) + \lambda\|\xi\|_1 + \frac{\beta}{2}\|\gamma - \xi\|_2^2. \tag{5}$$

In (2), the scaling factor vector γ is optimized for both model accuracy and sparsity, which can be difficult to balance when training a CNN. However, in (5), γ is optimized for only model accuracy because it is a variable of the overall loss function $\tilde{\mathcal{L}}(W,\gamma)$ while ξ is optimized only for sparsity because it is penalized by the ℓ_1 norm. The quadratic penalty enforces γ and ξ to be similar in values, thereby ensuring γ to be sparse.

Let (W,γ) be a concatenated vector of W and γ. We minimize (5) via alternating minimization, so for each epoch t, we solve the following subproblems:

$$(W^{t+1},\gamma^{t+1}) \in \arg\min_{W,\gamma} \tilde{\mathcal{L}}(W,\gamma) + \frac{\beta}{2}\|\gamma - \xi^t\|_2^2 \tag{6a}$$

$$\xi^{t+1} \in \arg\min_{\xi} \lambda\|\xi\|_1 + \frac{\beta}{2}\|\gamma^{t+1} - \xi\|_2^2. \tag{6b}$$

Below, we describe how to solve each subproblem in details.

(W,γ)-subproblem. The (W,γ)-subproblem given in (6a) cannot be solved in closed form because the loss function $\tilde{\mathcal{L}}(W,\gamma)$ is a composition of several nonlinear functions. Typically, when training a CNN, this subproblem would be solved by (stochastic) gradient descent. To formulate (6a) as a gradient descent step, we follow a prox-linear strategy as follows:

$$\begin{aligned}(W^{t+1},\gamma^{t+1}) \in \arg\min_{W,\gamma}\ & \tilde{\mathcal{L}}(W^t,\gamma^t) + \langle \nabla_W \tilde{\mathcal{L}}(W^t,\gamma^t), W - W^t\rangle \\ & + \langle \nabla_\gamma \tilde{\mathcal{L}}(W^t,\gamma^t), \gamma - \gamma^t\rangle + \frac{\alpha}{2}\|W - W^t\|_2^2 + \frac{\alpha}{2}\|\gamma - \gamma^t\|_2^2 + \frac{\beta}{2}\|\gamma - \xi^t\|_2^2,\end{aligned} \tag{7}$$

where $\alpha > 0$. By differentiating with respect to each variable, setting the partial derivative equal to zero, and solving for the variable, we have

$$W^{t+1} = W^t - \frac{1}{\alpha}\nabla_W \tilde{\mathcal{L}}(W^t,\gamma^t) \tag{8a}$$

$$\gamma^{t+1} = \frac{\alpha\gamma^t + \beta\xi^t}{\alpha+\beta} - \frac{1}{\alpha+\beta}\nabla_\gamma \tilde{\mathcal{L}}(W^t,\gamma^t). \tag{8b}$$

We see that (8a) is gradient descent on W^t with step size $\frac{1}{\alpha}$ while (8b) is gradient descent on a weighted average of γ^t and ξ^t with step size $\frac{1}{\alpha+\beta}$. These steps are straightforward to implement in practice when training a CNN because the gradient $(\nabla_W \tilde{\mathcal{L}}(W^t,\gamma^t), \nabla_\gamma \tilde{\mathcal{L}}(W^t,\gamma^t))$ can be approximated by backpropagation.

ξ-subproblem. To solve (6b), we perform a proximal update by minimizing the following subproblem:

$$\xi^{t+1} \in \arg\min_{\xi} \lambda\|\xi\|_1 + \frac{\alpha}{2}\|\xi - \xi^t\|_2^2 + \frac{\beta}{2}\|\gamma^{t+1} - \xi\|_2^2. \tag{9}$$

Algorithm 1. Proximal NS: proximal algorithm for minimizing (5)

Input: Regularization parameter λ, proximal parameter α, penalty parameter β
Initialize W^1, ξ^1 with random values.
Initialize γ^1 such that $\gamma_i = 0.5$ for each channel i.
1: **for** each epoch $t = 1, \ldots, T$ **do**
2: $W^{t+1} = W^t - \frac{1}{\alpha}\nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t)$ by stochastic gradient descent or variant.
3: $\gamma^{t+1} = \frac{\alpha\gamma^t + \beta\xi^t}{\alpha+\beta} - \frac{1}{\alpha+\beta}\nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t)$ by stochastic gradient descent or variant.
4: $\xi^{t+1} = \mathcal{S}\left(\frac{\alpha\xi^t + \beta\gamma^{t+1}}{\alpha+\beta}, \frac{\lambda}{\beta+\alpha}\right)$.
5: **end for**

Expanding it gives

$$\xi^{t+1} = \arg\min_{\xi} \|\xi\|_1 + \frac{1}{2\left(\frac{\lambda}{\beta+\alpha}\right)}\left\|\xi - \frac{\alpha\xi^t + \beta\gamma^{t+1}}{\alpha+\beta}\right\|_2^2 = \mathcal{S}\left(\frac{\alpha\xi^t + \beta\gamma^{t+1}}{\alpha+\beta}, \frac{\lambda}{\beta+\alpha}\right),$$

where $\mathcal{S}(x, \lambda)$ is the soft-thresholding operator defined by $(\mathcal{S}(x, \lambda))_i = \text{sign}(x_i)\max\{0, |x_i| - \lambda\}$ for each entry i. Therefore, ξ is updated by performing soft thresholding on the weighted average between ξ^t and γ^{t+1}.

We summarize the new algorithm for NS in Algorithm 1 as proximal NS.

4 Convergence Analysis

To establish global convergence of proximal NS, we present relevant definitions and assumptions.

Definition 1 ([5]). *A proper, lower-semicontinuous function $f : \mathbb{R}^m \to (-\infty, \infty]$ satisfies the Kurdyka-Łojasiewicz (KL) property at a point $\bar{x} \in dom(\partial f) := \{x \in \mathbb{R}^m : \partial f(x) \neq \varnothing\}$ if there exist $\eta \in (0, +\infty]$, a neighborhood U of $\bar{x}$, and a continuous concave function $\phi : [0, \eta) \to [0, \infty)$ with the following properties: (i) $\phi(0) = 0$; (ii) ϕ is continuously differentiable on $(0, \eta)$; (iii) $\phi'(x) > 0$ for all $x \in (0, \eta)$; and (iv) for any $x \in U$ with $f(\bar{x}) < f(x) < f(\bar{x}) + \eta$, it holds that $\phi'(f(x) - f(\bar{x}))dist(0, \partial f(x)) \geq 1$. If f satisfies the KL property at every point $x \in dom(\partial f)$, then f is called a KL function.*

Assumption 1. *Suppose that*

a) $\tilde{\mathcal{L}}(W, \gamma)$ is a proper, differentiable, and nonnegative function.
b) $\nabla\tilde{\mathcal{L}}(W, \gamma)$ is Lipschitz continuous with constant L.
c) $\tilde{\mathcal{L}}(W, \gamma)$ is a KL function.

Remark 1. Assumption 1 (a)-(b) are common in nonconvex analysis (e.g., [5]). For Assumption 1, most commonly used loss functions for CNNs are verified to be KL functions [38]. Some CNN architectures do not satisfy Assumption 1(a) when they contain nonsmooth functions and operations, such as the ReLU activation functions and max poolings. However, these functions and operations can be replaced with their smooth approximations. For example, the smooth approximation of ReLU is the softplus function $\frac{1}{c}\log(1 + \exp(cx))$ for some parameter

$c > 0$ while the smooth approximation of the max function for max pooling is the softmax function $\sum_{i=1}^{n} \frac{x_i e^{cx_i}}{\sum_{i=1}^{n} e^{cx_i}}$ for some parameter $c > 0$. Besides, Fu *et al.*[9] made a similar assumption to establish convergence for their algorithm designed for weight and filter pruning. Regardless, our numerical experiments demonstrate that our proposed algorithm still converges for CNNs containing ReLU activation functions and max pooling.

For brevity, we denote

$$F(W, \gamma, \xi) \coloneqq \tilde{\mathcal{L}}(W, \gamma) + \lambda \|\xi\|_1 + \frac{\beta}{2} \|\gamma - \xi\|_2^2.$$

Now, we are ready to present the main theorem:

Theorem 1. *Under Assumption 1, if* $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ *generated by Algorithm 1 is bounded and we have* $\alpha > L$*, then* $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ *converges to a critical point* (W^*, γ^*, ξ^*) *of* F*.*

The proof is delayed to the appendix. It requires satisfying the sufficient decrease property in F and the relative error property of ∂F [5].

5 Numerical Experiments

We evaluate proximal NS on VGG-19 [32], DenseNet-40 [14,15], and ResNet-110/164 [12] trained on CIFAR 10/100 [18]. The CIFAR 10/100 dataset [18] consists of 60,000 natural images of resolution 32×32 with 10/100 categories. The dataset is split into two sets: 50,000 training images and 10,000 test images. As done in recent works [12,23], standard augmentation techniques (e.g., shifting, mirroring, and normalization) are applied to the images before training and testing. The code for proximal NS is available at https://github.com/kbui1993/Official-Proximal-Network-Slimming.

5.1 Implementation Details

For CIFAR 10/100, the implementation is mostly the same as in [23]. Specifically, we train the networks from scratch for 160 epochs using stochastic gradient descent with initial learning rate at 0.1 that reduces by a factor of 10 at the 80th and 120th epochs. Moreover, the models are trained with weight decay 10^{-4} and Nesterov momentum of 0.9 without damping. The training batch size is 64. However, the parameter λ is set differently. In our numerical experiments, using Algorithm 1, we set $\xi \sim \text{Unif}[0.47, 0.50]$ for all networks while $\lambda = 0.0045$ and $\beta = 100$ for VGG-19, $\lambda = 0.004$ and $\beta = 100$ for DenseNet-40, and $\lambda = 0.002$ and $\beta = 1.0, 0.25$ for ResNet-110 and ResNet-164, respectively. We have initially $\alpha = 10$, the reciprocal of the learning rate, and it changes accordingly to the learning rate schedule. A model is trained five times on NVIDIA GeForce RTX 2080 for each network and dataset to obtain the average statistics.

Table 1. The average number of scaling factors equal to zero at the end of training. Each architecture is trained five times per dataset.

Architecture	Total Channels/γ_i	CIFAR 10	CIFAR 100
		Avg. Number of $\gamma_i = 0$	Avg. Number of $\gamma_i = 0$
VGG-19	5504	4105.2	3057.0
DenseNet-40	9360	6936.4	6071.6
ResNet-164	12112	8765.4	7115.8

5.2 Results

We apply proximal NS to train VGG-19, DenseNet-40, and ResNet-164 on CIFAR 10/100. According to Table 1, proximal NS drives a significant number of scaling factors to be exactly zeroes for each trained CNN. In particular, for VGG-19 and DenseNet-40, at least 55% of the scaling factors are zeroes while for ResNet-164, at least 58% are zeroes. We can safely remove the channels with zero scaling factors because they are unnecessary for inference. Unlike the original NS [23], proximal NS does not require us to select a scaling factor threshold based on how many channels to remove and how much accuracy to sacrifice.

We compare proximal NS with the original NS [23] and variational CNN pruning (VCP) [40], a Bayesian version of NS. To evaluate the effect of regularization and pruning on accuracy, we include the baseline accuracy, where the architecture is trained without any regularization on the scaling factors. For completeness, the models trained with original NS and proximal NS are fine tuned with the same setting as the first time training but without ℓ_1 regularization on the scaling factors. The results are reported in Tables 2a–2b.

After the first round of training, proximal NS outperforms both the original NS and VCP in test accuracy while reducing a significant amount of parameters and FLOPs. Because proximal NS trains a model towards a sparse structure, the model accuracy is less than the baseline accuracy by at most 1.56% and it remains the same between before and after pruning, a property that the original NS does not have. Although VCP is designed to preserve test accuracy after pruning, it does not compress as well as proximal NS for all architectures. With about the same proportion of channels pruned as the original NS, proximal NS saves more FLOPs for both VGG-19 and ResNet-164 and generally more parameters for all networks.

To potentially improve test accuracy, the pruned models from the original and proximal NS are fine tuned. For proximal NS, test accuracy of the pruned models improve slightly by at most 0.42% for DenseNet-40 and ResNet-164 while worsen for VGGNet-19. Moreover, proximal NS is outperformed by the original NS in fine-tuned test accuracy for all models trained on CIFAR 100.

A more accurate model from original NS might be preferable. However, the additional fine tuning step requires a few more training hours to obtain an accuracy that is up to 1.5% higher than the accuracy of a pruned model trained once by proximal NS. For example, for ResNet-164 trained on CIFAR 100, proximal NS takes about 7 h to attain an average accuracy of 75.26% while the original

Table 2. Results between the different NS methods on CIFAR 10/100. Average statistics are obtained by training the baseline architectures and original NS five times, while the results for variational NS are originally reported from [40].

(a) CIFAR 10

Architecture	Method	Avg. Training Time per Epoch (s) Pre-Pruned/Fine Tuned	% Channels Pruned	% Param. Pruned	% FLOPS Pruned	Test Accuracy (%) Post Pruned/Fine Tuned
VGG-19	Baseline	38.10/—	N/A	N/A	N/A	93.83/—
	Original NS [23]	40.39/29.40	74.00*	90.22	54.67	10.00/93.81
	Proximal NS (ours)	42.71/30.39	74.59	**91.17**	**57.54**	93.71/93.38
DenseNet-40	Baseline	117.45/—	N/A	N/A	N/A	94.25/—
	Original NS [23]	119.49/74.45	74.01	67.13	**60.46**	41.46/93.94
	VCP [40]	Not Reported	60.00	59.67	44.78	93.16/—
	Proximal NS (ours)	118.86/76.10	74.11	**67.75**	57.35	93.58/93.64
ResNet-164	Baseline	146.41/—	N/A	N/A	N/A	94.75/—
	Original NS [23]	151.62/112.80	71.98	52.95	59.27	16.61/93.21
	VCP [40]	Not Reported	74.00	56.70	49.08	93.16/—
	Proximal NS (ours)	150.13/114.26	72.37	**65.84**	**63.54**	93.19/93.41

(b) CIFAR 100

Architecture	Method	Avg. Training Time per Epoch (s) Pre-Pruned/Fine Tuned	% Channels Pruned	% Param. Pruned	% FLOPS Pruned	Test Accuracy (%) Post Pruned/Fine Tuned
VGG-19	Baseline	37.83	N/A	N/A	N/A	72.73/—
	Original NS [23]	39.98/30.74	55.00	78.53	38.66	1.00/72.91
	Proximal NS (ours)	42.31/30.04	55.54	**79.62**	**41.17**	72.81/72.70
DenseNet-40	Baseline	117.17	N/A	N/A	N/A	74.55/—
	Original NS [23]	119.32/77.95	65.01	**59.29**	**52.61**	25.96/74.50
	VCP [40]	Not Reported	37.00	37.73	22.67	72.19/—
	Proximal NS (ours)	120.89/82.92	64.87	59.15	45.00	73.70/73.98
ResNet-164	Baseline	145.37	N/A	N/A	N/A	76.79/—
	Original NS [23]	150.65/115.95	59.00	26.66	45.17	2.39/76.68
	VCP [40]	Not Reported	47.00	17.59	27.16	73.76/—
	Proximal NS (ours)	149.15/117.88	58.75	**42.28**	**47.93**	75.26/75.68

* This is the maximum possible for all five networks to remain functional for inference.

NS requires about 12 h to achieve 1.42% higher accuracy. Therefore, the amount of time and resources spent training for an incremental improvement may not be worthwhile.

Finally, we compare proximal NS with other pruning methods applied to Densenet-40 and ResNet-110 trained on CIFAR 10. The other pruning methods, which may require fine tuning, are L1 [19], GAL [22], and Hrank [21]. For DenseNet-40, proximal NS prunes the most parameters and the second most FLOPs while having comparable accuracy as the fine-tuned Hrank and post-pruned GAL-0.05. For ResNet-110, proximal NS has better compression than L1, GAL-0.5, and Hrank with its post-pruned accuracy better than GAL-0.5's fine-tuned accuracy and similar to L1's fine-tuned accuracy. Although GAL or Hrank might be advantageous to use to obtain a sparse, accurate CNN, they have additional requirements besides fine tuning. GAL [22] requires an accurate baseline model available for knowledge distillation. For Hrank [21], the compression ratio needs to be specified for each convolutional layer, thereby making hyperparameter tuning more complicated (Table 3).

Table 3. Comparison of Proximal NS with other pruning methods on CIFAR 10.

Architecture	Method	% Param./FLOPs Pruned	Test Accuracy (%) Post Pruned/Fine Tuned
DenseNet-40	Hrank [21]	53.80/61.00	—-/93.68
	GAL-0.05 [22]	56.70/54.70	93.53/94.50
	Proximal NS (Ours)	67.75/57.54	93.58/93.64
ResNet-110	L1 [19]	32.60/38.70	—-/93.30
	GAL-0.5 [22]	44.80/48.50	92.55/92.74
	Hrank [21]	39.40/41.20	—-/94.23
	Proximal NS (Ours)	50.70/48.54	93.25/93.27

Overall, proximal NS is a straightforward algorithm that yields a generally more compressed and accurate model than the other methods in one training round. Although its test accuracy after one round is slightly lower than the baseline accuracy, it is expected because of the sparsity–accuracy trade-off and being a prune-while-training algorithm (which automatically identifies the insignificant channels during training) as discussed in [30]. Lastly, the experiments show that fine tuning the compressed models trained by proximal NS marginally improves the test accuracy, which makes fine tuning wasteful.

6 Conclusion

We develop a channel pruning algorithm called proximal NS with global convergence guarantee. It trains a CNN towards a sparse, accurate structure, making fine tuning optional. In our experiments, proximal NS can effectively compress CNNs with accuracy slightly less than the baseline. Because fine tuning CNNs trained by proximal NS marginally improves test accuracy, we will investigate modifying the algorithm to attain significantly better fine-tuned accuracy.

For future direction, we shall study proximal cooperative neural architecture search [34,35] and include nonconvex, sparse regularizers, such as $\ell_1 - \ell_2$ [36] and transformed ℓ_1 [39].

A Appendix

First, we introduce important definitions and lemmas from variational analysis.

Definition 2 ([27]). *Let $f : \mathbb{R}^n \to (-\infty, +\infty]$ be a proper and lower semicontinuouous function.*

(a) The Fréchet subdifferential of f at the point $x \in domf := \{x \in \mathbb{R}^n : f(x) < \infty\}$ is the set

$$\hat{\partial} f(x) = \left\{ v \in \mathbb{R}^{n^2} : \liminf_{y \neq x, y \to x} \frac{f(y) - f(x) - \langle v, y - x \rangle}{\|y - x\|} \geq 0 \right\}.$$

(b) The limiting subdifferential of f at the point $x \in domf$ is the set

$$\partial f(x) = \left\{ v \in \mathbb{R}^{n^2} : \exists \{(x^t, y^t)\}_{t=1}^{\infty} \quad s.t. \quad x^t \to x, f(x^t) \to f(x), \hat{\partial} f(x^t) \ni y^t \to y \right\}.$$

Lemma 1 (Strong Convexity Lemma [4]**).** *A function $f(x)$ is called strongly convex with parameter μ if and only if one of the following conditions holds:*

a) $g(x) = f(x) - \frac{\mu}{2}\|x\|_2^2$ is convex.
b) $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|_2^2, \ \forall x, y.$

Lemma 2 (Descent Lemma [4]**).** *If $\nabla f(x)$ is Lipschitz continuous with parameter $L > 0$, then*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|x - y\|_2^2, \ \forall x, y.$$

For brevity, denote $\tilde{W} \coloneqq (W, \gamma)$, the overall set of weights in a CNN, and $Z \coloneqq (\tilde{W}, \xi) = (W, \gamma, \xi)$. Before proving Theorem 1, we prove some necessary lemmas.

Lemma 3 (Sufficient Decrease). *Let $\{Z^t\}_{t=1}^{\infty}$ be a sequence generated by Algorithm 1. Under Assumption 1, we have*

$$F(Z^{t+1}) - F(Z^t) \leq \frac{L - \alpha}{2}\|Z^{t+1} - Z^t\|_2^2. \tag{10}$$

for all $t \in \mathbb{N}$. In addition, when $\alpha > L$, we have

$$\sum_{t=1}^{\infty} \|Z^{t+1} - Z^t\|_2^2 < \infty. \tag{11}$$

Proof. First we define the function

$$L_t(\tilde{W}) = \tilde{\mathcal{L}}(\tilde{W}^t) + \langle \nabla \tilde{\mathcal{L}}(\tilde{W}^t), \tilde{W} - \tilde{W}^t \rangle + \frac{\alpha}{2}\|\tilde{W} - \tilde{W}^t\|_2^2 + \frac{\beta}{2}\|\gamma - \xi^t\|_2^2. \tag{12}$$

We observe that L_t is strongly convex with respect to $\tilde{W}$ with parameter α. Because $\nabla L_t(\tilde{W}^{t+1}) = 0$ by (7), we use Lemma 1 to obtain

$$\begin{aligned} L_t(\tilde{W}^t) &\geq L_t(\tilde{W}^{t+1}) + \langle \nabla L_t(\tilde{W}^{t+1}), \tilde{W}^t - \tilde{W}^{t+1} \rangle + \frac{\alpha}{2}\|\tilde{W}^{t+1} - \tilde{W}^t\|_2^2 \\ &\geq L_t(\tilde{W}^{t+1}) + \frac{\alpha}{2}\|\tilde{W}^{t+1} - \tilde{W}^t\|_2^2, \end{aligned} \tag{13}$$

which simplifies to

$$\begin{aligned} \tilde{\mathcal{L}}(\tilde{W}^t) + \frac{\beta}{2}\|\gamma^t - \xi^t\|_2^2 - \alpha\|\tilde{W}^{t+1} - \tilde{W}^t\|_2^2 \geq & \tilde{\mathcal{L}}(\tilde{W}^t) + \langle \nabla \tilde{\mathcal{L}}(\tilde{W}^t), \tilde{W}^{t+1} - \tilde{W}^t \rangle \\ & + \frac{\beta}{2}\|\gamma^{t+1} - \xi^t\|_2^2. \end{aligned} \tag{14}$$

Since $\nabla\tilde{\mathcal{L}}(\tilde{W})$ is Lipschitz continuous with constant L, we have

$$\tilde{\mathcal{L}}(\tilde{W}^{t+1}) \leq \tilde{\mathcal{L}}(\tilde{W}^t) + \langle \nabla\tilde{\mathcal{L}}(\tilde{W}^t), \tilde{W}^{t+1} - \tilde{W}^t \rangle + \frac{L}{2}\|\tilde{W}^{t+1} - \tilde{W}^t\|_2^2 \tag{15}$$

by Lemma 2. Combining the previous two inequalities gives us

$$\tilde{\mathcal{L}}(\tilde{W}^t) + \frac{\beta}{2}\|\gamma^t - \xi^t\|_2^2 + \frac{L-2\alpha}{2}\|\tilde{W}^{t+1} - \tilde{W}^t\|_2^2 \geq \tilde{\mathcal{L}}(\tilde{W}^{t+1}) + \frac{\beta}{2}\|\gamma^{t+1} - \xi^t\|_2^2.$$

Adding the term $\lambda\|\xi^t\|_1$ on both sides and rearranging the inequality give us

$$F(\tilde{W}^{t+1}, \xi^t) - F(Z^t) \leq \frac{L-2\alpha}{2}\|\tilde{W}^{t+1} - \tilde{W}^t\|_2^2 \tag{16}$$

By (9), we have

$$\lambda\|\xi^{t+1}\|_1 + \frac{\beta}{2}\|\gamma^{t+1} - \xi^{t+1}\|_2^2 + \frac{\alpha}{2}\|\xi^{t+1} - \xi^t\|_2^2 \leq \lambda\|\xi^t\|_1 + \frac{\beta}{2}\|\gamma^{t+1} - \xi^t\|_2^2.$$

Adding $\tilde{\mathcal{L}}(\tilde{W}^{t+1})$ on both sides and rearranging the inequality give

$$F(Z^{t+1}) - F(\tilde{W}^{t+1}, \xi^t) \leq -\frac{\alpha}{2}\|\xi^{t+1} - \xi^t\|_2^2 \tag{17}$$

Summing up (16) and (17) and rearranging them, we have

$$F(Z^{t+1}) - F(Z^t) \leq \frac{L-2\alpha}{2}\|\tilde{W}^{t+1} - \tilde{W}^t\|_2^2 - \frac{\alpha}{2}\|\xi^{t+1} - \xi^t\|_2^2 \leq \frac{L-\alpha}{2}\|Z^{t+1} - Z^t\|_2^2. \tag{18}$$

Summing up the inequality for $t = 1, \ldots, N-1$, we have

$$\sum_{t=1}^{N-1} \frac{\alpha - L}{2}\|Z^{t+1} - Z^t\|_2^2 \leq F(Z^1) - F(Z^N) \leq F(Z^1).$$

Because $\alpha > L$, the left-hand side is nonnegative, so as $N \to \infty$, we have (11).

Lemma 4 (Relative error property). *Let $\{Z^t\}_{t=1}^{\infty}$ be a sequence generated by Algorithm 1. Under Assumption 1, for any $t \in \mathbb{N}$, there exists some $w^{t+1} \in \partial F(Z^{t+1})$ such that*

$$\|w^{t+1}\|_2 \leq (3\alpha + 2L + \beta)\left\|Z^{t+1} - Z^t\right\|_2. \tag{19}$$

Proof. We note that

$$\nabla_W \tilde{\mathcal{L}}(\tilde{W}^{t+1}) \in \partial_W F(Z^{t+1}), \tag{20a}$$

$$\nabla_\gamma \tilde{\mathcal{L}}(\tilde{W}^{t+1}) + \beta(\gamma^{t+1} - \xi^{t+1}) \in \partial_\gamma F(Z^{t+1}), \tag{20b}$$

$$\lambda\partial_\xi\|\xi^{t+1}\|_1 - \beta(\gamma^{t+1} - \xi^{t+1}) \in \partial_\xi F(Z^{t+1}). \tag{20c}$$

By the first-order optimality conditions of (7) and (9), we obtain

$$\nabla_W \tilde{\mathcal{L}}(\tilde{W}^t) + \alpha(W^{t+1} - W^t) = 0, \tag{21a}$$

$$\nabla_\gamma \tilde{\mathcal{L}}(\tilde{W}^t) + \alpha(\gamma^{t+1} - \gamma^t) + \beta(\gamma^{t+1} - \xi^t) = 0, \tag{21b}$$

$$\lambda \partial_\xi \|\xi^{t+1}\|_1 + \alpha(\xi^{t+1} - \xi^t) - \beta(\gamma^{t+1} - \xi^{t+1}) \ni 0. \tag{21c}$$

Combining (20a) and (21a), (20b) and (21b), and (20c) and (21c), we obtain

$$\nabla_W \tilde{\mathcal{L}}(\tilde{W}^{t+1}) - \nabla_W \tilde{\mathcal{L}}(\tilde{W}^t) - \alpha(W^{t+1} - W^t) = w_1^{t+1} \in \partial_W F(Z^{t+1}), \tag{22a}$$

$$\nabla_\gamma \tilde{\mathcal{L}}(\tilde{W}^{t+1}) - \nabla_\gamma \tilde{\mathcal{L}}(\tilde{W}^t) - \alpha(\gamma^{t+1} - \gamma^t) - \beta(\xi^{t+1} - \xi^t) = w_2^{t+1} \in \partial_\gamma F(Z^{t+1}), \tag{22b}$$

$$-\alpha(\xi^{t+1} - \xi^t) = w_3^{t+1} \in \partial_\xi F(Z^{t+1}), \tag{22c}$$

where $w^{t+1} = (w_1^{t+1}, w_2^{t+1}, w_3^{t+1}) \in \partial F(Z^{t+1})$. As a result, by triangle inequality and Lipschitz continuity of $\nabla\tilde{\mathcal{L}}$, we have

$$\begin{aligned}\|w_1^{t+1}\|_2 &\le \alpha\|W^{t+1} - W^t\|_2 + \|\nabla_W \tilde{\mathcal{L}}(\tilde{W}^{t+1}) - \nabla_W \tilde{\mathcal{L}}(\tilde{W}^t)\|_2 \\ &\le \alpha\|W^{t+1} - W^t\| + L\|\tilde{W}^{t+1} - \tilde{W}^t\|_2 \le (\alpha + L)\|Z^{t+1} - Z^t\|_2,\end{aligned}$$

$$\begin{aligned}\|w_2^{t+1}\|_2 &\le \alpha\|\gamma^{t+1} - \gamma^t\|_2 + \beta\|\xi^{t+1} - \xi^t\|_2 + \|\nabla_\gamma \tilde{\mathcal{L}}(\tilde{W}^{t+1}) - \nabla_\gamma \tilde{\mathcal{L}}(\tilde{W}^t)\|_2 \\ &\le (\alpha + L)\|\tilde{W}^{t+1} - \tilde{W}^t\|_2 + \beta\|\xi^{t+1} - \xi^t\|_2 \le (\alpha + \beta + L)\|Z^{t+1} - Z^t\|_2,\end{aligned}$$

and

$$\|w_3^{t+1}\|_2 \le \alpha\|\xi^{t+1} - \xi^t\|_2 \le \alpha\|Z^{t+1} - Z^t\|_2.$$

Therefore, for all $t \in \mathbb{N}$, we have

$$\|w^{t+1}\|_2 \le \|w_1^{t+1}\|_2 + \|w_2^{t+1}\|_2 + \|w_3^{t+1}\|_2 \le (3\alpha + 2L + \beta)\left\|Z^{t+1} - Z^t\right\|_2.$$

Proof (Proof of Theorem 1). The result follows from Lemmas 3–4 combined with [5, Theorem 1]

References

1. Aghasi, A., Abdi, A., Nguyen, N., Romberg, J.: Net-trim: convex pruning of deep neural networks with performance guarantee. In: Advances in Neural Information Processing Systems, pp. 3177–3186 (2017)
2. Aghasi, A., Abdi, A., Romberg, J.: Fast convex pruning of deep neural networks. SIAM J. Math. Data Sci. **2**(1), 158–188 (2020)
3. Alvarez, J.M., Salzmann, M.: Learning the number of neurons in deep networks. In: Advances in Neural Information Processing Systems, pp. 2270–2278 (2016)
4. Beck, A.: First-order methods in optimization. SIAM (2017)
5. Bolte, J., Sabach, S., Teboulle, M.: Proximal alternating linearized minimization for nonconvex and nonsmooth problems. Math. Program. **146**(1), 459–494 (2014)

6. Bui, K., Park, F., Zhang, S., Qi, Y., Xin, J.: Nonconvex regularization for network slimming: compressing CNNs even more. In: Bebis, G., et al. (eds.) Advances in Visual Computing. ISVC 2020. LNCS, vol. 12509, pp. 39–53. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64556-4_4
7. Bui, K., Park, F., Zhang, S., Qi, Y., Xin, J.: Improving network slimming with nonconvex regularization. IEEE Access **9**, 115292–115314 (2021)
8. Bui, K., Park, F., Zhang, S., Qi, Y., Xin, J.: Structured sparsity of convolutional neural networks via nonconvex sparse group regularization. Front. Appl. Math. Stat. (2021)
9. Fu, Y., et al.: Exploring structural sparsity of deep networks via inverse scale spaces. IEEE Trans. Pattern Anal. Mach. Intell. (2022)
10. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
11. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
13. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: a data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
14. Huang, G., Liu, Z., Pleiss, G., Van Der Maaten, L., Weinberger, K.: Convolutional networks with dense connectivity. IEEE Trans. Pattern Anal. Mach. Intell. (2019)
15. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700–4708 (2017)
16. Huang, J., et al.: Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7310–7311 (2017)
17. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015)
18. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Technical report, University of Toronto (2009)
19. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
20. Li, Y., Gu, S., Mayer, C., Gool, L.V., Timofte, R.: Group sparsity: the hinge between filter pruning and decomposition for network compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020
21. Lin, M., et al.: Hrank: filter pruning using high-rank feature map. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1529–1538 (2020)
22. Lin, S., et al.: Towards optimal structured CNN pruning via generative adversarial learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2790–2799 (2019)
23. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2736–2744 (2017)

24. Meng, F., et al.: Pruning filter in filter. Adv. Neural Inf. Process. Syst. **33**, 17629–17640 (2020)
25. Noll, D.: Convergence of non-smooth descent methods using the Kurdyka-Łojasiewicz inequality. J. Optim. Theory Appl. **160**(2), 553–572 (2014)
26. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, pp. 91–99 (2015)
27. Rockafellar, R.T., Wets, R.J.B.: Variational Analysis, vol. 317. Springer Science & Business Media, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02431-3
28. Santurkar, S., Tsipras, D., Ilyas, A., Madry, A.: How does batch normalization help optimization? Adv. Neural Inf. Process. Syst. **31** (2018)
29. Scardapane, S., Comminiello, D., Hussain, A., Uncini, A.: Group sparse regularization for deep neural networks. Neurocomputing **241**, 81–89 (2017)
30. Shen, M., Molchanov, P., Yin, H., Alvarez, J.M.: When to prune? A policy towards early structural pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12247–12256 (2022)
31. Shor, N.Z.: Minimization Methods for Non-Differentiable Functions, vol. 3. Springer Science & Business Media, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-82118-9
32. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
33. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 2074–2082 (2016)
34. Xue, F., Qi, Y., Xin, J.: RARTS: an efficient first-order relaxed architecture search method. IEEE Access **10**, 65901–65912 (2022)
35. Xue, F., Xin, J.: Network compression via cooperative architecture search and distillation. In: IEEE International Conference on AI for Industries, pp. 42–43 (2021)
36. Yin, P., Lou, Y., He, Q., Xin, J.: Minimization of ℓ_{1-2} for compressed sensing. SIAM J. Sci. Comput. **37**(1), A536–A563 (2015)
37. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016)
38. Zeng, J., Lau, T.T.K., Lin, S., Yao, Y.: Global convergence of block coordinate descent in deep learning. In: International Conference on Machine Learning, pp. 7313–7323. PMLR (2019)
39. Zhang, S., Xin, J.: Minimization of transformed l_1 penalty: theory, difference of convex function algorithm, and robust application in compressed sensing. Math. Program. **169**(1), 307–336 (2018)
40. Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., Tian, Q.: Variational convolutional neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2780–2789 (2019)