



Article

MobilePrune: Neural Network Compression via ℓ_0 Sparse Group Lasso on the Mobile System

Yubo Shao ¹, Kaikai Zhao ², Zhiwen Cao ³, Zhehao Peng ¹, Xingang Peng ⁴, Pan Li ¹, Yijie Wang ^{2,*} and Jianzhu Ma ^{5,*}

- ¹ Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA; shao111@purdue.edu (Y.S.); peng272@purdue.edu (Z.P.); panli@purdue.edu (P.L.)
- ² Department of Computer Science, Indiana University at Bloomington, Bloomington, IN 47405, USA; kkai_zhao@yeah.net
- ³ Department of Computer Graphics, Purdue University, West Lafayette, IN 47907, USA; cao270@purdue.edu
- ⁴ Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing 100190, China; xingang.peng@gmail.com
- ⁵ Institute for Artificial Intelligence, Peking University, Beijing 100871, China
- * Correspondence: yijie.wang@iu.edu (Y.W.); majianzhu@pku.edu.cn (J.M.)

Abstract: It is hard to directly deploy deep learning models on today's smartphones due to the substantial computational costs introduced by millions of parameters. To compress the model, we develop an ℓ_0 -based sparse group lasso model called MobilePrune which can generate extremely compact neural network models for both desktop and mobile platforms. We adopt group lasso penalty to enforce sparsity at the group level to benefit General Matrix Multiply (GEMM) and develop the very first algorithm that can optimize the ℓ_0 norm in an exact manner and achieve the global convergence guarantee in the deep learning context. MobilePrune also allows complicated group structures to be applied on the group penalty (i.e., trees and overlapping groups) to suit DNN models with more complex architectures. Empirically, we observe the substantial reduction of compression ratio and computational costs for various popular deep learning models on multiple benchmark datasets compared to the state-of-the-art methods. More importantly, the compression models are deployed on the android system to confirm that our approach is able to achieve less response delay and battery consumption on mobile phones.

Keywords: mobile computing; model compression; pruning network; deep learning; convolutional neural network



Citation: Shao, Y.; Zhao, K.; Cao, Z.; Peng, Z.; Peng, X.; Li, P.; Wang, Y.; Ma, J. MobilePrune: Neural Network Compression via ℓ_0 Sparse Group Lasso on the Mobile System. *Sensors* **2022**, *22*, 4081. <https://doi.org/10.3390/s22114081>

Academic Editor: Paolo Bellavista

Received: 18 March 2022

Accepted: 25 May 2022

Published: 27 May 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Deep neural networks (DNNs) have achieved tremendous success in many real-world applications. However, the computational cost of DNN models significantly restricts their deployment on platforms with limited computational resources, such as mobile devices. To address this challenge, numerous model compression algorithms have been proposed to reduce the sizes of DNN models. The most popular solution is to prune the weights with small magnitudes by adding ℓ_0 or ℓ_1 penalties [1–4]. The non-zero weights selected by these methods are randomly distributed and do not reduce the memory consumption due to the matrix operations widely adopted in nowadays deep learning architectures as shown in Figure 1(b.2). The implementation of such a non-structured sparse matrix in cuDNN [5], which is the Basic Linear Algebra Subroutines library used by deep learning models, has similar memory consumption as the original matrix without pruning, as shown in Figure 1(b.2),c.2). To overcome the problem, structured pruning models [6–9] are proposed to enforce group sparsity by pruning a group of pre-defined variables together. By tying the weights connecting to the same neuron together, these approaches are able to prune a number of hidden neurons to reduce the sizes of weight matrices and benefit

General Matrix Multiply (GEMM) used in cuDNN as shown in Figure 1(b.3),(c.3). However, one of the main problems of the structured methods is that they do not consider and take advantage of the hardware accelerator architectures.

In this paper, we observe three key factors that could lead to an extremely compact deep network. First, we observe that most modern deep learning architectures rely on the General Matrix Multiply (GEMM) functions implemented in the cuDNN package. We, therefore, propose a new network compression algorithm to harmonize the group selections in the structured penalty and the implementation of GEMM in cuDNN as well as the hardware accelerator using sparse systolic tensor array [10,11]. Figure 1 demonstrates the basic rationale of our observation. In comparison to the pruned model in Figure 1(c.3),(c.4) needs additional sparsity within the remaining groups, which could be utilized by the sparse systolic tensor array for hardware acceleration.

The second observation is that recent studies [12,13] have demonstrated that ℓ_0 norm is the best sparsity-inducing penalty compared to lasso ℓ_1 [14], elastic net [15], SCAD [16], and MCP [17] models. Remarkably, even current ℓ_0 optimization techniques can not achieve the global optimal solution, these ℓ_0 -based methods with sub-optimal solutions still significantly outperform other sparsity norms, which can be solved to global optima [13,18]. Hence, the community has shown great interest in using ℓ_0 norm to compress the large-scale DNN models [8,19–24]. However, some ℓ_0 -based DNN pruning approaches [8,25,26] rely on different relaxation strategies to conquer the non-convex and non-differentiable challenges, which does not fully exploit the strength of ℓ_0 regularization. The other methods [19–23] rely on the alternating direction method of multipliers (ADMM) whose global convergence has not been proved so far when applied to ℓ_0 norm optimization [27–29].

Third, most of these algorithms are designed originally for mobile platforms, as the computational resources are relatively rich for desktop applications. However, few of them have been deployed on real mobile systems to test the running time and energy consumption to verify their assumptions.

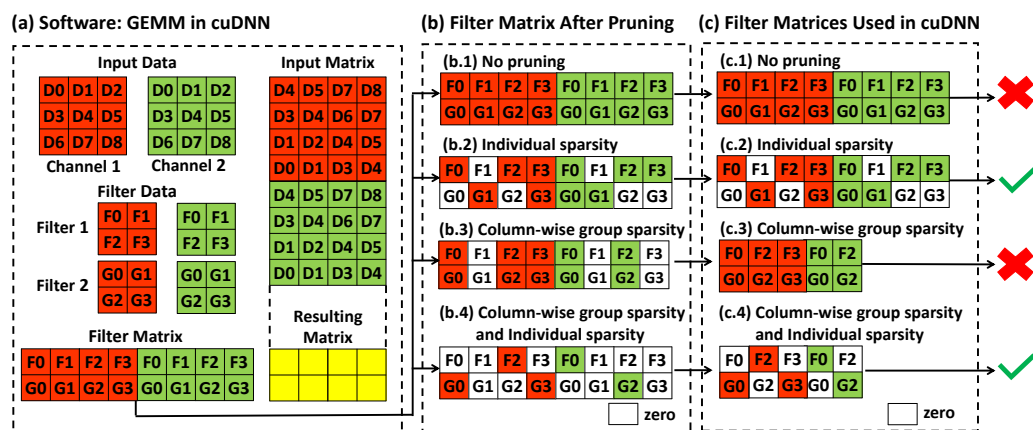


Figure 1. Observations of different strategies’ pruned filter matrix for hardware acceleration with software implementation of convolution in cuDNN. (a) General Matrix Multiply (GEMM) is applied in cuDNN. (b) Different strategies such as no pruning, individual sparsity, column-wise group sparsity, and both individual sparsity and column-wise group sparsity on pruning the filter matrix. (c) The pruned filter matrix implemented in cuDNN and determined whether it can be used for hardware acceleration or not.

Therefore, we develop the very first algorithm in this paper, named MobilePrune, which is able to solve the optimization of ℓ_0 sparse group lasso regularization in an exact manner. The main technical contribution is that we solve the proximal operators for ℓ_0 sparse group lasso, where groups in the group lasso term could have overlapping group structure and tree structure [30]. From the theoretical point of view, we prove our algorithm always converges to the critical point (local optimal point) under mild

conditions. In addition, we conduct extensive experiments on multiple public datasets and find MobilePrune achieves superior performance at sparsifying networks with both fully connected and convolutional layers. More importantly, we deploy our system on the real android system on several mobile devices and test the performance of the algorithm on multiple Human Activity Recognition (HAR) tasks. The results show that MobilePrune achieves much lower energy consumption and higher pruning rate while still retaining high prediction accuracy. Besides a powerful network compression algorithm, this work also provides a valuable platform and mobile dataset for further work to evaluate their methods in a very real scenario.

The rest of this paper is organized as follows. Section 2 provides the relevant background and related work. In Section 3, we give a brief overview of the proposed MobilePrune methods. In Section 4, we discuss the detailed information of the proposed methods and algorithms. In Section 5, we describe how the experiments are set up and evaluate the proposed methods from different perspectives. Section 6 discusses the future work and summarizes the paper.

2. Related Work

2.1. Sparsity for Deep Learning Models

Many pruning algorithms for deep learning models achieve slim neural networks by introducing sparse-inducing norms to the models. ℓ_1 regularization [31–33] and ℓ_0 regularization [8] were applied to induce sparsity on each individual weight. However, such individual sparsity has arbitrary structures, which cannot be utilized by software and hardware. Wei et al. [24] applied group sparsity to prune filters or channels, which can reduce the matrix size used in GEMM in cuDNN. Because the pruned models are compatible with cuDNN, they achieved large speedups. There are methods [34,35] aiming to find sparse models at both individual and group levels, which is similar to our goal. However, they all used ℓ_1 norm to induce individual sparsity in addition to group sparsity. We have performed a comprehensive comparison and demonstrated that our MobilePrune method is the best in inducing sparsity at both individual and group levels for pruning deep learning models in Section 5.

2.2. Learning Algorithms for ℓ_0 Norm

Recent studies [12,13] demonstrate that ℓ_0 norm is the best sparsity-inducing penalty comparing to lasso ℓ_1 [14], elastic net [15], SCAD [16], and MCP [17] models. Remarkably, even current ℓ_0 optimization techniques cannot achieve the global optimal solution, these ℓ_0 -based methods with sub-optimal solutions still significantly outperform ℓ_1 and elastic net models, which can be solved to global optima [13,18]. Therefore, the machine learning community has shown great interest in using ℓ_0 norm to compress the large-scale DNN models [8,19–24]. However, some ℓ_0 -based DNN pruning approaches [8,25] rely on different relaxation strategies to conquer the non-convex and non-differentiable challenges, which does not fully exploit the strength of ℓ_0 regularization. The other methods [19–23] rely on the alternating direction method of multipliers (ADMM) whose global convergence has not proved so far when applied to ℓ_0 norm optimization [27–29].

2.3. Software & Hardware Compatibility

In this paper, we aim to design an algorithm to make the pruned DNN models compatible with cuDNN library [5] and hardware accelerator architecture that uses the sparse systolic tensor array [11,36]. cuDNN is the GPU-accelerated library used in deep learning models [5]. As shown in Figure 1a, convolution used in the convolutional neural network is lowered to matrix multiplication. Therefore, the size of the filter matrix can be reduced when inducing group sparsity column-wise as shown in Figure 1(b.3),(b.4), which can reduce the memory of the DNN models to achieve practical performance improvement. The systolic tensor array is an efficient hardware accelerator for structured sparse matrix as shown in Figure 1(c.4). Specifically, each column in Figure 1(c.4) is sparse. To achieve a

pruned DNN model that is compatible with cuDNN and the systolic tensor array, sparsity needs to be induced on both the group level and within-group level. We will show how we achieve this in the following sections.

3. Overview

The central idea of MobilePrune is to compress the deep learning model in a way that is compatible with the architecture of data organization in the memory by combining ℓ_0 regularization and group lasso regularization. The group lasso regularization helps to keep important groups of weights that benefit cuDNN, while ℓ_0 regularization helps to achieve additional sparsity within those important groups that are needed for hardware acceleration. Figure 2 provides an overview of the proposed MobilePrune method. As illustrated in Figure 2a–c, the group lasso penalty will remove all the weights together with the i th neuron if it is less important for the prediction and if a group is selected, the ℓ_0 penalty further removes the weights with small magnitudes within the group. Note that zeroing out the weights connected to the i th neuron results in removing the i th neuron and all the associated weights entirely. We will discuss more detail information in next section.

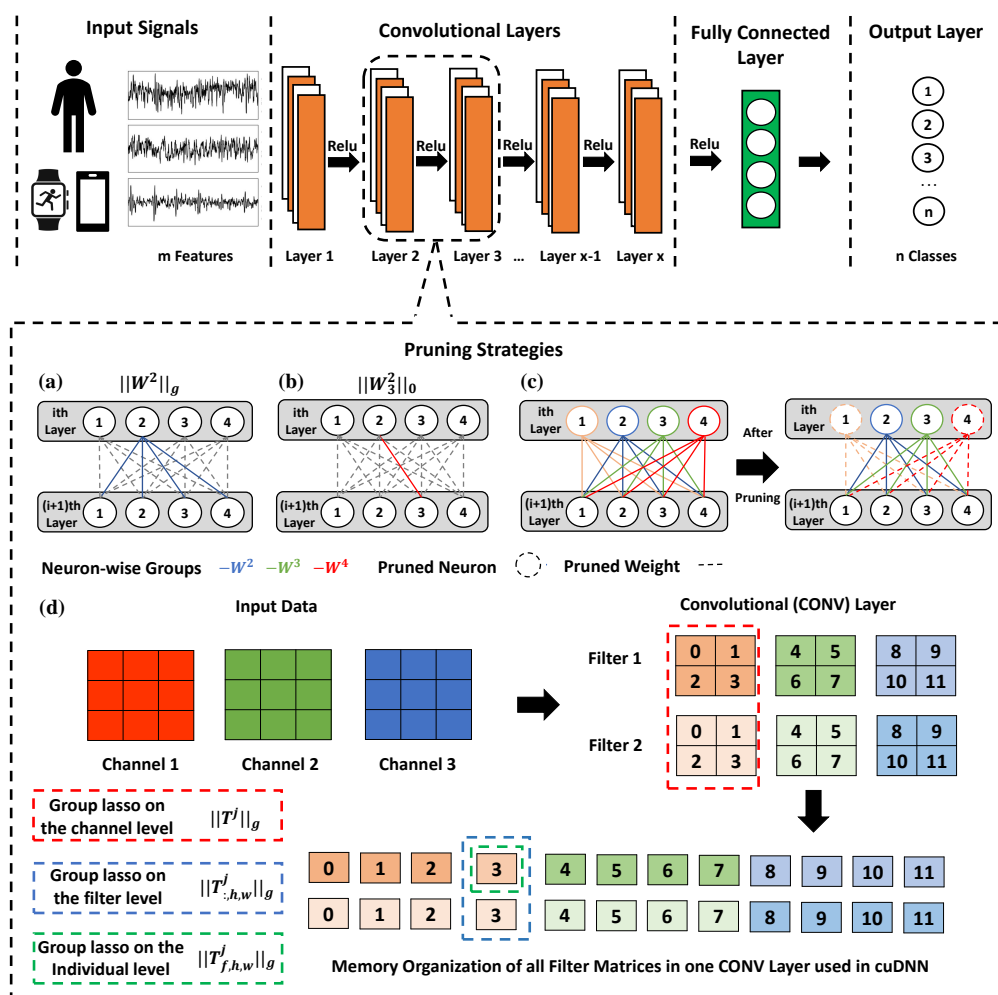


Figure 2. Overview of the proposed MobilePrune method. (a) Group sparsity for weights of a neuron for fully connected layers. (b) Sparsity on individual weights for fully connected layers. (c) Pruning strategy for fully connected layers and their effect where sparsity is induced on both neuron-wise groups and individual weights. (d) Group and individual sparsity for convolutional layers.

4. Methods

Our main objective is to obtain a sparse deep neural network with a significantly less number of parameters at both individual and group levels by using the proposed novel combined regularizer: ℓ_0 sparse group lasso.

4.1. ℓ_0 Sparse Group Lasso

We aim to prune a generic (deep) neural network, which includes fully connected feed-forward networks (FCN) and convolutional neural networks (CNN). Assuming the generic neural network has N neurons in FCN and M channels in CNN. Let W^i denote the outgoing weights of the i th neuron in FCN and T^j represent the 3D tensor of all filters in the j th channel, which can come from different layers. The training objective for the neural network is given as follows:

$$\min_{W,T} : \mathcal{L}(W, T; \mathcal{D}) + \Omega_{\lambda}^{\eta}(W) + \Gamma_{\beta,\gamma}^{\alpha}(T) \quad (1)$$

where $W = \{W^1, \dots, W^N\}$, $T = \{T^1, \dots, T^M\}$, $\mathcal{D} = \{x_i, y_i\}_{i=1}^P$ is a training dataset with P instances, \mathcal{L} is an arbitrary loss function parameterized by W and T , $\Omega_{\lambda}^{\eta}(W)$ and $\Gamma_{\beta,\gamma}^{\alpha}(T)$ represent the ℓ_0 sparse group lasso penalties for neurons and channels, respectively. Specifically, $\Omega_{\lambda}^{\eta}(W)$ is defined as

$$\Omega_{\lambda}^{\eta}(W) = \sum_{i=1}^N (\eta \|W^i\|_0 + \lambda \|W^i\|_g) \quad (2)$$

where $\eta \geq 0$ and $\lambda \geq 0$ are regularization parameters. Let $n(i)$ represent the set of outgoing edge weights of neuron i . Then, $\|W^i\|_0 = \sum_{j \in n(i)} \|W_j^i\|_0$ (W_j^i is the j th outgoing edge weight of neuron i ; $\|W_j^i\|_0 = 0$ when $W_j^i = 0$ and $\|W_j^i\|_0 = 1$ otherwise) computes the number of the non-zero edges in W^i and $\|W^i\|_g = \sqrt{\sum_{j \in n(i)} (W_j^i)^2}$ aggregates the weights associated with the i th neuron as a group. The core spirit of Equation (2) is illustrated in Figure 2a–c, the group lasso penalty $\|W^i\|_g$ tends to remove all the weights together with the i th neuron if it is less important. If a group is selected, the ℓ_0 penalty further removes the weights with small magnitudes within the group. Group sparsity $\|W^i\|_g$ can help remove neurons in the neural network, which reduces the size of the neural network and further improve efficiency. Individual sparsity $\|W^i\|_0$ helps to achieve additional sparsity within the remaining neurons. Such structured sparsity can be used by the systolic tensor array [10,11]. The other regularization term $\Gamma_{\beta,\gamma}^{\alpha}(T)$ is defined as following,

$$\Gamma_{\beta,\gamma}^{\alpha}(T) = \sum_{j=1}^M (\alpha \|T^j\|_0 + \beta \|T^j\|_g + \gamma \sum_{h,w} \|T_{:,h,w}^j\|_g), \quad (3)$$

where α, β , and γ are non-negative regularization parameters. Equation (3) defines a hierarchical-structured sparse penalty in which structure is guided by the memory organization of GEMM using cuDNN [5]. As demonstrated in Figure 2d, the pruning strategy encoded in Equation (3) explicitly takes advantage of the GEMM used in cuDNN. $\|T^j\|_g$ enforces the group sparsity of all the filters applied to the j th channel and $\|T_{:,h,w}^j\|_g$ enforces the group sparsity across the filters on the same channel and $\|T^j\|_0 = \sum_f \sum_h \sum_w \|T_{f,h,w}^j\|_0$ prunes the small weights within the remaining channels and filters. Equation (3) can help to achieve an extremely compact model as Figure 1(c.4). Therefore, the computation can be accelerated at both software and hardware levels.

4.2. Exact Optimization by PALM

In this subsection, we first briefly review the general PALM (Proximal Alternating Linearized Minimization) framework used in our MobilePrune algorithm. Then, we

introduce how we modify PALM to efficiently optimize the ℓ_0 sparse group lasso for neural network compression. PALM is designed to optimize a general optimization problem formulated as:

$$\min_{W, T} : F(W, T) + \Phi_1(W) + \Phi_2(T). \quad (4)$$

where $F(W, T)$ is a smooth function and $\Phi_1(W)$ and $\Phi_2(T)$ do not need to be convex or smooth, but are required to be lower semi-continuous. The PALM algorithm applies proximal forward-backward algorithm [37] to optimize Equation (4) with respect to W and T in an alternative manner. Specifically, at iteration $k + 1$, the temporal values of $W^{(k+1)}$ and $T^{(k+1)}$ for the proximal forward-backward mapping are derived by solving the following sub-problems,

$$W^{(k+1)} \in \min_W : \left\{ \frac{c^k}{2} \|W - U_i^{(k)}\|_F^2 + \Phi_1(W) \right\}, \quad (5)$$

$$T^{(k+1)} \in \min_T : \left\{ \frac{d^k}{2} \|T - V_i^{(k)}\|_F^2 + \Phi_2(T) \right\}, \quad (6)$$

where $U_i^{(k+1)} = W_i^{(k)} - \frac{1}{c^k} \nabla_W F(W^{(k)}, T^{(k)})$ and $V_i^{(k+1)} = T_i^{(k)} - \frac{1}{d^k} \nabla_T F(W^{(k+1)}, T^{(k)})$. Additionally, c^k and d^k are positive constants. This optimization process has been proven to converge to a critical point when functions F , Φ_1 , and Φ_2 are bounded [37]. We further extend the convergence proof in [37] and prove that the global convergence of PALM holds for training deep learning models under mild conditions. The detailed proof can be found in Appendix A.

To optimize Equation (1), we define two proximal operators for the two penalty terms as the following,

$$\pi_\lambda^\eta(y) \equiv \arg \min_x \left\{ \frac{1}{2} \|x - y\|_2^2 + \Omega_\lambda^\eta(x) \right\}, \quad (7)$$

$$\theta_{\beta, \gamma}^\alpha(y) \equiv \arg \min_x \left\{ \frac{1}{2} \|x - y\|_2^2 + \Gamma_{\beta, \gamma}^\alpha(x) \right\}. \quad (8)$$

Here, functions $\Omega_\lambda^\eta(\cdot)$ and $\Gamma_{\beta, \gamma}^\alpha(\cdot)$ take vectors as inputs, which are equivalent to Equations (2) and (3) once we vectorize W^i and T^j . The overall optimization process of MobilePrune is described in Algorithm 1. Once we can efficiently compute the optimal solution of $\pi_\lambda^\eta(\cdot)$ and $\theta_{\beta, \gamma}^\alpha(\cdot)$, the computational burden mainly concentrates on the partial derivative calculation of functions $H(\cdot)$ and $F(\cdot)$, which is the same as training a normal DNN model.

Algorithm 1 The framework of MobilePrune Algorithm.

Initialize $(W^i)^0, \forall i, (T^j)^0, \forall j$ and L_r .

for $k = 0, 2, \dots$ **do**

for $i = 1, 2, \dots, N$ **do**

$$H_k^i = \nabla_{W_k^i} \mathcal{L}(W_k^i, W_k^{j \neq i}, T_k).$$

$$W_{k+1}^i \in \pi_{\lambda/L_r}^{\eta/L_r} \left(W_k^i - \frac{1}{L_r} H_k^i \right) \text{ by solving Equation (7).}$$

end for

for $j = 1, 2, \dots, M$ **do**

$$F_k^j = \nabla_{T_k^j} \mathcal{L}(W_{k+1}, T_k^j, T_k^{l \neq j}).$$

$$T_{k+1}^j \in \theta_{\beta/L_r, \gamma/L_r}^{\alpha/L_r} \left(T_k^j - \frac{1}{L_r} F_k^j \right) \text{ by solving Equation (8).}$$

end for

end for

4.3. Efficient Computation of Proximal Operators

To the best of our knowledge, $\pi_\lambda^\eta(y)$ and $\theta_{\beta,\gamma}^\alpha(y)$ defined in Equations (7) and (8) are novel proximal operators that have not been attempted before. Solving $\pi_\lambda^\eta(y)$ and $\theta_{\beta,\gamma}^\alpha(y)$ is the key to apply MobilePrune in Algorithm 1. Therefore, this subsection elaborates the algorithmic contributions we made to efficiently calculate $\pi_\lambda^\eta(y)$ and $\theta_{\beta,\gamma}^\alpha(y)$.

4.3.1. Proximal Operator $\pi_\lambda^\eta(\cdot)$

The difficulty of solving $\pi_\lambda^\eta(y)$ in Equation (9) is that both $\|x\|_g$ and $\|x\|_0$ are not differentiable when the vector $x = \mathbf{0}$. Furthermore, $\|x\|_0$ calculates the number of non-zeros in the vector $x \in \mathbb{R}^n$ and there are $C(n,0) + C(n,1) + \dots + C(n,n) = 2^n$ ($C(n,k)$ computes the number of non-zero patterns in x where k elements in x are not zeros) possible combinations, which indicates the brute-force method needs 2^n computations to find the global optimal solution. However, here we prove that the $\pi_\lambda^\eta(y)$ in Equation (9) can be efficiently solved by a $O(n \log(n))$ algorithm in a closed form. We illustrate the algorithm in Algorithm 2 and prove its correctness in Theorem 1. To the best of our knowledge, it is the first efficient algorithm that can calculate this novel proximal operator.

Theorem 1. *The proximal operator $\pi_\lambda^\eta(y)$ can be written as*

$$\pi_\lambda^\eta(y) \equiv \arg \min_x \left\{ f(x) := \frac{1}{2} \|x - y\|_2^2 + \lambda \|x\|_g + \eta \|x\|_0 \right\} \quad (9)$$

The optimal solution of this proximal operator can be computed by Algorithm 2.

Algorithm 2 Efficient calculation of $\pi_\lambda^\eta(y)$

Input: A sorted vector y , such that $|y_1| \geq |y_2| \geq \dots$

Output: x^*

for $i = 0, \dots, n$ **do**

if $\|\vec{y}_i\|_g \leq \lambda$ **then**

$$U_i = \frac{1}{2} \|y\|_2^2$$

else

$$U_i = -\frac{1}{2} (\|\vec{y}_i\|_g - \lambda)^2 + i\eta + \frac{1}{2} \|y\|_2^2$$

end if

end for

Compute $k = \arg \min_j U_j$

if $U_k \geq \frac{1}{2} \|y\|_2^2$ **then**

$$x^* = \mathbf{0}$$

else

$$x^* = (\|\vec{y}_k\|_g - \lambda) \frac{\vec{y}_k}{\|\vec{y}_k\|_g}$$

end if

Proof. Without loss of generality, we assume $y = [y_1, y_2, \dots, y_n]^T \in \mathbb{R}^n$ is an ordered vector, where $|y_1| \geq |y_2| \geq \dots \geq |y_n|$. Then, we define $\vec{y}_k = [y_1, y_2, \dots, y_k, 0, \dots, 0]$, where the top k elements with largest absolute values are kept and all the rest elements are set to zeroes. We define another set $\Phi^k = \{x \mid \|x\|_0 = k, x \in \mathbb{R}^n\}$ to represent all n -dimensional vectors with exact k non-zero elements. For any $x = [x_1, x_2, \dots, x_n]^T \in \Phi^k$, we further define a mask function $e : e_i = \mathbb{1}\{x_i \neq 0\}$ to reveal the non-zero locations of x .

Since we do not know how many non-zero elements remain in the optimal solution of Equation (9), we need to enumerate all possible k and solve $n + 1$ sub-problems for all $x^k \in \Phi^k$ with $k = 0, \dots, n$. For each k , the sub-problem is defined as

$$\begin{aligned} \min_{x^k \in \Phi^k} f(x^k) &:= \frac{1}{2} \|x^k - y\|_2^2 + \Omega_\lambda^\eta(x^k) \\ \Leftrightarrow \min_{x^k \in \Phi^k} &\frac{1}{2} \|x^k - y^k\|_2^2 + \frac{1}{2} \|\bar{y}^k\|_2^2 + \Omega_\lambda^\eta(x^k) \end{aligned} \tag{10}$$

Based on Lemma A4 in Appendix A.2, we observe that if $\|y^k\|_g \leq \lambda$, then $x^{k*} = \mathbf{0}$ and $f(x^{k*}) = \frac{1}{2} \|y\|_2^2$. If $\|y^k\|_g > \lambda$, then $x^{k*} = (\|y^k\|_g - \lambda) \frac{y^k}{\|y^k\|_g}$ and the value of the objective function can be computed as

$$\begin{aligned} f(x^{k*}) &= \frac{1}{2} \|x^{k*} - y^k\|_2^2 + \frac{1}{2} \|\bar{y}^k\|_2^2 + \Omega_\lambda^\eta(x^{k*}) \\ &= -\frac{1}{2} (\|y^k\|_g - \lambda)^2 + \eta k + \frac{1}{2} \|y\|_2^2. \end{aligned} \tag{11}$$

Equation (11) tells us $f(x^{k*})$ is a function of y^k . The task of calculating the minimum value of function $f(x^{k*})$ is transformed into solving another optimization $-\frac{1}{2} (\|y^k\|_g - \lambda)^2 + \eta k + \frac{1}{2} \|y\|_2^2$ with respect to y^k , which is equivalent to ask which of the k components of y can achieve the minimum value of f . Since we assume $\|y^k\|_g > \lambda$, the optimal solution is clearly to select the top k components with the largest value from y . Therefore we have $\vec{y}_k = \arg \min_{y^k} -\frac{1}{2} (\|y^k\|_g - \lambda)^2 + \eta k + \frac{1}{2} \|y\|_2^2$ and the corresponding $x^{k*} = (\|\vec{y}_k\|_g - \lambda) \frac{\vec{y}_k}{\|\vec{y}_k\|_g}$. Furthermore, the objective function value is $-\frac{1}{2} (\|\vec{y}_k\|_g - \lambda)^2 + \eta k + \frac{1}{2} \|y\|_2^2$. Hence, problem (10) has a closed-form solution. \square

As shown in Algorithm 2, the heaviest computation is to sort the input vector y , therefore, the time complexity for solving Equation (9) is $O(n \log(n))$.

4.3.2. Proximal Operator $\theta_{\beta, \gamma}^\alpha(y)$

Similar as $\pi_\lambda^\eta(y)$, $\theta_{\beta, \gamma}^\alpha(y)$ is the solution of the following optimization problem:

$$\begin{aligned} \min_x : \kappa(x) &:= \frac{1}{2} \|x - y\|_2^2 + \alpha \|x\|_0 + \beta \|x\|_g + \gamma \sum_{i=1}^d \|x_{G_i}\|_g \\ \text{s.t. } &\bigcup_{i=1}^d G_i = \{1, 2, \dots, n\}, G_i \cap G_j = \emptyset, \forall i, j \end{aligned} \tag{12}$$

where we assume $x, y \in \mathbb{R}^n$. $G_i \subseteq \{1, \dots, n\}$, i is the index of a group and d represents the number of groups. Note that the grouping structures specified in Equation (12) is a special case of the grouped tree structures [30], where $\|x\|_g$ is the group lasso for the root of the tree and all the $\|x_{G_i}\|_g$ are the group lasso terms of its children. Notice that groups from the same depth on the tree do not overlap and furthermore $\|x\|_0 = \sum_{i=1}^d \|x_{G_i}\|_0$. To simplify the notation, assuming $\|x\|_0 \neq 0$ we define $h(x) = \frac{1}{2} \|x - y\|_2^2 + \beta \|x\|_g$ that is a convex and differentiable and rewrite the problem as

$$\begin{aligned} \min_x : &h(x) + \sum_{i=1}^d \Omega_\gamma^\alpha(x_{G_i}) \\ \text{s.t. } &x \neq \mathbf{0}, \bigcup_{i=1}^d G_i = \{1, 2, \dots, n\}, G_i \cap G_j = \emptyset, \forall i, j \in \{1, \dots, d\}, \end{aligned} \tag{13}$$

where $\sum_{i=1}^d \Omega_{\gamma}^{\alpha}(x_{G_i}) = \sum_{i=1}^d \alpha \|x_{G_i}\|_0 + \gamma \|x_{G_i}\|_g$. We can use the proximal method [38] to find a solution x^{\dagger} of Equation (13). In the proximal method, we need to estimate the Lipschitz constant $\mathcal{L}(x) = 1 + \frac{\beta}{\|x\|_g}$ and the partial derivative $\nabla_{x_{G_i}} h(x) = (1 + \frac{\beta}{\|x\|_g})x_{G_i} - y_{G_i}$. In addition, we need to use Algorithm 2 to solve $\pi_{\gamma}^{\alpha}(\cdot)$ for each group x_{G_i} . After obtaining x^{\dagger} , we can find the solution of Equation (12) by comparing $\kappa(\mathbf{0})$ with $\kappa(x^{\dagger})$. If $\kappa(\mathbf{0}) \leq \kappa(x^{\dagger})$, then the local optimal solution of Equation (12) is $x^* = \mathbf{0}$, otherwise, $x^* = x^{\dagger}$. We elaborate the algorithm for the proximal operator $\theta_{\beta, \gamma}^{\alpha}(y)$ in the Algorithm 3. The major computation cost is the proximal method, therefore, the convergence rate of Algorithm 3 is $O(1/k)$ [38].

Algorithm 3 Efficient calculation of $\theta_{\beta, \gamma}^{\alpha}(y)$

Input: \mathcal{L}^0 and x^0

Output: x^*

for $l = 0, 1, 2, \dots, k$ **do**

Let $\mathcal{L}^l = 1 + \frac{\beta}{\|x^l\|_g}$ and $u = x_{G_i}^l - \frac{1}{\mathcal{L}^l} \nabla_{x_{G_i}} h(x^l)$, then compute $x_{G_i}^l \in \pi_{\gamma/\mathcal{L}^l}^{\alpha/\mathcal{L}^l}(u)$, $\forall i$ by applying Algorithm 2.

end for

if $\kappa(\mathbf{0}) \leq \kappa(x^k)$ **then**

$x^* = \mathbf{0}$

else

$x^* = x^k$

end if

5. Experimental Setup and Results

5.1. Performance on Image Benchmarks

In this subsection, we compared our proposed MobilePrune approach with other state-of-the-art pruning methods in terms of prune rate, computational costs, and test accuracy. We mainly compared our methods with structured pruning methods because DNN models pruned by non-structure pruning methods could not obtain practical speedup as shown in Figure 1. Notably, we only compared the results that can be reproduced by the source codes provided by the competing methods. First, we briefly summarized their methodology. PF [32] and NN slimming [33] were simple magnitude-based pruning methods based on l_1 norm. BC [9], SBP [7], and VIBNet [39] cast the DNN pruning into probabilistic Bayesian models. C-OBP [40], C-OBS [2], Kron-OBP [40,41], Kron-OBS [2,41], and EigenDamage [42] are Hessian matrix-based methods. ℓ_0 norm penalized method [8] and group lasso penalized method [24] are also well-known methods.

In our experiments, we use NVIDIA Corporation as the GPU and the number of cores of the CPU is 12. All the baseline models were trained from scratch via stochastic gradient decent(SGD) with a momentum of 0.9. We trained the networks for 150 epochs on MNIST and 400 epochs on CIFAR-10 and Tiny-ImageNet with an initial learning rate of 0.1 and weight decay of 5×10^{-4} . The learning rate is decayed by a factor of 10 at 50, 100 on MNIST and at 100, 200 on CIFAR-10 and Tiny-ImageNet, respectively. The details of hyper-parameters for all experiments are summarized in Appendix B. We also provide the computational efficiency of our methods in Appendix C.

5.1.1. MNIST Dataset

We first applied MobilePrune to prune the LeNet-300-100 and LeNet-5 [7–9] models on the MNIST dataset [43]. LeNet-300-100 is a fully-connected neural network model with three layers and 267 K parameters. LeNet-5 is comprised of two [20, 50] convolutional layers and two [800, 500] fully-connected layers with 431K parameters. Here, we compared with the state-of-the-art structured network compression algorithms [7–9] in terms of pruned accuracy, remaining parameters, pruned architecture, and FLOPs of the pruned models.

As shown in the top half of the MNIST dataset of Table 1, our model achieves the least number of neurons after pruning the LeNet-300-100 model and the lowest drop of the prediction accuracy 0.01% compared to other methods. More importantly, our pruned model achieves the lowest FLOPs. Note that the architecture of our pruned model is as compact as L0-sep [8], but is extremely sparse with only 5252 weights left. This additional sparsity would be critical when applying hardware acceleration [10,11] to our pruned model.

In addition, we compared with SSL on pruning the first two convolutional layers as done in [24] in Table A2. SSL has the same group lasso penalty term as ours but without ℓ_0 norm regularization. As shown, our method decreases the sizes of the filters from 25 and 500 to 16 and 65, respectively, which dramatically lowers the FLOPs. In addition, the non-zero parameters in those remaining filters is very sparse in our model.

Table 1. Comparison of pruned models with state-of-the-art methods on different datasets – MNIST, CIFAR-10, and Tiny-ImageNet, respectively. (We highlight our MobilePrune results and mark the best performance as blue among different methods for each model in each dataset).

Dataset	Model	Methods	Base/Pruned Accuracy (%)	Original/Remaining Parameters (Mil)	FLPOs (Mil)
MNIST	LeNet-300-100	BC-GNJ [9]	98.40/98.20	267.00/28.73	28.64
		BC-GHS [9]	98.40/98.20	267.00/28.17	28.09
		L0 [8]	-/98.60	-	69.27
		L0-sep [8]	-/98.20	-	26.64
		MobilePrune	98.24/98.23	267.00/5.25	25.79
	LeNet-5	SBP [7]	-/99.14	-	212.80
		BC-GNJ [9]	99.10/99.00	431.00/3.88	282.87
		BC-GHS [9]	99.10/99.00	431.00/2.59	153.38
		L0 [8]	-/99.10	-	1113.40
		MobilePrune	99.12/99.11	431.00/2.31	113.50
CIFAR-10	VGG-like	Original [44]	-/92.45	15.00/-	313.5
		PF [32]	-/93.40	15.00/5.4	206.3
		SBP [7]	92.80/92.50	15.00/-	136.0
		SBPa [7]	92.80/91.00	15.00/-	99.20
		VIBNet [39]	-/93.50	15.00 /0.87	86.82
		MobilePrune	92.96/92.94	15.00/0.60	77.83
	ResNet32	C-OBD [40]	95.30/95.27	7.42/2.92	488.85
C-OBS [2]		95.30/95.30	7.42/3.04	378.22	
Kron-OBD [40,41]		95.30/95.30	7.42/3.26	526.17	
Kron-OBS [2,41]		95.30/95.46	7.42/3.23	524.52	
EigenDamage [42]		95.30/95.28	7.42/2.99	457.46	
MobilePrune		95.29/95.47	7.42/2.93	371.30	
Tiny-ImageNet	VGG-19	NN slimming [33]	61.56/40.05	20.12/5.83	158.62
		C-OBD [40]	61.56/47.36	20.12/4.21	481.90
		C-OBS [2]	61.56/39.80	20.12/6.55	210.05
		Kron-OBD [40,41]	61.56/44.41	20.12/4.72	298.28
		Kron-OBS [2,41]	61.56/44.54	20.12/5.26	266.43
		EigenDamage [42]	61.56/56.92	20.12/5.21	408.17
		MobilePrune	61.56/56.27	20.12/4.05	407.37

The bottom half of the MNIST dataset in Table 1 shows the performance comparison on pruning the LeNet-5 model. The LeNet-5 model pruned by our method achieves the lowest FLOPs (113.50 K) with the smallest predicting accuracy drop 0.01%. Moreover, our pruned model also has the smallest number of weights (around 2310). In addition, we compared with SSL on pruning the first two convolutional layers as done in [24]. SSL

has the same group lasso penalty term as ours, but without ℓ_0 norm regularization. More details about SSL can be found in Appendix C.2. As shown, our method decreases the sizes of the filters from 25 and 500 to 16 and 65, respectively, which dramatically lowers the FLOPs. In addition, the non-zero parameters in those remaining filters are very sparse in our model.

5.1.2. CIFAR-10 Dataset

We further evaluated our method on pruning more sophisticated DNN architectures, VGG-like [44] and ResNet-32 [42,45] and widen the network by a factor of 4, on CIFAR-10 [46]. Similarly, we compared with the state-of-the-art structured pruning methods [2,7,32,39–42] in terms of various metrics. As shown in the middle of Table 1, the pruned VGG-like model obtained by our method achieves the lowest FLOPs with the smallest test accuracy drop. Similar to previous results, our pruned model is able to keep the smallest number of weights in comparison to other methods, the key for hardware acceleration [10,11]. As presented in Table 1, the pruned ResNet-32 model achieved by our method outperforms other pruned models in terms of pruned test accuracy and FLOPs. In addition, in terms of the remaining weights, our pruned model is at the same sparsity level as C-OBDD [40] while our pruned accuracy outperforms C-OBDD by a large margin.

5.1.3. Tiny-ImageNet Dataset

Besides the experiments on MNIST and CIFAR-10 datasets, we further evaluated the performance of our method on a more complex dataset, Tiny-ImageNet [47], using VGG-19 [48]. Tiny-ImageNet is a subset of the full ImageNet, which consists of 100,000 images for validation. There are 200 classes in Tiny-ImageNet. We compared our method with some state-of-the-art methods [2,33,40,42] in Table 1. As shown in Table 1, the test accuracy of the pruned model derived from our method outperforms all the other methods by a significant margin, about 10%, except EigenDamage. Our proposed method obtains the same-level test accuracy as EigenDamage. However, our method achieves a much sparser DNN model with 1.16 million fewer weights than EigenDamage. Meanwhile, our pruned model achieves lower FLOPs.

5.2. Performance on Human Activity Recognition Benchmarks

To demonstrate the efficacy and effectiveness of our proposed MobilePrune method, we perform a series of comparison studies with other state-of-the-art pruning methods such as ℓ_0 norm, ℓ_1 norm, ℓ_2 norm, group lasso, and ℓ_1 sparse group lasso for all three datasets—WISDM [49,50], UCI-HAR [51,52], and PAMAP2 [53–55]. We evaluate the pruning accuracy and pruning rate of weights (parameters) and nodes for our proposed MobilePrune approach and all other state-of-the-art pruning methods using the same learning rate (0.0001) and the same number of epochs (150) for all three datasets. The pruning thresholds are 0.015, 0.005, 0.01 for the pruning methods in the WISDM, HCI-HAR, and PAMAP2 datasets, respectively. In addition, we evaluate the computational cost and battery consumption for our proposed method with all other state-of-the-art pruning methods as well. The details of the dataset descriptions and the hyper-parameters for all experiments are summarized in Appendix D.

5.2.1. Performance on the Desktop

We use Google Colab [56] to build a PyTorch backend on the above datasets. The GPU for Google Colab is NVIDIA Tesla K80 and the number of cores of the CPU for Google Colab is 2. As shown in Table 2, if we only use ℓ_0 norm penalty or ℓ_2 norm penalty, there is no effect on neurons or channels pruning as expected for all three datasets. Similarly, if we only employ group lasso penalty, the pruned model still has more weights or nodes left. For the UCI-HAR dataset, ℓ_1 norm penalty and ℓ_1 sparse group lasso penalty cannot sparse down the model while for the other two datasets, these two penalties could achieve better sparsity, but cannot be better than MobilePrune approach. There exists a trade-off between

the pruned accuracy and the pruning rate. As can be seen in Table 2, our MobilePrune method still has high pruned accuracy even if there are not too many parameters and nodes left. In addition, we compare our MobilePrune method with ℓ_1 sparse group lasso penalty. The ℓ_0 sparse group lasso model still significantly outperforms the ℓ_1 sparse group lasso model in weights and nodes pruning, which demonstrates its superiority in pruning CNN models.

Table 2. Comparison of pruning method on the desktop with state-of-the-art methods for pruning accuracy, pruning rate and response delay on HAR datasets—WISDM, HCI-HAR, and PAMAP2, respectively. (We highlight our MobilePrune results and mark the best performance as blue among different penalties for each dataset).

Dataset	Penalty	Base/Pruned Accuracy (%)	Parameter Nonzero (%)	Parameter Remaining (%)	Node Remaining (%)	Base/Pruned Response Delay (s)	Time Saving Percentage (%)
WISDM	l_0 norm	94.72/94.79	63.36	100.00	100.00	0.38/0.39	0.00
	l_1 norm	94.30/93.84	13.58	46.26	68.16	0.38/0.24	36.84
	l_2 norm	94.61/94.54	56.28	90.46	95.12	0.38/0.35	7.89
	Group lasso	94.68/94.32	48.23	89.73	94.73	0.38/0.35	7.89
	l_1 sparse Group lasso	94.81/94.79	17.91	53.41	73.83	0.41/0.26	36.59
	MobilePrune	94.97 / 94.65	9.52	28.03	52.52	0.50/0.17	66.00
	UCI-HAR	l_0 norm	91.52/91.48	88.49	100.00	100.00	0.84/0.80
l_1 norm		90.46/90.33	81.58	98.47	99.22	0.81/0.82	0.00
l_2 norm		91.01/90.94	88.35	100.00	100.00	0.79/0.80	0.00
Group lasso		90.80/90.84	82.91	100.00	100.00	0.83/0.78	6.02
l_1 sparse Group lasso		91.11/91.04	81.21	97.70	98.83	0.84/0.80	4.76
MobilePrune		90.06/89.96	23.00	46.83	68.75	1.01/0.43	57.43
PAMAP2		l_0 norm	93.15/93.07	69.27	100.00	100.00	0.41/0.41
	l_1 norm	95.22/95.29	1.46	7.28	19.73	0.40/0.08	80.00
	l_2 norm	92.08/ 92.09	65.32	94.93	97.27	0.41/0.39	4.88
	Group lasso	93.30/ 93.28	61.78	100.00	100.00	0.41/0.41	0.00
	l_1 sparse Group lasso	96.87/97.20	2.67	9.72	26.17	0.40/0.10	75.00
	MobilePrune	96.89/96.95	1.26	3.72	10.74	0.51/0.05	90.20

We also calculate the response delay and time saving percentage for all the above methods on the desktop platform. Response delay is the time needed for the desktop to run the pre-trained model after the raw input signal is ready. Here in Table 2, the response delay results are obtained after running 200 input samples. As can be seen in Table 2, MobilePrune could save up to 66.00%, 57.43%, 90.20% on response delay on WISDM, HCI-HAR, and PAMAP2 datasets, respectively.

Overall, if we apply MobilePrune method, the pruned CNN models can still achieve the best sparsity in terms of both neurons (or channel) and weights without loss of performance. Additionally, the results in Table 2 show that our MobilePrune method could achieve 28.03%, 46.83%, 3.72% on weight (parameter) sparsity, and 52.52%, 68.75%, and 10.74% on node sparsity for the WISDM, UCI-HAR, and PAMAP2 datasets, respectively.

5.2.2. Performance of Mobile Phones

We evaluate the computational cost and battery consumption for our proposed MobilePrune approach with all other state-of-the-art pruning methods. In order to obtain the final results about how these models perform on today's smartphone, we implement an Android Application using Android Studio on Huawei P20 and OnePlus 8 Pro. PyTorch Android API [57] is used here for running trained models on Android devices. Currently, the Android devices only support running machine learning models by using CPU only.

For the Huawei P20, the CPU is Cortex-A73. For the OnePlus 8 Pro, it is using Octa-Core as its CPU. We also use the Batterystats tool and the Battery Historian script [58] to test the battery consumption.

Table 3 shows the response delay results and battery usage for our proposed method and all other state-to-the-arts pruning methods. Response delay is the time needed for the smartphone's system to run the pre-trained model after the raw input signal is ready. Here, in Table 3, the response delay results are obtained after running 200 input samples and the battery consumption results are obtained after running 2000 input samples for each penalty in all three datasets. For the HCI-HAR dataset, our MobilePrune approach could save up to 40.14%, 22.22% on response delay and 34.52%, 19.44% on battery usage for Huawei P20 and OnePlus Pro 8, respectively, while the other pruning methods stay almost the same compared to the uncompressed version. For the WISDM and PAMAP2 datasets, l_0 norm penalty, l_2 norm penalty, and group lasso penalty cannot sparse down the model, and therefore they cannot provide any savings in both response delay and battery consumption. l_1 norm and l_1 sparse group lasso methods could provide better time saving and battery consumption saving compared to those three penalties, but they still cannot perform better than the MobilePrune method, which saves 61.94% and 88.15% in response time, and 37.50% and 36.71% in battery consumption for WISDM and PAMAP2 dataset, respectively, on Huawei P20. Additionally, it also saves 52.08% and 77.66% in response time, and 32.35% and 37.93% in battery consumption for WISDM and PAMAP2 dataset, respectively, on OnePlus 8 Pro. Overall, results in Table 3 demonstrate MobilePrune's superiority in pruning HAR CNN models for battery usage and computational cost on today's smartphone.

Table 3. Comparison of pruning method on the mobile devices with other state-of-the-art pruning methods for computational cost and battery usage on HAR dataset—WISDM, HCI-HAR, and PAMAP2, respectively. (We highlight our MobilePrune results and mark the best performance as blue among different penalties for each device in each dataset).

Dataset	Device	Penalty	Base/Pruned Response Delay (s)	Time Saving Percentage (%)	Based/Pruned Device Estimated Battery Use (%/h)	Battery Saving Percentage (%)
WISDM	Huawei P20	l_0 norm	1.40/1.27	9.29	0.71/0.70	1.41
		l_1 norm	1.33/0.71	46.62	0.74/0.65	12.16
		l_2 norm	1.28/1.21	5.47	0.74/0.77	0.00
		Group lasso	1.27/1.27	0.00	0.74/0.77	0.00
		l_1 sparse Group lasso	1.25/0.81	35.20	0.74/0.68	8.11
		MobilePrune	1.34/0.51	61.94	0.72/0.45	37.50
	OnePlus 8 Pro	l_0 norm	0.57/0.49	14.04	0.34/0.32	5.88
		l_1 norm	0.48/0.34	29.17	0.35/0.30	14.29
		l_2 norm	0.48/0.40	16.67	0.34/0.34	0.00
		Group lasso	0.49/0.45	8.16	0.34/0.35	0.00
l_1 sparse Group lasso		0.48/0.33	31.25	0.35/0.30	14.29	
	MobilePrune	0.48/0.23	52.08	0.34/0.23	32.35	
HCI-HAR	Huawei P20	l_0 norm	1.43/1.43	0.00	0.84/0.84	0.00
		l_1 norm	1.42/1.42	0.00	0.85/0.84	1.18
		l_2 norm	1.43/1.43	0.00	0.84/0.84	0.00
		Group lasso	1.43/1.43	0.00	0.84/0.82	2.38
		l_1 sparse Group lasso	1.42/1.41	0.70	0.85/0.82	3.53
		MobilePrune	1.42/0.85	40.14	0.84/0.55	34.52
	OnePlus 8 Pro	l_0 norm	0.53/0.53	0.00	0.35/0.35	0.00
		l_1 norm	0.54/0.51	5.56	0.37/0.36	2.70
		l_2 norm	0.54/0.53	1.85	0.37/0.37	0.00
		Group lasso	0.53/0.52	1.89	0.36/0.36	0.00
l_1 sparse Group lasso		0.53/0.52	1.89	0.36/0.36	0.00	
	MobilePrune	0.54/0.42	22.22	0.36/0.29	19.44	

Table 3. Cont.

Dataset	Device	Penalty	Base/Pruned Response Delay (s)	Time Saving Percentage (%)	Based/Pruned Device Estimated Battery Use (%/h)	Battery Saving Percentage (%)
PAMAP2	Huawei P20	l_0 norm	2.64/2.72	0.00	0.76/0.79	0.00
		l_1 norm	2.74/0.45	83.58	0.79/0.53	32.91
		l_2 norm	2.67/2.56	4.12	0.78/0.78	0.00
		Group lasso	2.67/2.68	0.00	0.78/0.78	0.00
		l_1 sparse Group lasso	2.69/0.55	79.55	0.79/0.57	27.85
		MobilePrune	2.70/0.32	88.15	0.79/0.50	36.71
	OnePlus 8 Pro	l_0 norm	0.94/0.93	1.06	0.88/0.88	0.00
		l_1 norm	0.93/0.25	73.12	0.87/0.55	36.78
		l_2 norm	0.93/0.91	2.15	0.88/0.87	1.14
		Group lasso	0.94/0.95	0.00	0.89/0.89	0.00
l_1 sparse Group lasso		0.95/0.29	69.47	0.88/0.59	32.95	
	MobilePrune	0.94/0.21	77.66	0.87/0.54	37.93	

5.3. Ablation Studies

To demonstrate the efficacy and effectiveness of the l_0 sparse group lasso penalty, we performed a series of ablation studies on various DNN models. As shown in Table 4, if we only use l_0 norm penalty, there is no effect on a neuron or channel pruning as expected. Similarly, if we only employ the group lasso penalty, the pruned model still has more weights left. However, if we apply l_0 sparse group lasso, we can achieve pruned DNN models that are sparse in terms of both neurons (or channel) and weights. In addition, we compare our l_0 sparse group lasso model with l_1 sparse group lasso [59] on pruning DNN models. Table 4 shows their comparison on pruning various DNN models. More details can be found in Appendix A.3 and Appendix C. As shown in Table 4 and the results in supplementary, the l_0 sparse group lasso model significantly outperforms the l_1 sparse group lasso model in all aspects, which demonstrates its superiority in pruning DNN models.

Table 4. Ablation studies on various network models. (We mark the best performance as blue among different penalties for each model).

Network Model	Penalty	Base/Pruned Accuracy (%)	Original/Remaining Parameters (Mil)	FLOPs	Sparsity (%)
LetNet-300	l_0 norm	98.24/98.46	267 K/57.45 K	143.20	21.55
	Group lasso	98.24/98.17	267 K/32.06 K	39.70	12.01
	l_1 sparse group lasso	98.24/98.00	267 K/15.80 K	25.88	5.93
	l_0 sparse group lasso	98.24/98.23	267 K/5.25 K	25.79	1.97
LetNet-5	l_0 norm	99.12/99.20	431 K/321.0 K	2293.0	74.48
	Group lasso	99.12/99.11	431 K/8.81 K	187.00	2.04
	l_1 sparse group lasso	99.12/99.03	431 K/9.98 K	183.83	2.32
	l_0 sparse group lasso	99.12/99.11	431 K/2.31 K	113.50	0.54
VGG-like	l_0 norm	92.96/93.40	15 M/3.39 M	210.94	22.6
	Group lasso	92.96/92.47	15 M/0.84 M	78.07	5.60
	l_1 sparse group lasso	92.96/92.90	15 M/0.61 M	134.35	4.06
	l_0 sparse group lasso	92.96/92.94	15 M/0.60 M	77.83	4.00
ResNet-32	l_0 norm	95.29/95.68	7.42 M/6.74 M	993.11	90.84
	Group lasso	95.29/95.30	7.42 M/3.03 M	373.09	40.84
	l_1 sparse group lasso	95.29/95.04	7.42 M/5.66 M	735.12	76.28
	l_0 sparse group lasso	95.29/95.47	7.42 M/2.93 M	371.30	39.49
VGG-19	l_0 norm	61.56/61.99	138 M/19.29 M	1519.23	13.98
	Group lasso	61.56/53.25	138 M/5.93 M	683.99	4.30
	l_1 sparse group lasso	61.56/53.97	138 M/0.21 M	1282.82	0.15
	l_0 sparse group lasso	61.56/56.27	138 M/4.05 M	407.37	2.93

6. Conclusions

In this work, we proposed a new DNN pruning method MobilePrune, which is able to generate compact DNN models that are compatible with both cuDNN and hardware

acceleration. MobilePrune compress DNN models at both group and individual levels by using the novel ℓ_0 sparse group lasso regularization. We further developed a global convergent optimization algorithm MobilePrune based on PALM to directly train the proposed compression models without any relaxation or approximation. Furthermore, we developed several efficient algorithms to solve the proximal operators associated with ℓ_0 sparse group lasso with different grouping strategies, which is the key computation of our MobilePrune. We have performed empirical evaluations on several public benchmarks. Experimental results show that the proposed compression model outperforms existing state-of-the-art algorithms in terms of computational costs and prediction accuracy. MobilePrune has a great potential to design slim DNN models that can be deployed on dedicated hardware that uses a sparse systolic tensor array. More importantly, we deploy our system on the real android system on both Huawei P20 and OnePlus 8 Pro, and the performance of the algorithm on multiple Human Activity Recognition (HAR) tasks. The results show that MobilePrune achieves much lower energy consumption and higher pruning rate while still retaining high prediction accuracy.

There are other options to further compress the neural network models such as Neural Logic Circuits and Binary Neural Networks, which all use binary variables to represent inputs and hidden neurons. These two models are orthogonal to our methods, which means our pruning model could be adopted on Neural Logic Circuits, Binary Neural Networks and other neural network architectures designed for mobile systems. We will explore which mobile neural network could be better integrated with our network compression model in the future.

Author Contributions: Conceptualization: Y.S., Y.W. and J.M.; Methodology: K.Z., Y.W. and J.M.; Software: Y.S.; Validation: Y.S. and K.Z.; Writing—original draft preparation: Y.S., Z.C., Z.P., Y.W. and J.M.; Writing—review and editing: Y.S., X.P. and K.Z.; Supervision: P.L., Y.W. and J.M.; Project administration: Y.W. and J.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: We used the publicly available datasets—MNIST (<http://yann.lecun.com/exdb/mnist/>), CIFAR-10 (<http://www.cs.toronto.edu/~kriz/cifar.html>), Tiny-ImageNet (<https://paperswithcode.com/dataset/tiny-imagenet>), WISDM (<https://archive.ics.uci.edu/ml/datasets/WISDM+Smartphone+and+Smartwatch+Activity+and+Biometrics+Dataset+>), UCI-HAR (<https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones>), and PAMAP2 (<http://archive.ics.uci.edu/ml/datasets/pamap2+physical+activity+monitoring>). Both the source codes for the proposed model results and mobile application can be found in <https://github.com/yuboyubo/CNNCompression0> (accessed on 24 May 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Appendix A.1. The Convergence Analysis of Applying PALM Algorithm to Deep Learning Models

For simplicity, we prove the convergence of the PALM algorithm on a neural network that only has fully connected layers, and the regularization is added on the weight matrix of each layer. The proof can be easily extended to DNN models with regularization added on each neuron and DNN with convolutional layers.

Given a feed-forward neural network with $N - 1$ hidden layers, there are d_i neurons in the i -th layer. Let d_0 and d_N represent the number of neurons in the input and output layers, respectively. Therefore, the input data can be presented by $X := \{x_1, \dots, x_n\} \in \mathbb{R}^{d_0 \times n}$ and the output data can be denoted as $Y := \{y_1, \dots, y_n\} \in \mathbb{R}^{d_N \times n}$. Let $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ be the weight matrix between the $(i - 1)$ -th layer and the i -th layer. Here, in order to simplify

the notation, we let W_i absorb the bias of the i -th layer. We denote the collection of W_i as $\mathbf{W} = \{W_i\}_{i=1}^N$. The DNN model training problem can be formulated as

$$\min_{\mathbf{W}} : \mathbf{R}(\mathbf{W}; X, Y) + \sum_i r_i(W_i), \quad (\text{A1})$$

where $\mathbf{R}(\mathbf{W}; X, Y) = \frac{1}{n} \sum_n \ell(\Phi(\mathbf{W}; x_i), y_i)$, and $\ell(\cdot)$ is the loss function. $\Phi(\mathbf{W}; x_i) = \sigma_N(W_N \sigma_{N-1}(\dots W_2 \sigma_1(W_1 x_i)))$ is the DNN model with N layers of model parameter \mathbf{W} and σ_i is the activation function for neurons in the i -th layer. r_i is the regularization function applied to W_i . We make the assumptions for our DNN model as follows:

Assumption A1. Suppose that the DNN model satisfies the following assumptions:

1. The regularization functions $r_i, i = 1, \dots, N$ are lower semi-continuous.
2. The derivatives of the loss function ℓ and all activation functions $\sigma_i, i = 1, \dots, N$ are bounded and Lipschitz continuous.
3. The loss function ℓ , activation function $\sigma_i, \forall i$, and the regularization function $r_i, \forall i$ are either real analytic or semi-algebraic [37], and continuous on their domains.

Remark A1. The DNN model that satisfies the assumptions made in Assumption A1 could have squared, logistic, hinge, cross-entropy, or soft-max loss function and smooth activation functions, such as sigmoid, or hyperbolic tangent, and ℓ_1 norm, ℓ_2 norm, or ℓ_0 norm regularization term, and the assumption requires the activation functions to be smooth. Then, activation function, such as rectified linear unit (ReLU), does not satisfy the requirement. However, we can use the Softplus activation function or Swish activation function to replace ReLU.

Since $\mathbf{R}(\mathbf{W}; X, Y)$ depends on weight matrices $W_i, i = 1, \dots, N$ between the DNN layers, we rewrite (A1) as the following format in terms of only the independent variables

$$\min_{W_1, \dots, W_N} : \Psi(\mathbf{W}) := \mathbf{H}(W_1, \dots, W_N) + \sum_i r_i(W_i), \quad (\text{A2})$$

where \mathbf{H} is exactly the same as \mathbf{R} in (A1), but appears in a different form. We propose Algorithm A1 to solve (A2) and prove the convergence via the following theorem. In Algorithm A1, we use the proximal operator $\text{prox}_i^\sigma(x)$ at each step, which is defined as

$$\text{prox}_i^\sigma(x) = \arg \min_u \left\{ \frac{t}{2} \|u - x\| + \sigma(u) \right\}. \quad (\text{A3})$$

Theorem A1. Suppose Assumption A1 holds. As in [37], the sequence $\mathbf{W}^k = (W_1^k, \dots, W_N^k)$ generated by Algorithm A1 converges to the critical point of (A2) if the following conditions hold:

1. $\Psi(\mathbf{W})$ is a Kurdyka-Lojasiewicz (KL) function.
2. The partial gradient $\nabla_{W_i} \mathbf{H}, \forall i$ is Lipschitz continuous and there exist positive constants $\underline{l}_i, \bar{l}_i$ such that $c_i^k \in (\underline{l}_i, \bar{l}_i), k = 1, 2, \dots$
3. $\nabla_{\mathbf{W}} \mathbf{H}(W_1, \dots, W_N)$ has Lipschitz constant on any bounded set.

Proof. For the first condition, utilizing Proposition 1 and Lemma 3–6 in [60], we can easily prove that under our assumption $\Psi(\mathbf{W})$ is a KL function. For Equation (1) in the main text, it is also a KL function because ℓ_0 and group lasso are semi-algebraic [60,61]. We will add the simple proof in the Appendix.

According to our model assumption and Remark 1 in [62], we can show that $\Psi(\mathbf{W})$ is a KL function. For the second condition, based on our model Assumption A1 and Lemma A1 provided in the following, we know that $\nabla_{W_i} \mathbf{H}, \forall i$ is Lipschitz continuous. In addition, in Algorithm A1, we use backtracking strategy to estimate L_i at each iteration, therefore, there exist $\underline{l}_i = L_i^0$ and $\bar{l}_i = \bar{L}_i$, such that $c_i^k \in (L_i^0, \bar{L}_i), k = 1, 2, \dots$

For the last condition, based on Lemma A3 provided in the following, we have $\nabla_{\mathbf{W}}\mathbf{H}(W_1, \dots, W_N)$ is Lipschitz continuous for any bounded set. \square

Algorithm A1 PALM Algorithm for Deep Learning Models

Initialize $\eta > 1, L_1^0, \dots, L_N^0$, and W_1^0, \dots, W_N^0 .

for $k = 1, 2, \dots$ **do**

for $i = 1$ **to** N **do**

 Find the smallest i^k such that with $\bar{L}_i = \eta^{i^k} L_i^{k-1}$

$$\|\nabla_{W_i}\mathbf{H}(W_i^{k-1}) - \nabla_{W_i}\mathbf{H}(W_i^k)\| \leq \bar{L}_i \|W_i^{k-1} - W_i^k\| \tag{A4}$$

 Set $c_i^k = \eta^{i^k} L_i^{k-1}$ and compute

$$W_i^k = \text{prox}_{\frac{c_i^k}{c_i^k}}^{r_i} \left\{ W_i^{k-1} - \frac{1}{c_i^k} \nabla_{W_i}\mathbf{H}(W_i^{k-1}) \right\} \tag{A5}$$

end for

end for

Lemma A1. According to Assumption A1, the derivatives of the loss function and all the activation functions used in function \mathbf{R} in (A1) are bounded and Lipschitz continuous, then $\nabla_{W_i}\mathbf{H} = \nabla_{W_i}\mathbf{R}$ is also bounded and Lipschitz continuous.

Proof. The partial derivative of W_i can be written as

$$\nabla_{W_i}\mathbf{H} = \nabla_{W_i}\mathbf{R} = \frac{\partial \ell}{\partial \Phi} \cdot \frac{\partial \Phi}{\partial \sigma_N} \dots \frac{\partial \sigma_{i+1}}{\partial \sigma_i} \cdot \frac{\partial \sigma_i}{\partial W_i} \tag{A6}$$

From Assumption A1, we know both the derivatives of the loss function $\frac{\partial \ell}{\partial \Phi}$ and the derivatives of any activation function $\frac{\partial \sigma_{i+1}}{\partial \sigma_i}, \forall i$ are bounded and Lipschitz continuous. Based on Lemma A2 and (A6), we know the multiplication of bounded and Lipschitz continuous functions is still bounded and Lipschitz continuous. Therefore, $\nabla_{W_i}\mathbf{H}$ is bounded and Lipschitz continuous. \square

Lemma A2. If $f_i : \mathbb{R}^N \rightarrow \mathbb{R}, \forall i$ is bounded $|f_i(X)| < M_i, \forall i$ and Lipschitz continuous $\|f_i(X) - f_i(Y)\| \leq L_i \|X - Y\|$, then their multiplication $f_1(X)f_2(X)\dots f_n(X)$ is still Lipschitz continuous.

Proof. We first prove that the multiplication of two bounded Lipschitz continuous functions is still Lipschitz continuous as follows.

$$\begin{aligned} \|f_1(X)f_2(X) - f_1(Y)f_2(Y)\| &= \|f_1(X)f_2(X) - f_1(X)f_2(Y) + f_1(X)f_2(Y) - f_1(Y)f_2(Y)\| \\ &\leq \|f_1(X)f_2(X) - f_1(X)f_2(Y)\| + \|f_1(X)f_2(Y) - f_1(Y)f_2(Y)\| \\ &\leq |f_1(X)|L_2\|X - Y\| + |f_2(Y)|L_1\|X - Y\| \\ &\leq (M_1L_2 + M_2L_1)\|X - Y\|. \end{aligned} \tag{A7}$$

We can then extend the above to the multiplication of multiple functions and prove the lemma. \square

Lemma A3. If $\nabla_{W_i}\mathbf{H}, \forall i \in \{1, \dots, N\}$ is bounded and Lipschitz continuous, $\nabla_{\mathbf{W}}\mathbf{H}$ is also Lipschitz continuous.

Proof. Lemma A1 has shown that $\nabla_{W_i}\mathbf{H}, \forall i \in \{1, \dots, N\}$ is bounded and Lipschitz continuous. Here we want to show that $\nabla_{\mathbf{W}}\mathbf{H}$ is Lipschitz continuous on any bounded set. Define $\text{vec}(\mathbf{W}) = [\text{vec}(W_1)^T, \dots, \text{vec}(W_N)^T]^T$ as a vector where $\text{vec}()$ vectorizes a matrix by stacking its columns. Specifically, there exists a constant $M > 0$, such that for any $\mathbf{W}, \widehat{\mathbf{W}}$

$$\begin{aligned}
\|\nabla_{\mathbf{W}}\mathbf{H} - \nabla_{\widehat{\mathbf{W}}}\mathbf{H}\|^2 &= \|\nabla_{\text{vec}(\mathbf{W})}\mathbf{H} - \nabla_{\text{vec}(\widehat{\mathbf{W}})}\mathbf{H}\|^2 \\
&= \left\| [(\nabla_{\text{vec}(W_1)}\mathbf{H})^T, \dots, (\nabla_{\text{vec}(W_N)}\mathbf{H})^T] - [(\nabla_{\text{vec}(\widehat{W}_1)}\mathbf{H})^T, \dots, (\nabla_{\text{vec}(\widehat{W}_N)}\mathbf{H})^T] \right\|^2 \\
&= \sum_i \left\| (\nabla_{\text{vec}(W_i)}\mathbf{H})^T - (\nabla_{\text{vec}(\widehat{W}_i)}\mathbf{H})^T \right\|^2 \\
&\leq \sum_i L_i^2 \left\| \text{vec}(W_i) - \text{vec}(\widehat{W}_i) \right\|^2 \\
&\leq L_{\max}^2 \sum_i \left\| \text{vec}(W_i) - \text{vec}(\widehat{W}_i) \right\|^2 \\
&= L_{\max}^2 \left\| \mathbf{W} - \widehat{\mathbf{W}} \right\|^2,
\end{aligned} \tag{A8}$$

where L_{\max} is the largest Lipschitz constant among L_1, \dots, L_N . Then, we define $M = L_{\max}$ and we have

$$\|\nabla_{\mathbf{W}}\mathbf{H} - \nabla_{\widehat{\mathbf{W}}}\mathbf{H}\| \leq M \left\| \mathbf{W} - \widehat{\mathbf{W}} \right\|, \tag{A9}$$

as desired. \square

Theorem A2. *The sequence generated by Algorithm A1 $\{(W_1^k, \dots, W_N^k)\}$ converges to a critical point (W_1^*, \dots, W_N^*) of Equation (A2) at least in the sub-linear convergence rate, i.e. there exists some $\omega > 0$, such that*

$$\|(W_1^k, \dots, W_N^k) - (W_1^*, \dots, W_N^*)\| \leq \omega k^{\frac{1-\theta}{2\theta-1}}, \tag{A10}$$

where $\theta \in (\frac{1}{2}, 1)$.

Proof. Based on Theorem 2 in [63] and Theorem 6.4 in [61], we can prove the above theorem. \square

Appendix A.2. The Lemma Used in the Proof of Theorem 1

Lemma A4. *The proximal operator associated with the Euclidean norm $\|\cdot\|$ has a closed form solution:*

$$\arg \min_{\mathbf{x}} \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|^2 + \lambda \|\mathbf{x}\| = \max(\|\mathbf{y}\| - \lambda, 0) \frac{\mathbf{y}}{\|\mathbf{y}\|}, \tag{A11}$$

where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$ are both n -dimensional vectors.

This is a known result and has been used in previous study [64].

Appendix A.3. The Algorithm for ℓ_1 Sparse Group Lasso

In Section 5.3, we conduct experiments regarding ℓ_1 norm sparse group Lasso for comparison with our proposed ℓ_0 sparse group lasso. Here, we elaborate the ℓ_1 norm sparse group Lasso algorithm. We first consider the following proximal operator associated with ℓ_1 overlapping group Lasso regularization:

$$\pi_{\lambda_2}^{\lambda_1}(\mathbf{v}) = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \left\{ g_{\lambda_2}^{\lambda_1}(\mathbf{x}) \equiv \frac{1}{2} \|\mathbf{x} - \mathbf{v}\|^2 + \lambda_1 \|\mathbf{x}\|_1 + \lambda_2 \sum_{i=1}^k \|\mathbf{x}_{G_i}\| \right\}, \tag{A12}$$

where the regularization coefficients, λ_1 and λ_2 , are non-negative values, $i = 1, 2, \dots, k$, $G_i \subseteq \{1, 2, \dots, n\}$ denotes the indices corresponding to the i -th group. According to Theorem 1 in [59], $\pi_{\lambda_2}^{\lambda_1}(\cdot)$ can be derived from the following $\pi_0^{\lambda_1}(\cdot)$ and we present this conclusion in Lemma A5.

Lemma A5. Let $\mathbf{u} = \text{sgn}(\mathbf{v}) \odot \max(|\mathbf{v}| - \lambda_1, 0)$, and

$$\pi_{\lambda_2}^0(\mathbf{u}) = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \left\{ h_{\lambda_2}(\mathbf{x}) \equiv \frac{1}{2} \|\mathbf{x} - \mathbf{u}\|^2 + \lambda_2 \sum_{i=1}^g \|\mathbf{x}_{G_i}\| \right\}. \quad (\text{A13})$$

Then, $\pi_{\lambda_2}^{\lambda_1}(\mathbf{v}) = \pi_{\lambda_2}^0(\mathbf{u})$ holds.

According to Lemma A4, it is easy to verify that given $\mathbf{x}_{G_i} \cap \mathbf{x}_{G_j} = \emptyset$, the optimal \mathbf{x}_{G_i} minimizing $h_{\lambda_2}(\mathbf{x})$ is given by

$$\mathbf{x}_{G_i} = \max(\|\mathbf{u}_{G_i}\| - \lambda_2, 0) \frac{\mathbf{u}_{G_i}}{\|\mathbf{u}_{G_i}\|}, \quad (\text{A14})$$

where $i = 1, 2, \dots, k$. For the fully connected layers, we define the ℓ_1 overlapping group Lasso regularizer as

$$\begin{aligned} \Psi(W) &= \sum_{q=1}^Q \psi_{\lambda_2}^{\lambda_1}(W_{:,q}) = \sum_{q=1}^Q \lambda_1 \|W_{:,q}\|_1 + \lambda_2 \|W_{:,q}\| \\ &= \lambda_1 \|W\|_1 + \sum_{q=1}^Q \lambda_2 \|W_{:,q}\|, \end{aligned} \quad (\text{A15})$$

where $W_{:,q}$ represents the output weights of the q -th neuron of W . According to Lemma A5, let $\widehat{W} = \text{sgn}(W) \odot \max(|W| - \lambda_1, 0)$, and then $\psi_{\lambda_2}^{\lambda_1}(W_{:,q})$ can be reduced to $\psi_{\lambda_2}^0(\widehat{W}_{:,q})$, which can be solved via (A14) as follows.

$$W_{:,q} = \max(\|\widehat{W}_{:,q}\| - \lambda_2, 0) \frac{\widehat{W}_{:,q}}{\|\widehat{W}_{:,q}\|} \quad (\text{A16})$$

For the convolutional layers, we denote the ℓ_1 overlapping group Lasso regularizer as

$$\begin{aligned} \Phi(T) &= \sum_{c=1}^{N_c} \phi_{\beta_b}^{\beta_a}(T_{:,c,:}) = \sum_{c=1}^{N_c} \beta_a \|T_{:,c,:}\|_1 + \beta_b \sum_{h=1}^{N_h} \sum_{w=1}^{N_w} \|T_{:,c,h,w}\| \\ &= \beta_a \|T\|_1 + \sum_{c=1}^{N_c} \sum_{h=1}^{N_h} \sum_{w=1}^{N_w} \beta_b \|T_{:,c,h,w}\|, \end{aligned} \quad (\text{A17})$$

Likewise, let $\widehat{T}_{:,c,:} = \text{sgn}(T_{:,c,:}) \odot \max(|T_{:,c,:}| - \beta_a, 0)$, and then $\phi_{\beta_b}^{\beta_a}(T_{:,c,:})$ can be reduced to $\phi_{\beta_b}^0(\widehat{T}_{:,c,:})$, which can be solved via (A14) as follows.

$$T_{:,c,h,w} = \max(\|\widehat{T}_{:,c,h,w}\| - \beta_b, 0) \frac{\widehat{T}_{:,c,h,w}}{\|\widehat{T}_{:,c,h,w}\|} \quad (\text{A18})$$

By incorporating (A15) and (A17) into our model, the ℓ_1 norm sparse group Lasso regularized problem can be formulated as

$$\min_{\mathcal{W}} : L(\mathcal{W}; X, Y) + \sum_{i=1}^{N_f} \sum_{q=1}^Q \psi_{\lambda_2}^{\lambda_1}(W_{:,q}^{(i)}) + \sum_{j=1}^{N_l} \sum_{c=1}^{N_c} \phi_{\beta_b}^{\beta_a}(T_{:,c,:}^{(j)}). \quad (\text{A19})$$

We elaborate the DNN_PALM algorithm associated with ℓ_1 norm sparse group lasso in Algorithm A2, in which the partial derivatives are denoted as $H(W_{:,q}^{(i)}) = \nabla_{W_{:,q}^{(i)}} L(W_{:,q}^{(i)})$ and $F(T_{:,c,h,w}^j) = \nabla_{T_{:,c,h,w}^j} L(T_{:,c,h,w}^j)$ for fully connected layers and convolutional layers, respectively.

Algorithm A2 DNN_PALM Algorithm for ℓ_1 norm Group Lasso

```

Initialize  $\mu > 1, L_i^0 > 0, (W^{(i)})^0, \forall i$  and  $L_j^0 > 0, (T^{(j)})^0, \forall j$ .
for  $k = 1, 2, \dots$  do
  for  $i = 1$  to  $N_f$  do
    Let  $\omega^{k-1} = (W_{:,q}^{(i)})^{k-1}$  and  $\omega^k = (W_{:,q}^{(i)})^k, \forall q$ .
    Find the smallest  $\mathcal{L} = \mu^b L_i^{k-1}, b \in \mathbb{N}$ , such that  $\|H(\omega^{k-1}) - H(\omega^k)\| \leq \mathcal{L} \|\omega^{k-1} - \omega^k\|$ , where
     $\omega^k$  is computed via (A16).
  end for
  for  $j = 1$  to  $N_l$  do
    for  $c = 1$  to  $N_c$  do
      Let  $\gamma^{k-1} = (T_{:,c,h,w}^j)^{k-1}$  and  $\gamma^k = (T_{:,c,h,w}^j)^k, \forall h, w$ .
      Find the smallest  $\mathcal{L} = \mu^b L_j^{k-1}, b \in \mathbb{N}$ , such that  $\|F(\gamma^{k-1}) - F(\gamma^k)\| \leq \mathcal{L} \|\gamma^{k-1} - \gamma^k\|$ , where
       $\gamma^k$  is computed via (A18).
    end for
  end for
end for

```

Appendix B*Appendix B.1. Iterative Method*

Iterative pruning [4] is another effective method for obtaining a sparse network while maintaining high accuracy. As iterative pruning is orthogonal to our method, we can couple the two methods to obtain even better performance per number of parameters used; specifically, we replace the usual weight decay regularizer used in [4] with our ℓ_0 sparse group lasso regularizer. In practice, we find that, empirically, the iterative method is able to achieve better performance. All results reported in the paper are from the iterative method.

Appendix B.2. Hyper-Parameter Settings

In our experiments, all the baseline models were trained from scratch via stochastic gradient decent(SGD) with a momentum of 0.9. We trained the networks for 150 epochs on MNIST and 400 epochs on CIFAR-10 and Tiny-ImageNet with an initial learning rate of 0.1 and weight decay of 5e-4. The learning rate is decayed by a factor of 10 at 50, 100 on MNIST and at 100, 200 on CIFAR-10 and Tiny-ImageNet, respectively.

The experimental settings regarding hyper-parameters for all DNN models we used in the paper are summarized in Table A1. We employ iterative pruning strategy to prune all the models. Namely, the pruning process and the retraining process are performed alternately.

Table A1. List of hyper-parameters and their values(“-” denotes “not applicable”).

Hyper-Parameter	LeNet300	LeNet5	VGG-Like	ResNet-32	VGG-19	Description
learning rate	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	The learning rate used in retraining process
gradient momentum	0.9	0.9	0.9	0.9	0.9	The gradient momentum used in retraining process
weight decay	1×10^{-4}	1×10^{-5}	5×10^{-4}	1×10^{-4}	1×10^{-4}	The weight decay factor used in retraining process
minibatch size	1×10^2	6×10^2	1×10^3	3×10^2	4×10^2	The number of training samples over which each SGD update is computed during the retraining process
ℓ_0 norm factor	4×10^{-4}	2×10^{-4}	1×10^{-6}	1×10^{-8}	1×10^{-10}	The shrinkage coefficient for ℓ_0 norm regularization
channel factor	-	1×10^{-3}	$1 \times 10^{-3} - 1 \times 10^{-2}$ ¹	5×10^{-2}	5×10^{-2}	The shrinkage coefficient of channels for group Lasso
neuron factor	2×10^{-4}	2×10^{-4}	1×10^{-4}	0	1×10^{-2}	The shrinkage coefficient of neurons for group Lasso
filter size factor	-	1×10^{-3}	1×10^{-4}	1×10^{-4}	1×10^{-4}	The shrinkage coefficient of filter shapes for group Lasso
pruning frequency (epochs/minibatches)	10	10	1	2	1	No. of epochs(LeNet)/minibatches(VGGNet/ResNet) for pruning before retraining
retraining epochs	30	30	20	30	15	The number of retraining epochs after pruning
iterations	74	102	63	2	66	The number of iterations for obtaining the final results

¹ On VGG-like, the channel factor is adaptive and it is increased by 0.001 if its cross-entropy loss is not greater than the loss before performing pruning for the current mini-batch. Its range is [0.001,0.01].

Appendix C

Appendix C.1. Computational Efficiency

We want to mention that our main focus here is to compress DNN models via ℓ_0 sparse group lasso. Our main contribution is to solve the corresponding optimizations. Speed is not our primary focus. However, we still provide the run time of our methods as a reference. We compared the run time of our MobilePrune with the baseline methods (original methods without pruning involved), and we found the ratio of the run time of MobilePrune to the run time of the baseline method is around 5 on average on the same machine.

Appendix C.2. Results about the SSL

Table A2. Results about learning filter shapes in LeNet-5. (We highlight our MobilePrune results).

Method	Base/Pruned Accuracy (%)	Filter Size	Remaining Filters	Remaining Parameters	FLOPs (K)
Baseline	-	25–500	20–50	500–25,000	2464
SSL [24]	99.10/99.00	7–14	1–50	-	63.82
MobilePrune	99.12/99.03	14–9	4–16	46–26	51.21

Appendix C.3. Additional Ablation Studies

In this section, we perform ablation studies to compare DNN models regularized by the proposed ℓ_0 sparse group lasso and other DNN models that are regularized by its individual components. Specifically, we compare DNN models regularized by the proposed ℓ_0 sparse group lasso with DNN models regularized by ℓ_0 norm penalty (set group Lasso penalty to 0) and DNN models regularized by group Lasso penalty (set ℓ_0 norm penalty to 0), respectively. For fair comparison, for all regularized DNN models, we use the same hyper-parameter setting.

From Table A3 to Table A6, we observe that ℓ_0 norm penalty has no effect on structured pruning as expected and group Lasso penalty can effectively remove redundant structure components. Furthermore, the combination of ℓ_0 norm and group Lasso (our proposed ℓ_0 sparse group lasso penalty) can yield sparser models at both structure level and individual weight level. Notably, ℓ_0 norm can help group Lasso to remove more redundant structure components. Therefore, better acceleration in terms of FLOPs can be obtained by applying our proposed ℓ_0 sparse group lasso penalty. We want to mention that when we compute FLOPs, we do not take the individual weight sparsity into account. However, based on [11,65], lower FLOPs could be achieved by unitizing the sparsity on weight level on dedicated architectures.

Table A3. Ablation studies on LeNet-5 (Architecture: 20-50-800-500).

Penalty	Base/Pruned Accuracy (%)	Original/Remaining Parameters (K)	Pruned Architecture	Filter Size	FLOPs (K)	Sparsity (%)
ℓ_0 norm	99.12/99.20	431/321.00	20-50-800-500	25–500	2293.0	74.48
Group Lasso	99.12/99.11	431/8.81	4-19-301-29	25–99	187.00	2.04
ℓ_1 Group Lasso	99.12/99.03	431/9.98	4-17-271-82	23–99	183.83	2.32
ℓ_0 sparse group lasso	99.12/99.11	431/2.31	5-14-151-57	16–65	113.50	1.97

Table A4. Ablation studies on VGG-like.

Penalty	Base/Pruned Accuracy (%)	Original/Remaining Parameters (Mil)	Pruned Architecture	FLOPs (Mil)
ℓ_0 norm	92.96/93.40	15/3.39	18-43-92-99-229-240-246-507-504-486-241-114-428-168	210.94
Group Lasso	92.96/92.47	15/0.84	17-43-89-99-213-162-93-42-32-28-8-5-429-168	78.07
ℓ_1 Group Lasso	92.96/92.90	15/0.61	17-43-92-99-229-240-246-323-148-111-41-39-159-161	134.35
ℓ_0 sparse group lasso	92.96/92.94	15/0.60	17-43-87-99-201-185-80-37-27-25-9-4-368-167	77.83

Table A5. Ablation studies on ResNet-32.

Penalty	Base/Pruned Accuracy (%)	Original/Remaining Parameters (Mil)	FLOPs (Mil)	Sparsity (%)
ℓ_0 norm	95.29/95.68	7.42/6.74	993.11	90.84
Group Lasso	95.29/95.30	7.42/3.43	393.09	45.95
ℓ_1 Group Lasso	95.29/95.04	7.42/5.66	735.12	76.28
ℓ_0 sparse group lasso	95.29/95.47	7.42/2.93	371.30	39.49

Table A6. Ablation studies on VGG19.

Penalty	Test Accuracy (%)	Remaining Parameters (Mil)	Pruned Architecture	FLOPs (Mil)
Baseline	61.56	20.12	64-64-128-128-256-256-256-512-512-512-512-512-512-512	1592.53
ℓ_0 norm	61.99	19.29	45-64-114-128-256-256-256-512-511-512-509-512-512-512-512	1519.23
Group Lasso	53.25	5.93	23-61-80-128-122-114-164-253-255-322-412-462-23-93-129-512	683.99
ℓ_1 Group Lasso	53.97	0.21	29-64-109-128-254-246-254-256-510-509-509-509-512-512-484-512	1282.82
ℓ_0 sparse group lasso	56.27	4.05	19-48-57-102-79-83-100-179-219-273-317-341-256-158-116-512	407.37

Appendix C.4. Additional Comparison between ℓ_0 Sparse Group Lasso and ℓ_1 Norm Sparse Group Lasso

In addition, we compare the proposed ℓ_0 sparse group lasso with ℓ_1 norm group Lasso. The algorithm for DNN models with ℓ_1 norm group Lasso penalty is introduced in Algorithm A2. For hyper-parameter setting, we use the same parameters as ℓ_0 sparse group lasso penalty (as shown in Table A1) except the parameter for ℓ_1 norm. We search the parameter of ℓ_1 norm in [0.0001, 0.01] and report the best results in terms of the pruned test accuracy.

From Table A3 to Table A6, we find that ℓ_0 sparse group lasso penalized models outperform ℓ_1 sparse group Lasso penalized models in terms of test accuracy and FLOPs. For VGG19 model (Table A6), ℓ_1 sparse group Lasso penalized model can achieve the fewest number of parameters, but the pruned test accuracy and FLOPs are much worse than the ℓ_0 sparse group lasso penalized model.

Appendix C.5. The Effect of the Coefficient of ℓ_0 Norm Regularizer

In our proposed ℓ_0 sparse group lasso, ℓ_0 norm regularizer plays an important role of facilitating pruning networks effectively and efficiently, which has been shown through the results in ablation studies with and without ℓ_0 norm penalty. We further explore the effect of the strength of the ℓ_0 norm coefficient on the pruning performance. We vary the shrinkage strength for ℓ_0 norm penalty by a factor of 10 while keeping the other settings fixed.

As can be seen from Tables A7 and A8, the larger the ℓ_0 norm coefficient is, the more parameters are pruned as expected. Additionally, there is a trade-off between the shrinkage coefficients for ℓ_0 norm penalty and group Lasso penalty, which depends on the practical demand.

Table A7. The effect of the coefficient of ℓ_0 norm penalty on VGG-like.

ℓ_0 Penalty Coefficient	Base/Pruned Accuracy (%)	Original/Remaining Parameters (Mil)	Pruned Architecture	FLOPs (Mil)
1×10^{-4}	92.96/89.77	15/0.06	17-43-83-99-161-105-57-28-24-15-11-4-104-157	56.43
1×10^{-5}	92.96/92.19	15/0.30	16-43-85-99-171-155-75-33-23-18-10-3-264-167	66.82
1×10^{-6}	92.96/92.94	15/0.60	17-43-87-99-201-185-80-37-27-25-9-4-368-167	77.83
1×10^{-7}	92.96/92.54	15/0.74	17-43-87-99-213-188-91-40-26-27-9-4-400-168	81.64

Table A8. The effect of the coefficient of ℓ_0 norm penalty on ResNet-32.

ℓ_0 Penalty Coefficient	Base/Pruned Accuracy (%)	Original/Remaining Parameters (Mil)	FLOPs (Mil)	Sparsity
1×10^{-6}	95.29/95.11	7.42/2.06	330.90	27.76
1×10^{-7}	95.29/95.33	7.42/2.72	369.36	36.66
1×10^{-8}	95.29/95.47	7.42/2.93	77.83	39.49
1×10^{-9}	95.29/95.44	7.42/3.02	372.98	40.70

Appendix D

Appendix D.1. Har Dataset Description

Appendix D.1.1. Wisdm Dataset

The WISDM dataset [49], publicly available in [50], includes six activities (walking, jogging, walking upstairs, walking downstairs, sitting, and standing) that contain 3D (x, y, z) raw signals collected from the smartphone's accelerometer at a sampling rate of 20Hz. The total number of participants involved in the experiment is 36. These participants performed certain daily activities with an Android phone in their front leg pockets. This dataset has a total of 1,098,209 samples and each sample consists of a timestamp, a user ID, an activity ID, and the acceleration (x, y, z) raw data. Here for this dataset, 3 features are used—the gravitational acceleration (x, y, z) toward the center of the Earth. A sliding window approach with a window size of 80 readings (4 seconds) is used for segmenting the sequences with a 50% overlapping.

Appendix D.1.2. UCI-HAR Dataset

The UCI-HAR[51] dataset, publicly available in [52], includes six activities (walking, walking upstairs, walking downstairs, sitting, standing, and jogging) that contains 3D (x, y, z) raw signals extracted from the accelerometer and gyroscope of a smartphone at a constant rate of 50 Hz strapped to the waist of a subject. These raw signals were applying a noise filter to remove the noise first and then sampled in fixed-width sliding windows of 2.56 s (128 readings). This dataset was collected from a group of 30 volunteers. And all volunteers were instructed to follow an activity protocol and wore a Samsung Galaxy S II smartphone on their waist. The dataset includes a total of 10,299 samples including 7352 training samples (71.39%) and 2947 testing samples (28.61%). The dimension for each sample is 128 readings \times number of features with a 50% overlapping. Here for this dataset, 9 features are used—the acceleration signals (x, y, z) collected by the smartphone accelerometer in standard gravity units, the body acceleration signals (x, y, z) obtained by subtracting the gravity from the total acceleration, and the angular velocity vector (x, y, z) measured by the smartphone gyroscope.

Appendix D.1.3. PAMAP2 Dataset

The PAMAP2 dataset [53,54], publicly available in [55], contains data of different physical activities, performed by 9 subjects wearing 3 inertial measurement units and a heart rate monitor with a sampling rate of 100 Hz. According to the dataset's protocol, there are 12 physical activities—lying, sitting, standing, walking, running, cycling, nordic walking, ascending stairs, descending stairs, vacuum cleaning, and rope jumping. All the collected data above include two 3-axis accelerometer data, 3-axis gyroscope data, 3-axis magnetometer data, 3-axis orientation data, and temperature. This dataset has a total of 3,850,505 samples and each sample has a timestamp, a user id, an activity id, and the corresponding features. Here we pick 40 features listed by the dataset. Similar to the WISDM dataset, a sliding window approach with a window size of 128 readings (1.28 s) is used for segmenting the sequences with a 50% overlapping.

Appendix D.2. 1D CNN Model

In general, sensor data, such as accelerometers and gyroscopes, can be classified as time-series data. We can encode these time-series data as images to allow machines to recognize human behavior virtually. Inspired by the recent successes of deep learning techniques in computer vision, we convert sequence data into image data according to Gramian Angular Field transform algorithm [66] to obtain a one-dimensional convolutional neural network (1D CNN) model. The benefit of using CNNs for sequence classification is that we can learn from the raw time series data directly, and this method does not require domain expertise to manually engineer input features. This 1D CNN model can learn an internal representation of the time series data and could achieve good performance to models that fit on the version of the dataset with engineered features.

Similar to the general CNN model, the convolutional layer that uses the convolution kernel for the input data is the most essential part in CNN. The convolution kernel works as a filter and is activated by a non-linear activation function. In this paper, a sequential model with a PyTorch backend is built via Google Colab [56] and 5 consecutive 1D convolutional layers with 128 neurons each are selected. Each layer uses a ReLU activation function. In order to compress this CNN model, we cannot add a 1D max-pooling because of the random down-sampling. For the WISDM and PAMAP2 datasets, 10 convolution kernels are used for each layer. For the UCI-HAR dataset, 5 convolution kernels are used. All these values are selected based on tons of experiments. In addition, for the sake of efficiency, we select the number of epochs to be 150 for all three datasets during the training stage.

Appendix D.3. Data Pre-Processing

In order to provide a certain data dimension and improve the performance of the proposed 1D CNN Model, the above collected raw data need to be pre-processed as the following.

Appendix D.3.1. Re-Scaling and Standardization

If we use the above datasets' raw data directly to train our model, the final results may cause training bias because of those large values. In order to remove such bias, standardizing a dataset is necessary. Standardizing a dataset involves re-scaling the distribution of the values for each channel such that the mean is 0 and the standard deviation is 1, as shown in Equation (A20):

$$X_{ij} = \frac{X_{ij} - \text{mean}(X_i)}{\text{std}(X_i)} \quad (\text{A20})$$

where $i = 1, 2, \dots, n$ and n denotes the number of channels, $j = 1, 2, \dots, m$ and m denotes the number of elements in each channel.

Appendix D.3.2. Segmentation

As mentioned above, the input to the model consists of a data sequence extracted from the raw sensor data. The data were recorded continuously in the data collection process. In order to preserve the temporal relationship between the collected data points and their corresponding activity, a sliding window approach is used to segment the collected data points. For the HCI-HAR and PAMAP2 datasets, a fixed-length of 128 sliding windows with an overlapping rate of 50% is applied. For the WISDM dataset, a fixed-length of 80 sliding windows with an overlapping rate of 50% is applied. After segmenting the raw data, we obtain 27,455 samples for the WISDM dataset, 10,299 samples for the UCI-HAR dataset, and 30,356 samples for the PAMAP2 dataset. Next, we select 80% data samples randomly from WISDM and PAMAP2 datasets as the training samples while the remaining 20% data samples are the testing samples. For the UCI-HAR dataset, it is already split.

Appendix D.3.3. K-Fold Cross-Validation

In order to improve the proposed model's final performance, k-fold cross-validation is used after the above segmentation step. The principle of the k-fold cross-validation method is to split the input samples as the number of k groups. It can lead to a less biased or less positive assessment of the ability of the model than other methods [67]. All the training data samples are considered for both training and validation in a k-fold cross-validation approach. First, we divide the training data samples into k equal subsets. Then, we pick one subset as the validation set and the remaining $k - 1$ subsets as the training set. There are k different ways to select the validation set, and therefore we have k different pairs of testing and validation datasets. In this paper, we choose $k = 5$ and evaluate all the 5 different pairs of testing and validation datasets for our proposed method and all state-of-the-art pruning methods. The final training and validation dataset will be selected according to the final performance with the testing data set.

Appendix D.4. Hyper-Parameters Tuning

Hyper-parameters have a great impact on the deep learning model performance. In the following context, we will present how to select the training subset, validation subset based on the 5-fold cross-validation before the training stage, how to pick the learning rate during the training stage, how to select the model by the epochs during the training stage, and how to pick pruning threshold to compress the final model. The experiments are implemented on all three datasets and the model performance is evaluated by varying several model parameters.

Appendix D.4.1. Cross-Validation Tuning

In order to improve the proposed model's final performance, k-fold cross-validation is used after segmenting the input samples. This approach can lead to a less biased or less positive assessment of the ability of the model than other methods [67]. Table A9 shows the results on the test set corresponding to different validation set choices. The pruned accuracy results are obtained when we use the fold numbers 4, 5, and 1 as the validation set based on both pruned accuracy and the nonzero parameters' percent for the WISDM, UCI-HAR, PAMAP2 datasets, respectively.

Appendix D.4.2. Learning Rate Tuning

The learning rate is a hyper-parameter that controls how much to change the model in response to the estimated error each time the model weights are updated [68]. Table A9 demonstrates the experiment results of different learning-rate settings. For the WISDM dataset, we observe that the best performance of pruned accuracy and parameter remaining percentage is achieved when the learning rate equals 1.5×10^{-4} . For the UCI-HAR dataset, when the learning rate is 2×10^{-4} , we obtain the best results of pruned accuracy and parameter remaining percentage. For the PAMAP2 dataset, the best-pruned accuracy is achieved when the learning rate is 1.5×10^{-4} . To make the experiment settings consistent and comparable, we set the learning rate to be 1.0×10^{-4} for all three datasets.

Table A9. Impact of different cross-validation fold numbers and learning rates on the proposed ℓ_0 sparse group lasso approach on each HAR dataset—WISDM, UCI-HAR, and PAMAP2, respectively. (We highlight our selection in both fold number and learning rate for each dataset).

Dataset	Type	Value	Base/Pruned Accuracy (%)	Parameter Nonzero (%)	Parameter Remaining (%)	Node Remaining (%)
WISDM	Fold Number	1	93.52/92.68	11.64	32.49	57.42
		2	94.88/93.70	10.03	30.35	55.08
		3	94.45/93.48	9.45	27.97	52.13
		4	94.97/94.65	9.52	28.03	53.52
		5	93.52/92.68	11.64	32.49	57.42
	Learning Rate	1.0×10^{-5}	89.55/86.72	27.09	93.50	96.68
		5.0×10^{-5}	92.93/84.36	9.41	40.44	64.06
		1.0×10^{-4}	94.97 / 94.65	27.09	28.03	53.52
		1.5×10^{-4}	94.96/94.88	10.38	27.26	52.54
		1.0×10^{-4}	94.65/94.57	10.54	32.38	56.84
UCI-HAR	Fold Number	1	78.42/78.08	15.53	31.99	56.64
		2	89.89/89.28	32.49	64.29	80.27
		3	79.13/79.37	16.02	32.25	56.84
		4	78.22/78.22	18.69	40.02	63.48
		5	90.06/89.96	23.00	46.83	68.75
	Learning Rate	1.0×10^{-5}	85.27/85.51	77.98	94.66	97.27
		5.0×10^{-5}	89.38/89.24	16.69	85.77	92.58
		1.0×10^{-4}	90.06/89.96	23.00	46.83	68.75
		1.5×10^{-4}	90.94/90.91	16.69	31.04	56.45
		2.0×10^{-4}	90.40/90.43	13.24	29.10	54.10
PAMAP2	Fold Number	1	96.89/96.95	1.26	3.72	10.74
		2	92.29/92.28	1.27	3.15	10.35
		3	96.49/96.28	1.81	4.74	14.84
		4	95.08/94.99	1.20	3.42	10.55
		5	94.81/94.81	1.46	3.71	11.52
	Learning Rate	1.0×10^{-5}	93.63/85.80	7.93	28.61	49.22
		5.0×10^{-5}	94.25/93.89	3.90	11.81	28.32
		1.0×10^{-4}	96.89/96.95	1.26	3.72	10.74
		1.5×10^{-4}	96.57/96.62	1.12	2.36	7.62
		2.0×10^{-4}	94.89/94.99	0.68	2.02	7.62

Appendix D.4.3. Number of Epochs Tuning

The number of epochs is a hyper-parameter that defines the number of times that the learning algorithm will work through the entire training dataset. Figure A1a–c show the training accuracy, validation accuracy, and testing accuracy versus the number of epochs for WISDM, UCI-HAR, and PAMAP2 datasets, respectively. For all three datasets, the number of epochs is 150; however, we only pick the epoch number based on the highest accuracy of the validation set for each dataset. The validation dataset is different from the test dataset, but is instead used to give an unbiased estimate of our final model. Based on the final results, the highest validation accuracy occurs at 150, 102, 113 epochs for WISDM, UCI-HAR, and PAMAP2 datasets, respectively.

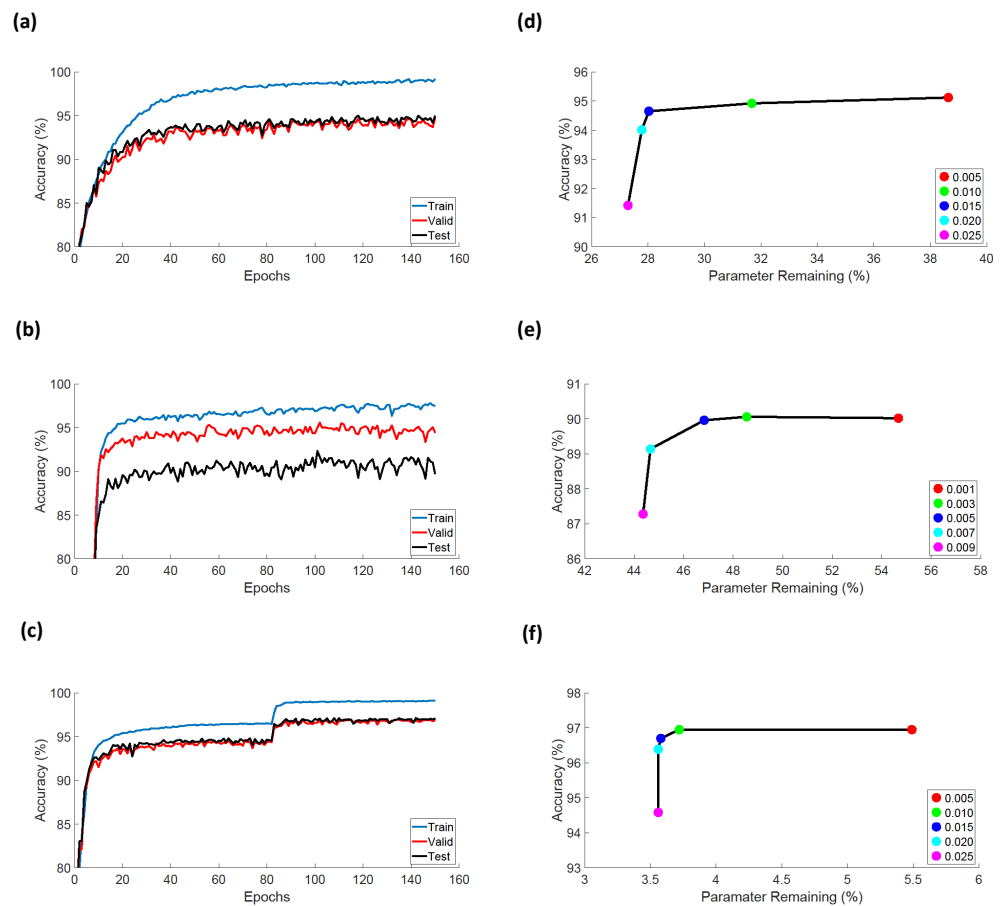


Figure A1. Impact of the different number of epochs (a–c) and prune thresholds (d–f) on the proposed ℓ_0 sparse group lasso approach on each HAR dataset—WISDM (a,d), UCI-HAR (b,e), and PAMAP2 (c,f), respectively.

Appendix D.4.4. Prune Threshold Tuning

For all the pruning methods including ℓ_0 sparse group lasso, after pruning, the weights cannot be exact zero due to the binary bits computation. Therefore, if weight is less than a costumed prune threshold, we set the weight to be zero in those pruned models. Figure A1d–f shows the experiment results of different prune threshold settings for WISDM, UCI-HAR, and PAMAP2 datasets, respectively. For the WISDM dataset, we observe that the best performance of pruned accuracy and parameter remaining percentage is achieved when the threshold equals 0.015. For the UCI-HAR dataset, when the pruned threshold is 0.005, we obtain the best results of pruned accuracy and parameter remaining percentage. For the PAMAP2 dataset, the best-pruned accuracy is achieved when the pruned threshold is 0.01.

References

1. Frankle, J.; Carbin, M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019. Available online: <https://arxiv.org/abs/1803.03635> (accessed on 24 May 2022).
2. Hassibi, B.; Stork, D. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5 (NIPS 1992)*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1993.
3. Castellano, G.; Fanelli, A.M.; Pelillo, M. An iterative pruning algorithm for feedforward neural networks. *IEEE Trans. Neural Netw.* **1997**, *8*, 519–531. [[CrossRef](#)] [[PubMed](#)]
4. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28*; Curran Associates, Inc.: Red Hook, NY, USA, 2015; pp. 1135–1143.

5. Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E. cuDNN: Efficient Primitives for Deep Learning. *arXiv* **2014**, arXiv:1410.0759.
6. Ding, X.; Ding, G.; Guo, Y.; Han, J.; Yan, C. Approximated oracle filter pruning for destructive cnn width optimization. In Proceedings of the 36th International Conference on Machine Learning, PMLR 97, Long Beach, CA, USA, 9–15 June 2019; pp. 1607–1616.
7. Neklyudov, K.; Molchanov, D.; Ashukha, A.; Vetrov, D.P. Structured Bayesian Pruning via Log-Normal Multiplicative Noise. In *Advances in Neural Information Processing Systems 30*; Curran Associates, Inc.: Red Hook, NY, USA, 2017; pp. 6775–6784.
8. Louizos, C.; Welling, M.; Kingma, D.P. Learning Sparse Neural Networks through L₀ Regularization. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
9. Louizos, C.; Ullrich, K.; Welling, M. Bayesian compression for deep learning. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 3288–3298.
10. Liu, Z.G.; Whatmough, P.N.; Mattina, M. Sparse Systolic Tensor Array for Efficient CNN Hardware Acceleration. *arXiv* **2020**, arXiv:2009.02381.
11. Liu, Z.; Whatmough, P.N.; Mattina, M. Systolic Tensor Array: An Efficient Structured-Sparse GEMM Accelerator for Mobile CNN Inference. *IEEE Comput. Archit. Lett.* **2020**, *19*, 34–37. [[CrossRef](#)]
12. Pilanci, M.; Wainwright, M.J.; El Ghaoui, L. Sparse learning via Boolean relaxations. *Math. Prog.* **2015**, *151*, 63–87. [[CrossRef](#)]
13. Bertsimas, D.; Pauphilet, J.; Parys, B.V. Sparse Regression: Scalable algorithms and empirical performance. *arXiv* **2019**, arXiv:1902.06547.
14. Tibshirani, R. Regression Shrinkage and Selection Via the Lasso. *J. R. Stat. Soc. Ser. B* **1994**, *58*, 267–288. [[CrossRef](#)]
15. Zou, H.; Hastie, T. Regularization and variable selection via the Elastic Net. *J. R. Stat. Soc. Ser. B* **2005**, *67*, 301–320. [[CrossRef](#)]
16. Fan, J.; Li, R. Variable selection via nonconcave penalized likelihood and its oracle properties. *J. Am. Stat. Assoc.* **2001**, *96*, 1348–1360. [[CrossRef](#)]
17. Zhang, C.H. Nearly unbiased variable selection under minimax concave penalty. *Ann. Stat.* **2010**, *38*, 894–942. [[CrossRef](#)]
18. Hazimeh, H.; Mazumder, R. Fast Best Subset Selection: Coordinate Descent and Local Combinatorial Optimization Algorithms. *arXiv* **2018**, arXiv:1803.01454.
19. Guo, Y.; Yao, A.; Chen, Y. Dynamic Network Surgery for Efficient DNNs. In *Advances in Neural Information Processing Systems 29*; Curran Associates, Inc.: Red Hook, NY, USA, 2016; pp. 1379–1387.
20. Ding, X.; Ding, G.; Zhou, X.; Guo, Y.; Han, J.; Liu, J. Global Sparse Momentum SGD for Pruning Very Deep Neural Networks. In *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 6379–6391.
21. Xiao, X.; Wang, Z.; Rajasekaran, S. AutoPrune: Automatic Network Pruning by Regularizing Auxiliary Parameters. In *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 13681–13691.
22. Alvarez, J.M.; Salzmann, M. Learning the Number of Neurons in Deep Networks. In *Advances in Neural Information Processing Systems 29*; Curran Associates, Inc.: Red Hook, NY, USA, 2016; pp. 2270–2278.
23. Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; Pensky, M. Sparse Convolutional Neural Networks. In Proceedings of the Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015. Available online: <https://ieeexplore.ieee.org/document/7298681> (accessed on 24 May 2022).
24. Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; Li, H. Learning Structured Sparsity in Deep Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016. Available online: <https://arxiv.org/abs/1608.03665> (accessed on 24 May 2022).
25. Yang, C.; Yang, Z.; Khattak, A.M.; Yang, L.; Zhang, W.; Gao, W.; Wang, M. Structured Pruning of Convolutional Neural Networks via L1 Regularization. *IEEE Access* **2019**, *7*, 106385–106394. [[CrossRef](#)]
26. Yang, H.; Gui, S.; Zhu, Y.; Liu, J. Automatic Neural Network Compression by Sparsity-Quantization Joint Learning: A Constrained Optimization-based Approach. *arXiv* **2020**, arXiv:1910.05897.
27. Zhang, T.; Ye, S.; Zhang, K.; Tang, J.; Wen, W.; Fardad, M.; Wang, Y. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers. *arXiv* **2018**, arXiv:1804.03294.
28. He, Y.; Liu, P.; Wang, Z.; Hu, Z.; Yang, Y. Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019.
29. Ren, A.; Zhang, T.; Ye, S.; Li, J.; Xu, W.; Qian, X.; Lin, X.; Wang, Y. ADMM-NN: An Algorithm-Hardware Co-Design Framework of DNNs Using Alternating Direction Methods of Multipliers. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), Providence, RI, USA, 13–17 April 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 925–938. [[CrossRef](#)]
30. Liu, J.; Ye, J. Moreau-Yosida Regularization for Grouped Tree Structure Learning. In *Advances in Neural Information Processing Systems 23*; Lafferty, J.D., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2010; pp. 1459–1467.
31. Collins, M.D.; Kohli, P. Memory Bounded Deep Convolutional Networks. *arXiv* **2014**, arXiv:1412.1442.
32. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning Filters for Efficient ConvNets. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.

33. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2736–2744.
34. Yoon, J.; Hwang, S.J. Combined Group and Exclusive Sparsity for Deep Neural Networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; PMLR: Proceedings of Machine Learning Research; Precup, D., Teh, Y.W., Eds.; International Convention Centre: Sydney, Australia, 2017; Volume 70, pp. 3958–3966.
35. Scardapane, S.; Comminiello, D.; Hussain, A.; Uncini, A. Group sparse regularization for deep neural networks. *Neurocomputing* **2017**, *241*, 81–89. [[CrossRef](#)]
36. Liu, Z.G.; Whatmough, P.N.; Zhu, Y.; Mattina, M. S2TA: Exploiting Structured Sparsity for Energy-Efficient Mobile CNN Acceleration. In Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Korea, 2–6 April 2022.
37. Bolte, J.; Sabach, S.; Teboulle, M. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Math. Program.* **2014**, *146*, 459–494. [[CrossRef](#)]
38. Beck, A.; Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *Siam J. Imaging Sci.* **2009**, *2*, 183–202. [[CrossRef](#)]
39. Dai, B.; Zhu, C.; Guo, B.; Wipf, D. Compressing Neural Networks using the Variational Information Bottleneck. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018.
40. LeCun, Y.; Denker, J.S.; Solla, S.A. Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2*; Touretzky, D.S., Ed.; Morgan-Kaufmann: Burlington, MA, USA, 1990; pp. 598–605.
41. Zeng, W.; Urtasun, R. MLPPrune: Multi-Layer Pruning for Automated Neural Network Compression. 2019. Available online: <https://openreview.net/forum?id=r1g5b2RcKm> (accessed on 24 May 2022).
42. Wang, C.; Grosse, R.; Fidler, S.; Zhang, G. EigenDamage: Structured Pruning in the Kronecker-Factored Eigenbasis. In Proceedings of the Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 6566–6575.
43. Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [[CrossRef](#)]
44. Zagoruyko, S. 92.45% on CIFAR-10 in Torch. 2015. Available online: <http://torch.ch/blog/2015/07/30/cifar.html> (accessed on 26 May 2022).
45. Zhang, G.; Wang, C.; Xu, B.; Grosse, R. Three Mechanisms of Weight Decay Regularization. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
46. Krizhevsky, A.; Nair, V.; Hinton, G. CIFAR-10 (Canadian Institute for Advanced Research). Available online: <http://www.cs.toronto.edu/~kriz/cifar.html> (accessed on 24 May 2022).
47. Le, Y.; Yang, X.S. Tiny ImageNet Visual Recognition Challenge. *CS 231N* **2015**, *7*, 3.
48. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
49. Kwapisz, J.R.; Weiss, G.M.; Moore, S.A. Activity Recognition Using Cell Phone Accelerometers. *SIGKDD Explor. Newsl.* **2011**, *12*, 74–82. [[CrossRef](#)]
50. WISDM: Wireless Sensor Data Mining. Available online: <https://www.cis.fordham.edu/wisdm/dataset.php> (accessed on 24 May 2022).
51. Anguita, D.; Ghio, A.; Oneto, L.; Parra, X.; Reyes-Ortiz, J. A Public Domain Dataset for Human Activity Recognition using Smartphones. In Proceedings of the 21th international European symposium on artificial neural networks, computational intelligence and machine learning, Bruges, Belgium, 24–26 April 2013.
52. Human Activity Recognition Using Smartphones Data Set. Available online: <https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones> (accessed on 24 May 2022).
53. Reiss, A.; Stricker, D. Introducing a New Benchmarked Dataset for Activity Monitoring. In Proceedings of the 2012 16th International Symposium on Wearable Computers, Newcastle, UK, 18–22 June 2012. [[CrossRef](#)]
54. Reiss, A.; Stricker, D. Creating and Benchmarking a New Dataset for Physical Activity Monitoring. In *PETRA '12, Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments, Crete, Greece, 6–8 June 2012*; Association for Computing Machinery: New York, NY, USA, 2012. [[CrossRef](#)]
55. PAMAP2 Physical Activity Monitoring Data Set. Available online: <https://archive.ics.uci.edu/ml/datasets/PAMAP2+Physical+Activity+Monitoring> (accessed on 24 May 2022).
56. Google Colab. Available online: <https://research.google.com/colaboratory/faq.html> (accessed on 24 May 2022).
57. Pytorch Mobile. Available online: <https://pytorch.org/mobile/android/> (accessed on 24 May 2022).
58. Profile Battery Usage with BatteryStats and Battery Historian. Available online: <https://developer.android.com/topic/performance/power/setup-battery-historian> (accessed on 24 May 2022).
59. Yuan, L.; Liu, J.; Ye, J. Efficient Methods for Overlapping Group Lasso. In *Advances in Neural Information Processing Systems 24*; Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2011; pp. 352–360.

60. Zeng, J.; Lau, T.T.K.; Lin, S.B.; Yao, Y. Global convergence of block coordinate descent in deep learning. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, CA, USA, 9–15 June 2019. Available online: <https://arXiv:1803.00225> (accessed on 24 May 2022).
61. Bao, C.; Ji, H.; Quan, Y.; Shen, Z. L0 norm based dictionary learning by proximal methods with global convergence. In Proceedings of the Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014. [[CrossRef](#)]
62. Lau, T.T.K.; Zeng, J.; Wu, B.; Yao, Y. A Proximal Block Coordinate Descent Algorithm for Deep Neural Network Training. In Proceedings of the 6th International Conference on Learning Representations, ICLR 2018—Workshop Track Proceedings, Vancouver, BC, Canada, 3 May–30 April 2018. Available online: arxiv.org/abs/1803.09082 (accessed on 24 May 2022).
63. Attouch, H.; Bolte, J. On the convergence of the proximal algorithm for nonsmooth functions involving analytic features. *Math. Program.* **2009**, *116*, 5–16. [[CrossRef](#)]
64. Bach, F.R.; Mairal, J.; Ponce, J. Convex Sparse Matrix Factorizations. *arXiv* **2008**, arXiv:0812.1869
65. Shomron, G.; Weiser, U. Non-Blocking Simultaneous Multithreading: Embracing the Resiliency of Deep Neural Networks. In Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Athens, Greece, 17–21 October 2020. Available online: arxiv.org/abs/2004.09309 (accessed on 24 May 2022).
66. Wang, Z.; Oates, T. Imaging Time-Series to Improve Classification and Imputation. *arXiv* **2015**, arXiv:1506.00327.
67. Tamilarasi, P.; Rani, R. Diagnosis of Crime Rate against Women using k-fold Cross Validation through Machine Learning. In Proceedings of the 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 11–13 March 2020; pp. 1034–1038. [[CrossRef](#)]
68. Brownlee, J. Understand the Impact of Learning Rate on Neural Network Performance. 2019. Available online: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks> (accessed on 24 May 2022).