Applied Intelligence (2023) 53:24603–24618
https://doi.org/10.1007/s10489-023-04740-z

Recursive least squares method for training and pruning convolutional neural networks

Tianzong Yu[1] · Chunyuan Zhang[1] · Meng Ma[1] · Yuan Wang[1]

Accepted: 25 May 2023 / Published online: 26 July 2023
© The Author(s) 2023

Abstract

Convolutional neural networks (CNNs) have shown good performance in many practical applications. However, their high computational and storage requirements make them difficult to deploy on resource-constrained devices. To address this issue, in this paper, we propose a novel iterative structured pruning algorithm for CNNs based on the recursive least squares (RLS) optimization. Our algorithm combines inverse input autocorrelation matrices with weight matrices to evaluate and prune unimportant input channels or nodes in each CNN layer and performs the next pruning operation when the testing loss is tuned down to the last unpruned level. Our algorithm can be used to prune feedforward neural networks (FNNs) as well. The fast convergence speed of the RLS optimization allows our algorithm to prune CNNs and FNNs multiple times in a small number of epochs. We validate its effectiveness in pruning VGG-16 and ResNet-50 on CIFAR-10 and CIFAR-100 and pruning a three-layer FNN on MNIST. Compared with four popular pruning algorithms, our algorithm can adaptively prune CNNs according to the learning task difficulty and can effectively prune CNNs and FNNs with a small or even no reduction in accuracy. In addition, our algorithm can prune the original sample features in the input layer.

Keywords Convolutional neural network · Model compression · Structured pruning · Iterative pruning · Recursive least squares

1 Introduction

Convolutional neural networks (CNNs) are the most widely used class of deep neural networks (DNNs) [1–3]. CNNs are well suited for handling computer vision tasks [4–7] since they can extract sample features from images at different abstract levels through convolutional and pooling mechanisms [8]. However, CNNs generally have high computational and storage costs, which hinder their widespread applications to some extent [9, 10]. In particular, in the past decade, mobile devices, such as smartphones, wearable devices and drones, have been increasingly used. There remains a growing demand to deploy CNNs with these devices, but their computational and storage capacities are much lower than those of conventional computers [11]. Therefore, how to compress CNNs has become an important research focus in deep learning.

✉ Chunyuan Zhang
zcy@hainanu.edu.cn

[1] School of Computer Science and Technology, Hainan University, Renmin Street, Haikou 570228, Hainan, China

At present, five categories of model compression algorithms have been proposed for CNNs [12]. The first category is network pruning, which prunes redundant channels or nodes [13, 14]. The second category is parameter quantization, which reduces the bits of all parameters to reduce the computational and storage costs [15, 16]. The third category is low-rank factorization, which decomposes three-dimensional filters into two-dimensional filters [17]. The fourth category is filter compacting, which uses compact filters to replace loose and overparameterized filters [18]. The last category is knowledge distillation, in which knowledge is acquired from the original network and used to generate a smaller network [19, 20]. Among these categories, network pruning has received the most research attention [21].Thus, we focus on this type of model compression algorithms in this paper.

Network pruning methods can be further classified into unstructured and structured pruning methods [22]. In theory, unstructured pruning methods can prune arbitrary redundant nodes in convolutional layers and achieve high compression ratios. However, unstructured pruning methods are difficult to implement since they destroy the form of weight matrices.

To address this issue, almost all existing unstructured pruning algorithms, such as Optimal Brain Damage [23], Soft Channel Pruning [24] and ℓ_0 Minimization [25], zero out the unimportant weights instead of really pruning the redundant nodes in simulation experiments. In contrast, structured pruning methods aim to remove unimportant channels. These methods preserve the structure of weight matrices and are thus more practical and popular, although their pruning granularity is coarse.

Structured pruning methods usually include three stages: training, pruning and fine-tuning (also called retraining in some papers) [26, 27]. According to the number of pruning operations, these methods can be divided into one-shot structured pruning and iterative structured pruning [28]. The former performs pruning and fine-tuning only once and thus requires fewer epochs to obtain the compressed model. However, its compression ratio and accuracy rely heavily on the given pruning ratio. In other words, it is often difficult to obtain the optimal compressed model with one-shot pruning. In contrast, the latter performs multiple pruning and fine-tuning operations, which may lead to better results; however, multiple operations are very time-consuming, especially for large-scale neural networks. There is still much debate about what kind of structured pruning approach is best for different scenarios.

In recent years, researchers have proposed many structured pruning algorithms. For example, Li et al. proposed a one-shot pruning algorithm called the ℓ_1-norm [29], which evaluates and prunes unimportant output channels by using ℓ_1 regularization for the weights of the convolutional layers. Liu et al. proposed another one-shot pruning algorithm called Network Slimming [30], which prunes channels by using ℓ_1 regularization for the scaling factors in the batch normalization layers. Molchanov et al. proposed two iterative pruning algorithms called Taylor FO [31] and Taylor SO [32], which use the first- and second-order Taylor expansions to estimate the contribution of each channel to the final loss, respectively, and remove the channels with scores smaller than a given threshold. Chen et al. proposed another iterative pruning algorithm called Collaborative Channel Pruning [33], which evaluates and removes unimportant channels by combining the convolution layer weights and batch normalization layer scaling factors.

Although researchers claim that these algorithms can effectively compress CNNs, they still have three common shortcomings. The first shortcoming is that they use stochastic gradient descent (SGD) to optimize CNNs during the training and fine-tuning stages. It is well known that SGD converges slowly and can be difficult to tune [34], which results in these algorithms requiring more training epochs. The second shortcoming is that they mainly use the weight magnitude to prune unimportant output channels. In fact, the training and fine-tuning results are influenced by the dataset, and the redundancy of the input features in each layer is the main reason to prune channels. Therefore, the weight magnitude cannot be used to evaluate the redundancy accurately. The third shortcoming is that the pruning ratio is manually and empirically set to a fixed value by users, which may cause underpruning or overpruning. In addition to these three shortcomings, existing iterative structured pruning algorithms have another shortcoming in that the pruning times and the repruning timing are manually set by users.

To overcome the above shortcomings, we propose a novel iterative structured pruning algorithm in this paper. In our previous work [35], we proposed a recursive least squares (RLS) optimization algorithm, which can be viewed as a special SGD algorithm with the inverse input autocorrelation matrix as the learning rate. Compared with SGD and Adam optimization, the RLS optimization has better convergence speed and quality. Our proposed algorithm is based on this optimization algorithm. In addition to using the RLS optimization to train and fine-tune CNNs, it combines inverse input autocorrelation matrices with weight matrices to evaluate and prune unimportant input channels or nodes and automatically performs the next pruning operation when the testing loss is tuned down to the last unpruned level. Our algorithm can also be used for pruning feedforward neural networks (FNNs). We validate its effectiveness in pruning VGG-16, ResNet-50 and an FNN on the CIFAR-10, CIFAR-100 and MNIST datasets. Compared with existing iterative pruning algorithms, our algorithm can prune CNNs and FNNs multiple times in a smaller number of epochs and more effectively prune CNNs and FNNs with a smaller accuracy loss. In addition, it can adaptively prune CNNs according to the difficulty of the learning task.

The key contributions of this paper can be summarized as follows:

1) We use the RLS optimization rather than SGD to accelerate our algorithm and all comparative pruning algorithms used in the experiments.
2) We present a novel iterative structured pruning algorithm that combines inverse input autocorrelation matrices and weight matrices to evaluate and prune unimportant input channels or nodes.
3) We suggest the testing loss as the repruning criterion of our algorithm and all comparative iterative pruning algorithms. Each repruning operation is performed when the testing loss is tuned down to the last unpruned level.
4) We conduct extensive experiments to verify the effectiveness of our algorithm. Our algorithm can effectively reduce both the number of floating-point operations (FLOPs) and number of parameters for CNNs and FNNs with a small accuracy loss. In particular, the experiments on FNNs show that our algorithm can prune the original sample features.

The remainder of this paper is organized as follows: In Section 2, we review the RLS algorithm and the RLS optimization for CNNs. In Section 3, we introduce our algorithm in detail. In Section 4, we present the experimental settings and results. Finally, we summarize this paper in Section 5.

2 Background

In this section, we introduce the background knowledge and some notations used in this paper. We first review the derivation of the RLS algorithm and then review the learning mechanism of CNNs with the RLS optimization.

2.1 Recursive least squares

RLS is a popular adaptive filtering algorithm with fast convergence speed. This algorithm recursively determines the weights that minimize the weighted linear least squares loss function based on the input signal [36]. Compared with the linear least squares algorithm, it is more suitable for online learning.

Let $\mathbb{X}_t = \{\mathbf{x}_1, \cdots, \mathbf{x}_t\}$ denote all sample inputs from the starting step to the current step, and let $\mathbb{Y}_t^* = \{y_1^*, \cdots, y_t^*\}$ denote the corresponding target outputs. On this basis, the quadratic minimization problem solved by the RLS algorithm over time t is defined as

$$\mathbf{w}_t = \arg\min_{\boldsymbol{w}} \frac{1}{2} \sum_{i=1}^{t} \lambda^{t-i} (y_i^* - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)^2 \tag{1}$$

where $\mathbf{w}$ is the weight vector, and $\lambda \in (0, 1]$ is the forgetting factor which enhances the importance of recent data over older data [37]. Let $\nabla_{\boldsymbol{w}} \frac{1}{2} \sum_{i=1}^{t} \lambda^{t-i} (y_i^* - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)^2 = \mathbf{0}$. Then, we easily obtain

$$\boldsymbol{w} = (\sum_{i=1}^{t} \lambda^{t-i} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}})^{-1} \sum_{i=1}^{t} \lambda^{t-i} \boldsymbol{x}_i y_i^* \tag{2}$$

We define $\mathbf{A}_t$ and $\mathbf{b}_t$ as follows:

$$\mathbf{A}_t = \sum_{i=1}^{t} \lambda^{t-i} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}} \tag{3}$$

$$\mathbf{b}_t = \sum_{i=1}^{t} \lambda^{t-i} \mathbf{x}_i y_i^* \tag{4}$$

Based on (2), the solution $\mathbf{w}_t$ to (1) can be derived as

$$\mathbf{w}_t = \mathbf{A}_t^{-1} \mathbf{b}_t \tag{5}$$

To avoid calculating the inverse of $\mathbf{A}_t$ in (5), we define the inverse input autocorrelation matrix $\mathbf{P}_t = (\mathbf{A}_t)^{-1}$. Equations (3) and (4) show that $\mathbf{A}_t$ and $\mathbf{b}_t$ can be computed recursively as follows:

$$\mathbf{A}_t = \lambda \mathbf{A}_{t-1} + \mathbf{x}_t \mathbf{x}_t^{\mathrm{T}} \tag{6}$$

$$\mathbf{b}_t = \lambda \mathbf{b}_{t-1} + \mathbf{x}_t y_t^* \tag{7}$$

Using Sherman-Morrison matrix inversion formula [38] with (6), we obtain

$$\mathbf{P}_t = \frac{1}{\lambda} \mathbf{P}_{t-1} - \frac{1}{\lambda h_t} \mathbf{u}_t (\mathbf{u}_t)^{\mathrm{T}} \tag{8}$$

where $\mathbf{u}_t$ and h_t are defined as follows:

$$\mathbf{u}_t = \mathbf{P}_{t-1} \mathbf{x}_t \tag{9}$$

$$h_t = \lambda + \mathbf{u}_t^{\mathrm{T}} \mathbf{x}_t \tag{10}$$

Substituting (7) and (8) into (5), we obtain

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \frac{1}{h_t} \mathbf{u}_t e_t \tag{11}$$

where e_t is defined as

$$e_t = \mathbf{w}_{t-1}^{\mathrm{T}} \mathbf{x}_t - y_t^* \tag{12}$$

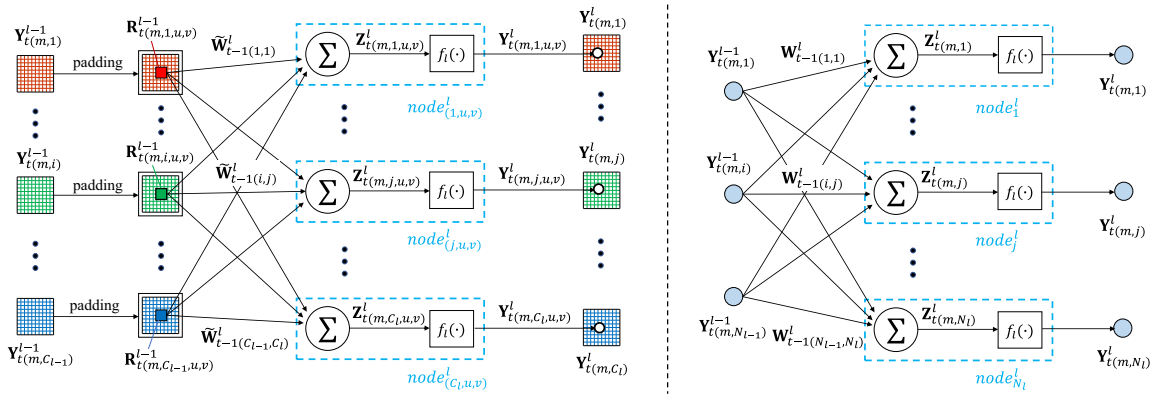
Finally, we obtain the RLS algorithm, which is defined by (8) to (12).

2.2 CNNs with RLS optimization

The RLS optimization is a special type of SGD algorithm with the inverse input autocorrelation matrix as the learning rate [35]. Due to the fast convergence speed of the RLS algorithm, it can efficiently optimize CNNs. A CNN generally consists of an input layer followed by some number of convolutional layers, pooling layers and fully-connected layers [39]. Since the pooling layers have no learnable weights, we need to review the RLS optimization only for the convolutional and fully-connected layers.

Let $\mathbf{Y}_t^0$ and $\mathbf{Y}_t^*$ denote the input and the target output of the current training minibatch, respectively, and let L denote the total number of convolutional and fully-connected layers. The forward-propagation learning of the m^{th} sample in the current minibatch in the l^{th} CNN layer is illustrated in Fig. 1. For brevity, we omit the bias term in each layer. Based on these notations, we briefly introduce the RLS optimization rules for CNNs. According to [35], the recursive update rule of the inverse input autocorrelation matrix $\mathbf{P}_t^l$ in the l^{th} layer is defined as

$$\mathbf{P}_t^l \approx \frac{1}{\lambda} \mathbf{P}_{t-1}^l - \frac{k}{\lambda h_t^l} \mathbf{u}_t^l (\mathbf{u}_t^l)^{\mathrm{T}} \tag{13}$$



(a) A convolutional layer (b) A fully-connected layer

Fig. 1 Forward-propagation learning of the m^{th} sample in the current minibatch in the l^{th} CNN layer. (a): $\mathbf{Y}^{l-1}_{t(m,i)}$ and $\mathbf{Y}^{l}_{t(m,j)}$ are the i^{th} channel input and the j^{th} channel output, $\mathbf{W}^{l}_{t-1(i,j)}$ is the filter weight matrix between the i^{th} input channel and the j^{th} output channel, $\mathbf{R}^{l-1}_{t(m,i,u,v)}$ is the receptive field matrix in the i^{th} input channel for the output $\mathbf{Y}^{l}_{t(m,j,u,v)}$, C_{l-1} and C_l are the number of input and output channels, and $f_l(\cdot)$ is the activation function. (b): $\mathbf{Y}^{l-1}_{t(m,i)}$ and $\mathbf{Y}^{l}_{t(m,j)}$ are the i^{th} node input and the j^{th} node output, $\mathbf{W}^{l}_{t-1(i,j)}$ is the weight between the i^{th} input node and the j^{th} output node, and N_{l-1} and N_l are the number of input and output nodes

where $k > 0$ is the average scaling factor and $\mathbf{u}^l_t$ and h^l_t are defined as follows:

$$\mathbf{u}^l_t = \mathbf{P}^l_{t-1}\mathbf{x}^l_t \tag{14}$$

$$h^l_t = \lambda + k(\mathbf{x}^l_t)^{\mathrm{T}}\mathbf{u}^l_t \tag{15}$$

where $\mathbf{x}^l_t$ is the average vector. If the l^{th} layer is a convolutional layer, $\mathbf{x}^l_t \in \mathbb{R}^{C_{l-1}H_lW_l}$ is defined as

$$\mathbf{x}^l_t = \frac{1}{M_t U_l V_l}\sum_{m=1}^{M_t}\sum_{u=1}^{U_l}\sum_{v=1}^{V_l}(flatten(\mathbf{R}^{l-1}_{t(m,:,u,v,:,:)}))^{\mathrm{T}} \tag{16}$$

where H_l and W_l denote the height and width of the filters, M_t denotes the current minibatch size, U_l and V_l denote the height and width of the output channels, and $flatten(\cdot)$ denotes reshaping the given matrix or tensor into a column vector. If the l^{th} layer is a fully-connected layer, $\mathbf{x}^l_t$ is defined as

$$\mathbf{x}^l_t = \frac{1}{M_t}\sum_{m=1}^{M_t}(\mathbf{Y}^{l-1}_{t(m,:)})^{\mathrm{T}} \tag{17}$$

Note that if the preceding layer of this layer is a convolutional or pooling layer, $\mathbf{Y}^{l-1}_{t(m,:)}$ will denote the flattened vector of the preceding layer's output. In the RLS optimization algorithm, tensor $\mathbf{W}^l_{t-1}$ is converted to matrix $\mathbf{W}^l_{t-1}$ by defining $\mathbf{W}^l_{t-1(:,j)} = flatten(\mathbf{W}^l_{t-1(:,j,:,:)})$. In addition, the algorithm uses the momentum to accelerate the convergence rate. Thus, regardless of whether the l^{th} layer is a convolutional layer or a fully-connected layer, the recursive update rule of $\mathbf{W}^l_{t-1}$ is defined as follows:

$$\mathbf{\Psi}^l_t = \alpha\mathbf{\Psi}^l_{t-1} - \frac{\eta^l}{h^l_t}\mathbf{P}^l_{t-1}\nabla_{\mathbf{W}^l_{t-1}} \tag{18}$$

$$\mathbf{W}^l_t \approx \mathbf{W}^l_{t-1} + \mathbf{\Psi}^l_t \tag{19}$$

where $\mathbf{\Psi}^l_t$ is the velocity matrix of the l^{th} layer at step t, α is the momentum factor, $\eta^l > 0$ is the gradient scaling factor, and $\nabla_{\mathbf{W}^l_{t-1}}$ is the equivalent gradient of the linear output loss function $\mathcal{L}_t$ with respect to $\mathbf{W}^l_{t-1}$. $\mathcal{L}_t$ is defined as

$$\mathcal{L}_t = \frac{1}{2M_t}\left\|\mathbf{Z}^L_t - \mathbf{Z}^*_t\right\|^2_{\mathrm{F}} \tag{20}$$

where $\mathbf{Z}^L_t = f_L^{-1}(\mathbf{Y}^L_t)$ is the linear output matrix and $\mathbf{Z}^*_t = f_L^{-1}(\mathbf{Y}^*_t)$ is the desired linear output matrix corresponding to $\mathbf{Z}^L_t$. Note that the RLS optimization assumes that $f_L(\cdot)$ is monotonic in the output layer [35]. In addition, the RLS optimization can be used for FNNs, since the above equations except for (16) can also be used for fully-connected layers and the last part of a CNN can generally be viewed as an FNN.

3 The proposed algorithm

In this section, we present our iterative structured pruning algorithm. We first introduce the theoretical foundation of our algorithm. Then, we describe our pruning strategy in detail and show the overall pseudocode of our algorithm.

3.1 Theoretical foundation

As discussed in Section 2.1, $\mathbf{A}_t$ is the input autocorrelation matrix. Suppose that $\mathbf{x}_t$ has n features, namely, $\mathbf{x}_t = [\mathbf{x}_{t(1)}, \mathbf{x}_{t(2)}, \cdots, \mathbf{x}_{t(n)}]^{\mathrm{T}}$. Then, $\mathbf{x}_t\mathbf{x}_t^{\mathrm{T}}$ can be expressed as

$$\mathbf{x}_t\mathbf{x}_t^{\mathrm{T}} = \begin{bmatrix} \mathbf{x}_{t(1)}\mathbf{x}_{t(1)} & \mathbf{x}_{t(1)}\mathbf{x}_{t(2)} & \cdots & \mathbf{x}_{t(1)}\mathbf{x}_{t(n)} \\ \mathbf{x}_{t(2)}\mathbf{x}_{t(1)} & \mathbf{x}_{t(2)}\mathbf{x}_{t(2)} & \cdots & \mathbf{x}_{t(2)}\mathbf{x}_{t(n)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{t(n)}\mathbf{x}_{t(1)} & \mathbf{x}_{t(n)}\mathbf{x}_{t(2)} & \cdots & \mathbf{x}_{t(n)}\mathbf{x}_{t(n)} \end{bmatrix} \tag{21}$$

Let $s_{\mathbf{A}_{t(i)}}$ denote the sum of the i^{th} row (or column) in $\mathbf{A}_t$. According to (3), (6) and (21), $s_{\mathbf{A}_{t(i)}}$ can be defined as

$$s_{\mathbf{A}_{t(i)}} = \sum_{k=1}^{t} \lambda^{t-k}\mathbf{x}_{k(i)}s_{\mathbf{x}_k} = \lambda s_{\mathbf{A}_{t-1(i)}} + \mathbf{x}_{t(i)}s_{\mathbf{x}_t} \tag{22}$$

where $s_{\mathbf{x}_t}$ is the sum of all features in $\mathbf{x}_t$. In deep learning, we typically assume that all samples are independently and identically distributed [39]. CNNs are usually used for computer vision tasks, and they often use the ReLU activation function in the hidden layers; thus, we can suppose that $\mathbf{x}_{t(i)} \geq 0$. In addition, the forgetting factor λ is generally close to 1. Therefore, $s_{\mathbf{x}_1}, \cdots, s_{\mathbf{x}_t}$ are approximately equal, and $s_{\mathbf{A}_{t(i)}}$ is approximately proportional to the sum of $\mathbf{x}_{1(i)}, \cdots, \mathbf{x}_{t(i)}$. In other words, if $s_{\mathbf{A}_{t(i)}}$ is small, the i^{th} features in $\mathbf{x}_1, \cdots, \mathbf{x}_t$ are probably small, and their influence on the outputs is probably small. Since $\mathbf{P}_t$ is the inverse of $\mathbf{A}_t$, we can easily draw the following conclusion: If the sum of the i^{th} row (or column) in $\mathbf{P}_t$ is large, the importance of the i^{th} features in $\mathbf{x}_1, \cdots, \mathbf{x}_t$ will probably be small.

Thus, we can use $\mathbf{P}_t^l$ to evaluate the importance of the input nodes in the l^{th} layer since $\mathbf{P}_t^l$ has the same meaning as $\mathbf{P}_t$. For fully-connected layers in CNNs, we can directly use this conclusion. However, for convolutional layers in CNNs, structured pruning methods aim to prune unimportant channels rather than nodes. Fortunately, according to (16) and Fig. 1(a), we know that $\mathbf{x}_{t((i-1)H_lW_l+1)}, \mathbf{x}_{t((i-1)H_lW_l+2)}, \cdots, \mathbf{x}_{t(iH_lW_l)}$ in $\mathbf{x}_t^l$ comes from the i^{th} input channel. Thus, we can use the sum of the $(i-1)H_lW_l + 1$ to iH_lW_l rows (columns) in $\mathbf{P}_t^l$ to evaluate the importance of the i^{th} input channel.

Based on the above analysis, we define a vector $\mathbf{s}_{\mathbf{P}_t^l}$ to evaluate the importance of the input channels or nodes in the l^{th} layer. The i^{th} element in $\mathbf{s}_{\mathbf{P}_t^l}$ is defined as

$$\mathbf{s}_{\mathbf{P}_t^l(i)} = \begin{cases} \sum_{j=N_{l(i)}^{cb}}^{N_{l(i)}^{ce}} \sum_{k=1}^{N_l^{re}} \mathbf{P}_{t(k,j)}^l & l \leq L_c \\ \sum_{k=1}^{N_{l-1}} \mathbf{P}_{t(k,i)}^l & l > L_c \end{cases} \tag{23}$$

where $N_{l(i)}^{cb} = (i-1)H_lW_l + 1$, $N_{l(i)}^{ce} = iH_lW_l$, $N_l^{re} = C_{l-1}H_lW_l$, and L_c denotes the total number of convolutional layers. Equation (23) can be explained as follows: If the l^{th} layer is a convolutional layer (i.e. $l \leq L_c$), $\mathbf{s}_{\mathbf{P}_t^l(i)}$ denotes the importance of the i^{th} input channel. If the l^{th} layer is a fully-connected layer (i.e. $l > L_c$), $\mathbf{s}_{\mathbf{P}_t^l(i)}$ denotes the importance of the i^{th} input node. Furthermore, the larger the value of $\mathbf{s}_{\mathbf{P}_t^l(i)}$ is, the more likely it is that the i^{th} input channel or node is unimportant.

Algorithm 1 RLS Method for Training and Pruning CNNs.

Input: $\lambda, k, \alpha, \{\eta^l\}_{l=1}^L, \xi, epoch_{prune}$
Initialize: $\{\mathbf{W}_0^l\}_{l=1}^L, \{\mathbf{\Psi}_0^l\}_{l=1}^L, \{\mathbf{P}_0^l\}_{l=1}^L, loss_{min} = +\infty$

for $epoch = 1$ to MaxEpoch **do**
  for $t = 1$ to MaxBatch **do**
    Get the minibatch $(\mathbf{Y}_t^0, \mathbf{Y}_t^*)$ from the training dataset $\mathbb{D}_{train}$
    Perform the forward propagation and get $\{\mathbf{Y}_t^l\}_{l=1}^L$
    Compute the training loss $\mathcal{L}_t$ by (20)
    for $l = L$ to 1 **do**
      Update $\mathbf{\Psi}_{t-1}^l, \mathbf{W}_{t-1}^l, \mathbf{P}_{t-1}^l$ by (18), (19), (13)
    end for
  end for
  Compute $loss$ by using the testing dataset $\mathbb{D}_{test}$
  if $epoch \geq epoch_{prune}$ and $loss \leq loss_{min}$ **then**
    Update $loss_{min} = loss$
    for $l = 1$ to L **do**
      Compute $\mathbf{s}_{\mathbf{P}_t^l}, \mathbf{s}_{\mathbf{W}_t^l}$ by (23), (24)
      Compute $\mathbb{I}_t^l(\xi)$ by (25)
      Prune the input channels or nodes indexed by $\mathbb{I}_t^l(\xi)$, and prune the corresponding components in $\mathbf{P}_t^l, \mathbf{W}_t^l, \mathbf{\Psi}_t^l, \mathbf{W}_t^{l-1}$ and $\mathbf{\Psi}_t^{l-1}$[1]
    end for
  end if
  for $l = 1$ to L **do**
    Reset $\mathbf{W}_0^l = \mathbf{W}_t^l$, $\mathbf{\Psi}_0^l = \mathbf{\Psi}_t^l$ and $\mathbf{P}_0^l = \mathbf{P}_t^l$
  end for
end for

[1] $\mathbf{W}_t^{l-1}$ and $\mathbf{\Psi}_t^{l-1}$ are pruned only when $l \geq 2$

3.2 RLS-based pruning

As mentioned in Section 1, existing structured pruning algorithms have some shortcomings. In this subsection, we consider how to overcome these shortcomings and present our pruning algorithm.

One shortcoming of existing algorithms is that they cannot prune the unimportant channels accurately since most algorithms use only the weight magnitude to evaluate the channel importance. To address this problem, we combine inverse input autocorrelation matrices with weight matrices to evaluate and prune the unimportant input channels in the convolutional layers and unimportant nodes in the fully-connected layers.

In Section 3.1, we defined $\mathbf{s}_{\mathbf{P}_t^l}$ by using the inverse input autocorrelation matrix $\mathbf{P}_t^l$. Next, we define another vector $\mathbf{s}_{\mathbf{W}_t^l}$ by using the weight matrix $\mathbf{W}_t^l$. Li et al. proposed the ℓ_1-norm algorithm and demonstrated that the sum of the absolute filter weights can be used to evaluate the importance of the output channels [29]. It is well known that the output of the $(l-1)^{th}$ layer is the input of the l^{th} layer in CNNs. Thus, we can modify this approach to evaluate the importance of the input channels or nodes in the l^{th} layer. By using this approach, the i^{th} element in $\mathbf{s}_{\mathbf{W}_t^l}$ is defined as

$$\mathbf{s}_{\mathbf{W}_t^l}(i) = \begin{cases} \sum_{k=1}^{N_{l-1}^{re}} |\mathbf{W}_{t(k,i)}^{l-1}| & 2 \le l \le L_c \\ \sum_{k=1}^{N_{l-1}^{re}} |\mathbf{W}_{t(k,c_{(i)})}^{l-1}| & l = L_c + 1 \\ \sum_{k=1}^{N_{l-1}} |\mathbf{W}_{t(k,i)}^{l-1}| & l > L_c + 1 \end{cases} \tag{24}$$

where $N_{l-1}^{re} = C_{l-2}H_{l-1}W_{l-1}$, $c_{(i)} = \lfloor i/(U_{l-1}V_{l-1}) \rfloor + 1$, and $|\cdot|$ denotes the absolute value of a real number. According to [29], the i^{th} input channel or node is more likely to be unimportant when $\mathbf{s}_{\mathbf{W}_t^l(i)}$ is smaller. Note that we only consider the convolutional and fully-connected layers in this paper since a pooling layer has no learnable weights and the same number of output channels as its preceding convolutional layer.

Next, we use $\mathbf{s}_{\mathbf{P}_t^l}$ and $\mathbf{s}_{\mathbf{W}_t^l}$ to select unimportant input channels and nodes. Let $\mathbb{A}_t^l(\mathbf{s}_{\mathbf{P}_t^l}, \xi)$ and $\mathbb{B}_t^l(\mathbf{s}_{\mathbf{W}_t^l}, \xi)$ denote the index sets of the top ξ unimportant input channels and nodes, which are determined by $\mathbf{s}_{\mathbf{P}_t^l}$ and $\mathbf{s}_{\mathbf{W}_t^l}$, respectively. Then, the index set of the pruned input channels or nodes in the l^{th} layer can be defined as

$$\mathbb{I}_t^l(\xi) = \begin{cases} \mathbb{A}_t^l(\mathbf{s}_{\mathbf{P}_t^l}, \xi) \cap \mathbb{B}_t^l(\mathbf{s}_{\mathbf{W}_t^l}, \xi) & l \ge 2 \\ \mathbb{A}_t^l(\mathbf{s}_{\mathbf{P}_t^l}, 0.5\xi) & l = 1 \end{cases} \tag{25}$$

where ξ is the preset pruning ratio. Note that we only use $\mathbb{A}_t^l(\mathbf{s}_{\mathbf{P}_t^l}, 0.5\xi)$ to prune the input channels in the first layer. This is because (24) shows that $\mathbf{s}_{\mathbf{W}_t^l}$ cannot be used to evaluate the importance of these channels. In addition, we find that the size of $\mathbb{I}_t^l(\xi)$ is approximately one half of the size of $\mathbb{A}_t^l(\mathbf{s}_{\mathbf{P}_t^l}, \xi)$ or $\mathbb{B}_t^l(\mathbf{s}_{\mathbf{W}_t^l}, \xi)$ if $l \ge 2$ in the following experiments presented in Section 4.2. Some readers may argue that our preset pruning ratio ξ is also set to a fixed value. However, according to (25), $\mathbb{I}_t^l(\xi)$ is the intersection of $\mathbb{A}_t^l(\mathbf{s}_{\mathbf{P}_t^l}, \xi)$ and $\mathbb{B}_t^l(\mathbf{s}_{\mathbf{W}_t^l}, \xi)$, except for in the first layer. Hence, our actual pruning ratio is lower than ξ during each pruning operation. In fact, in our experiments, we find that our actual pruning ratio gradually decreases as the iterative pruning process continues, and the final compression ratio of the CNNs is adaptively adjusted according to the difficulty of the learning task.

The remaining shortcomings of existing algorithms are addressed with the following two approaches: 1) We use the RLS optimization to improve the convergence rate of the

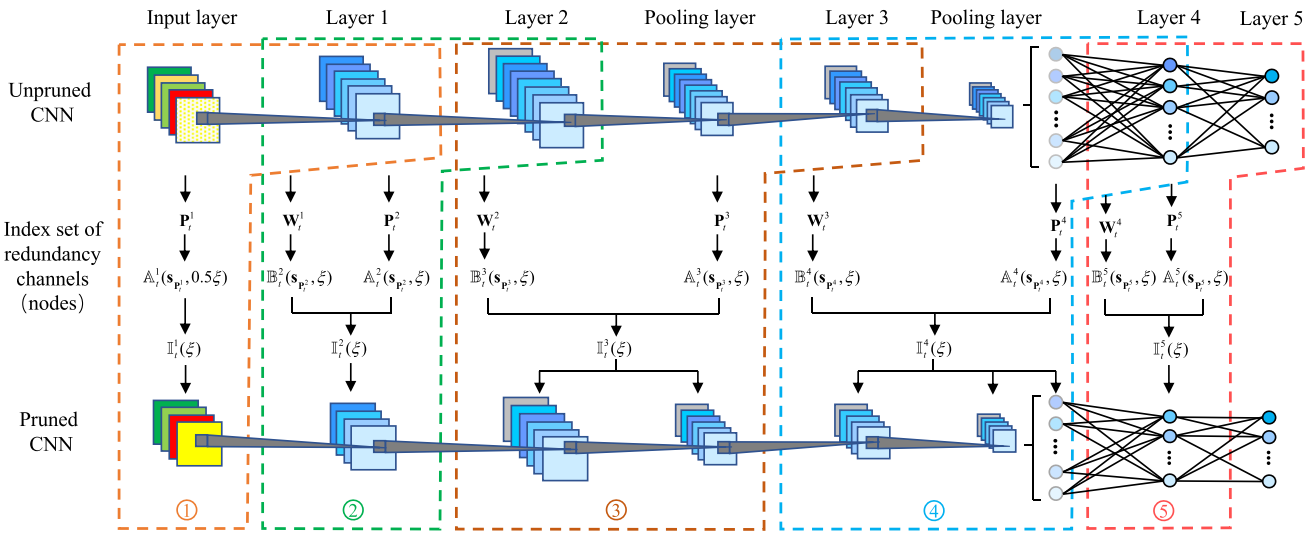


Fig. 2 Detail pruning of each layer in a five-layer CNN with our algorithm. The l^{th} dashed border is to show how $\mathbb{I}_t^l(\xi)$ is computed and the input channels or nodes are pruned in the l^{th} layer

training and fine-tuning stages. This is also convenient for the implementation of our pruning strategy. 2) We use the testing loss to determine the repruning timing at the end of a training epoch. Each repruning operation is performed only when the testing loss is tuned down to the last unpruned level.

Based on the above information, our iterative structured pruning algorithm is summarized in Algorithm 1, where $epoch_{prune}$ denotes the first pruning epoch, and $loss$ denotes the testing loss defined by the mean squared error (MSE). Our algorithm can also be used to prune FNNs since it includes the training and pruning of the fully-connected layers. In addition, the RLS optimization can be used in existing pruning algorithms to replace the SGD optimization. To clarify how our algorithm prunes CNNs, the detailed pruning process of each layer in a five-layer CNN is illustrated in Fig. 2. It clearly shows that our algorithm can prune the input layer in theory. To the best of our knowledge, there is no existing pruning algorithm that can do this.

4 Experiments

In this section, we validate the effectiveness of our algorithm. We first introduce our experimental settings. Then, we evaluate the influence of the preset pruning ratio ξ on the performance of our algorithm. Finally, we present the comparison results of our algorithm versus four popular pruning algorithms.

Table 1 Comparison on the reductions in FLOPs, parameters and testing accuracy for all models pruned by our algorithm with different ξ values on different datasets

Pruning Ratio	Pruned Acc(%)	FLOPs↓(%)	Parameters↓(%)	Acc↓(%)
(a) Pruning the VGG-16 model on CIFAR-10, Baseline Acc(%)=91.63				
ξ=0.2	92.13	48.09	56.43	-0.52
ξ=0.3	91.92	81.44	87.36	-0.31
ξ=0.4	91.73	84.87	88.74	-0.12
ξ=0.5	90.98	88.74	91.22	0.63
(b) Pruning the VGG-16 model on CIFAR-100, Baseline Acc(%)=70.08				
ξ=0.2	69.16	30.86	18.07	0.92
ξ=0.3	67.98	58.56	44.70	2.10
ξ=0.4	68.35	68.81	61.20	1.73
ξ=0.5	67.55	80.85	80.48	2.53
(c) Pruning the ResNet-50 model on CIFAR-10, Baseline Acc(%)=94.11				
ξ=0.2	93.40	27.96	25.53	0.71
ξ=0.3	93.55	31.88	30.36	0.56
ξ=0.4	93.42	55.29	54.15	0.69
ξ=0.5	93.09	64.79	64.27	1.02
(d) Pruning the ResNet-50 model on CIFAR-100, Baseline Acc(%)=74.53				
ξ=0.2	73.71	17.17	16.73	0.82
ξ=0.3	73.30	27.55	25.19	1.23
ξ=0.4	73.02	39.35	34.94	1.51
ξ=0.5	72.12	53.98	50.26	2.41
(e) Pruning the three-layer FNN model on MNIST, Baseline Acc(%)=98.93				
ξ=0.2	98.52	58.42	58.42	0.31
ξ=0.3	98.64	74.84	74.84	0.21
ξ=0.4	98.68	82.68	82.68	0.25
ξ=0.5	98.57	90.79	90.79	0.36

Baseline Acc and Pruned Acc denote the testing accuracies of the original model and the pruned model at the last epoch, the ↓ denotes the reduced percentage between the pruned model and the original model, and the negative Acc↓ values denote the increases in the testing accuracy

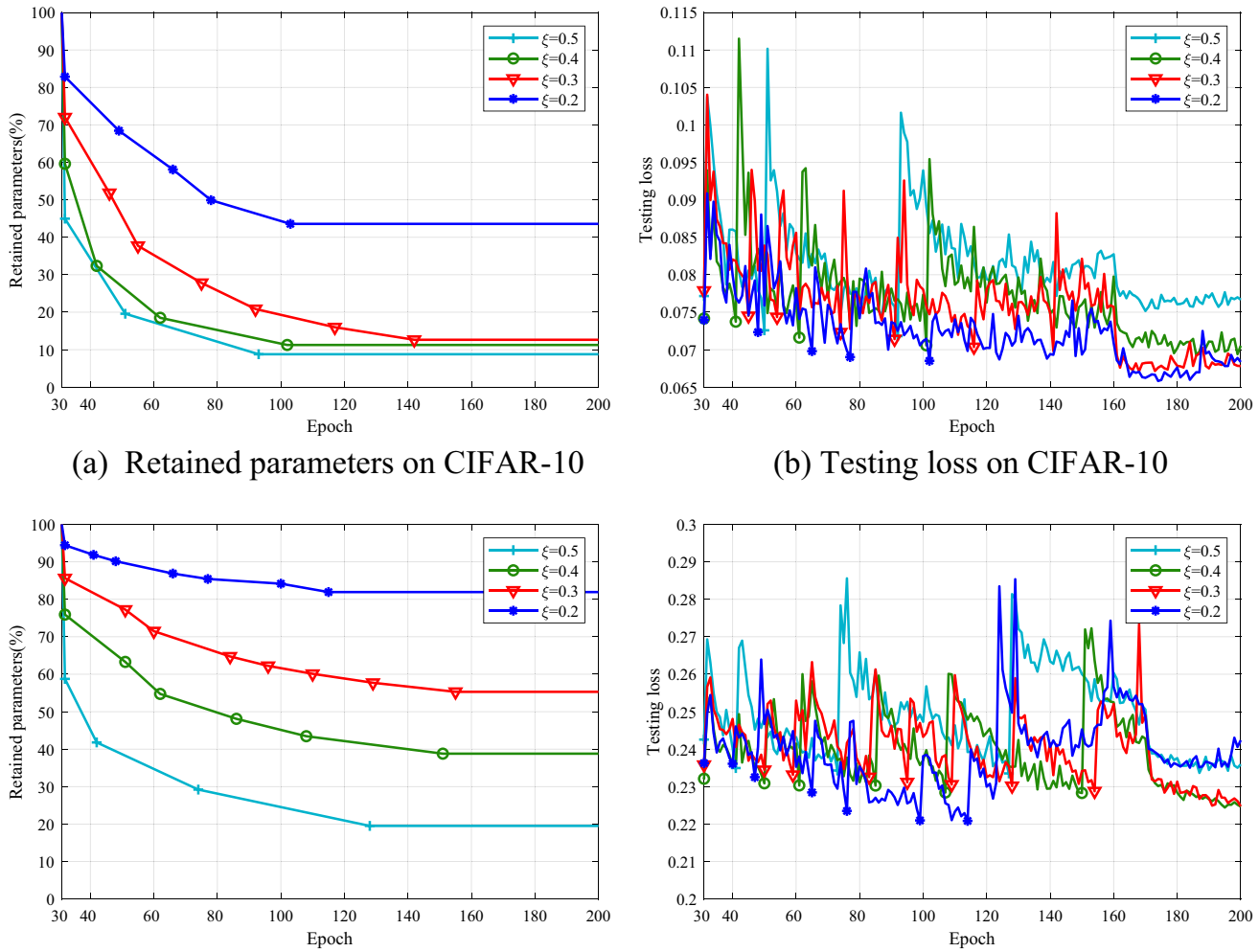


Fig. 3 Comparison on retained parameters and testing loss versus the number of epochs for the VGG-16 model pruned by our algorithm with different ξ values on CIFAR-10 and CIFAR-100. The marker point denotes the model is pruned at the corresponding epoch

4.1 Experimental settings

In our experiments, we use three benchmark datasets: CIFAR-10, CIFAR-100 [40] and MNIST [41]. CIFAR-10 and CIFAR-100 both include 60,000 32×32 three-channel colour images, with 50,000 images used for training and 10,000 images used for testing. The CIFAR-100 classification task is more difficult than the CIFAR-10 classification task since CIFAR-100 has 100 image classes, while CIFAR-10 has only 10 image classes. MNIST consists of 10 classes of 70,000 28×28 greyscale images, with 60,000 images used for training and 10,000 images used for testing.

For the CIFAR-10 and CIFAR-100 datasets, we use VGG-16 and ResNet-50 models, which have the same architectures as the models presented in [42] and [43], respectively. The VGG-16 model has 3.4×10^7 parameters and requires 3.3×10^8 FLOPs, and the ResNet-50 model has 2.3×10^7 parameters and requires 1.3×10^9 FLOPs. For using (20) as the training loss function, we delete their Softmax activation function. In addition, we prune only the second layer in each residual block of the ResNet-50 model, since the first and last layers use 1×1 filters to adjust the number of channels and have a small number of parameters and FLOPs. For the MNIST dataset, we use a three-layer FNN model, which has 1024 ReLU nodes, 512 ReLU nodes and 10 Identity nodes. The FNN model has 1.3×10^6 parameters and requires 1.3×10^6 FLOPs.

We compare the performance of our algorithm with that of four popular pruning algorithms: ℓ_1-norm, Network Slimming, Taylor FO and Taylor SO. The first two algorithms are one-shot pruning algorithms, and the last two are iterative pruning algorithms. These algorithms originally used SGD for training and fine-tuning and were designed to prune only convolutional layers. In some CNNs, such as VGG-16,

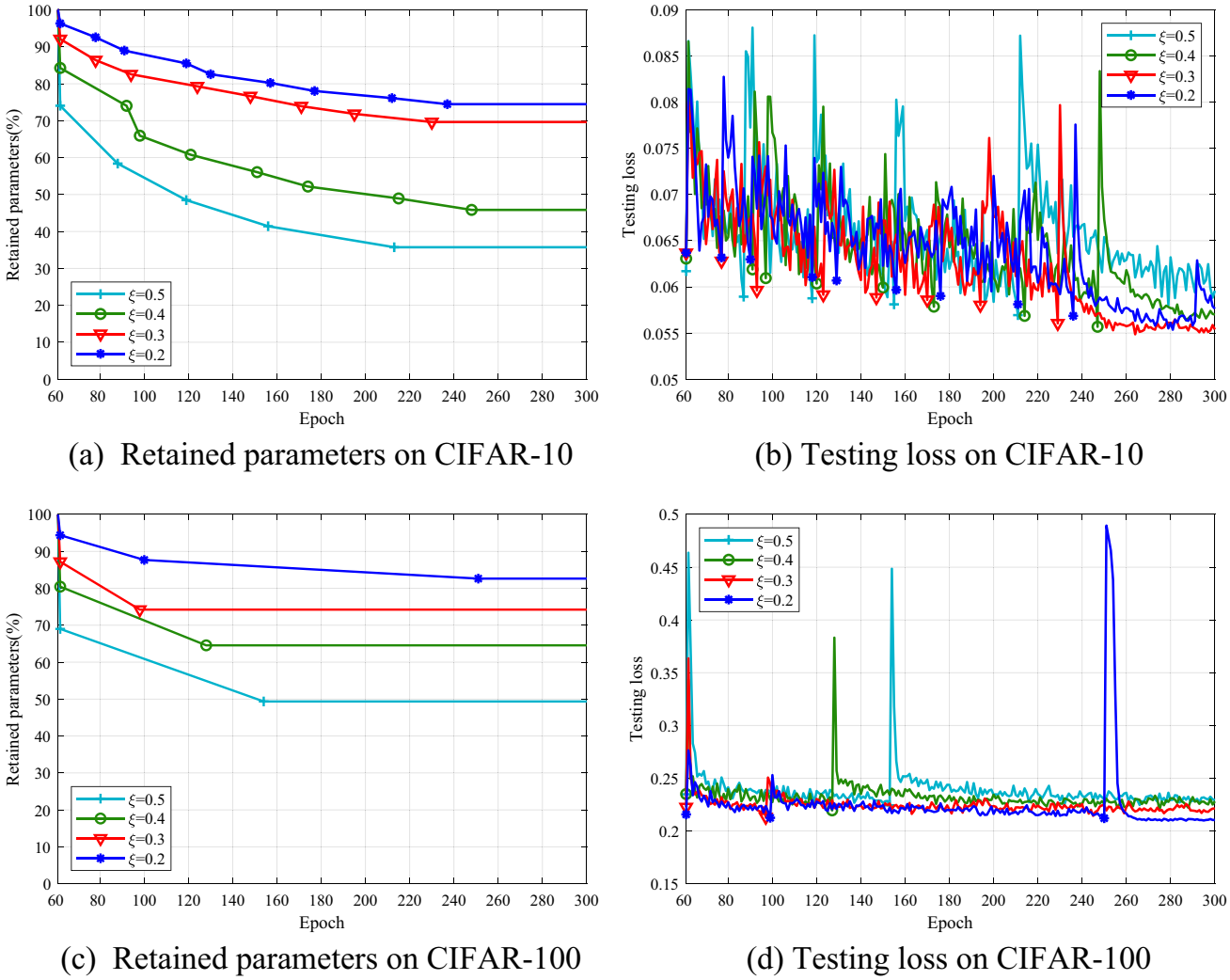


Fig. 4 Comparison on retained parameters and testing loss versus the number of epochs for the ResNet-50 model pruned by our algorithm with different ξ values on CIFAR-10 and CIFAR-100

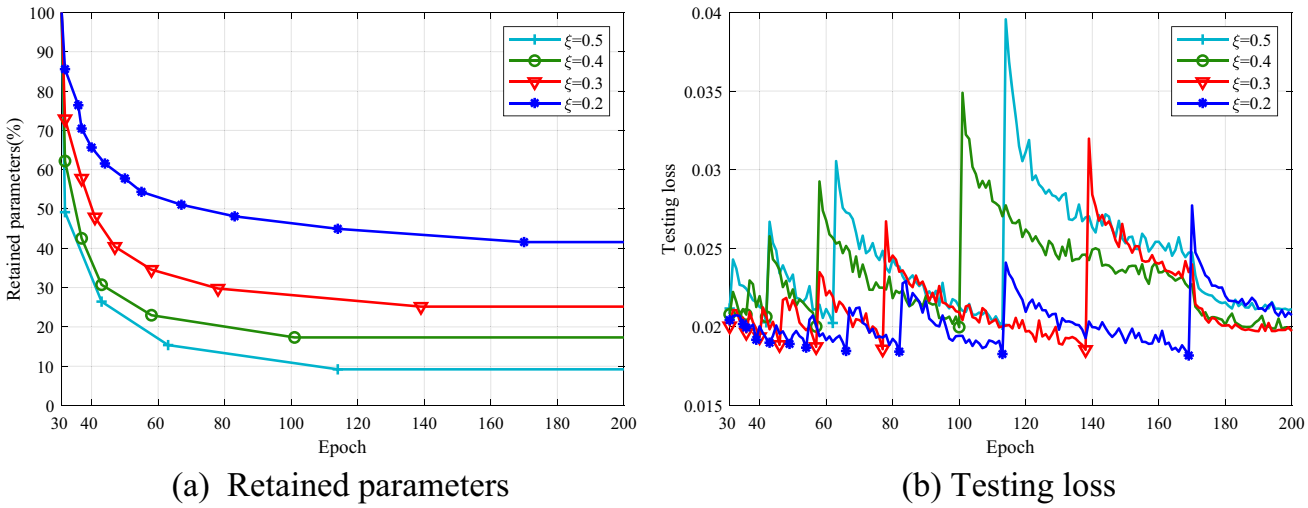


Fig. 5 Comparison on retained parameters and testing loss versus the number of epochs for the three-layer FNN model pruned by our algorithm with different ξ values on MNIST

fully-connected layers have more parameters than convolutional layers, and they are more likely to cause overfitting than convolutional layers. To ensure a fair comparison, we modify these algorithms to prune convolutional and fully-connected layers and replace the SGD optimization with the RLS optimization. In addition, for the Taylor FO and Taylor SO algorithms, we also use the testing loss as the repruning criterion.

All experiments are conducted by using PyTorch on an NVIDIA GeForce 1080Ti GPU with a minibatch size of 64. All algorithms use the RLS optimization, in which $\mathbf{W}_0^l$ is initialized to the default value by PyTorch, and λ, k, α, η^l, $\mathbf{\Psi}_0^l$ and $\mathbf{P}_0^l$ are set or initialized to 1, 0.1, 0.5, 1, the zero matrix and the identity matrix, respectively. Their pruning ratios are determined and presented in Section 4.2. For the VGG-16, ResNet-50 and three-layer FNN models, all algorithms are run for 200, 300 and 200 epochs, all the first or one-shot pruning operations are performed at the end of the 30^{th}, 60^{th} and 30^{th} epoch, and all the repruning operations are performed before the 160^{th}, 250^{th} and 170^{th} epoch, respectively. This is because all iterative pruning algorithms require some epochs to finely tune the models after the last repruning operation and the ResNet-50 model converges slower than the VGG-16 and FNN models.

4.2 Influence of the preset pruning ratio

The pruning ratio is perhaps the most important hyperparameter in pruning algorithms. In this subsection, we evaluate the influence of the preset pruning ratio ξ on the performance of our algorithm and determine the pruning ratios of the other four comparative pruning algorithms.

The comparison results of the reductions in FLOPs, parameters and testing accuracy for the VGG-16, ResNet-50 and three-layer FNN models pruned by our algorithm with different ξ values on CIFAR-10, CIFAR-100 and MNIST datasets are summarized in Table 1. As ξ increases, the FLOPs↓ and the Parameters↓ of all models increase significantly, but their pruned accuracies have small or even no reductions. These results prove that our algorithm can effectively prune the unimportant input channels and nodes in CNNs and FNNs. Moreover, the pruned VGG-16 and pruned ResNet-50 models have higher FLOPs↓ and Parameters↓ on CIFAR-10 than on CIFAR-100. In other words, by using our algorithm, the models retain more parameters for CIFAR-100 than for CIFAR-10. This proves that our algorithm can adaptively prune CNNs according to the learning task difficulty.

In addition, the comparison results of retained parameters and testing loss versus the number of epochs for all models pruned by our algorithm with different ξ values are shown in Figs. 3, 4 and 5, respectively. Our algorithm can prune all models multiple times in 30 to 160 or 60 to 250 epochs. After each pruning operation, the testing loss is reduced in a few epochs. This proves that our algorithm with different ξ values retains the fast convergence speed of the RLS optimization. As ξ increases, the pruning times gradually decrease in most cases. In addition, as the iterative pruning process continues, the actual pruning ratio gradually deceases. For example, for the VGG-16 model on CIFAR-10, our algorithm performs

Table 2 Comparison on the reductions in FLOPs, parameters and testing accuracy for the VGG-16 model pruned by different pruning algorithms on CIFAR-10 and CIFAR-100

Algorithm	Pruned Acc(%)	FLOPs↓(%)	Parameters↓(%)	Acc↓(%)
(a) On CIFAR-10, Baseline Acc(%)=91.63				
ℓ_1-norm	91.77	63.50	63.86	-0.16
Network Slimming	**91.81**	60.26	64.51	**-0.20**
Taylor FO	91.59	70.00	90.54	0.02
Taylor SO	90.50	55.53	75.98	1.11
Ours$_{\xi=0.4}$	91.73	84.78	88.74	-0.12
Ours$_{\xi=0.5}$	90.98	**88.74**	**91.22**	0.63
(b) On CIFAR-100, Baseline Acc(%)=70.08				
ℓ_1-norm	66.46	64.60	65.08	3.62
Network Slimming	67.75	63.70	68.63	2.33
Taylor FO	68.24	64.40	**84.21**	1.84
Taylor SO	68.30	52.68	75.89	1.78
Ours$_{\xi=0.4}$	**68.35**	68.81	61.20	**1.73**
Ours$_{\xi=0.5}$	67.55	**80.85**	80.48	2.53

four pruning operations when ξ is 0.4. The actual pruning ratios of these pruning operations are 0.220, 0.220, 0.219 and 0.202, respectively. These results prove that our algorithm can prevent overpruning to some extent when ξ is too large.

Based on Table 1 and Figs. 3 to 5, considering the trade-off between the accuracy loss and the compression ratio, we set ξ to be 0.4 and 0.5 in the following comparative experiments. According to the above statistical data from the VGG-16 model on CIFAR-10, our actual pruning ratio is approximately one half of ξ. Thus, to ensure a fair comparison, we set the pruning ratios of Taylor FO and Taylor SO to be 0.2. However, in contrast to our algorithm, the ℓ_1-norm and Network Slimming methods are one-shot pruning algorithms, meaning that they prune the CNNs and FNNs only once. Thus, we experimentally determine their pruning ratios to be 0.4 for the VGG-16 and ResNet-50 models and to be 0.6 for the three-layer FNN model.

Table 3 Comparison on the reductions in channels or nodes for each layer in the VGG-16 model pruned by different pruning algorithms on CIFAR-10 and CIFAR-100

Layer	Channels or nodes ↓(%)					
	ℓ_1-norm	Network Slimming	Taylor FO	Taylor SO	Ours$_{\xi=0.4}$	Ours$_{\xi=0.5}$
(a) On CIFAR-10						
Conv64-1	40.6	57.8	65.6	75.0	60.9	64.1
Conv64-2	40.6	45.3	0.0	6.2	60.9	67.2
Conv128-1	40.6	44.5	16.4	7.0	64.1	68.8
Conv128-2	40.6	24.2	4.7	0.0	61.7	67.2
Conv256-1	40.6	29.7	53.9	35.2	53.5	62.1
Conv256-2	40.6	26.2	41.8	25.0	54.3	61.3
Conv256-3	40.6	29.3	43.4	14.8	59.8	66.0
Conv512-1	40.6	37.3	78.9	51.8	65.2	67.6
Conv512-2	40.6	40.6	48.9	57.4	67.6	70.5
Conv512-3	40.6	36.5	73.2	51.0	58.2	67.2
Conv512-4	40.6	52.0	84.0	67.6	68.0	71.5
Conv512-5	40.6	48.2	78.5	59.6	65.6	69.9
Conv512-6	40.6	42.8	78.3	66.0	58.2	64.5
FC4096-1	40.6	41.3	57.0	34.5	50.9	52.1
FC4096-2	40.6	38.7	77.5	63.1	83.6	86.3
FC10	0.0	0.0	0.0	0.0	0.0	0.0
(b) On CIFAR-100						
Conv64-1	40.6	37.5	79.7	87.5	54.7	51.6
Conv64-2	40.6	43.7	9.4	28.1	40.6	48.4
Conv128-1	40.6	46.1	25.0	21.1	50.8	57.0
Conv128-2	40.6	41.4	14.8	7.0	40.6	59.4
Conv256-1	40.6	33.6	51.6	18.0	36.7	54.3
Conv256-2	40.6	36.7	43.0	12.5	46.9	57.0
Conv256-3	40.6	37.1	43.4	5.5	44.1	59.0
Conv512-1	40.6	37.3	74.6	23.8	36.3	55.5
Conv512-2	40.6	39.5	74.0	21.7	45.9	57.0
Conv512-3	40.6	38.5	73.8	96.7	57.0	68.2
Conv512-4	40.6	45.3	79.3	48.8	46.1	54.7
Conv512-5	40.6	45.1	78.5	84.2	40.8	52.5
Conv512-6	40.6	43.7	84.6	88.1	56.4	65.2
FC4096-1	40.6	11.0	64.8	65.3	32.6	58.1
FC4096-2	40.6	70.2	69.7	32.4	29.0	49.3
FC100	0.0	0.0	0.0	0.0	0.0	0.0

4.3 Comparison with other pruning algorithms

4.3.1 Pruning VGG-16 on CIFAR-10 and CIFAR-100

The comparison results of the reductions in FLOPs, parameters and testing accuracy for the VGG-16 model pruned by our algorithm and other four algorithms on CIFAR-10 and CIFAR-100 are summarized in Table 2, with the best results shown in bold. Our algorithm effectively reduces the FLOPs and the number of parameters for the VGG-16 model with a small accuracy loss. Under similar pruned accuracy conditions, the FLOPs↓ and Parameters↓ of our algorithm are significantly higher than those of the ℓ_1-norm and Network Slimming algorithms. Taylor FO and Taylor SO also effectively reduce the number of parameters for the VGG-16 model. However, they cannot reduce the FLOPs as effectively as they reduce the number of parameters, and the FLOPs↓ of Taylor SO is even smaller than those of the ℓ_1-norm and Network Slimming algorithms. In addition, in Section 4.2, we discuss that our algorithm can adaptively prune CNNs according to the learning task difficulty. Comparing the Parameters↓ values of the four comparative algorithms in Table 2, we determine that only Taylor FO makes the VGG-16 model retain more parameters for CIFAR-100 than for CIFAR-10, and the ℓ_1-norm, Network Slimming and Taylor SO algorithms do not have this property.

In addition, the comparison results of the reductions in channels or nodes for each layer in the VGG-16 model pruned by all algorithms on CIFAR-10 and CIFAR-100 are summarized in Table 3. Our algorithm, ℓ_1-norm and Network Slimming prune each VGG-16 layer more uniformly than Taylor FO and Taylor SO. This is the reason why our algorithm achieves both high FLOP reduction and high parameter reduction.

4.3.2 Pruning ResNet-50 on CIFAR-10 and CIFAR-100

The comparison results of the reductions in FLOPs, parameters and testing accuracy for the ResNet-50 model pruned by our algorithm and other four algorithms on CIFAR-10 and CIFAR-100 are summarized in Table 4, with the best results shown in bold. Our algorithm with $\xi = 0.5$ effectively reduces the FLOPs and the number of parameters for the ResNet-50 model with a small accuracy loss. Under similar pruned accuracy conditions, the FLOPs↓ and the Parameters↓ of the ℓ_1-norm and Network Slimming algorithms are significantly lower than those of our algorithm. Moreover, Taylor FO and Taylor SO cannot reduce the FLOPs as effectively as they reduce the number of parameters. Although Taylor SO has the best FLOPs↓ and Parameters↓ on CIFAR-100, its pruning accuracy is reduced by 4.22%. Among the considered algorithms, our algorithm with $\xi = 0.4$ has the best pruned accuracies on CIFAR-10 and CIFAR-100. In contrast, the four comparative algorithms have considerably larger accuracy losses for pruning the ResNet-50 model than for pruning the VGG-16 model. This proves that our algorithm has better adaptability for pruning different CNNs. In addition, our algorithm and Taylor FO can make the ResNet-50 model retain more parameters for

Table 4 Comparison on the reductions in FLOPs, parameters and testing accuracy for the ResNet-50 model pruned by different pruning algorithms on CIFAR-10 and CIFAR-100

Algorithm	Pruned Acc(%)	FLOPs(%)↓	Parameters(%)↓	Acc(%)↓
(a) On CIFAR-10, Baseline Acc(%)=94.11				
ℓ_1-norm	93.20	47.01	46.58	0.91
Network Slimming	93.02	41.45	51.77	1.09
Taylor FO	92.32	27.56	55.45	1.79
Taylor SO	91.95	27.51	55.01	2.16
Ours$_{\xi=0.4}$	**93.42**	55.29	54.15	**0.69**
Ours$_{\xi=0.5}$	93.09	**64.79**	**64.27**	1.02
(b) On CIFAR-100, Baseline Acc(%)=74.53				
ℓ_1-norm	42.34	47.00	45.79	32.19
Network Slimming	70.26	44.66	49.30	4.27
Taylor FO	71.20	23.58	51.24	3.33
Taylor SO	70.31	**60.63**	**77.77**	4.22
Ours$_{\xi=0.4}$	**73.02**	39.35	34.94	**1.51**
Ours$_{\xi=0.5}$	72.12	53.98	50.26	2.41

Table 5 Comparison on the reductions in channels or nodes for each layer in the ResNet-50 model pruned by different pruning algorithms on CIFAR-10 and CIFAR-100

Layer	Channels or nodes ↓(%)											
	ℓ_1-norm		Network Slimming		Taylor FO		Taylor SO		Ours$_{\xi=0.4}$		Ours$_{\xi=0.5}$	
	Input	Output	Input	Output	Input	Output	Input	Output	Input	Output	Input	Output
(a) On CIFAR-10												
block64-1	39.1	39.1	34.4	29.7	18.7	7.8	28.1	7.8	59.4	57.8	65.6	57.8
block64-2	39.1	39.1	34.4	39.1	7.8	1.6	29.7	28.1	64.1	45.3	67.2	75.0
block64-3	39.1	39.1	34.4	53.1	0.0	4.7	21.9	17.2	64.1	50.0	56.3	56.3
Block128-1	39.8	39.8	30.5	27.3	25.8	11.7	19.5	14.1	50.0	48.4	57.0	65.6
Block128-2	39.8	39.8	33.6	28.1	21.9	20.3	21.1	6.2	43.7	35.9	56.3	56.3
Block128-3	39.8	39.8	30.5	26.6	21.1	14.1	14.8	16.4	49.2	50.0	60.9	57.0
Block128-4	39.8	39.8	27.3	25.8	19.5	14.1	10.2	15.6	52.3	53.1	60.9	58.6
Block256-1	39.8	39.8	29.7	18.4	55.1	60.2	28.9	35.5	42.6	47.3	55.9	64.1
Block256-2	39.8	39.8	21.9	20.7	54.7	55.1	28.5	35.5	44.1	40.2	54.3	55.5
Block256-3	39.8	39.8	21.5	21.5	53.9	61.7	38.3	64.8	39.8	46.5	56.3	59.0
Block256-4	39.8	39.8	24.2	21.9	62.5	70.3	60.9	86.3	42.2	47.7	54.3	57.4
Block256-5	39.8	39.8	19.5	31.2	63.3	73.4	66.8	93.4	41.4	44.5	55.5	57.0
Block256-6	39.8	39.8	38.3	40.6	73.4	76.6	73.8	94.5	44.1	50.0	60.2	58.6
Block512-1	39.8	39.8	52.1	46.7	86.5	88.7	86.9	90.2	52.3	28.9	55.9	53.5
Block512-2	39.8	39.8	79.5	52.7	86.7	85.2	85.5	93.2	54.3	44.9	77.1	64.1
Block512-3	39.8	39.8	71.7	47.1	88.7	86.1	95.1	93.4	61.1	57.4	63.5	64.6
(b) On CIFAR-100												
block64-1	39.1	39.1	35.9	37.5	0.0	0.0	65.6	64.1	32.8	32.8	54.7	39.1
block64-2	39.1	39.1	42.2	48.4	0.0	0.0	35.9	18.7	40.6	29.7	53.1	51.6
block64-3	39.1	39.1	35.9	40.6	0.0	0.0	31.2	21.9	35.9	23.4	51.6	45.3
Block128-1	39.8	39.8	26.6	25.8	0.8	0.0	43.0	20.3	31.2	35.9	48.4	44.5
Block128-2	39.8	39.8	31.2	34.4	0.0	0.8	39.8	28.9	35.9	29.7	48.4	43.8
Block128-3	39.8	39.8	32.8	39.1	0.0	0.8	12.5	16.4	35.9	33.6	48.4	50.0
Block128-4	39.8	39.8	34.4	36.7	1.6	0.0	26.6	17.2	39.1	35.9	50.8	39.1
Block256-1	39.8	39.8	33.6	31.2	12.5	13.7	58.2	48.0	38.3	38.7	50.0	51.2
Block256-2	39.8	39.8	33.6	34.0	13.7	21.5	57.8	60.9	38.7	35.5	53.1	46.9
Block256-3	39.8	39.8	26.6	37.5	9.0	19.9	52.3	61.7	33.2	26.6	55.9	47.3
Block256-4	39.8	39.8	25.4	44.9	14.5	22.7	69.1	78.1	31.2	27.0	46.1	39.8
Block256-5	39.8	39.8	24.2	42.2	16.4	30.9	91.4	91.4	30.1	29.3	47.3	42.6
Block256-6	39.8	39.8	26.6	44.9	26.2	42.6	89.1	90.2	31.6	30.5	50.8	44.1
Block512-1	39.8	39.8	34.6	49.6	61.3	67.8	97.7	95.7	41.6	28.5	49.6	40.8
Block512-2	39.8	39.8	36.7	63.9	71.3	64.1	96.5	96.7	18.6	25.8	37.7	40.6
Block512-3	39.8	39.8	38.9	67.2	73.4	70.7	98.4	98.8	24.2	27.7	40.8	47.5

Table 6 Comparison on the reductions in FLOPs, parameters and testing accuracy for the three-layer FNN model pruned by different pruning algorithms on MNIST

Algorithm	Pruned Acc(%)	FLOPs(%)↓	Parameters(%)↓	Acc(%)↓
Baseline Acc(%): 98.93				
ℓ_1-norm	98.42	69.51	69.51	0.51
Network Slimming	98.40	68.75	68.75	0.53
Taylor FO	98.67	77.10	77.10	0.26
Taylor SO	98.67	76.99	76.99	0.26
Ours$_{\xi=0.4}$	**98.68**	82.68	82.68	**0.25**
Ours$_{\xi=0.5}$	98.57	**90.79**	**90.79**	0.36

CIFAR-100, but the FLOPs↓ with Taylor FO on CIFAR-100 is only 23.58%.

In addition, the comparison results of the reductions in channels or nodes for each layer in the ResNet-50 model pruned by all algorithms on CIFAR-10 and CIFAR-100 are summarized in Table 5. Our algorithm, ℓ_1-norm and Network Slimming prune each ResNet-50 layer more uniformly than Taylor FO and Taylor SO.

4.3.3 Pruning FNNs on MNIST

The comparison results of the reductions in FLOPs, parameters and testing accuracy for the three-layer FNN model pruned by different pruning algorithms on MNIST are summarized in Table 6, with the best results shown in bold. Our algorithm achieves the best pruning accuracy, FLOPs↓ and Parameters↓. In particular, the FLOPs↓ and the Parameters↓ of our algorithm are significantly higher than those of the four comparative algorithms. These results prove that our algorithm can effectively prune FNNs with a small accuracy loss.

In addition, the comparison results of the reductions in nodes for each layer in the three-layer FNN model pruned by all algorithms on MNIST are summarized in Table 7. Our algorithm, ℓ_1-norm and Network Slimming prune each FNN layer more uniformly than Taylor FO and Taylor SO. More importantly, the results show that our algorithm can not only prune the nodes in fully-connected hidden layers but also prune the original sample features in the input layer. In fact, according to (25), our algorithm can also prune the original channels of multichannel samples if the number of channels is greater than or equal to $2/\xi$. In Sections 4.3.1 and 4.3.2, our algorithm does not prune the input layers of the VGG-16 and ResNet-50 models because all images in CIFAR-10 and CIFAR-100 have only three channels.

5 Conclusion

In this paper, we studied structured pruning algorithms for CNNs. To address the shortcomings of existing algorithms, we proposed an RLS-based iterative structured pruning algorithm. Our algorithm employs the RLS optimization to accelerate the convergence rate of the training and fine-tuning stages, combines inverse input autocorrelation matrices with weight matrices to evaluate and prune unimportant input channels or nodes in CNN layers, and uses the testing loss to automatically determine the repruning timing. We demonstrated the effectiveness of our algorithm in pruning VGG-16 and ResNet-50 on CIFAR-10 and CIFAR-100 and pruning an FNN on MNIST. The experimental results show that our algorithm can prune CNNs and FNNs multiple times in a small number of epochs. Compared with four popular pruning algorithms, our algorithm can effectively reduce both the FLOPs and number of parameters for CNNs and FNNs with small or even no reduction in the accuracy. Furthermore, our algorithm can adaptively prune CNNs according to the learning task difficulty and has better adaptability for pruning different networks. Moreover, our algorithm can prune the original sample features.

Table 7 Comparison on the reductions in nodes for each layer in the three-layer FNN model pruned by different pruning algorithms on MNIST

Layer	Nodes ↓(%)					
	ℓ_1-norm	Network Slimming	Taylor FO	Taylor SO	Ours$_{\xi=0.4}$	Ours$_{\xi=0.5}$
Input	0.0	0.0	0.0	0.0	67.0	68.2
FC1024	60.1	58.0	63.1	62.7	59.1	73.8
FC512	60.2	64.8	95.7	96.5	44.5	60.9
FC10	0	0	0	0	0	0

Acknowledgements This work is supported by the National Natural Science Foundation of China (grant nos. 61762032 and 11961018).

Data Availability All data included in this study are available upon request by contact with the corresponding author.

Declarations

Competing interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Gabor M, Zdunek R (2023) Compressing convolutional neural networks with hierarchical tucker-2 decomposition. Appl Soft Comput 132:109856. https://doi.org/10.1016/j.asoc.2022.109856
2. Liu H, Liu T, Zhang Z, Sangaiah AK, Yang B, Li Y (2022) ARHPE: Asymmetric relation-aware representation learning for head pose estimation in industrial human-computer interaction. IEEE Trans Industr Inform 18(10):7107–7117. https://doi.org/10.1109/TII.2022.3143605
3. Liu, H, Liu, T, Chen, Y, Zhang, Z, Li Y-F(2022) EHPE: Skeleton cues-based gaussian coordinate encoding for efficient human pose estimation. IEEE Trans Multimedia, pp 1–12. https://doi.org/10.1109/TMM.2022.3197364
4. Liu T, Wang J, Yang B, Wang X (2021) NGDNet: Nonuniform gaussian-label distribution learning for infrared head pose estimation and on-task behavior understanding in the classroom. Neurocomputing 436:210–220. https://doi.org/10.1016/j.neucom.2020.12.090
5. Liu H, Fang S, Zhang Z, Li D, Lin K, Wang J (2022) MFDNet: Collaborative poses perception and matrix fisher distribution for head pose estimation. IEEE Trans Multimedia 24:2449–2460. https://doi.org/10.1109/TMM.2021.3081873
6. Liu H, Nie H, Zhang Z, Li Y (2021) Anisotropic angle distribution learning for head pose estimation and attention understanding in human-computer interaction. Neurocomputing 433:310–322. https://doi.org/10.1016/j.neucom.2020.09.068
7. Li Z, Liu H, Zhang Z, Liu T, Xiong NN (2022) Learning knowledge graph embedding with heterogeneous relation attention networks. IEEE Trans Neural Netw Learn Syst 33(8):3961–3973. https://doi.org/10.1109/TNNLS.2021.3055147
8. LeCun Y, Bengio Y, Hinton GE (2015) Deep learning. Nat 521(7553):436–444. https://doi.org/10.1038/nature14539
9. Li, S, Sun, Y, Yen, GG, Zhang M (2021) Automatic design of convolutional neural network architectures under resource constraints. IEEE Trans Neural Netw Learn Syst, pp 1–15 . https://doi.org/10.1109/TNNLS.2021.3123105
10. Liu H, Zheng C, Li D, Shen X, Lin K, Wang J, Zhang Z, Zhang Z, Xiong NN (2022) EDMF: Efficient deep matrix factorization with review feature learning for industrial recommender system. IEEE Trans Industr Inform 18(7):4361–4371. https://doi.org/10.1109/TII.2021.3128240
11. Kocacinar B, Tas B, Akbulut FP, Catal C, Mishra D (2022) A real-time cnn-based lightweight mobile masked face recognition system. IEEE Access 10:63496–63507. https://doi.org/10.1109/ACCESS.2022.3182055
12. Cheng J, Wang P, Li G, Hu Q, Lu H (2018) Recent advances in efficient computation of deep convolutional neural networks. Front Inf Technol Electron Eng 19(1):64–77. https://doi.org/10.1631/FITEE.1700789
13. Liang T, Glossner J, Wang L, Shi S, Zhang X (2021) Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing 461:370–403. https://doi.org/10.1016/j.neucom.2021.07.045
14. Cheng Y, Wang X, Xie X, Li W, Peng S (2022) Channel pruning guided by global channel relation. Appl Intell 52(14):1–12. https://doi.org/10.1007/s10489-022-03198-9
15. Hasan MS, Alam R, Adnan MA (2023) Compressed neural architecture utilizing dimensionality reduction and quantization. Appl Intell 53(2):1271–1286. https://doi.org/10.1007/s10489-022-03221-z
16. Yu Z, Shi Y (2022) Kernel quantization for efficient network compression. IEEE Access 10:4063–4071. https://doi.org/10.1109/ACCESS.2022.3140773
17. Wang J, Zhu L, Dai T, Xu Q, Gao T (2021) Low-rank and sparse matrix factorization with prior relations for recommender systems. Appl Intell 51(6):3435–3449. https://doi.org/10.1007/s10489-020-02023-5
18. Chen Y, Wu H, Chen Y, Liu R, Ye H, Liu S (2021) Design of new compact multi-layer quint-band bandpass filter. IEEE Access 9:139438–139445. https://doi.org/10.1109/ACCESS.2021.3116807
19. Gou J, Yu B, Maybank SJ, Tao D (2021) Knowledge distillation: A survey. Int J Comput Vis 129:1789–1819. https://doi.org/10.1007/s11263-021-01453-z
20. Xu C, Gao W, Li T, Bai N, Li G, Zhang Y (2023) Teacher-student collaborative knowledge distillation for image classification. Appl Intell 53(2):1997–2009. https://doi.org/10.1007/s10489-022-03486-4
21. Yang W, Xiao Y (2022) Structured pruning via feature channels similarity and mutual learning for convolutional neural network compression. Appl Intell 52(12):14560–14570. https://doi.org/10.1007/s10489-022-03403-9
22. Yang C, Liu H (2022) Channel pruning based on convolutional neural network sensitivity. Neurocomputing 507:97–106. https://doi.org/10.1016/j.neucom.2022.07.051
23. LeCun, Y, Denker, JS, Solla SA (1989) Optimal brain damage. In: Touretzky, DS (ed) Advances in neural information processing systems 2, NIPS Conference, Denver, Colorado, USA, November 27-30, 1989, pp 598–605. https://dl.acm.org/doi/10.5555/109230.109298
24. He Y, Dong X, Kang G, Fu Y, Yan C, Yang Y (2020) Asymptotic soft filter pruning for deep convolutional neural networks. IEEE Trans Cybern 50(8):3594–3604. https://doi.org/10.1109/TCYB.2019.2933477
25. Li G, Xu G (2021) Providing clear pruning threshold: A novel CNN pruning method via ℓ_0 regularisation. IET Image Process 15(2):405–418. https://doi.org/10.1049/ipr2.12030
26. Xu S, Chen H, Gong X, Liu K, Lü J, Zhang B (2021) Efficient structured pruning based on deep feature stabilization. Neural

Comput Appl 33(13):7409–7420. https://doi.org/10.1007/s00521-021-05828-8
27. Wei H, Wang Z, Hua G, Sun J, Zhao Y (2022) Automatic group-based structured pruning for deep convolutional networks. IEEE Access 10:128824–128834. https://doi.org/10.1109/ACCESS.2022.3227619
28. Frankle, J, Carbin M (2019) The lottery ticket hypothesis: Finding sparse, trainable neural networks. Paper presented at the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019
29. Li, H, Kadav, A, Durdanovic, I, Samet, H, Graf HP (2017) Pruning filters for efficient convNets. Paper presented at the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017
30. Liu, Z, Li, J, Shen, Z, Huang, G, Yan, S, Zhang C (2017) Learning efficient convolutional networks through network slimming. Paper presented at the IEEE International conference on computer vision, ICCV 2017, Venice, Italy, October 22-29, 2017
31. Molchanov, P, Tyree, S, Karras, T, Aila, T, Kautz J (2017) Pruning convolutional neural networks for resource efficient inference. Paper presented at the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017
32. Molchanov, P, Mallya, A, Tyree, S, Frosio, I, Kautz J (2019) Importance estimation for neural network pruning. Paper presented at the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019
33. Chen Y, Wen X, Zhang Y, Shi W (2021) CCPrune: Collaborative channel pruning for learning compact convolutional networks. Neurocomputing 451:35–45. https://doi.org/10.1016/j.neucom.2021.04.063
34. Li X (2018) Preconditioned stochastic gradient descent. IEEE Trans Neural Netw Learn Syst 29(5):1454–1466. https://doi.org/10.1109/TNNLS.2017.2672978
35. Zhang, C, Song, Q, Zhou, H, Ou, Y, Deng, H, Yang LT (2021) Revisiting recursive least squares for training deep neural networks. Preprint at https://arxiv.org/abs/2109.03220
36. Chen Y, Hero AO (2012) Recursive $\ell_{1,\infty}$ group lasso. IEEE Trans Signal Process 60(8):3978–3987. https://doi.org/10.1109/TSP.2012.2192924
37. Bruce, AL, Goel, A, Bernstein DS (2020) Recursive least squares with matrix forgetting. Paper presented at the 2020 American Control Conference, ACC 2020, Denver, CO, USA, July 1-3, 2020
38. Sherman J, Morrison WJ (1950) Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. Ann Math Stat 21:124–127. https://doi.org/10.1214/aoms/1177729893
39. Goodfellow I, Bengio Y, Courville A (2018) Deep learning. MIT press. https://doi.org/10.1007/s10710-017-9314-z
40. Krizhevsky A (2009) Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto
41. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324. https://doi.org/10.1109/5.726791
42. Simonyan, K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. Paper presented at the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015
43. Zhang G, Xu S, Li J, Guo AJX (2022) Group-based network pruning via nonlinear relationship between convolution filters. Appl Intell 52(8):9274–9288. https://doi.org/10.1007/s10489-021-02907-0

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

M.D. Tianzong Yu is a graduate of Hainan University within the School of Computer Science and Technology at Hainan University, China. His research is primarily focused on Deep Learning and Reinforcement Learning.

Dr. Chunyuan Zhang is the Associate Professor within the School of Computer Science and Technology at Hainan University, China. His research is primarily focused on Deep Learning and Reinforcement Learning.

M.D. Meng Ma is a graduate of Hainan University within the School of Computer Science and Technology at Hainan University, China. His research is primarily focused on Deep Learning and Reinforcement Learning.

M.D. Yuan Wang is a graduate of Hainan University within the School of Computer Science and Technology at Hainan University, China. His research is primarily focused on Deep Learning and Reinforcement Learning.