International Journal of Computational Intelligence Systems (2023) 16:17
https://doi.org/10.1007/s44196-023-00202-z

RESEARCH ARTICLE

Compression of Deep-Learning Models Through Global Weight Pruning Using Alternating Direction Method of Multipliers

Kichun Lee[1] · Sunghun Hwangbo[1] · Dongwook Yang[1] · Geonseok Lee[1]

Received: 24 October 2022 / Accepted: 12 February 2023

Abstract
Deep learning has shown excellent performance in numerous machine-learning tasks, but one practical obstacle in deep learning is that the amount of computation and required memory is huge. Model compression, especially in deep learning, is very useful because it saves memory and reduces storage size while maintaining model performance. Model compression in a layered network structure aims to reduce the number of edges by pruning weights that are deemed unnecessary during the calculation. However, existing weight pruning methods perform a layer-by-layer reduction, which requires a predefined removal-ratio constraint for each layer. Layer-by-layer removal ratios must be structurally specified depending on the task, causing a sharp increase in the training time due to a large number of tuning parameters. Thus, such a layer-by-layer strategy is hardly feasible for deep layered models. Our proposed method aims to perform weight pruning in a deep layered network, while producing similar performance, by setting a global removal ratio for the entire model without prior knowledge of the structural characteristics. Our experiments with the proposed method show reliable and high-quality performance, obviating layer-by-layer removal ratios. Furthermore, experiments with increasing layers yield a pattern in the pruned weights that could provide an insight into the layers' structural importance. The experiment with the LeNet-5 model using MNIST data results in a higher compression ratio of 98.8% for the proposed method, outperforming existing pruning algorithms. In the Resnet-56 experiment, the performance change according to removal ratios of 10–90% is investigated, and a higher removal ratio is achieved compared to other tested models. We also demonstrate the effectiveness of the proposed method with YOLOv4, a real-life object-detection model requiring substantial computation.

Keywords Network compression · Weight pruning · Non-convex optimization · Parallel computing

Abbreviations

DL	Deep learning
DNN	Deep neural network
CNN	Convolutional neural network
ADMM	Alternating direction method of multipliers
RPP	Relaxant probabilistic projection
LGD	L0-norm constrained gradient descent

Kichun Lee and Sunghun Hwangbo contributed equally to this work.

✉ Geonseok Lee
lgs5228@hanyang.ac.kr

Kichun Lee
skylee@hanyang.ac.kr

Sunghun Hwangbo
thehb01@gmail.com

Dongwook Yang
dongwook412@naver.com

[1] Department of Industrial Engineering, Hanyang University, Seongdong-gu, Seoul, South Korea

1 Introduction

As data accumulation and storage become easier and more processing methods are developed, studies related to deep learning that require a significant amount of computational power are being actively conducted. Deep learning models are used in various fields, such as for visual processing technology that generates useful information by analyzing images, natural language processing technology that understands and analyzes human language, and speech processing technology that synthesizes or converts human speech. Deep learning outperforms many existing techniques and continues to be developed. The superior performance of deep-learning models is possible because data-processing speeds driven by GPUs have advanced rapidly. However,

Published online: 24 February 2023

achieving high-quality performance of deep learning leads to an inherent problem where the number of weights of the model (i.e., the model size) increases. Because of this problem, the required memory increases with the model size, and the training time becomes longer. In addition, it may be difficult to deploy and apply a trained model on a real-time basis because the speed deteriorates as the amount of calculations increase. For example, the YOLOv4 model, which is known to be the fastest among object-detection techniques, achieved about 65 FPS (frame per second) in MS COCO data; however, this was only possible when using an expensive GPU (Tesla V100).

Many approaches have been proposed to select an optimal or sub-optimal ensemble of traditional ML classifiers [1–4]. Recently, due to the inherent model-size issues in deep learning, network compression techniques have emerged as a new and challenging area to alleviate the problem of rapidly increasing memory and computational requirements. In particular, deploying deep neural networks (DNNs) to devices requiring real-time processing is a very promising research subject. Model compression aims to save memory, reduce the storage size of the model, and reduce computational requirements while taking full advantage of pretrained models. In the past few years, various model compression techniques have been in development, taking into account the tradeoff between the degree of compression and the accuracy.

According to [5], research into model compression in DNNs can be classified into four major categories: compact-model methods, tensor decomposition, data quantization, and network sparsification. A compact-model technique aims to create a smaller model itself that achieves acceptable performance among several candidates. Different from compact-model methods, the three other categories compress a DNN model by modifying the existing model in training sessions rather than creating a new model. Tensor decomposition decomposes an existing matrix (or tensor in general) into a matrix with smaller dimensionality. Data quantization is a method of compressing a DNN model by reducing the bit-width of data. Lastly, network sparsification simplifies the computational graph used for training a DNN model. Each of these four categories can be combined for better performance. Sharing a similar spirit to accomplish DNN model compression, each category has different characteristics in terms of the accuracy preservation degree, compression degree, structural information, and utilization method.

In particular, weight pruning in network sparsification aims to obtain sparse weights by removing edges from a deep-learning network graph. A prior work [6] solves a non-convex minimization problem using the alternating direction method of multipliers (ADMM) to maintain performance while sparsing weights based on existing CNN models, effectively pruning the weight of the model. This method has some advantages over other methods; for example, it has a higher compression ratio and can quickly reach a convergence rate by setting removal ratios. However, since model compression is generally used for large models, the method is practically limited in that it has to set removal ratios for each layer to [6] perform layer-wise pruning. For example, YOLOv4 has about 100 convolution layers, and it takes a substantial amount of time to experiment with the removal ratio of each layer. Obviously, the situation worsens for larger models because various hyperparameters (optimizer, learning rate, epoch, etc.) are used to find the optimal model and several tuning hyperparameters are used to prune weights by ADMM.

Our research aims to solve the above mentioned problems. Specifically, our approach formulates an optimization problem by applying ADMM to the entire layer instead of using layer-by-layer pruning. When structurally pruning a large network model with layer-by-layer and filter-by-filter pruning ratios, for example, it is difficult to find optimal parameters through a relatively small number of experiments. In fact, this issue often becomes a reality because a great number of experiments need to be conducted to find appropriate removal ratios when applying this technique to a large-size, real-life DNN model. Our method is able to easily grasp trends in the pruning degrees for the layers in a DNN model (e.g., different pruning ratios for the layers close to the input, those close to the output, and those for the intermediate layers). As a result, the application of our method can provide a base policy for pruning ratios in the layers of a DNN model. According to the findings in [6], the layers in charge of feature extraction, which are usually located near the input in tasks dealing with images, must be pruned at a small removal ratio. Our method prunes a DNN model without structural removal ratios, and our experiments show that it effectively prunes a deep layered model with a global removal ratio.

The structure of this paper is as follows. In Sect. 2, we briefly provide related studies on weight pruning and preliminaries of ADMM. Section 3 contains a detailed description of our proposed model. Section 4 compares the proposed model with a few selected pruning models through experiments, showing that the proposed model has a higher compression ratio. This section also shows the proposed weight pruning technique applied to YOLOv4, demonstrating the model's effectiveness for a large and practical model. Section 5 concludes the paper by suggesting future research directions.

2 Background

2.1 Related Works

Among the categories of neural network model compression, network sparsification reduces the number of computations required and the size of the model by pruning unnecessary information. Pruning can be divided into weight pruning and neuron pruning. Weight pruning is a method of reducing the number of edges in a computational graph (to prune relatively insignificant or redundant weights), and neuron pruning is a method of reducing the number of nodes (to prune unnecessary nodes). In addition, pruning methods also can be divided in various ways according to the sparse structure: element-wise, vector-wise, and block-wise [7]. An element-wise method, also called unstructured pruning, evaluates the contribution of each weight element to the entire network. Removing insignificant connections without assumptions on the network structures, this method achieves gains in both the model flexibility and the predictive power. On the other hand, vector-wise and block-wise methods reveal compact network structures effectively by eliminating parameter groups instead of individual weights. Vector-wise methods [8, 9] estimate the importance of column vectors in the weight matrix and then prune a fixed set of groups by their priority. Similarly, block-wise methods [10, 11] divide the weight matrix into subblocks and consider each of the subblocks as a basic pruning unit. Unfortunately, these structured sparsity methods often fail to escalate the model accuracy due to the excessive loss of information. Our proposed model corresponds to ADMM-based weight pruning with an element-wise sparse method.

Most of the element-wise pruning methods in network sparsification are performed based on heuristic search. A heuristic-based method does not guarantee that it can effectively maintain the performance, so large performance decreases may occur. Therefore, in recent years, studies that perform pruning through optimization rather than heuristic methods have been preferred. Optimization-based methods can find less important or redundant information more effectively than heuristic-based methods, and they can also obtain higher performance. The proposed method adopts an optimization-based method to perform pruning while maintaining the existing performance.

Indeed, weight pruning was inspired by [12] and has been studied extensively. This work uses a method called optimal brain damage (OBD), which reduces the size of the model by removing information with small saliency of the second derivative of the objective function related to the weight. The optimal brain surgeon (OBS) method [13] incorporated weight pruning, which was a new technology that complemented the disadvantages of [12]. Since these two studies, weight pruning using various other methods has also been proposed. The study in [14] prunes the weights through the sensitivity of each layer based on the genetic algorithm, and then performs fine tuning on the pruned model based on the knowledge distillation framework. Another study [15] attempts effective weight pruning by solving the L0-norm constrained optimization problem through relaxant probabilistic projection (RPP) and L0-norm constrained gradient descent (LGD).

The algorithm proposed in this paper performs weight pruning based on optimization in an element-wise sparse structure that eliminates structural settings. Accordingly, we compare our method with other weight pruning methods performed on an element-wise structure in this experiment. Deep compression [16] attempts model compression using three stage pipelines consisting of pruning, trained quantization, and Huffman coding. In the weight pruning step, small weights are heuristically pruned and retrained, and model capacity is reduced by 9 to 13 times. Netpruning [17] eliminates redundant connections through three steps. The first step is to train the importance of connections. The second step removes unnecessary connections, and the third step finally retrains the network. Synthesizing DNN in the seed architecture, NeST [18] removes connections that are considered unnecessary through magnitude values to avoid duplication. To verify the effectiveness of the proposed global pruning, we further compare it with other structured pruning methods. Filter pruning proposed by Li et al. [19] removed filters with low weight magnitudes to reduce the redundancy in CNNs. NISP [20] measured the importance of filters based on their corresponding reconstruction errors in the next layer. HRank [21] mathematically proved that filters with lower ranks are less important to accuracy. CNN-FCF [22] presented an effective CNN compression approach which performs filter selection and filter learning jointly in a unified optimization scheme. DCP [23] proposed an iterative greedy algorithm to solve the channel selection problem considering both reconstruction error and discriminative power.

ADMM [24], an effective method for solving optimization problems, is widely used because of its parallel computing abilities. ADMM, which shows good performance, is often adopted for composite optimization problems, while gradient descent methods are mainly used for simple optimization problems. Noticeably, the optimization problem of the weight pruning method used in this study cannot be solved by gradient descent because differentiation is impossible and non-convex functions are included. Therefore, when using ADMM, we divided the original problem into two sub-optimization problems: one can be solved with gradient descent, and the other can be solved analytically. The study in [6] performs ADMM by constructing an optimization problem for each layer in weight pruning. Specifically, this method adds a cardinality function to the constraint of the optimization problem, and then performs pruning by setting a removal ratio for each layer. StructADMM [25] prunes weights for various structure types, such as filter-wise, shape-wise, and channel-wise sparsity, and similarly constructs an optimization problem and solves it through ADMM. We configure our method similarly to other ADMM-based methods. However, when comparing it with other methods while adopting a global removal ratio, we achieve effective sparsity without loss of performance in a short time.

2.2 Preliminary: ADMM

ADMM is a popular technique used for solving convex optimization problems in machine learning and deep leaning, making possible a large-scale optimization [26]. Recent works also demonstrate that under certain conditions, the ADMM is guaranteed to converge for non-convex problems [27]. Specifically, the ADMM can separate the variables and decompose the problem into two subproblems. We notice that the loss function associated with a constraint in this study includes a non-convex cardinality function to induce the sparsity of the weights.

Basically, the loss function of DNN consists of a basic loss $f_0(x)$ and a regularizer $h(x)$. ADMM separates the variable, x, in problem (1) and transforms it into problem (2). After that, we induce the augmented Lagrangian in (3):

$$\min_{x} f_0(x) + h(x), \tag{1}$$

$$\min_{x,z} \quad f_0(x) + h(x) \quad \text{s.t.} \quad x - z = 0, \tag{2}$$

$$L_\rho(x, z, \nu) = f_0(x) + h(z) + \frac{\rho}{2}\|x - z\|_2^2 + \nu^T(x - z). \tag{3}$$

In (3), we update the primal variables x and z and the Lagrangian multiplier ν while performing ADMM iterations, as shown in (4), (5), and (6):

$$x^{(k+1)} = \arg\min_{x} L_\rho(x, z^{(k)}, \nu^{(k)}), \tag{4}$$

$$z^{(k+1)} = \arg\min_{z} L_\rho(x^{(k+1)}, z, \nu^{(k)}), \tag{5}$$

$$\nu^{(k+1)} = \nu^{(k)} + \rho(x^{(k+1)} - z^{(k+1)}). \tag{6}$$

If ν is transformed into $\mu = \frac{1}{\rho}\nu$, $\frac{\rho}{2}\|x - z\|_2^2 + \nu^T(x - z)$ can be transformed into $\frac{\rho}{2}\|x - z + \mu\|_2^2 - \frac{\rho}{2}\|\mu\|_2^2$. As a result, (4), (5), and (6) can be changed to (7), (8), and (9):

$$x^{(k+1)} = \arg\min_{x} f_0(x) + \frac{\rho}{2}\|x - z^{(k)} + \mu^{(k)}\|_2^2, \tag{7}$$

$$z^{(k+1)} = \arg\min_{z} h(z) + \frac{\rho}{2}\|x^{(k+1)} - z + \mu^{(k)}\|_2^2, \tag{8}$$

$$\mu^{(k+1)} = \mu^{(k)} + x^{(k+1)} - z^{(k+1)}. \tag{9}$$

Finally, the optimal x can be obtained by sequentially solving the Eqs. (7), (8), and (9).

3 Global Weight Pruning

To solve the shortcomings of the technique described in [6], the proposed algorithm in this paper performs weight pruning with a global removal ratio, denoted by global weight pruning, rather than layer-wise removal ratios. Although it prunes a network without structural information, it runs sufficiently fast, even when applied to a large model. In addition, layers closer to the network input need to be pruned less than other layers to maintain input diversity and maintain performance. Our experiments show that the proposed model can automatically prune the first layer less.

3.1 Steps of the Proposed Method

The proposed method proceeds in four steps, as shown in Fig. 1. First, we train a DNN model to find the weight that increases the model accuracy, similar to the training of a general deep-learning model. In the ADMM step, we decompose the global weight pruning problem into two subproblems and solve them iteratively, which is the essence of the proposed method. The resulting solutions force the value of unnecessary weights to converge towards zero. In the next pruning step, we keep the weights with the large magnitudes and set the rest to

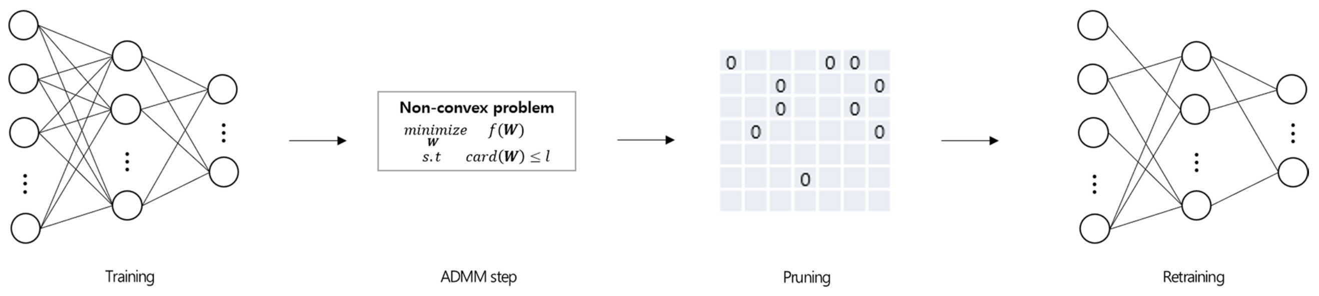

Fig. 1 Steps of the proposed method

zero. Finally, we fine-tune the remaining non-zero weights to recover the inference accuracy.

3.2 Formulation of the Proposed Model

In this section, we describe the proposed model in detail mathematically. The first step involves a general DNN training step, and we assume that weights, $\boldsymbol{W}$, of the pretrained model exist. The pretrained model possibly includes all machine-learning tasks through (original) loss functions, such as classification, regression, object detection, and segmentation.

For the second ADMM step, we first set the original loss function with an additional regularization term and follow the sequence described in Sect. 2.2:

$$Loss(\boldsymbol{W}) = origin\_loss + \lambda \sum_{i=1}^{l} \|W_i\|_2^2, \tag{10}$$

where $\boldsymbol{W} = \{W_1, \dots, W_l\}$ is the set of vectorized multi-dimensional tensor weights for layer i and W_i corresponds to the vector $W_i \in \mathbb{R}^{d_i}$ of the $d_i \in \mathbb{R}$ dimension. The vectorization of tensor weights can be a concatenation of the row vectors in the weight matrix between two layers, and the total number of layers is l. The first term can be thought of as the cross entropy loss in the case of classification, the mean-squared error in the case of regression, and some specific loss function adopted by each algorithm. For example, the loss function of the YOLOv4 object-detection model is a combination of the coordinates of a bounding box, the confidence of whether an object is included or not, and the class information of an object. The second term pertains to regularization with the Frobenius norm by default. It is easily differentiable and represents the squared energy.

To perform weight pruning, we introduce the cardinality function as a constraint in Eq. (10). Here, cardinality means the number of non-zero elements. To control the number of pruned weights in the network, we constrain the number of non-zero elements in the entire network to be less than a global parameter, n. The formulation is as follows:

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & Loss(\boldsymbol{W}) \\ \text{s.t.} \quad & cardinality(\boldsymbol{W}) < \sum_{i}^{l} n_i = n, \end{aligned} \tag{11}$$

where the number of elements that have not been removed for all layers is n. As a tuning parameter, n is specified in advance by the user; for small n, the network will be highly sparse. One needs to set n carefully to avoid either too sparse or too dense networks in the results. Inherently, the number of elements that have not been removed for layer i is n_i, which automatically emerges during pruning. Equation (10) can be easily solved with gradient descent, but the non-convex cardinality function in Eq. (11) means the gradient descent technique cannot guarantee the optimal solution. Thus, we modify the equation to use ADMM, which can be applied to non-convex optimization problems as follows:

$$\min_{\boldsymbol{W}} \quad Loss(\boldsymbol{W}) + h(\boldsymbol{W}), \tag{12}$$

where $h(\boldsymbol{W})$ is an indicator function for the cardinality constraint as follows:

$$h(\boldsymbol{W}) = \begin{cases} 0, & cardinality(\boldsymbol{W}) < n \\ \infty, & cardinality(\boldsymbol{W}) \geq n. \end{cases} \tag{13}$$

We separate the variable from Eq. (12) and modify it as shown in Eq. (14):

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & Loss(\boldsymbol{W}) + h(\boldsymbol{Z}) \\ \text{s.t.} \quad & \boldsymbol{W} = \boldsymbol{Z}. \end{aligned} \tag{14}$$

Next, the augmented Lagrangian of Eq. (14) is written as follows:

$$\begin{aligned} L_\rho(\boldsymbol{W}, \boldsymbol{Z}) = & Loss(\boldsymbol{W}) + h(\boldsymbol{Z}) \\ & + \frac{\rho}{2}\|\boldsymbol{W} - \boldsymbol{Z} + \boldsymbol{\mu}\|_F^2 - \frac{\rho}{2}\|\boldsymbol{\mu}\|_F^2, \end{aligned} \tag{15}$$

where ρ is a penalty parameter indicating the step size. The scaled Lagrangian $\boldsymbol{\mu} = \frac{\boldsymbol{\nu}}{\rho}$ is a variable expressed to simplify the equation with the original Lagrangian multiplier $\boldsymbol{\nu}$. Finally, the ADMM equation is rewritten and $\boldsymbol{W}$, $\boldsymbol{Z}$, and $\boldsymbol{\mu}$ are updated for iteration k:

$$\boldsymbol{W}^{(k+1)} = \arg\min_{\boldsymbol{W}} Loss(\boldsymbol{W}) + \frac{\rho}{2}\|\boldsymbol{W} - \boldsymbol{Z}^{(k)} + \boldsymbol{\mu}^{(k)}\|_F^2, \tag{16}$$

$$\boldsymbol{Z}^{(k+1)} = \arg\min_{\boldsymbol{Z}} h(\boldsymbol{Z}) + \frac{\rho}{2}\|\boldsymbol{W}^{(k+1)} - \boldsymbol{Z} + \boldsymbol{\mu}^{(k)}\|_F^2, \tag{17}$$

$$\boldsymbol{\mu}^{(k+1)} = \boldsymbol{\mu}^{(k)} + \boldsymbol{W}^{(k+1)} - \boldsymbol{Z}^{(k+1)}. \tag{18}$$

In the above equations, we obtain $\boldsymbol{W}$ through Eq. (16) and then find $\boldsymbol{Z}$ through Eq. (17). After that, $\boldsymbol{\mu}$ can be obtained simply by gradient ascent through Eq. (18).

The first term of Eq. (16) uses the loss function, which can be differentiated from the pretrained model. The second term can also be differentiated. In general, it can be solved easily by gradient descent as in Eq. (19):

$$W_i^{(k+1)} = W_i^{(k)} - \alpha\Big(\frac{\partial Loss(\boldsymbol{W}^{(k)})}{\partial W_i^{(k)}} + \rho(W_i^{(k)} - Z_i^{(k)} + \mu_i^{(k)})\Big). \tag{19}$$

Instead of inner iterations generated by gradient descent, depending on the form of $Loss(\cdot)$, one can obtain a closed form solution for updating $W_i^{(k+1)}$. Equation (17) cannot be solved through gradient descent, so we solve it using projection, similar to the approach used in [6]:

$$\begin{aligned} \boldsymbol{Z}^{(k+1)} &= \arg\min_{\boldsymbol{Z}} h(\boldsymbol{Z}) + \sum_{i=1}^{l} \frac{\rho}{2} \| W_i^{(k+1)} - Z_i + \mu_i^{(k)} \|_2^2 \\ &= \arg\min_{\boldsymbol{Z} \in \boldsymbol{C}} \sum_{i=1}^{l} \frac{\rho}{2} \| W_i^{(k+1)} - Z_i + \mu_i^{(k)} \|_2^2, \\ &= Proj_{\boldsymbol{C}}(\boldsymbol{W}^{(k+1)} + \boldsymbol{\mu}^{(k)}) \end{aligned} \tag{20}$$

where $\boldsymbol{C} = \{\boldsymbol{Z} \mid cardinality(\boldsymbol{Z}) < n\}$. Notice that $\boldsymbol{C}$ is non-convex, and the projection operator $Proj_{\boldsymbol{C}}(\cdot)$ is not unique. To make the number of non-zero elements in $\boldsymbol{W}^{(k+1)} + \boldsymbol{\mu}^{(k)}$ less than the number specified by the user (n), we shrink the elements in $\boldsymbol{W}^{(k+1)} + \boldsymbol{\mu}^{(k)}$ zero except for the first $n-1$ elements in descending order by absolute value. In addition, we use the initial value $\boldsymbol{W}^{(0)}$ as the trained weight, the initial value $\boldsymbol{Z}^{(0)}$ as $Proj_{\boldsymbol{C}}(\boldsymbol{W}^{(0)})$, and $\boldsymbol{\mu}^{(0)}$ as a matrix with all zero elements. Being a good optimization technique in many applications, ADMM might undergo a great number of iterations to converge to a final solution when handling non-convex problems. To alleviate the computational burden, we mask the zero weights, and then retrain the DNN with the remaining non-zero weights while freezing the masked ones to 0. Noticeably, the retraining step allows fast convergence to a desired solution from the good initial point with only a few parameters to be fine-tuned. In this way, we can restore the accuracy of the pruned network such that it may achieve performance better than or at least comparable with the pretrained model.

The proposed global weight pruning automatically seeks a sparse set of weights without specifying layer-by-layer removal ratio. Algorithm 1 describes the overall process of our proposed method which consists of four steps: pretraining, ADMM iterations, pruning, and retraining. Algorithm 1 takes data as the input and then returns the pruned weight. The initial value settings of $\boldsymbol{W}$,$\boldsymbol{Z}$, and $\boldsymbol{\mu}$ used in the ADMM step correspond to lines 4–6. The ADMM step corresponding to Eqs. 16, 17, and 18 proceeds in lines 7–13, and a sparse matrix can be obtained by performing pruning on $\boldsymbol{W}$ in line 14. After that, it freezes the zero weight and finally performs retraining to obtain a final model with sparsity and comparable performance (lines 15–18). Algorithm 2 describes a function that performs projection, setting all elements equal to zero except for the first $n-1$ largest elements. The *Flatten* and *Reshape* functions flatten, or vectorize, the input $\boldsymbol{X}$ into a 1D sequence and recover the size of *flatten_X* back to the original input size, respectively. The *Top_n* function takes a vector as an input and returns the n-th value with a large value, which corresponds to the *threshold* of line 6. The projection ends with the process of setting weights smaller than the calculated *threshold* to zero (lines 7–9).

Algorithm 1 Proposed Model

```
1:  Input: Data
2:  Output: $W$
3:
4:  $W^0 = trained\_Model(Data)$                                          ▷ pretraining
5:  $Z^0 = Proj_C(W)^0)$
6:  $\mu^0$ = Matrix of zeros equal size to $W^0$
7:  for $k = 0, \ldots, k\_step - 1$ do                                    ▷ ADMM iterations
8:      for $epoch = 0, \ldots, epochs$ do
9:          $W^{(k+1)} = \arg\min_W Loss(W) + \frac{\rho}{2}\|W - Z^{(k)} + \mu^{(k)}\|_F^2$
10:     end for
11:     $Z^{(k+1)} = Proj_C(W^{(k+1)} + \mu^{(k)})$
12:     $\mu^{(k+1)} = \mu^{(k)} + W^{(k+1)} - Z^{(k+1)}$
13: end for
14: $W = Proj_C(W^{k\_step})$                                              ▷ pruning
15: Freezing zero weights
16: for $retrain\_epoch = 0,\ldots,retrain\_epochs$ do                      ▷ retraining
17:     $W = \arg\min_W Loss(W)$
18: end for
```

Algorithm 2 Projection
```
1: Input: $\boldsymbol{n}, \boldsymbol{X}$
2: Output: $projected\_X$
3:
4: $dim\_X = Dimension(\boldsymbol{X})$
5: $flatten\_X = Flatten(\boldsymbol{X})$
6: $threshold = Top\_n(abs(flatten\_X), \boldsymbol{n})$
7: $under\_threshold = abs(flatten\_X) < threshold$
8: $flatten\_X[under\_threshold] = 0$
9: $projected\_X = Reshape(flatten\_X, dim\_X)$
```

4 Experiments

We conducted experiments with neural networks of various sizes and three known models: LeNet-5, Resnet-56, and YOLOv4. In Sect. 4.1, we observe the effect of the previous-layer weight and the removal ratio for each layer by training the proposed model by gradually increasing layers. In Sect. 4.2, we compare our model with existing element-wise weight pruning models. In Sect. 4.3, we conduct experiments with various weight pruning models, such as filter-wise and channel-wise methods. Finally, Sect. 4.4 provides a real-life application to YOLOv4, which is a large object-detection model that actually needs weight pruning. The experiments are run on TensorFlow 2 using one NVIDIA RTX 3090 GPU and two RTX 6000. For the sake of simplicity, we denote the proposed pruning methods as global pruning.

Table 2 Removal ratio comparisons using different weight pruning models on LeNet-5

Model	Top-1 accuracy (base) (%)	Top-1 accuracy (pruned) (%)	Removal ratio (%)
Deep compression [16]	99.2	99.26	92.0
Netpruning [17]	99.2	99.23	91.7
NeST [18]	99.2	99.23	98.65
ADMM pruning [6]	99.2	99.2	98.6
Global pruning	99.43	99.54	98.8

Table 1 Removal ratio for each layer according to the number of layers

	Number of layers				
Layer	2	4	6	8	10
CONV1	76.74	85.24	89.06	77.78	82.64
CONV2	94.43	96.00	97.96	98.20	96.25
CONV3		90.59	95.51	98.15	96.58
CONV4		93.27	94.44	97.31	97.75
CONV5			93.20	96.16	98.34
CONV6			91.61	95.78	95.38
CONV7				94.21	93.94
CONV8				91.85	95.97
CONV9					94.31
CONV10					92.48
DENSE	89.69	89.28	88.33	86.95	86.26
Number of weights (all layers)	539,200	612,928	686,656	760,384	834,112
Number of weights (after pruning)	53,920	61,293	68,666	76,039	83,412
Accuracy (base, %)	98.76	99.05	99.12	99.09	99.21
Accuracy (pruned, %)	98.71	98.80	99.19	98.99	99.03

4.1 Convolution Neural Networks with Various Layers

In this experiment, we construct convolution neural network models by increasing the number of layers from two to ten to see if layers close to the input are relatively important among all the layers of the model. The experiment is conducted on MNIST data. We also investigate whether the removal ratios and performance of pruned models evolve according to the number of layers. Each layer, except the final fully connected (dense) layer, consists of a convolutional layer with filter configuration (3, 3, 64) and batch normalization. We increased the number of layers accordingly, applying weight pruning to examine the evolution of weight removal ratios for the layers. For simplicity, we denote the models as cnn-model k, where k is the total number of layers. In Table 1, we observe a pattern where the weights of the layers close to the input are not removed as much compared to those of the other layers. Specifically, the removal ratios of the first convolutional layer (CONV1) change from 76.74% for CNN-model 2 to 89.06% for CNN-model 6. For the following convolutional layers (CONV2 to CONV10), the minimum (90.59%) and the maximum (98.34%) removal ratios are greater than all of the removal ratios of the first convolutional layer (CONV1). In addition, the removal ratios for the dense layers range from 86.25% to 89.69%, which are not

Table 3 Weight pruning results on LeNet-5

Layer	Number of weights (base)	Number of weights (after pruning)	Removal ratio
CONV1	500	255	49.0%
CONV2	25,000	1,723	93.1%
DENSE 1	400,000	2,738	99.3%
DENSE 2	5,000	450	91.0%
Total	430,500	5,166	98.8%

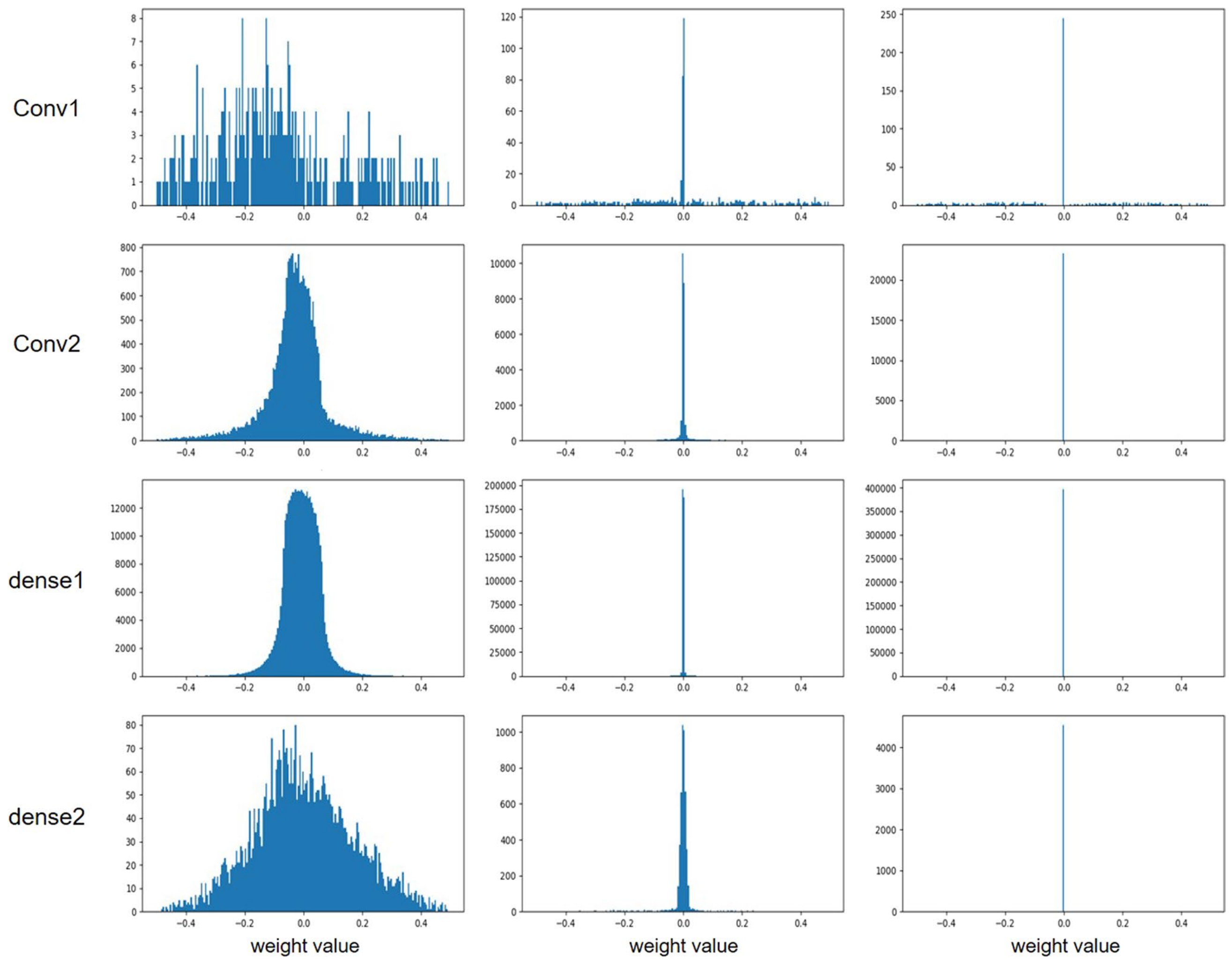

Fig. 2 Weight distributions on LeNet-5

much different from those of CNN-model 2 to CNN-model 10. We also notice that the accuracy of the pruned model is comparable with that of the original model.

4.2 LeNet-5

LeNet-5 is an image classification model that uses 28 by 28 images as an input, and it has a structure consisting of two convolutional layers and two dense layers. The experiment is conducted on MNIST data.

Before weight pruning, when the comparison models use LeNet-5, they all show the same performance. The experiment revealed the optimal parameters of the proposed model to be $\lambda = 0.01$ and $\rho = 0.004$. Table 2 compares the proposed model (global pruning) with element-wise weight pruning models, showing not only that the highest removal ratio is achieved by the proposed model but also that the highest accuracy, being greater than that of the initial model, is obtained by our model. We performed the proposed global pruning on the pretrained baseline model, producing 99.43% Top-1 accuracy, which is slightly higher than that of previous works [6, 16–18]. It is known that CNNs provide impressive performance on many visual tasks, yet their architectures are usually over-parameterized. Therefore, we aim to compress the pretrained models while preserving the discriminative ability.

In addition, Table 3 shows the removal ratio of each layer for the optimal model. In the convolutional layer responsible for feature extraction, it is interesting to observe that the layer close to the input has a low removal ratio, indicating the input variables initially possess valuable information. This result is consistent with findings of the previous experiment, where the layers are gradually added.

After convergence, we visualize the weight distribution for each layer to confirm the change in the weight distribution for LeNet-5. In Fig. 2, the left column shows the distribution from the initial model, the middle column shows that from the model of the ADMM steps, and the right column shows that from the model of the retraining steps. Since 98.8% of the weights are removed, it is clear that a large number of weights are close to zero in the model of the ADMM steps. The retraining steps further shrink the weights to zero. Given the highly compressed model, it is worthwhile mentioning that the accuracy of the final pruned model exceeds that of the initial model. This shows the ability of the proposed model to effectively compress deep layered networks without sacrificing performance.

4.3 ResNet-56

In this experiment, we apply weight pruning to ResNet-56 [28] using the CIFAR-10 dataset consisting of 10 classes and 32 by 32 images. ResNet, based on VGGNet [29] stacking of 3 by 3 convolutional layers, uses a residual block to solve the problem of improper training when the number of layers of the model increases. The ResNet model used in the experiment consists of a total of 56 layers, the number of parameters is 0.85 M, and the accuracy is 93.07%.

For the ResNet-56 model, we compared the experimental results of our weight pruning with existing filter or channel pruning methods [19–23]. Table 4 reports the accuracy and removal ratio of different models before and after pruning. Each model shows the accuracy of the base model and the accuracy after pruning, and NISP [20] shows the difference in accuracy between the pruned model and the base model. We conducted an experiment to remove 10% to 90% of the total weight for the Resnet-56 model. After weight pruning, 10% to 40% removal ratio further increased accuracy. From 50% to 90%, the accuracy decreased by 0.09%, 0.34%, 0.43%, 1.11%, and 2.29%, respectively, in comparison with that of the base ResNet-56 model.

All weight pruning models achieved a removal ratio of less than 50% while maintaining accuracy performance. The results show that our proposed model maintains the accuracy performance sufficiently, even at a removal ratio of 50%. For example, when the removal ratio of HRank is 68.10%, the accuracy is 90.72%. In this case, the performance preservation is 97.2% (90.72/93.26). In contrast, the proposed method produces a performance preservation of 99.5% (92.64/93.07) when the removal ratio is 70%, which

Table 4 Removal ratio comparisons using different weight pruning models on ResNet-56

Model	Top-1 accuracy (base) (%)	Top-1 accuracy (pruned) (%)	Removal ratio (%)
Filter pruning [19]	93.04	93.10	9.40
		93.06	13.70
NISP [20]	93.04	93.01	43.61
HRank [21]	93.26	93.17	42.40
		90.72	68.10
CNN-FCF [22]	93.14	93.38	43.09
		91.92	69.74
DCP [23]	93.80	93.49	49.24
Global pruning	93.07	93.14	10
		93.17	20
		93.17	30
		93.11	40
		92.98	50
		92.73	60
		92.64	70
		91.96	80
		90.78	90

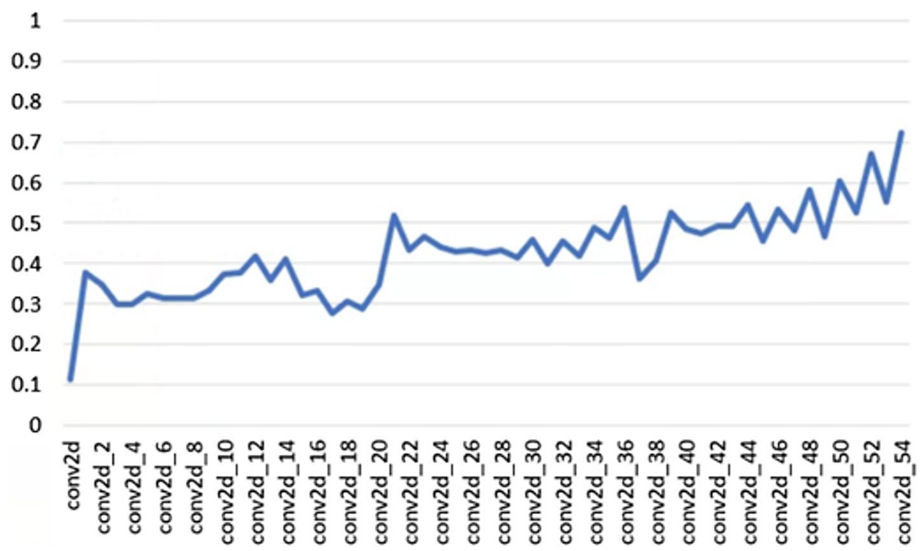

Fig. 3 Weight distributions on LeNet-5

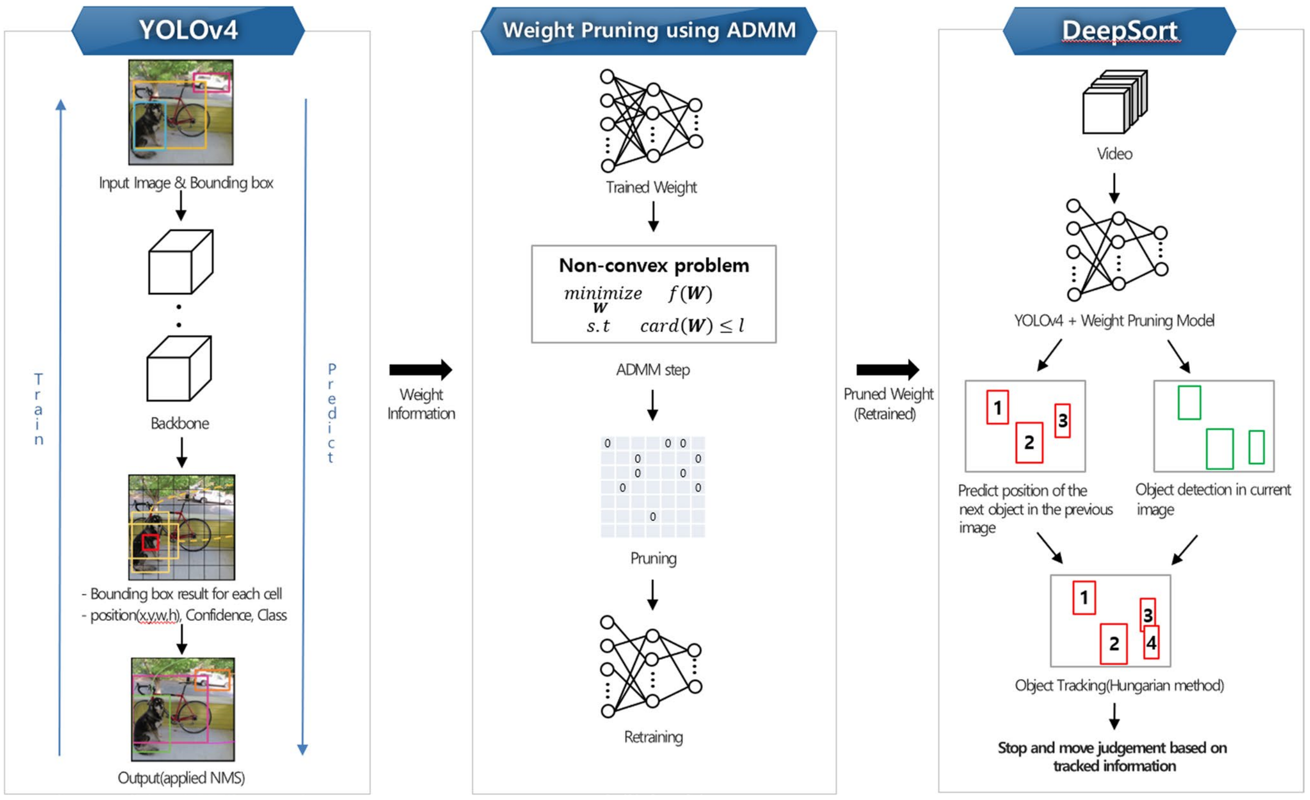
YOLOv4
Input Image & Bounding box
Backbone
- Bounding box result for each cell
- position(x,y,w,h), Confidence, Class
Output(applied NMS)
Train
Predict
Weight Information
Weight Pruning using ADMM
Trained Weight
Non-convex problem
$\underset{W}{minimize} \quad f(W)$
$s.t \quad card(W) \le l$
ADMM step
Pruning
Retraining
Pruned Weight (Retrained)
DeepSort
Video
YOLOv4 + Weight Pruning Model
Predict position of the next object in the previous image
Object detection in current image
Object Tracking(Hungarian method)
Stop and move judgement based on tracked information

Fig. 4 Application workflow

surpasses HRank. When comparing it with CNN-FCF, similar results are observed; global pruning achieves better performance preservation, even when the removal ratio is larger.

In addition, Fig. 3 shows the weight removal ratio for each layer for the Resnet-56 model when the removal ratio is set to 50%. As in the previous experiment, the removal ratio of the frontmost layer is the lowest at 11.57%. In addition, the removal ratio tends to increase as it approaches the last layer.

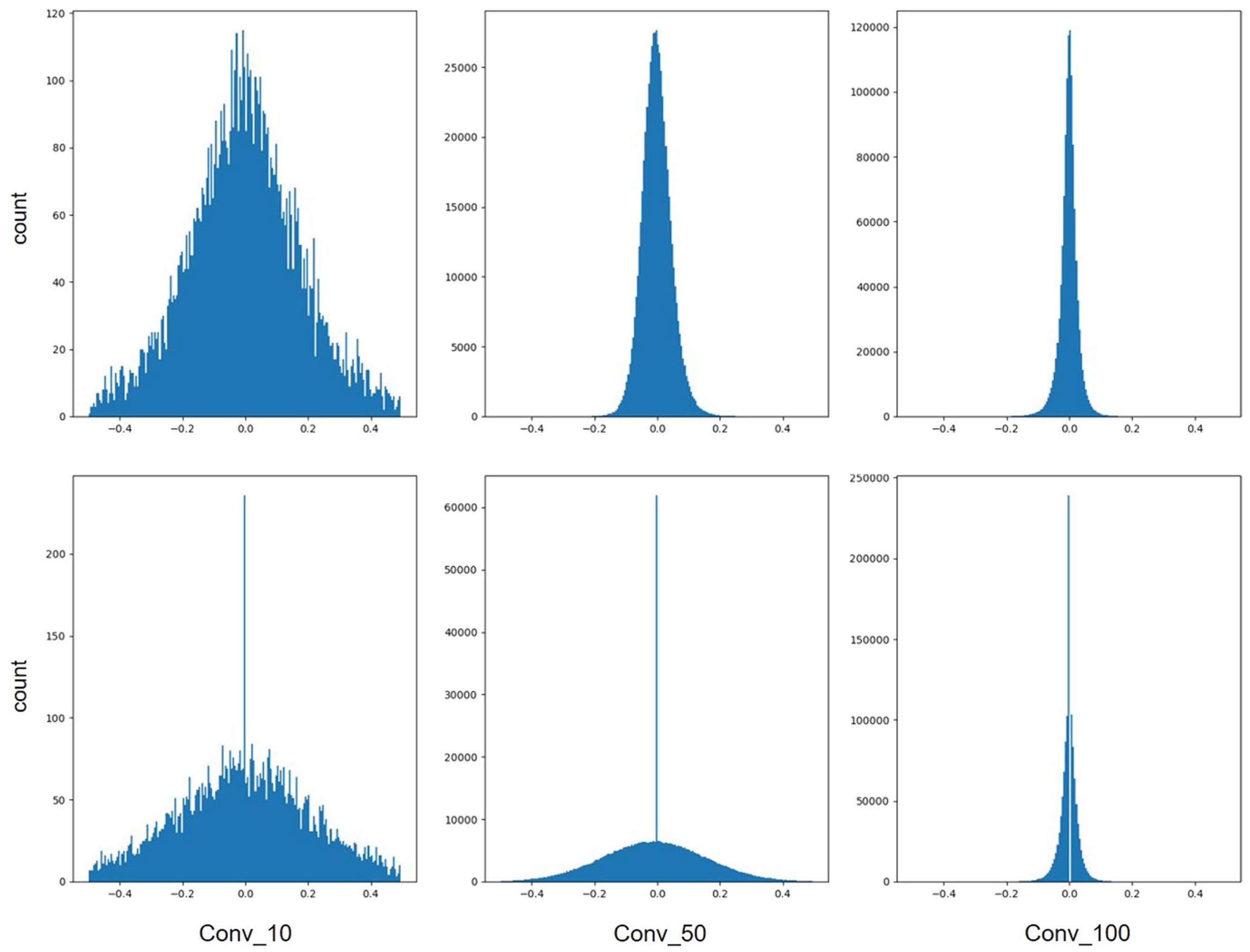

Fig. 5 Weight distributions on YOLOv4 are shown. The first column indicates the distribution of the 10th layer located in the beginning part, the second column is the distribution of the 50th layer located in the middle part, and the third column is the distribution of the 100th layer located in the last part

4.4 Real-Life Application

Among deep learning tasks with images and video frames, object detection and object tracking are very popular. Research in these areas is being actively conducted. To show the practicality of this study, we apply weight pruning to YOLOv4 [30], using the COCO dataset and combining Deepsort [31], which is an object tracking model. In short, YOLO is a popular object-detection algorithm, possessing similar performance to Fast R-CNN [32], which has shown high performance in the object-detection field. It has recently achieved tremendous speed improvement. While Fast R-CNN has a speed performance of 0.5 FPS, YOLO has a speed performance of 45 FPS, enabling real-time object detection. YOLO is constantly evolving. The fourth version, YOLOv4, uses a variety of the latest deep learning techniques to improve performance.

The aim is to compress video by removing unnecessary (i.e., no movement) states from CCTV video data utilizing YOLOv4 and Deepsort. To determine the unnecessary state of a certain object, the object tracking model (Deepsort) uses object information detected by YOLOv4. Afterwards, it goes through video compression by removing the state of no movement from the identified movement information. The entire workflow is depicted in Fig. 4.

As mentioned, YOLO is faster than other object-detection models, making real-time object detection possible. However, when used on a device with low computing power, such as a mobile device, real-time detection is hardly possible due to the large amount of computation required. In the case of YOLOv4, approximately 100 convolution layers exist, and many experiments are needed to structurally set the layer-by-layer removal ratios. Therefore, when using such a large model for an object-detection task, it is appropriate to apply our model.

The experimental results are as follows. We find that a removal ratio of 20%, as a result of weight pruning on YOLOv4, keep the mAP (mean average precision) of the base model similar. mAP is measurement of object-detection performance, and it describes the average of the area below the graph plotted through precision and recall for each class. Figure 5 shows the weight distribution of the representative layers located at the beginning (conv2d_10), middle (conv2d_50), and end (conv2d_100). The first row shows the weight distribution from the base YOLOv4, and the second row shows that from the pruned model. In the conv2d_10 layer, 164 of 8192 weights, which is 2%, become zero. In the conv2d_50 layer, 55493 of 589824 weights, which is 9.4%, become zero. In the conv2d_100 layer, 239034 of 1179648 weights, which is 20.2%, become zero. Therefore, the conv2d_10 layer has a small ratio of zeros, while the remaining layers have a larger ratio of weights to zero.

5 Conclusions

In this study, we propose an ADMM-based element-wise weight pruning method that sets only the removal ratio of the entire layer during the training process. Weight pruning using traditional ADMM-based optimization methods requires structurally setting a large number of removal ratios, such as by using a layer-wise, filter-wise, or channel-wise method. Therefore, in large models that actually require weight pruning, it is difficult to find the optimal removal ratio and the training times can be very large. We prune the weights simply by using one removal ratio, making only a small variation to the existing ADMM model. This achieves similar performance to ADMM-based models but with less training time.

In the model with convolution neural networks with various layers, we show that the layer closest to the input achieves a smaller removal ratio. This means that if we remove many of those layers, we will lose important information. The LeNet-5 experiment achieves higher removal ratios than element-wise based methods. In the Resnet-56 experiment, which is compared with various weight pruning methods (e.g., filter and channel-wise methods), a removal ratio of 50% is achieved. The higher the removal ratio, the lower the accuracy, which can be selected at the user's discretion. In addition, the proposed model is also applied to a project that used YOLOv4, which is a very large model. This shows that our proposed technique can be applied to a large model to provide sufficient weight pruning. However, as a limitation, the ADMM optimization method does not guarantee optimum in non-convex problems. However, we note that this limitation is universal for objective functions in deep learning. In addition, though we attempted to verify the improved speed through our model compression, admittedly we were unable to observe speed improvement it due to the limitations of software and hardware. In the future, we envision verifying it when a comparison experiment on speed is possible with software support.

In future research, it is necessary to develop a model that can achieve a higher compression ratio while reducing the number of experiments by changing the ρ value, which is a very sensitive parameter in our model, to learnable parameter.

Author Contributions KL designed and supervised the research. SH performed experiments and wrote the manuscript. DY helped in performing experiments and writing the manuscript. GL reviewed and edited the manuscript.

Funding This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(*MSIT) (No.2018R1A5A7059549), *Ministry of Science and ICT. This work was also supported by "Human Resources Program in Energy Technology" of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resource from the Ministry of Trade, Industry & Energy, Republic of Korea. (No. 20204010600090).

Availability of Data and Materials The datasets are available in the web. Descriptions are in the text.

Declarations

Conflict of Interest All the authors declare that there is no conflict of interest.

Ethics Approval and Consent to Participate All the authors agreed with the content of this paper.

Consent for Publication All the authors reviewed the final manuscript and approved it for publication.

References

1. Brzezinski, D., Stefanowski, J.: Reacting to different types of concept drift: the accuracy updated ensemble algorithm. IEEE Transac. Neural Netw. Learn. Syst. **25**(1), 81–94 (2013)

2. Guo, H., Liu, H., Li, R., Changan, W., Guo, Y., Mingliang, X.: Margin & diversity based ordering ensemble pruning. Neurocomputing **275**, 237–246 (2018)
3. Petchrompo, S., Coit, D.W., Brintrup, A., Wannakrairot, A., Parlikad, A.K.: A review of Pareto pruning methods for multi-objective optimization. Computers Ind. Eng. **19**, 108022 (2022)
4. Goel, K., Batra, S.: Two-level pruning based ensemble with abstained learners for concept drift in data streams. Expert. Syst. **38**(3), e12661 (2021)
5. Deng, L., Li, G., Han, S., Shi, L., Xie, Y.: Model compression and hardware acceleration for neural networks: a comprehensive survey. Proc. IEEE **108**(4), 485–532 (2020)
6. 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. In: Proceedings of the European conference on computer vision (ECCV), pp. 184–199. Springer (2018)
7. Qingbei, G., Xiao-Jun, W., Josef, K., Zhiquan, F.: Weak sub-network pruning for strong and efficient neural networks. Neural Netw. **144**, 614–626 (2021)
8. Zhuliang, Y., Shijie, C., Wencong, X., Chen, Z., Lanshun, N.: Balanced sparsity for efficient dnn inference on gpu. In: Proceedings of the AAAI conference on artificial intelligence. pp. 5676–5683 (2019)
9. Maohua, Z., Tao, Z., Zhenyu, G., Yuan, X.: Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pp 359–371 (2019)
10. Ji, Y., Liang, L., Deng, L., Zhang, Y., Zhang, Y., Xie, Y.: Tetris: tile-matching the tremendous irregular sparsity. In: Advances in neural information processing systems, p. 31. MIT Press (2018)
11. Lin, S., Ji, R., Li, Y., Deng, C., Li, X.: Toward compact convnets via structure-sparsity regularized filter pruning. IEEE Transac. Neural Netwo. Learning Syst. **31**, 574–588 (2019)
12. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. In: Advances in neural information processing systems, p. 2. MIT Press (1989)
13. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: optimal brain surgeon. Morgan Kaufmann, Rome (1993)
14. Yiming, H., Siyang, S., Jianquan, L., Xingang, W., Qingyi, G.: A novel channel pruning method for deep neural network compression. arXiv preprint arXiv:1805.11394, (2018)
15. Liang, L., Deng, L., Zeng, Y., Xing, H., Ji, Y., Ma, X., Li, G., Xie, Y.: Crossbar-aware neural network pruning. IEEE Access **6**, 58324–58337 (2018)
16. Song, H., Huizi, M., William J, D.: Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, (2015)
17. Song, H., Jeff, P., John, T., William J, D.: Learning both weights and connections for efficient neural networks. arXiv preprint arXiv:1506.02626, (2015)
18. Dai, X., Yin, H., Jha, N.K.: Nest: a neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transac. Computers **68**(10), 1487–1497 (2019)
19. Hao, L., Asim, K., Igor, D., Hanan, S., Hans Peter G.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710, (2016)
20. Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V.I., Han, X., Gao, M., Lin, C.-Y., Davis, L.S.: Nisp: pruning networks using neuron importance score propagation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 9194–9203. IEEE (2018)
21. Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L.: Hrank: filter pruning using high-rank feature map. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1529–1538. IEEE (2020)
22. Li, T., Wu, B., Yang, Y., Fan, Y., Zhang, Y., Liu, W.: Compressing convolutional neural networks via factorized convolutional filters. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 3977–3986. IEEE (2019)
23. Zhuangwei, Z., Mingkui, T., Bohan, Z., Jing, L., Yong, G., Qingyao, W., Junzhou, H., Jinhui Z.: Discrimination-aware channel pruning for deep neural networks. arXiv preprint arXiv:1810.11809, (2018)
24. Boyd, S., Parikh, N., Chu, E.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Now Publishers Inc (2011)
25. Tianyun, Z., Shaokai, Y., Kaiqi, Z., Xiaolong, M., Ning, L., Linfeng. Z., Jian, T., Kaisheng, M., Xue L., Makan F.: et al. Structadmm: a systematic, high-efficiency framework of structured weight pruning for dnns. arXiv preprint arXiv:1807.11091, (2018)
26. Chen, T.-A., Yang, D.-N., Chen, M.-S.: AlignQ: alignment quantization with ADMM-based correlation preservation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 12538–12547. IEEE (2022)
27. Kumar, C., Rajawat, K.: Network dissensus via distributed ADMM. IEEE Transac. Signal Process. **68**, 2297–2301 (2020)
28. Kaiming, H., Xiangyu, Z., Shaoqing, R., Jian, S.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778. IEEE (2016)
29. Karen, S., Andrew, Z.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, (2014)
30. Bochkovskiy, A., Wang, C.-Y., Mark Liao, H.-Y.: Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934, (2020)
31. Wojke, N., Bewley, A., Paulus, D.: Simple online and realtime tracking with a deep association metric. In: 2017 IEEE international conference on image processing (ICIP), pp. 3645–3649. IEEE (2017)
32. Girshick, R.: r-cnn fast. In: Proceedings of the IEEE international conference on computer vision, pp. 1440–1448. IEEE (2015)