Neural Computing and Applications
https://doi.org/10.1007/s00521-024-09719-6

ORIGINAL ARTICLE

A roulette wheel-based pruning method to simplify cumbersome deep neural networks

Kit Yan Chan[1] · Ka Fai Cedric Yiu[2] · Shan Guo[2] · Huimin Jiang[3]

Received: 23 May 2023 / Accepted: 25 March 2024
© The Author(s) 2024

Abstract
Deep neural networks (DNNs) have been applied in many pattern recognition or object detection applications. DNNs generally consist of millions or even billions of parameters. These demanding computational storage and requirements impede deployments of DNNs in resource-limited devices, such as mobile devices, micro-controllers. Simplification techniques such as pruning have commonly been used to slim DNN sizes. Pruning approaches generally quantify the importance of each component such as network weight. Weight values or weight gradients in training are commonly used as the importance metric. Small weights are pruned and large weights are kept. However, small weights are possible to be connected with significant weights which have impact to DNN outputs. DNN accuracy can be degraded significantly after the pruning process. This paper proposes a roulette wheel-like pruning algorithm, in order to simplify a trained DNN while keeping the DNN accuracy. The proposed algorithm generates a branch of pruned DNNs which are generated by a roulette wheel operator. Similar to the roulette wheel selection in genetic algorithms, small weights are more likely to be pruned but they can be kept; large weights are more likely to be kept but they can be pruned. The slimmest DNN with the best accuracy is selected from the branch. The performance of the proposed pruning algorithm is evaluated by two deterministic datasets and four non-deterministic datasets. Experimental results show that the proposed pruning algorithm generates simpler DNNs while DNN accuracy can be kept, compared to several existing pruning approaches.

Keywords Deep neural network · Network simplification · Network compression · Heuristic method

1 Introduction

Deep neural networks (DNNs) have been found in numerous pattern recognition applications since they have superior performance in many recognition tasks, such as computer vision, image processing, natural language processing [1]. However, DNNs, especially those high-performing ones with many layers and large width, have a large number DNN parameters such as network connections, channels, weights and biases. They require large memory storage; consequently they are unsuitable for resource-limited applications such as mobile or embedded devices. To achieve a higher performance, deeper and wider DNNs are usually considered; hence more memory and computational resources are required. This is a main obstacle to implement a high-performance DNN on mobile or embedded devices.

To relieve the cumbersome DNN memories, pruning approaches is commonly used to sparse insignificant

✉ Kit Yan Chan
kit.chan@curtin.edu.au

Ka Fai Cedric Yiu
macyiu@polyu.edu.hk

Shan Guo
s.guo@polyu.edu.hk

Huimin Jiang
hmjiang@must.edu.mo

[1] School of Electrical Engineering, Computing and Mathematics Sciences, Curtin University, Bentley, Australia

[2] Department of Applied Maths, The Hong Kong Polytechnic University, Hong Kong, China

[3] School of Business, Macau University of Science and Technology, Macao, China

Published online: 02 May 2024

weights to be zero [2–5], or remove insignificant kernels or channels [6–10]. After the cumbersome components of DNN is pruned, the DNN is more feasible for hardware implementations [11–13]. Gradient-based or magnitude-based pruning approaches are commonly used. For gradient-based pruning approaches, LeCun et al. [14] was the first to prune insignificant weights between network links based on the second derivatives to the loss function; Hessian matrix is used to compute the second derivatives which are computationally demanding. Hassibi et al. [15] simplified the second derivative computations by ignoring the diagonal assumption since Hessian matrices are usually non-diagonal. However, computational complexities of second derivatives are demanding which exponentially increase with network sizes. The approaches only suit small DNNs but not the modern DNNs which have more than billions of weights. Magnitude-based pruning approaches do not have this limitation of demanding computations. Magnitude-based pruning methods simply assume that large weights are more significant than smaller weights [16, 17]. Magnitude-based pruning methods prune all zero weights and smaller weights below an absolute threshold value; those pruned weights can be in network links, kernels or channels. However, DNN accuracies are usually degraded by pruning.

More recent pruning methods fine-tune or retrain pruned DNNs in order to improve accuracies after the DNNs are pruned [18]. Han et al. [19] proposed a pruning approach which first removes network connections uncorrelated to network outputs and further prunes small weights. After the pruning process, the DNN is retrained to improve accuracy. A similar approach has been proposed by Luo et al. [20], where a three-stage approach is proposed to develop a compressed DNN with high accuracy. The approach first dropouts small weights during the training; a less cumbersome DNN is trained. The processes of pruning, fine-tuning and retraining are performed simultaneously in order to further slim the DNN but keep the DNN accuracy. However, there are still lack of approaches to determine when to prune and when to terminate the pruning. Frankle and Carbin [21] have proposed a lottery ticket hypothesis which determines when to terminate the pruning and when to restart the fine-tuning. The winning tickets help the DNN being pruning faster and learning more accurately than the original DNNs.

The aforementioned pruning strategies assume weight importance can be quantified by weight values. Small weights are pruned and large weights are kept. They do not consider the connections between small weights and large weights. Since small weights are possible to link with significant weights which have impact to DNN predictions, connections with significant weights are broken. DNN accuracy can be degraded significantly after smaller weights are pruned. In the paper, a branching tree pruning algorithm is proposed based on a roulette wheel operator. The branching tree pruning algorithm starts with a trained DNN. The trained DNN is pruned into a branch of pruned DNNs using a roulette wheel operator; the pruned DNNs are not identical and are retrained. The best retrained DNN is selected to generate another branch of pruned DNNs and they are retrained to generate another branch. To select which DNN weights are pruned, the roulette wheel operator is proposed. The roulette wheel is used to determine which weights are more likely to prune based on the weight values. Similar to roulette wheel selection in genetic algorithms [22], small weights are more likely to be pruned but are possible to be kept; large weights are likely to be kept but are possible to be pruned. The pruning process repeats iteratively until the accuracy of the pruned DNN is poorer than the original un-pruned DNN. The approach attempts to seek for a slimmer DNN and keep the DNN accuracy, unlike the aforementioned pruning methods which prune weights with small values or weights with small gradients when training. The proposed branching tree pruning algorithm also overcomes the limitations of the existing pruning-retraining approaches which terminate the pruning process prematurely [21, 23].

The performance of proposed approaches are evaluated with two types of data, deterministic data (MNIST dataset[1] and CIFAR-10 dataset[2]) and non-deterministic (four datasets from TID [24]). The labels of the deterministic data are the same when the same inputs are given. The deterministic data evaluates whether the DNN accuracy can be kept for invariant labels. The labels of the non-deterministic data are subjective and are different when the same inputs are given. The non-deterministic data evaluates whether the performance of the pruned DNN is robust to varying labels. Experimental results show that the proposed approach outperforms the several commonly used pruning approaches [21, 23] for both deterministic data and non-deterministic data.

2 Pruning of deep neural networks

The pruning rate, Θ, indicates how much the DNN is simplified. Θ is quantified by the pruning rate function $\boldsymbol{\Theta}(W)$ in (1), where W is the weight set of the pruned DNN:

[1] http://yann.lecun.com/exdb/mnist/.

[2] https://www.cs.toronto.edu/~kriz/cifar.html.

$$\boldsymbol{\Theta}(W) = \left(1 - \frac{\sum_{w \in W} sgn(|w|)}{\sum_{w \in W^O} sgn(|w|)}\right), \quad (1)$$

$$\text{with } sgn(|w|) = \begin{cases} 1 & \text{if } |w| > 0 \\ 0 & \text{if } |w| = 0 \end{cases}$$

In (1), W^O is the weight set of the original cumbersome DNN. w is the weight in the weight set. For the fraction in (1), the nominator is the number of non-zero weights of the pruned DNN; the denominator is the number of non-zero weights of the original DNN. This fraction indicates whether the size of W^O is close to that of W. When the fraction is close to one, the sizes of both W^O and W are close. When the fraction is small, the number of non-zero weights in W is small and many weights are pruned from the original DNN. Since $\boldsymbol{\Theta}(W)$ is the subtraction of the fraction from 1, $\boldsymbol{\Theta}(W)$ is large when more weights are pruned.

In order to simplify the DNN and keep the DNN accuracy, the pruning problem is formulated by the objective function (2) with the constraint (3):

$$\max_{W} \boldsymbol{\Theta}(W) \quad (2)$$

$$\text{subject to} \sum_{i=1}^{N} |f(x_i|W) - y_i| \leq \sum_{i=1}^{N} |f(x_i|W^O) - y_i| \quad (3)$$

where (2) maximizes $\boldsymbol{\Theta}(W)$ by optimizing W. (3) ensures that the accuracy of the pruned DNN is not less than that of the original DNN. N is the data size. Given that x_i is the input of the DNNs, y_i is the actual label of the data; $f(x_i|W)$ and $f(x_i|W^O)$ are the predictions of the pruned DNN and the original DNN, respectively. The left and right sides of (3) indicate the sum absolute errors between the actual labels and predictions; they indicate the errors obtained by the pruned DNN and the original DNN, respectively. After (2) with (3) is solved, the pruned DNN is simpler than the original cumbersome DNN but the accuracy can be kept.

To solve (2), the commonly used pruning algorithms rank the weights from small to large values [2, 6, 25, 26]. Weights with smaller values are pruned; weights with larger values are kept. Alternatively some pruning algorithms rank the weights which have small gradients when training with the backpropagation [14, 15, 27–29]. They prune the weights with small training gradients. The remaining weights, which are not pruned, are retrained with respect to the loss function. In those commonly used algorithms, weights with small values or small training gradients are considered to have small impacts to the predictions; those insignificant weights can be ignored and can be pruned. However, DNNs have large numbers of nodes and connections. A weight with small value is connected with many nodes and some nodes can be linked with weights which have significant impact to the predictions. When small weights connecting with significant nodes are pruned, predictions of DNN can change significantly. The accuracy of the pruned DNN can be declined. Alternatively, some approaches reinitialize the remaining weights randomly after the pruning and keep the pruned weights to be zeros [18–20]. Those approaches retrain the remaining weights of DNN again with respect to the loss function. However, experimental results show that those randomly pruning approaches only achieve poor results [21].

To overcome the common limitations of the state-of-art algorithms, a branching tree pruning algorithm is proposed in Sect. 4. The branching tree pruning algorithm is an iterative method. It first generates several pruned DNNs based on a trained DNN. It again prunes the weights and retrains the remaining weights of those pruned DNNs. Each retrained DNN generates several pruned DNNs with different pruning sets of which a proposed roulette pruning operator in Sect. 1 is used. The proposed roulette prune operator prunes the small weights more frequently and the large weights are more likely to be kept. The algorithm checks which DNNs have more the higher accuracy; the DNN with the highest accuracy is further pruned with several pruning sets. The process continues, until the DNN accuracy is declined compared to the original DNN. Unlike the commonly used algorithms which prune the smallest weights and weights with small gradients when training, the proposed algorithm uses a roulette prune operator in order to prune some weights where the small weights are more likely to be pruned and the large weights are more likely to be kept. The remaining weights are retrained. The process continues until the DNN accuracy declines. Unlikely the commonly used methods which only target for the sub-optimal pruning sets, the proposed algorithm is similar to the roulette wheel selection in genetic algorithms which attempts to search for the global optimum [22]. The proposed algorithm is more likely to search a set of weights which is redundant and is worth to be pruned. The algorithm also attempts to keep the DNN accuracy.

3 Branching tree pruning algorithm

This section presents the proposed branching tree pruning algorithm and discusses how it seeks and prunes the redundant weights while keeping the accuracy of the original DNN which has not been pruned. The main component of the proposed algorithm namely roulette prune operator is proposed in Sect. 3.1. The algorithmic flow of the proposed branching tree pruning algorithm is presented in Sect. 3.2.

3.1 Roulette prune operator

The roulette prune operator is illustrated in Algorithm 1. The proposed operator uses the roulette wheel approach to select redundant weights to be pruned. The selection approach is similar to roulette wheels in the casino. Each interval of the roulette wheel is assigned as each of the possible selections of which weights are to be pruned. Weights with large value are assigned with small intervals; they have smaller chances to be selected. Weights with small values are assigned with large intervals; they have larger chances to be selected. Large weights are likely to be kept but they still have chances to be pruned. Small weights are more likely to be pruned but they still have chances to be kept. Unlike the commonly used approach which only prunes small weights and generates DNNs with only large weights, the proposed operator attempts to explore a better pruning solution with a good combination of small and large weights.

In the roulette prune operator illustrated in Algorithm 1, the operator input is a DNN weight set P with weights, $p_i \in P$, and a pruning rate Θ, which is the required number of weights to be pruned. Θ is set by the proposed branching tree pruning algorithm in Sect. 3.2. The operator output is the pruned weight set namely W, where the pruned weights, $w_i = 0 \in W$, are set to be zero. Steps 1 and 2 initialize the empty roulette wheel interval for each weight. In Steps 3 to 5, the roulette prune operator computes the roulette interval for each weight based on the inverted weight value, where small weights occupy large intervals and large weights occupy small intervals. Steps 6 and 7 generate the roulette wheel interval for each weight. The selection of weights is similar to the roulette wheel game in the casino. When the weight is large, its inverted value is small and its roulette wheel interval is small; the chance of being pruned is small. For a small weight, its inverted value is large and its roulette wheel interval is large. The chance of being pruned is large.

The random selection in the while-loop simulates how the roulette wheel is rotated and how a ball falls in an interval. A random number is generated by *rand*(). When the random number is between the range of the jth roulette interval, Step 9 prunes the jth weight by setting p_j to be zero. The number of pruned weights is counted by Step 10. The roulette selection is repeated until the pre-defined number of weights is pruned. In Step 11, a pruned weight set namely W is initialized by using the remaining weights which have not been pruned, while some weights, $w_j \in W$, are set as zeros as $p_j = w_j = 0$. The backpropagation algorithm is used to tune the W with respect to a loss function and the pruned weights with $w_j = 0$ are still kept as zeros.

In the proposed operator, weights with a small value are likely to be pruned but they still have chances to be kept. Hence, probabilities of keeping small weights are not zero. Unlike the commonly used methods, they prune a fixed percentage of the smallest weights and smallest weights have to be pruned. In the proposed method, smallest weights still have chances to be survived. This is an advantage for the whole DNN. Since there is a chance that small weights are possible to be kept, useful information from small weights is possible to pass to other nodes to perform predictions. The proposed operator is incorporated with the branching tree pruning algorithm in Sect. 3.2, in order to seek for a better pruning configuration.

Algorithm 1 ROULETTE-PRUNE$(P, \Theta) \longrightarrow W$

Input: P, the weight set where the weight $p_i \in P$ with $i = 1, 2, ..., |P|$

Θ, the pruning rate

Output: W, the pruned weight set with some weights $w_i = 0, w_j \neq 0 \in W$

▷ $|P|$ is the total number of weights

▷ The numbers of zero and nonzero weights are $\Theta \times |P|$ and $(1 - \Theta) \times |P|$ respectively

Step 1: Initialize the position of the roulette wheel for the i^{th} weight, p_i, for **Step 6**:

$Tem\_total_i \leftarrow 0$

Step 2: Initialize the roulette wheel intervals for all weights in **Step 7**:

$Prune\_prob \leftarrow [.]$

Step 3: Convert the original weight, p_i, from a small value to a large value:

$Ind\_w_i \leftarrow (\max_{\forall j} p_j - p_i)$

Step 4: Sum all the converted weights, $Total\_w \leftarrow \sum_{\forall i} Ind\_w_i$.

Step 5: Compute the length of each roulette wheel interval, $Norm\_w_i$, for each converted weight, Ind_w_i:

$Norm\_w_i \leftarrow \frac{Ind\_w_i}{Total\_w}$

for $i \leftarrow 1$ to $|P|$ **do**

Step 6: Compute the position of the i^{th} weight, p_i, in the roulette wheel interval:

$Tem\_total_{i+1} \leftarrow Tem\_total_i + Norm\_w_i$

Step 7: Generate the roulette wheel interval for the i^{th} weight, p_i:

$Prune\_prob[i] \leftarrow [Tem\_total_i, Tem\_total_{i+1}]$

end for

Step 8: Initialize a counter integer, $Count \leftarrow 0$.

while $(Count < |P| \times \Theta)$ **do**

if $(Tem\_total_i > rand() \geq Tem\_total_i)$ **then**

▷ $rand()$ is a random number ranged from 0 to 1

Step 9: Prune the j^{th} weight by setting $p_j \leftarrow 0$.

Step 10: Set $Count \leftarrow Count + 1$.

end if

end while

Step 11: Use P as the initial weight set with some $p_j = 0 \in P$, and use the backpropagation with the loss function (4) to tune P and generate the pruned weight set W with some $p_j = w_j = 0 \in W$.

Return W

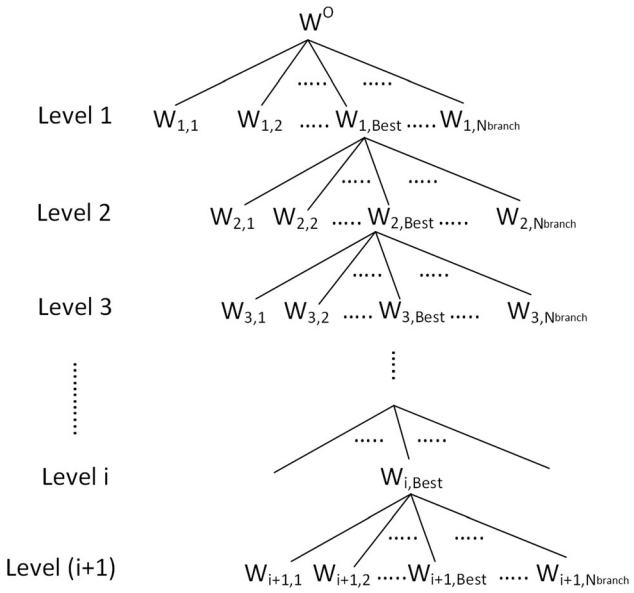


Fig. 1 Branching tree pruning algorithm

3.2 Algorithmic flow

The algorithmic flow of the proposed branching tree pruning algorithm is illustrated in Fig. 1. The algorithm is used to generate a pruned weight set, W_{final}, which has a smaller number of weights compared to the original weight set W^O. The original DNN accuracy can be kept. The number of weights of W_{final} is Θ_{final} time smaller than that of W^O, where Θ_{final} is the final pruning rate which is small than one. In Level 1, the proposed algorithm first reproduces N_{branch} pruned weight sets namely $W_{1,1}$, $W_{1,2}$,..., and $W_{1,N_{\text{branch}}}$ by ROULETTE-PRUNE in Sect. 3.1, where the pruned weight sets in level 1 are Θ time slimmer than W^O. Since ROULETTE-PRUNE is a non-deterministic algorithm, the pruned weight sets in level 1 are not identical. Those nonidentical weight sets attempt to explore different pruning configurations, in order to search for a pruning configuration which has a smaller number of weights and keeps the original DNN accuracy.

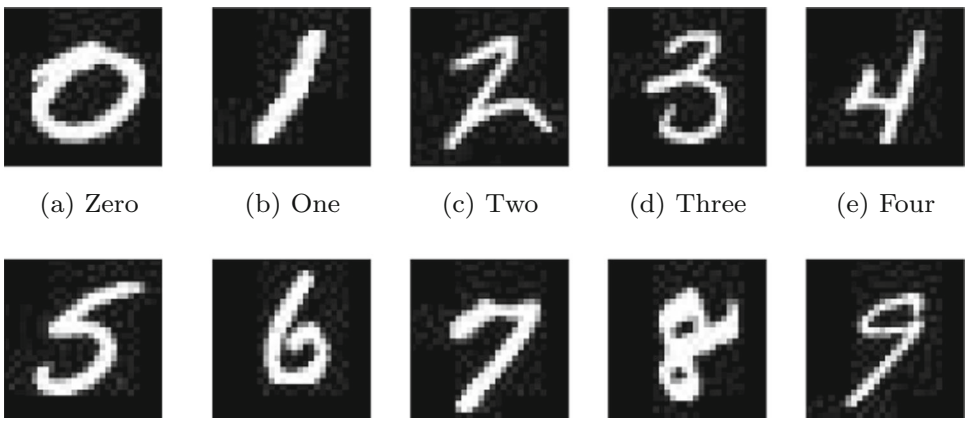


Fig. 2 Ten figures from MNIST database

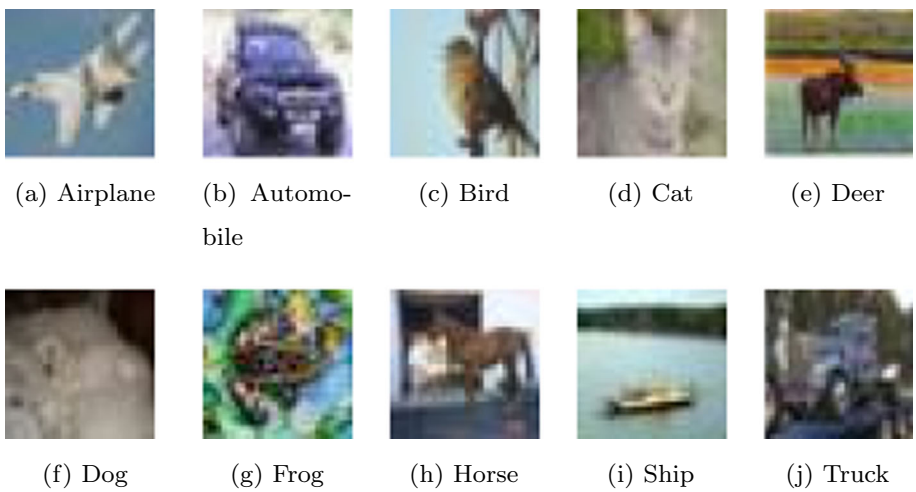


Fig. 3 Ten figures from CIFAR database

In Level 1, the best pruned weight set $W_{1,\text{Best}}$ is selected to explore Level 2. The N_{branch} pruned weight sets namely $W_{2,1}$, $W_{2,2}$,..., and $W_{2,N_{\text{branch}}}$ in Level 2 are explored and the best weight set namely $W_{2,\text{Best}}$ is selected to explore weight sets for Level 3. The process continues until no more weight set can be pruned while the prediction accuracy can be kept. In Level $(i+1)$, the best weight set is finally selected as the output of the proposed branching tree pruning algorithm. The best weight set is slimmer than W^O and the original accuracy can be kept, when the best weight set is used in the DNN. The proposed branching tree pruning algorithm is similar to the genetic algorithm with roulette wheel selection. The proposed algorithm attempts to overcome the limitations of the commonly used pruning approaches which keep exploring a single pruning solution and are not likely to explore a better pruning configuration.

Algorithm 2 BRANCH-TREE$\left(W^O, \Theta_{\text{req}}\right) \longrightarrow \left(W_{\text{final}}, \Theta_{\text{final}}\right)$

Input: W^O, the original weight set of the DNN

Θ_{req}, the minimum pruning rate which is the minimum pruning target

Output: W_{final}, the pruned weight set with some weights $w_m = 0 \in W_{final}$ and other weights $w_n \neq 0 \in W_{final}$

Θ_{final}, the final pruning rate

Step 1: Initialize two tree parameters, namely level i and branch j, as $i \leftarrow 0$ and $j \leftarrow 0$ respectively.

Step 2: Set the original weight set as the initial weight set: $W^O \rightarrow W_{i,j}$.

Step 3: Initialize the pruning rate as $\Theta \leftarrow 1$.

Step 4: Compute the loss of the original DNN with $W_{i,j} = W^O$, using $L_{i,j} \leftarrow F(W_{i,j})$ in (4).

do

for $k \leftarrow 1$ to N_{branch} **do**

▷ //The for-loop selects the best pruned weight set from the branches;

▷ //N_{branch} is the number of branches

Step 5: Initialize the first branch as the best branch, $Best \leftarrow 1$.

Step 6: Set the pruning rate and the counter as, $\Theta \leftarrow \Theta_{req} \times \Theta$ and $counter \leftarrow counter + 1$ respectively.

Step 7: Generate the pruned weight set of the k^{th} branch at level $(i+1)$ using ROULETTE-PRUNE$\left(W_{i,j}, \Theta\right) \longrightarrow W_{i+1,k}$.

if $\left(F(W_{i+1,k}) < F(W_{i+1,Best})\ AND\ (K > 1)\right)$ **then**

▷ //Steps 8 and 9 select the best pruned weights;

Step 8: Set the best pruned weight set as $W_{i+1,Best} \leftarrow W_{i+1,k}$.

Step 9: Set the best branch as, $Best \leftarrow k$ and level as $i \leftarrow i+1$.

end if

end for

while $\left(\exists L_{i,k} < L_{i-1,k}\right)$

Step 10: Set $W_{final} \leftarrow W_{i,Best}$ and $\Theta_{final} \leftarrow \Theta$

Return $\left(W_{final}, \Theta_{final}\right)$

The proposed branching tree pruning algorithm is detailed in Algorithm 2. W^O, is the input of the branching tree pruning algorithm. The outputs are W_{final} and Θ_{final}. Steps 1 to 3 and Step 5 initialize the algorithmic parameters and set the level number, i, and the branch, j, to be zero. They also set W^O, as the initial weight set, $W_{i,j}$ which is used as the initial searching point of the algorithm. Step 4 uses the loss function (4) to compute the loss value of the DNN with $W_{i,j} = W^O$.

$$L_{i,j} = F(W_{i,j}) = \sum_{k=1}^{N_D} \left(f(x_k|W_{i,j}) - y_k\right)^2 \tag{4}$$

where $L_{i,j}$ is the loss value with respect to the pruned weight $W_{i,j}$ which is at the jth branch of the ith level.

Step 5 initializes the best branch index, Best, as 1. Step 6 sets the pruning rate in the current tree level. Step 7 uses $ROULETTE - PRUNE$ to generate the pruned weight set, $W_{i+1,k}$, for the kth branch at level $(i+1)$, in order to explore a better pruning configuration. The if-condition is used to check whether the loss value of the pruned DNN with $W_{i+1,k}$ is better than the DNN with the best $W_{i+1,\text{Best}}$. If the condition is satisfactory, Step 8 replaces $W_{i+1,\text{Best}}$ with $W_{i+1,k}$ and the index, Best, is set as k, since the kth branch generates the best weight set. Steps 7 to 9 repeat continuously to explore a better pruning configuration, until a lower loss value cannot be explored for all branches. Step 10 sets the best weight set as the final weight set, W_{final}, and it also sets Θ as the final pruning rate, Θ_{final}. Finally, W_{final} and Θ_{final} are returned as the algorithmic outputs, where a smaller weight set W_{final} is used in the DNN and the accuracy of the original DNN can be kept.

4 Algorithmic performance evaluations

To evaluate the performance of the proposed branching tree pruning algorithm, both deterministic datasets and non-deterministic datasets are used and are described in Sect. 4.1. Then four commonly used pruning algorithms described in Sect. 4.2 are used to compare with performance of the proposed algorithm. The results obtained for the deterministic datasets and non-deterministic datasets are discussed in Sects. 4.3 and 4.4, respectively.

4.1 Benchmark datasets

The deterministic datasets in Sect. 4.1.1 are commonly used to evaluate the pruning performance, where the data labels are the same when the same data is given. They evaluate whether the DNN accuracy can be kept when the DNN is pruned. For the non-deterministic datasets in Sect. 4.1.2, the data labels are similar but not identical when the same data is given. They evaluate whether the performance of the pruned DNN is robust to the varying labels.

4.1.1 Deterministic datasets

For the deterministic datasets, the commonly used benchmark classification datasets namely MNIST[3] and CIFAR10[4] are used. Some figures for MNIST and CIFAR10 are shown in Figs. 2 and 3, respectively. These two datasets have been used to evaluate the performance of many pruning algorithms [23, 27, 30–35]. The MNIST database of handwritten digits has a training set of 60,000 gray images with the size of 28 × 28 pixels in 10 classes from zero to nine, and a test set of 10,000 examples. The CIFAR-10 dataset consists of 60 000 color images with the size of 32 × 32 pixels, and 6000 images per class. The classes consist of the items of airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. These two datasets are deterministic since the recognition outcomes are the same when the same images are given. For example, when an image with digit number one in MNIST is shown, the group true is one; when an image with the animal cat is shown, the group true is a cat. In the two deterministic datasets, each image has an exact ground truth.

4.1.2 Nondeterministic datasets

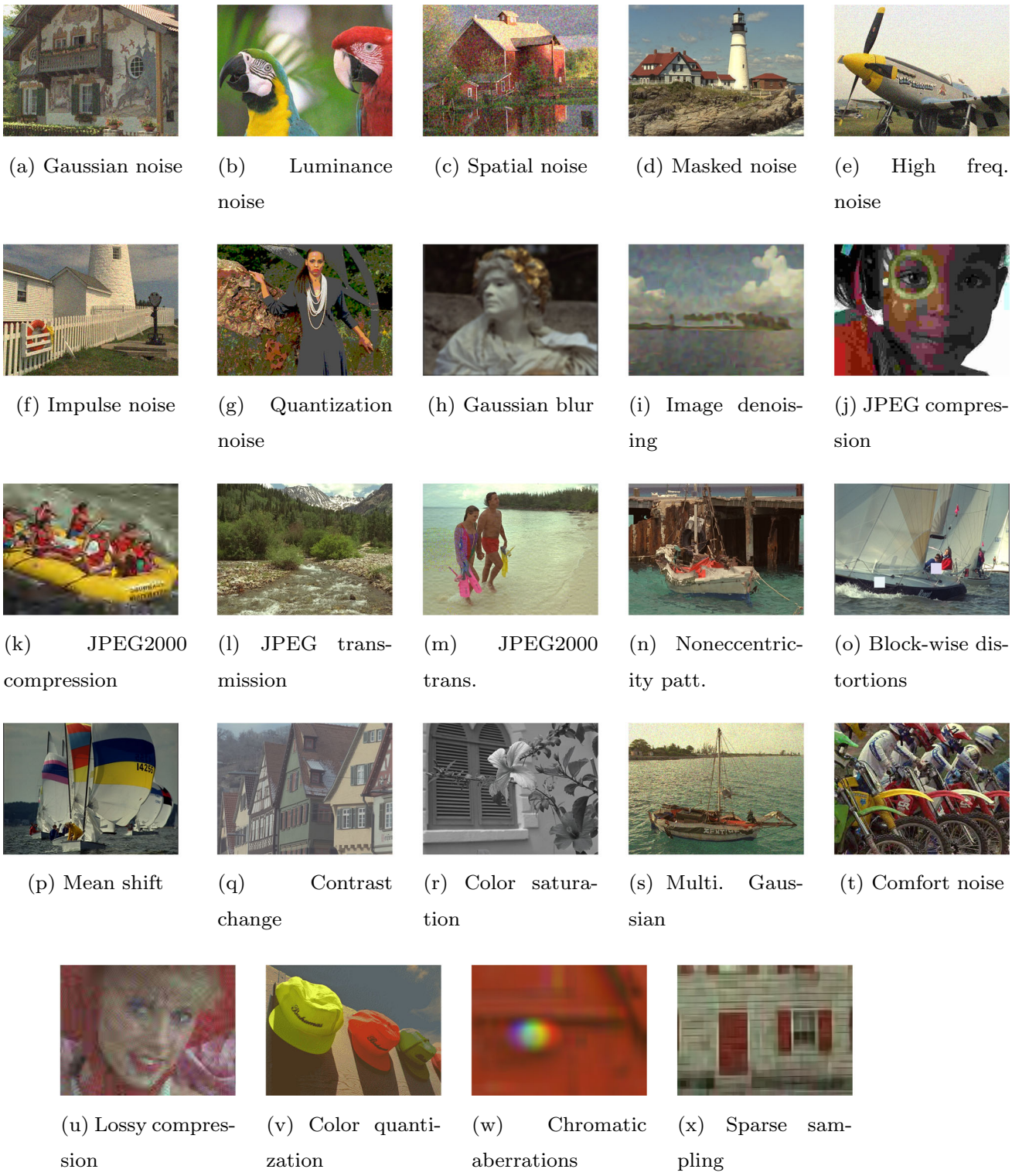
For the non-deterministic dataset, we use the TID image quality evaluation dataset to evaluate the performance of the pruning algorithms. Some of those images with different distortion types are shown in Fig. 4. For example, when the image shown in Fig. 4a distorted with Gaussian noise was shown to many persons, the scores evaluated by persons are different. These types of images evaluate whether the pruned DNN is robust for the predictions when the non-deterministic scores are used as the labels.

The TID database has 3000 distorted images of which each image was contaminated by a distortion on a reference image. There are 25 reference images and 24 distortion types with 5 levels. The 24 distortion types are caused by either camera operations, image transmissions, image compression/decompression, or conventional image processing. There are totally 3000 distorted images in the TID database. Some distortions are common in many applications; some are rare in real applications but they are particularly generated to evaluate the prediction performance [24].

In the TID database, the image qualities were scored from the range between 1 to 5 by the human observers

[3] http://yann.lecun.com/exdb/mnist/.

[4] https://www.cs.toronto.edu/~kriz/cifar.html.



(a) Gaussian noise (b) Luminance noise (c) Spatial noise (d) Masked noise (e) High freq. noise

(f) Impulse noise (g) Quantization noise (h) Gaussian blur (i) Image denoising (j) JPEG compression

(k) JPEG2000 compression (l) JPEG transmission (m) JPEG2000 trans. (n) Noneccentricity patt. (o) Block-wise distortions

(p) Mean shift (q) Contrast change (r) Color saturation (s) Multi. Gaussian (t) Comfort noise

(u) Lossy compression (v) Color quantization (w) Chromatic aberrations (x) Sparse sampling

Fig. 4 25 distortion types for TID2013 database

which are from 5 countries, Finland, France, Italy, Ukraine, and USA. We further ranked the image scores as integer values either 1, 2,..., or 5. To further increase the training samples, each distortion image was patched into 16 × 12 blocks with a resolution of 32 × 32 pixels. Therefore, 576 000 image blocks are used to test the performance of the pruning algorithms. Based on the TID database which is non-deterministic, we can evaluate whether the pruning

Table 1 Distortion types for TID2013 and its subsets

	Distortion type	Full	Noise	Actual	New
(a)	Gaussian noise	✓	✓	✓	×
(b)	Luminance noise	✓	✓	×	×
(c)	Spatial noise	✓	✓	✓	×
(d)	Masked noise	✓	✓	✓	×
(e)	High freq. noise	✓	✓	✓	×
(f)	Impulse noise	✓	✓	✓	×
(g)	Quantization noise	✓	✓	×	×
(h)	Gaussian blur	✓	✓	✓	×
(i)	Image denoising	✓	×	✓	×
(j)	JPEG compression	✓	×	✓	×
(k)	JPEG2000 compression	✓	×	✓	×
(l)	JPEG transmission	✓	×	✓	×
(m)	JPEG2000 trans.	✓	×	×	×
(n)	Noneccentricity patt	✓	×	×	×
(o)	Block-wise distortions	✓	×	×	×
(p)	Mean shift	✓	×	×	×
(q)	Contrast change	✓	×	×	×
(r)	Color saturation	✓	×	×	✓
(s)	Multi. Gaussian	✓	×	✓	✓
(t)	Comfort noise	✓	×	×	✓
(u)	Lossy compression	✓	×	✓	✓
(v)	Color quantization	✓	×	×	✓
(w)	Chromatic aberrations	✓	×	×	✓
(x)	Sparse sampling	✓	×	×	✓
Total number of images		3000	1000	1500	875
Total number of patches		576 000	192 000	288 000	168 000

✓: the corresponding noise type is included

×: the corresponding noise type is excluded

algorithms is sensitive to nonidentical scores when the same image block is presented.

Despite using the full TID dataset, three datasets namely Noise, Actual and New described in Table 1 are used to evaluate the performance of the pruning algorithms. These three datasets contain some noise types of the full TID datasets which are designed for particular image applications. The Noise, Actual and New datasets contain 192 000, 288 000 and 168 000 image blocks, respectively.

4.2 Algorithmic comparison

For the deterministic databases, MNIST and CIFAR10, the architecture of convolution neural networks has commonly been used to evaluate the pruning algorithms [23, 27, 30–35]. Hence, pruning of convolution neural networks is used to evaluate the pruning algorithm performance. In this research, the network architecture for evaluating the performance of recent pruning algorithms is used [23]. The DNN consists of two convolution layers, one pooling layer and one fully connected layer. The first convolution layer has 32 kernels with the size of 3×3. The second convolution layer has 64 kernels with the size of 3×3. The third layer is the max pooling layer and the final layer is the full connected layer with 128 hidden nodes. The total number of weights is 1.2 million.

For the non-deterministic database, TID, the convolution neural networks have been used to perform the classifications with reasonable results [36–38] but large numbers of parameters are required. The VGGnet architecture is used [39]. The VGGnet architecture consists of eight convolution layers with 3×3 cascaded convolution kernels and four maxpool layers [38]. The DNN was developed to process image blocks with the resolution of 32×32 pixels. To unify the image resolution of VGGnet input, images are resized by two convolution layers and a single maxpool layer. The image features of the DNN are inputted to two cascaded fully connected neural networks and a single pooling layer. The DNN consists of 5.2 million parameters [38].

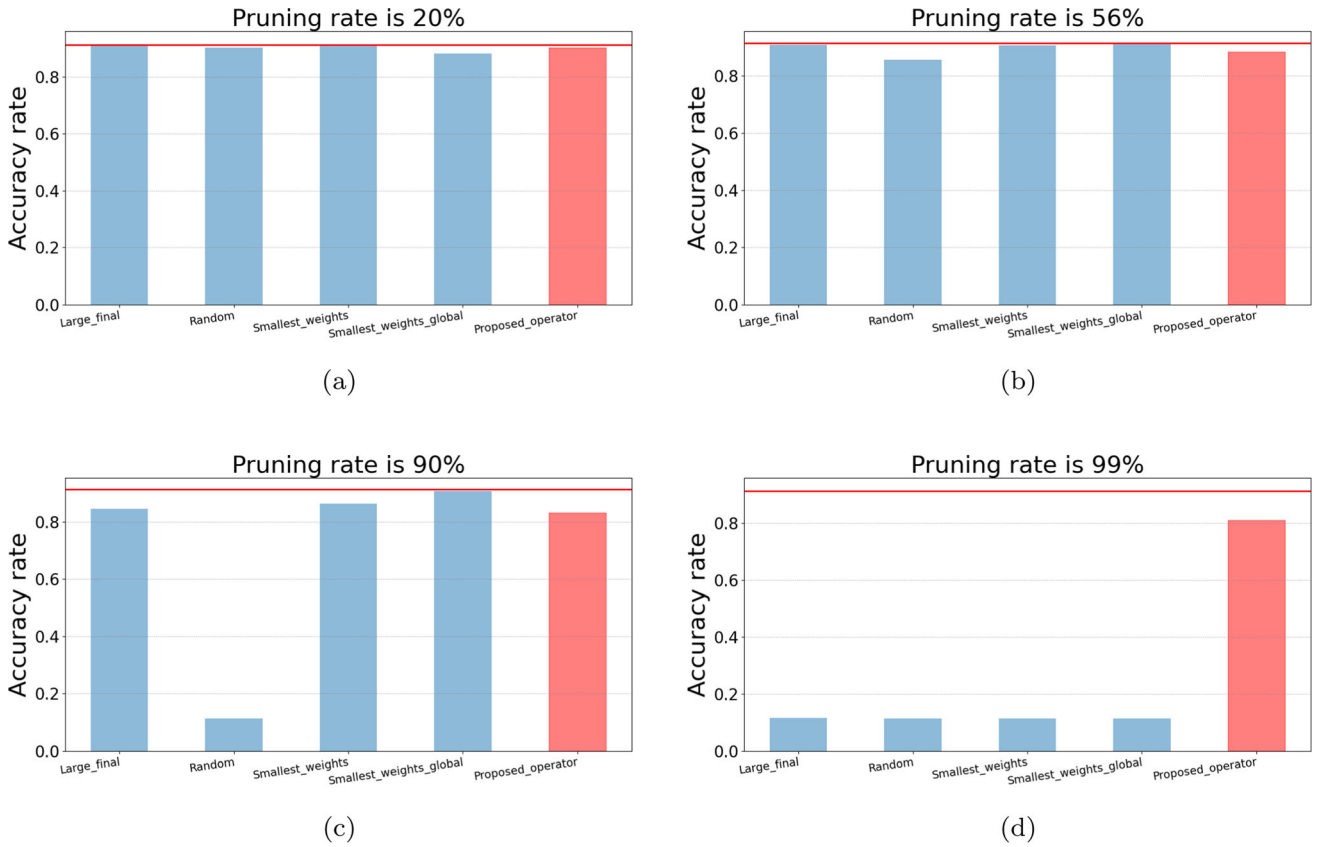


Fig. 5 Pruning of DNNs with initial weights for MINST data

The performance of the proposed pruning operator, ROULETTE-PRUNE, and the proposed algorithm, branching tree pruning algorithm, is compared with those obtained by the commonly used pruning operators [21] [23]:

- **Large final:** Weights are pruned randomly while weights ranked with larger values are kept from the fully trained DNN. When $c\%$ of weights are set to be pruned, $c\%$ of the larger weights are kept and $c\%$ of the remaining weights are selected randomly to be pruned.
- **Random:** The weights in the fully trained DNN are randomly selected to be pruned. When the pruning rate, $c\%$, is pre-defined, $c\%$ of the weights are selected randomly to be pruned.
- **Smallest weights**: Weights with the smallest absolute values in the fully trained DNN are pruned. $c\%$ of the smallest weights are pruned.
- **Smallest weights global**: Smallest weights across all layers are pruned in the fully trained DNN. For a DNN with N layers, $(\frac{c}{N})\%$ of the smallest weights in each layer are pruned when the overall pruning rate is set as $c\%$.

To evaluate the performance of the four commonly used pruning operators and the proposed ROULETTE-PRUNE in Sect. 3.1, pretrained DNNs are used. The number of epochs used to train the DNNs is 50. The comparisons were conducted by two commonly used approaches which were suggested by Frankle et la. [21] [23] to evaluate the pruning performance of DNNs.

- **Pruning with reinitialized weights**: The DNN was trained with a reasonable accuracy with respect to a dataset. Pruning is performed to sparse some non-zero weights into zeros. The DNN is reinitialized by regenerating the non-zero weights with random numbers and keeping the sparsed weights as zeros. The DNN is retrained by the reinitialized non-zero weights, while the sparsed weights are kept zeros. The DNN weights are pruned iteratively after training with several epochs until the DNN accuracy is smaller than the original DNN. The approach attempts to evaluate whether re-initializing values in unsparsed weights is more effective than keeping the weight values.
- **Pruning with trained weights**: Similar to the above approach, pruning with reinitialized weights, a DNN was trained with a reasonable accuracy with respect to a dataset. Pruning is performed to sparse some non-zero weights into zeros. Both the original trained weights and the sparsed weights are kept. The DNN is retrained

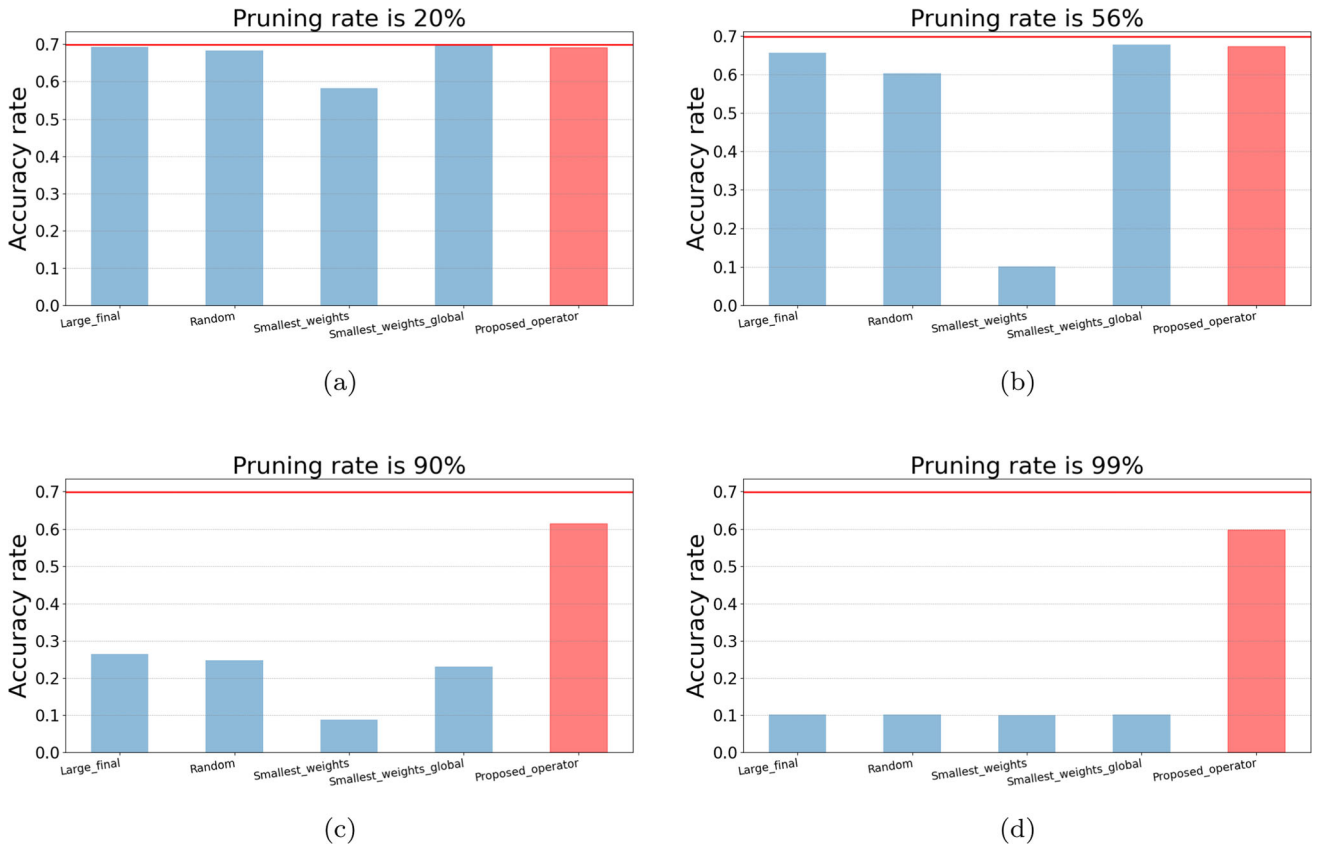


Fig. 6 Pruning of DNNs with initial weights for CIFAR data

based on the kept weights. The DNN weights are pruned and then retraining with pre-defined number of epochs iteratively, until the DNN accuracy is poorer than the original DNN. The approach attempts to evaluate whether using the original un-pruned weights to retrain the DNN is effective.

For the proposed branching tree pruning algorithm[5] in Sect. 3.2, the number of branches in each level is set as 5. The proposed algorithm is terminated when no improvement is obtained in a level, $(i+1)$. Hence, the total number of searches of the proposed algorithm is $5 \times (i+1)$. To achieve a fair comparison, the other tested operators are performed with $5 \times (i+1)$ searches. The best pruned DNN is used for the comparisons.

4.3 Performance evaluations with deterministic data

This subsection evaluates the numerical results obtained by the proposed ROULETTE-PRUNE operator in Sect. 3.1 and the proposed branching tree pruning algorithm in Sect. 3.2. The performance was evaluated by the two deterministic datasets, MINST and CIFAR-10. The results obtained by proposed methods are compared with the four tested operators, large final, random, smallest weights, and smallest weights global. The pruning results with reinitialized weights and trained weights are shown in Sects. 4.3.1 and 4.3.2, respectively.

4.3.1 Pruning results with reinitialized weights

The results of pruning with reinitialized weights are shown in Figs. 5 and 6 for the MINST and CIFAR data, respectively. Figure 5 shows the pruning performance for the MINST dataset where the un-pruned weights are randomly reinitialized before the DNN is retrained. In the figures, the red lines show the accuracy rate of the trained DNN which is not pruned; the accuracy rate is close to 0.95. Figure 5a, b, and c show the accuracy rates for the DNNs which are pruned with 20%, 56%, and 90%, respectively. The figures show that the accuracy rates for the five tested methods are close to the original accuracy rate of which the trained DNN is not pruned. However, the accuracy rate achieved by the random operator is less than 20% which is significantly poorer than the other four tested when the pruning rate is 90%. Figure 5d shows the results when the pruning rate is 99%. The results show that the proposed

[5] The proposed branching tree pruning algorithm is coded by python tensorflow. The source code is available by requesting the first author.

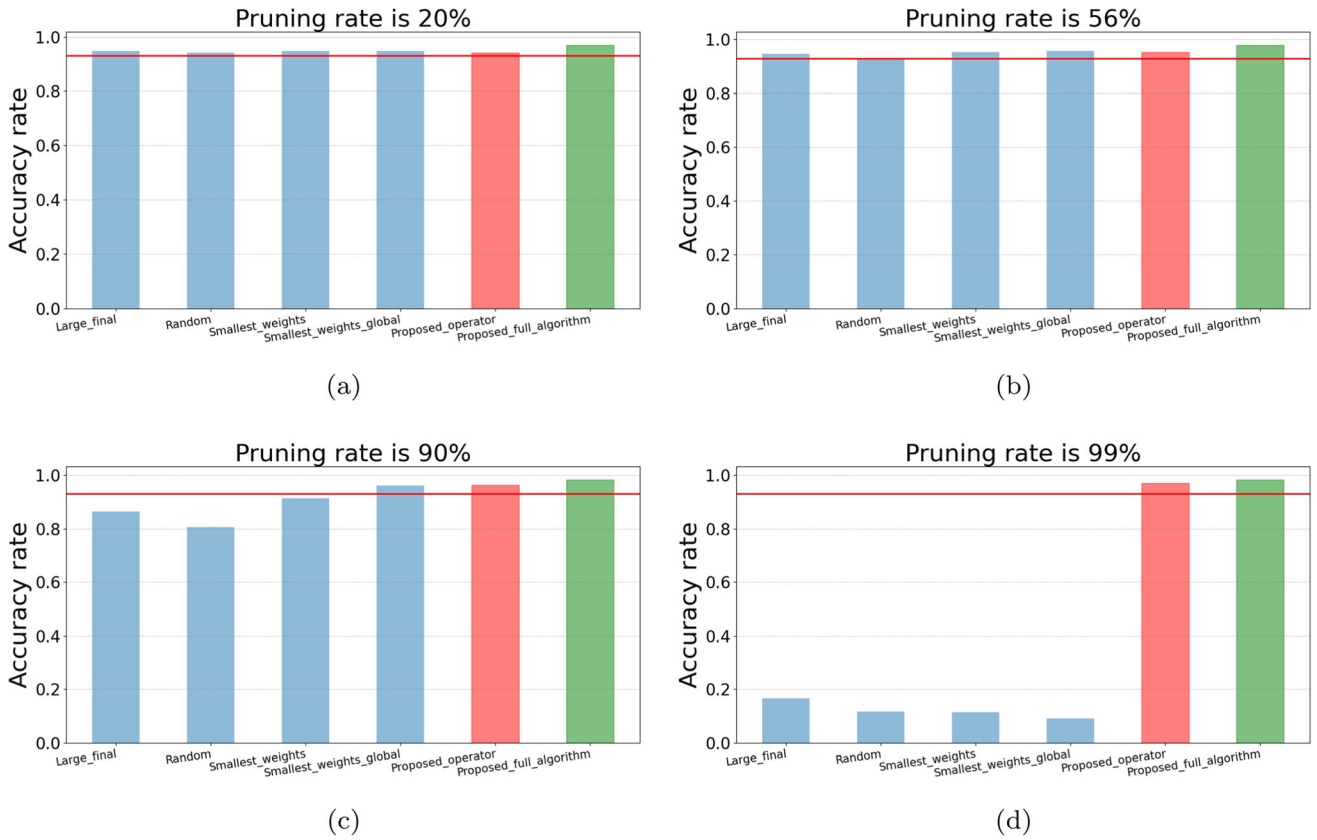


Fig. 7 Pruning of DNNs with trained weights for MINST data

ROULETTE PRUNE operator achieves better accuracy rate which is close to 80% compared to other tested operators. These results indicate that the proposed operator achieves a better result when the pruning rate is high.

Figure 6 shows the pruning performance of DNNs which are trained with CIFAR data. The red lines show the accuracy rates of the trained DNN which is not pruned; the accuracy rate is close to 70%. Figure 6a shows the results of which the DNN is pruned by 20% of weights. The accuracy rates for the five tested methods are close to the original accuracy rate, except that the accuracy rate achieved by the smallest weights operator is slightly poor than the others. Figure 6b shows that the results of which the DNN is pruned by 55% of weights. The accuracy rates obtained by the four operators, large final, random, smallest weights global and the proposed operator are close to the original accurate rate, while significantly poorer accuracy is obtained by the smallest weights operator. Figure 6c shows the results of which the DNN is pruned by 90% of weights. It shows that significantly poorer results are achieved by the tested operators while the proposed operator achieves a better accuracy rate which is close to the original accuracy rate. Similar results are obtained for the DNN pruned with 99% of weights. Figure 6d shows that the proposed operator achieves a significantly better accuracy rate compared to the four tested operators. Therefore, these results further indicate that the proposed operator achieves better results when the pruning rates are high.

4.3.2 Pruning results with trained weights

Figure 7 shows the pruning performance of the DNNs trained with the MNIST data, where the trained weights are used as the initial weights for retraining after some weights are pruned. The red lines show the accuracy rate of the original trained DNN which is close to 0.95. Figure 5a, b, c and d show the accuracy rates for the DNNs which are pruned with 20%, 56%, 90% and 99%, respectively. The DNNs are retrained with the trained weights while the pruned weights are kept to be zero. Figure 7a, b, and c shows that the accuracy rates for the six tested pruning methods are close to the original accuracy rates of which the original DNN is not pruned. Figure 5d shows that the performance of the four tested methods decreases significantly when the pruning rate is 99%, compared to the proposed operator and the proposed algorithm. These results indicate that the proposed ROULETTE-PRUNE operator and the proposed branching tree pruning

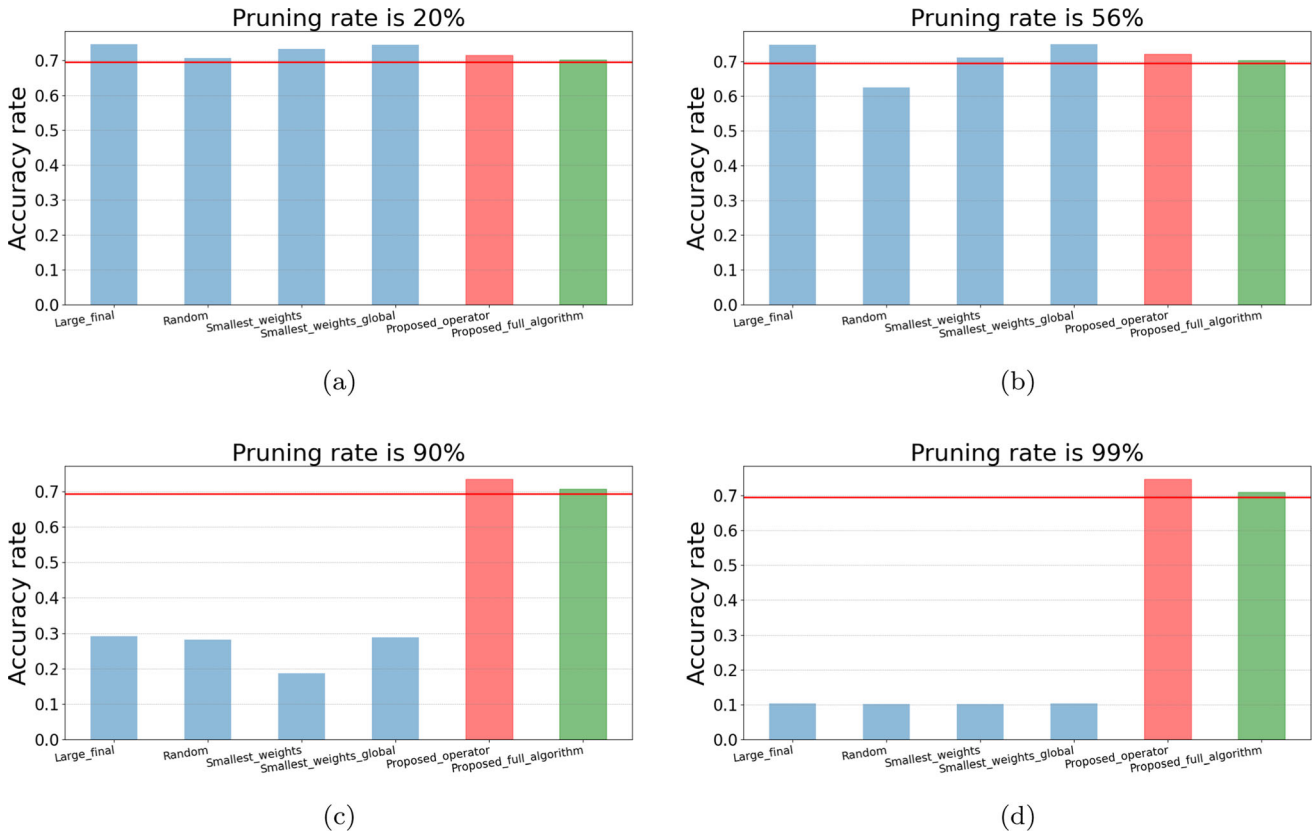


Fig. 8 Pruning of DNNs with trained weights for CIFAR data

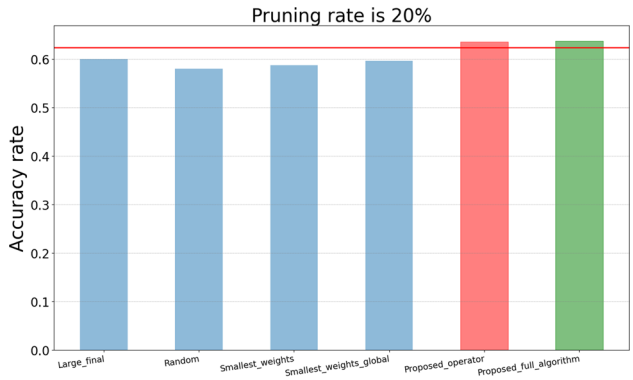


Fig. 9 Accuracies of pruned DNNs with trained weights for TID data

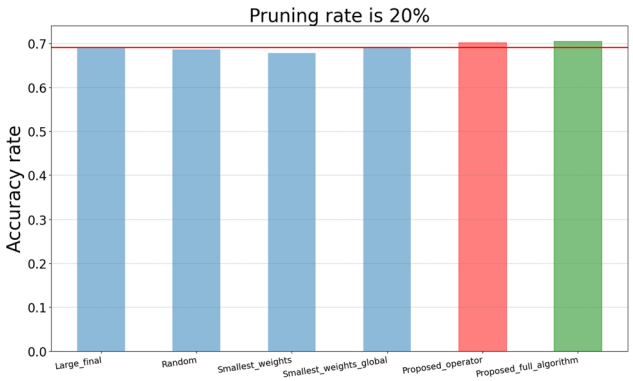


Fig. 10 Accuracies of pruned DNNs with trained weights for Noise data

algorithm outperforms the tested methods, when the pruning rates are high.

Figure 8 shows the pruning performance of DNNs trained with CIFAR data, where the trained weights are used as the initial weights for retraining. Figure 8a and b show the results when 20% and 56% of weights are pruned, respectively. The two figures show that the accuracy rates obtained by the four tested methods and the two proposed methods are close to the original accuracy rates. Figure 8c and d show the results when the DNN is pruned by 90% and 99% of weights, respectively. These two figures show that the proposed ROULETTE-PRUNE operator and the proposed branching tree pruning algorithm achieve better results which are close to the original accuracy rate, while the performance of the other four tested methods decrease significantly. These results further indicate that the proposed operator and the proposed method are able to keep the original accuracy rate when more weights are pruned.

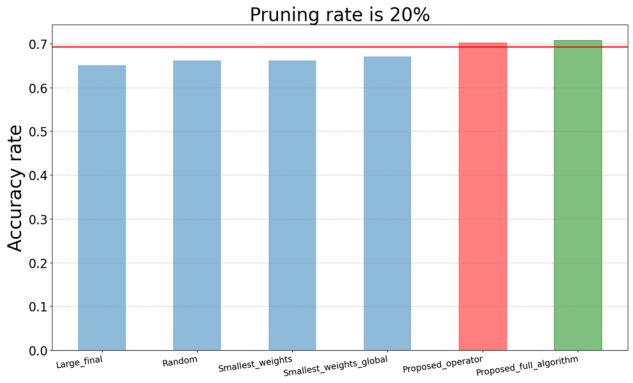


Fig. 11 Accuracies of pruned DNNs with trained weights for Actual data

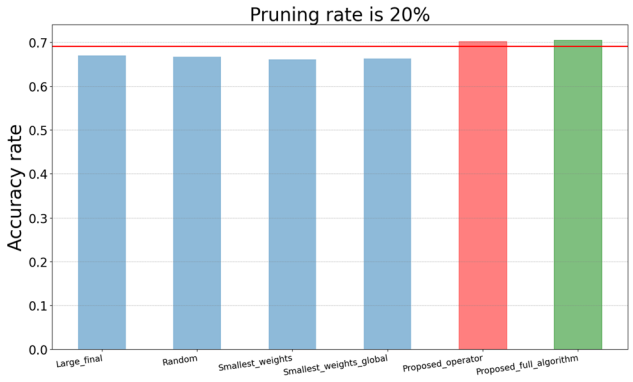


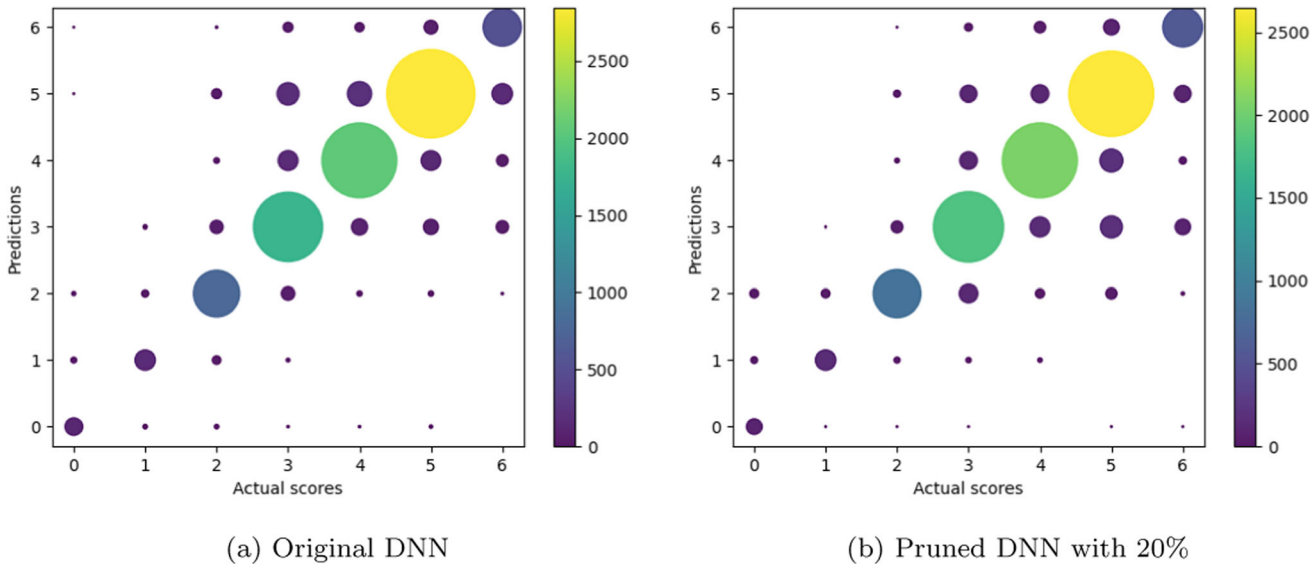
Fig. 12 Accuracies of pruned DNNs with trained weights for New data

4.4 Performance evaluations with non-deterministic data

The performance of the proposed methods are evaluated by the non-deterministic data, the four image quality evaluations datasets, namely TID, Noise, Actual and New [24] in Table 1. For these four datasets, 80% of image patches are used for training and the rest of the 20% are used to test the pruned DNN performance.

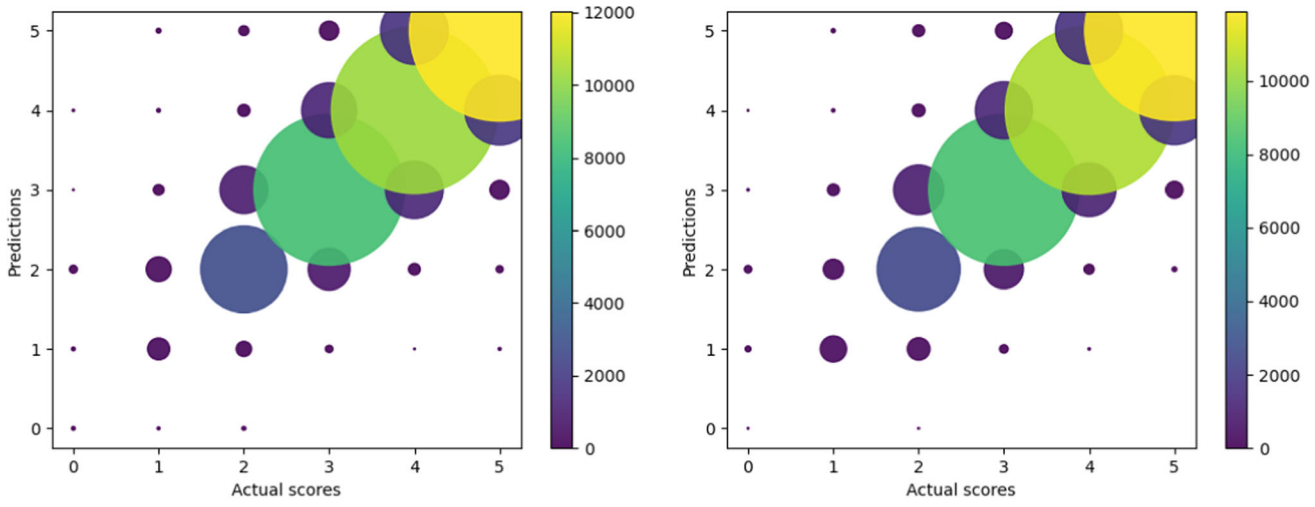
Figure 9 shows the pruning performance of the DNNs trained with the TID data, where the trained weights are used as the initial weights to retrain the DNNs after the original DNN is pruned by 20%. The accuracy rates are illustrated. The figure shows that the accuracy rate of the original DNN is 62.33% which are better than the pruned DNNs generated by the four tested methods. Therefore, the accuracies are poorer after the DNN is pruned when the tested methods are used. However, better accuracies can be achieved by the two proposed methods, ROULETTE-PRUNE operator and branching tree pruning algorithm, which obtain higher accuracies compared with the original DNN. Although 20% of weights are pruned by the proposed methods, better accuracies can be achieved compared to the original DNN. The red line shows the accuracy rate of the original DNN. It clearly shows that the proposed methods achieve better accuracies compared to the other four tested methods.

Figure 10 shows the pruning performance of DNNs trained by the Noise data. It shows that the accuracy rate of the original DNN without pruning is higher than 75 %.



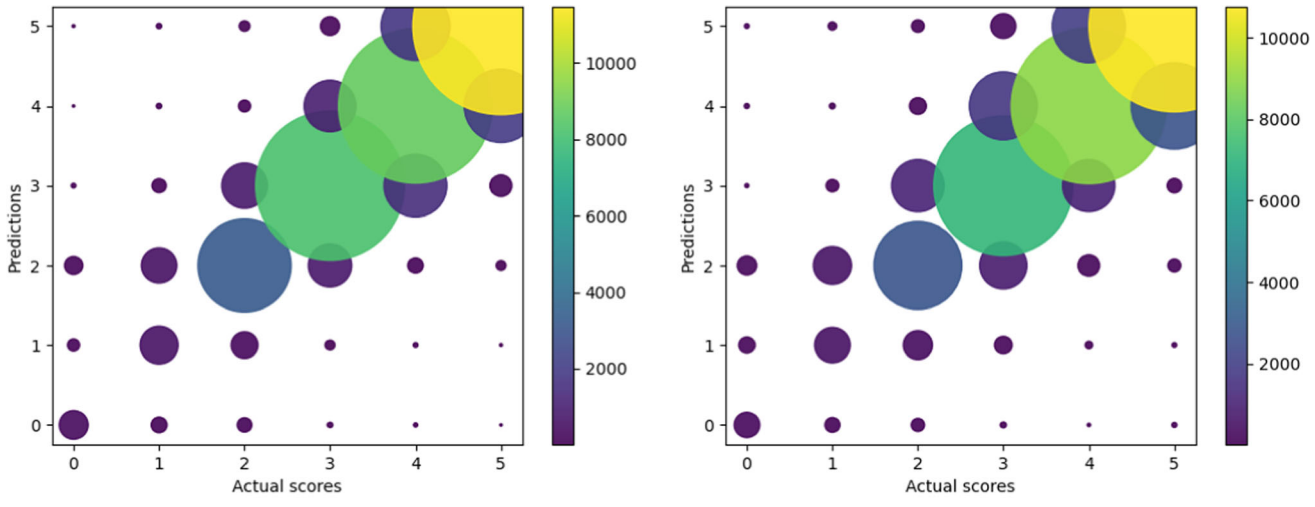
(a) Original DNN (b) Pruned DNN with 20%

Fig. 13 Predictions for TID dataset



(a) Original DNN (b) Pruned DNN with 20%

Fig. 14 Predictions for Noise dataset

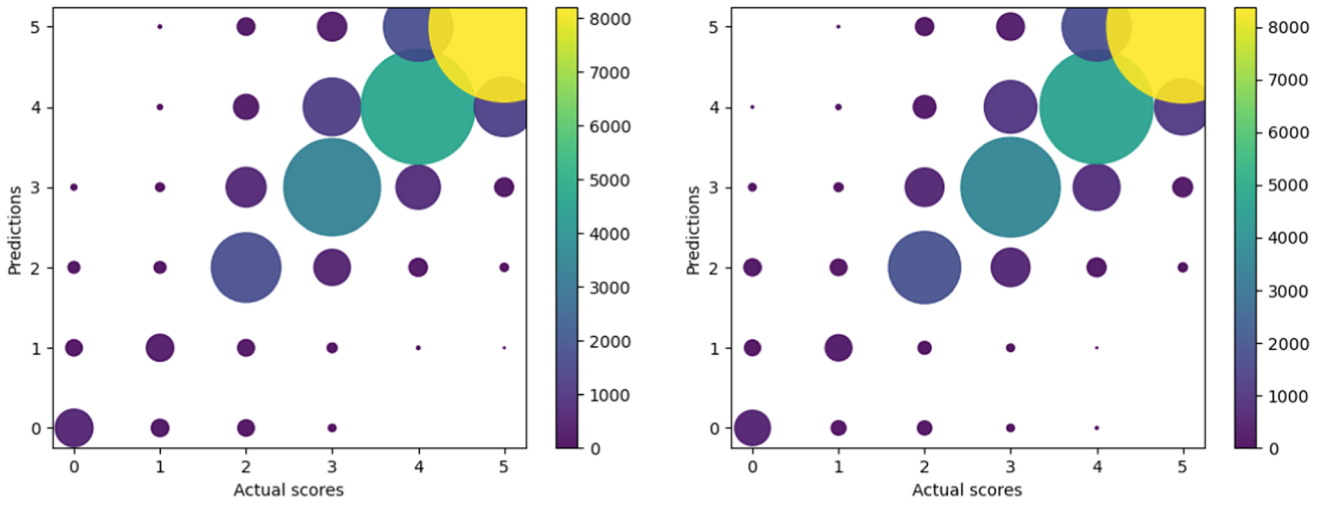


(a) Original DNN (b) Pruned DNN with 20%

Fig. 15 Predictions for Actual dataset

Poorer accuracy rates are obtained by the four tested methods, large final, random, smallest weights and smallest weights global, compared to the original DNN. Therefore, the accuracies of the tested methods are poorer after the DNN is pruned. However, better accuracies can be achieved by the proposed ROULETTE-PRUNE operator and proposed branching tree pruning algorithm which obtain higher accuracies compared with the original DNNs, after 20% of weights are pruned. Also the accuracy achieved by the proposed methods is better than those achieved by the other four tested methods. Similar results can be obtained for the Noise, Actual and New datasets which are shown in Figs. 10, 11 and 12, respectively. For the three datasets, the accuracy rates of the original DNNs without pruning are about 69 %. Poorer accuracy rates are obtained by the four tested methods while better accuracy rates are achieved by the two proposed methods, after 20% of the weights are pruned.

For the TID dataset, the scatter plots in Fig. 13a and b show the predictions of the original DNN and the slimmed DNN which are pruned by the proposed branching tree pruning algorithm, respectively. In the scatter plot, the x-axis shows the actual scores and the y-axis shows the predictions, where both actual scores and predictions are divided into the grids from 0 to 6. The circle size and color indicate the number of samples that match the



(a) Original DNN (b) Pruned DNN with 20%

Fig. 16 Predictions for New dataset

Table 2 Accuracy rates of pruned DNNs with trained weights for the six datasets (MINST, CIFAR, TID, Noise, Actual and New)

		Tested methods				Proposed methods	
Datasets	Original DNN (%)	Large final (%)	Random (%)	Smallest weights (%)	Smallest weights global (%)	Proposed operator (%)	Proposed full algorithm (%)
MINST	93.29	16.29	11.35	11.13	8.92	95.06	98.25
CIFAR	69.84	10.03	12.75	11.81	12.87	72.62	70.93
TID	62.33	58.84	56.86	57.57	58.43	63.67	65.06
Noise	69.15	69.05	68.65	68.07	69.01	70.89	71.94
Actual	69.01	65.03	66.08	66.16	67.05	70.13	70.76
New	68.50	66.92	66.62	66.03	66.28	70.20	70.46

corresponding prediction and actual score. When more samples match, the corresponding circle size is larger and the color is lighter. For example, we consider the coordinates (5,5) and (5,6). (5,5) indicates the number of samples which predict 5 when the actual score is 5. (5,6) indicates the number of samples which predict 6 when the actual score is 5. The two figures clearly show that the circle size of (5,5) is larger and also the color is lighter compared to (5,6). In the diagonal grids, the figures clearly indicate that the circle sizes are larger and the circle colors are lighter than the off-diagonal grids. The figures show that the patterns or grid distributions of both the original DNN and the slimmed DNN are similar. Therefore, similar predictions can be achieved by the slimmed DNN and the original DNN. 20% of memory can be saved when the DNN pruned with 20% of weights is used.

Similar results can be found for Noise, Actual and New datasets which are shown in Figs. 14, 15 and 16, respectively. The figures show that the patterns or grid distributions of both original DNN and pruned DNN are similar for those of the Noise, Actual and New datasets. Therefore, these results further show that similar predictions can be obtained by the pruned DNNs and the original DNNs. These results further demonstrate the effectiveness of the proposed branching tree pruning algorithm.

Table 2 summarizes the accuracy rates for the original DNNs which have not been pruned and the accuracy rates achieved by the tested methods, Large final, Random, Smallest weights and Smallest weights global, and the proposed methods, Proposed operator and Proposed full algorithm. For the MINST and CIFAR data, the original DNNs are pruned with 99% of the weights, where the network architecture for evaluating the performance of pruning algorithms is used [23]. For MINST and CIFAR, the table shows that the accuracy rates of the DNNs pruned by the tested methods decrease significantly, compared to the original DNNs. The Proposed operator and Proposed full algorithm achieve better results. For the TID, Noise,

Actual and New data, the DNN architecture is implemented for image quality evaluations [38]. The results show that the accuracy rates decrease for the tested methods compared to the original DNNs. Better accuracy rates can be achieved by the proposed methods. These results further indicate that the proposed methods outperform the tested methods.

5 Conclusion

In this paper, a branching tree pruning algorithm incorporated with a ROULETTE-PRUNE operator was proposed to simplify cumbersome DNNs by reducing the number of weights while keeping the DNN accuracy. The proposed algorithm attempts to overcome the limitation of the commonly used approaches that only pruned small weights which are possible to be connected with significant weights and have impact to DNN outputs. DNN accuracies can be degraded significantly after the pruning. The proposed algorithm is similar to the genetic algorithm with roulette wheel selection. It generates a few DNNs which prune both large and small weights simultaneously. Small weights have higher chances to be pruned but have chances to be kept; large weights are likely to be kept but have chances to be pruned. The process continues for many iterations to explore simpler DNNs while the DNN accuracy can be kept.

Experiments were conducted with two deterministic datasets namely MNIST and CIFAR-10, and four non-deterministic datasets, namely full TID, Noise, Actual and New. The performance of the proposed algorithm was compared to those of the four commonly used pruning methods. For the deterministic datasets, the proposed algorithm is able to generate simpler DNNs; the DNN accuracy can be kept particularly when the pruning rate is high. For the non-deterministic datasets, results show that more accurate DNNs can be generated by the proposed algorithm after the DNNs are pruned. The other tested methods generated poorer DNNs after the pruning. These results show that similar or better predictions can be obtained by the pruned DNNs compared to the original DNNs, when the proposed algorithm is used. Less memory is required, when the pruned DNNs are used but the accuracy can be kept.

In this paper, the performance of the proposed algorithm has only been validated by 2D images or offline predictions, where 3D images and online predictions have not be used to validate the algorithmic performance. In the future, we will further validate the algorithmic performance by implementing the proposed algorithm on our current research which involves DNNs with 3D images [40] and online predictions [41]. We will use the proposed algorithm to slim DNNs which are used for medical diagnosis with 3D images [40]. We will validate the prediction performance when the insignificant weights of the DNNs are pruned. We will evaluate the algorithmic performance by implementing the proposed algorithm on acoustic communication systems which require real time predictions [41]. We will validate whether prediction accuracy can be kept and how much computation time can be reduced when the DNNs are pruned by the proposed algorithm.

Acknowledgements The second author is supported by RGC Grant PolyU 15203923, the Projects of Strategic Importance (1-ZE1Y), Faculty of Science Dean's Reserve (1-ZVT5), and Projects (4-ZZPT, 1-WZ0E) of The Hong Kong Polytechnic University, Hong Kong.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Data availability All data generated or analyzed during this study are included in this published article.

Declarations

Conflict of interest This study was not funded by any grants. The author and the coauthors have no Conflict of interest. This article does not contain any studies with human participants performed by any of the authors.

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. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444
2. Bao RX, Yuan X, Chen ZK, Ma RX (2018) Cross-entropy pruning for compressing convolutional neural networks. Neural Comput 30(11):3128–3149
3. Liang T, Glossner J, Wanga L, Shi SB, Zhang XT (2021) Pruning and quantization for deep neural network acceleration: a survey. Neurocomputing 461:370–403
4. Alhalabi B, Gaber MM, Basura S (2021) Weights with the smallest magnitude values are set to zero micronets: a multi-phase pruning pipeline to deep ensemble learning in iot devices. Comput Electr Eng 96:107581
5. Pei SW, Wu YS, Guo J, Qiu MK (2022) Neural network pruning by recurrent weights for finance market. ACM Trans Internet Technol 22(3):56

6. Liu Z, Li J, Shen Z, Huang G, Yan SM, Zhang CS (2017) Learning efficient convolutional networks through network slimming, in: Proc. ICCV, pp. 2736–2744
7. Yang Y, Lang J, Wu J, Zhang YY, Su LJ, Song XM (2022) Wind speed forecasting with correlation network pruning and augmentation: a two-phase deep learning method. Renew Energy 198:267–282
8. Gao Y, Miyata SH, Akashi Y (2023) How to improve the application potential of deep learning model in hvac fault diagnosis: based on pruning and interpretable deep learning method. Appl Energy 348:121591
9. Jiang PC, Xue Y, Neri F (2023) Convolutional neural network pruning based on multi-objective feature map selection for image classification. Appl Soft Comput 139:110229
10. Fernandes F, Yen GG (2021) Pruning deep convolutional neural networks architectures with evolution strategy. Inf Sci 552:29–47
11. Li HY, Lu H, Wang HX, Deng SJ, Li XW (2023) Bitxpro: regularity-aware hardware runtime pruning for deep neural networks. IEEE Trans Very Large Scale Integr Syst 31(1):90–103
12. Fernandes F, Yen GG (2023) The hardware impact of quantization and pruning for weights in spiking neural networks. IEEE Trans Circuits Syst II Express Briefs 70(5):1789–1793
13. Fernandes F, Yen GG (2021) Automatic searching and pruning of deep neural networks for medical imaging diagnostic. IEEE Trans Neural Netw Learn Syst 32(12):5664–5674
14. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86:2278–2323
15. Hassibi B, Stork DG, Wolff GJ (1993) Optimal brain surgeon and general network pruning. In: IEEE international conference on neural networks, pp 293–299
16. Lei W, Chen H, Wu Y (2017) Compressing deep convolutional networks using kmeans based on weights distribution. In: proceedings of international conference on intelligent information processing, pp 1–6
17. Zhao M, Tong X, Wu WX, Wang Z, Zhou BX, Huang XD (2022) A novel deep-learning model compression based on filter-stripe group pruning and its iot application. Sensors 22:5623
18. Yu J, Lukefahr A, Palframan D, Dasika G, Das R, Scalpel SM (2017) Customizing dnn pruning to the underlying hardware parallelism. ACM SIGARCH Comput Archit News 45:548–560
19. Han S, Pool J, Narang S, Mao H, Gong E, Tang S, Elsen E, Vajdaand P, Paluri M, Catanzaro JB, Dally WJ (2017) Dsd: Dense-sparse-dense training for deep neural networks. In: international conference on learning representations
20. Luo JH, Wu J (2020) Autopruner: an end-to-end trainable filter pruning method for efficient deep model inference. Pattern Recogn 107:1–10
21. Frankle J, Carbin M (2019) The lottery ticket hypothesis: finding sparse, trainable neural networks. In: international conference on learning representations
22. Blickle T, Thiele L (1996) A comparison of selection schemes used in evolutionary algorithms. Evol Comput 4(4):361–394
23. Rachwan J, Zugner D, Charpentier B, Geisler S, Ayle M, Gunnemann S (2022) Winning the lottery ahead of time: efficient early network pruning. In: proceedings of the 39th international conference on machine learning, pp 25668–25683
24. Ponomarenkoa N, Jinb L, Ieremeieva O, Lukina V, Egiazarianb K, Astolab J, Vozelc B, Chehdic K, Carlid M, Battistid F, Kuo CCJ (2015) Image database tid2013: peculiarities, resultsand perspectives. Signal Process Image Commun 30:57–77
25. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958
26. Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012) Improving neural networks by preventing co-adaptation of feature detectors, CoRR
27. Lee N, Ajanthan T, Torr PHS (2020) Snip: single-shot network pruning based on connection sensitivity. In: proceedings of the international conference on learning representations
28. Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2017) Pruning convolutional neural networks for resource efficient inference. In: proceedings of the international conference on learning representations
29. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2017) Pruning filters for efficient convnets. In proceedings of international conference on learning representations
30. da Cunha ACW, Natale E (2022) Proving the strong lottery ticket hypothesis for convolutional neural networks. In: international conference on learning representations
31. He Z, Xie Z, Zhu Q, Qin Z (2022) Sparse double descent: where network pruning aggravates overfitting. In: proceedings of the 39th international conference on machine learning, pp 8635–8659
32. Pietron M, Wielgosz M (2020) Retrain or not retrain?-efficient pruning methods of deep cnn networks. In: IEEE proceedings of computational science, IEEE, pp 452–46
33. Noy A, Nayman N, Ridnik T, Zamir N, Doveh S, Friedman I, Giryes R, Zelnik L (2020) Asap: architecture search, anneal and prune. In: proceedings of the twenty third international conference on artificial intelligence and statistics, pp 452–46
34. Renda A, Frankle J, Carbin M (2020) Comparing rewinding and fine-tuning in neural network pruning. In: proceedings of the international conference on learning representations
35. Yu X, Serra T, Ramalingam S, Zhe S (2022) The combinatorial brain surgeon: pruning weights that cancel one another in neural networks. In: proceedings of the 39th international conference on machine learning, pp 25668–25683
36. Kim J, Lee S (2017) Fully deep blind image quality predictor. IEEE J Selected Top Sig Process 11(1):206–220
37. Kang L, Ye P, Li Y, Doermann D (2014) Convolutional neural networks for no-reference image quality assessment. In: proceedings of IEEE international conference of conference on computer vision and pattern recognition, pp 1733–1740
38. Bosse S, Maniry D, Muller TWKR, Samek W (2018) Deep neural networks for no-reference and full-reference image quality assessment. IEEE Trans Image Process 27(1):206–219
39. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. In: IEEE proceedings of imagenet challenge, IEEE, pp 1–10
40. Man N, Guo S, Yiu KFC, Leung CSK (2023) Multi-layer segmentation of retina oct images via advanced u-net architecture. Neurocomputing 515:185–200
41. Hassan S, Chen P, Rong Y, Chan KY (2023) Doppler shift compensation using an lstm-based deep neural network in underwater acoustic communication systems. In: proceedings of the IEEE OCEANS conference

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