

Article

Activation-Based Pruning of Neural Networks

Tushar Ganguli *  and Edwin K. P. Chong 

Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523, USA; edwin.chong@colostate.edu

* Correspondence: tushar.ganguli@colostate.edu

Abstract: We present a novel technique for pruning called *activation-based* pruning to effectively prune fully connected feedforward neural networks for multi-object classification. Our technique is based on the number of times each neuron is activated during model training. We compare the performance of activation-based pruning with a popular pruning method: magnitude-based pruning. Further analysis demonstrated that activation-based pruning can be considered a dimensionality reduction technique, as it leads to a sparse low-rank matrix approximation for each hidden layer of the neural network. We also demonstrate that the rank-reduced neural network generated using activation-based pruning has better accuracy than a rank-reduced network using principal component analysis. We provide empirical results to show that, after each successive pruning, the amount of reduction in the magnitude of singular values of each matrix representing the hidden layers of the network is equivalent to introducing the sum of singular values of the hidden layers as a regularization parameter to the objective function.

Keywords: machine learning; network pruning; dimensionality reduction; computer vision



Citation: Ganguli, T.; Chong, E.K.P. Activation-Based Pruning of Neural Networks. *Algorithms* **2024**, *17*, 48. <https://doi.org/10.3390/a17010048>

Academic Editor: Frank Werner

Received: 18 December 2023

Revised: 11 January 2024

Accepted: 19 January 2024

Published: 21 January 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Deep neural networks are used to solve real-world problems in various domains such as image classification, text classification, and speech recognition. These networks often require millions of parameters and billions of floating-point operations to make accurate predictions. Network pruning has emerged as an important technique for improving the efficiency of deep neural networks by removing redundant structures. Pruning reduces the number of parameters of a neural network, resulting in a reduction of the computational resource required to run the network. Some of the most-popular pruning methods are magnitude-based pruning [1,2], structured pruning [3,4], pruning based on the lottery ticket hypothesis [5], and dynamic pruning [6].

Of these methods, magnitude-based pruning [2,7] has been proven to be successful for producing compact models and has witnessed widespread acceptance. However, prior work on magnitude-based pruning contains certain deficiencies. As we demonstrate in our study, magnitude-based pruning does not inherently induce a low-rank structure in the hidden layers of neural networks. More-rigorous constraints are needed to drive rank reduction. Integrating it with structured pruning or low-rank regularizers is likely necessary to fully exploit the compression property.

To address these limitations, we propose a novel technique for pruning called activation-based pruning to reduce the number of parameters in a feedforward neural network. Our objective was to achieve results comparable to magnitude-based pruning [8] in terms of training, validation, and testing accuracy, while also demonstrating that activation-based pruning functions as a dimensionality reduction technique. This method effectively reduces the hidden layers of neural networks to sparse low-rank matrix approximations. Activation-based pruning can be achieved using both labeled and unlabeled data, provided the data are representative of the same distribution as the training data. Our empirical evidence supports the assertion that activation-based pruning is equivalent to introducing

a weighted nuclear norm as a regularization parameter during the minimization of the objective function in image classification tasks. Consequently, activation-based pruning eliminates the need for additional regularizers to induce a low-rank structure in the hidden layers of the feedforward network.

To further analyze the low-rank structure of the hidden layers, we conducted an empirical comparison between the low-rank structures generated using activation-based pruning and those obtained through principal component analysis (PCA) [9]. PCA is a linear dimensionality reduction technique that seeks to find orthogonal axes, known as principal components, in the high-dimensional input space. These principal components capture the maximum variance in the data while reducing dimensionality. In the context of a trained feedforward neural network, each hidden layer functions as a representation of input data capturing specific features inherent to the input. In this study, we applied rank- k reduction using PCA to each hidden layer, where the value of k is layer-specific. For a trained feedforward neural network, we conducted a comparative analysis of the network's training accuracy after reducing the dimensions of the hidden layers through PCA versus employing activation-based pruning to achieve a sparse low-rank matrix approximation.

2. Related Work

2.1. Low-Rank Matrix Approximation

Various pruning methods [2,10,11] have resulted in sparse matrices, low-rank matrix approximation, or a hybrid of both. Han et al. [2] compressed deep neural networks by simultaneously pruning both network weights and connections to reduce computational cost and memory usage by inducing a regularization term that encourages sparsity during training. Weights with small magnitudes are pruned based on a specified threshold. This technique results in sparse weight matrices. Swaminathan et al. [10] proposed a novel method called sparse low rank (SLR) to compress the dense layers of deep neural networks by improving upon truncated singular-value decomposition (SVD). The key idea is to induce structured sparsity into the decomposed matrices from SVD based on the significance of the input/output neurons. Neuron significance is estimated by absolute weights, activations, or a change in cost when removed. Yang et al. [11] proposed a method called SVD training, which first decomposes each layer into the form of its full-rank SVD and, then, performs training on the decomposed weights. Low rank is encouraged by applying sparsity-inducing regularizers on the singular values of each layer. Singular-value pruning is applied at the end to explicitly reach a low-rank model. Activation-based pruning achieves sparsity in low-rank matrix approximation by assigning scores to each neuron to identify significant and insignificant neurons. Our method avoids the computationally expensive process of decomposing matrices using SVD, making it advantageous for large matrices.

2.2. Structured/Unstructured Pruning

Pruning can be classified into structured [3–5,12–16], unstructured [2,17], and semi-structured [18] pruning. Structured pruning removes filters, channels, or layers to induce structured sparsity patterns. It is commonly used in convolutional neural networks (CNNs), where entire filters or channels (groups of neurons) can be pruned. In unstructured pruning, individual weights without structural constraints are considered for removal. It can be applied to any layer of a neural network, including fully connected layers and convolutional layers. Semi-structured pruning is a hybrid approach that combines aspects of both structured and unstructured pruning. It involves removing entire structures, such as filters or channels, but within those structures, individual weights may be pruned. Activation-based pruning falls under unstructured pruning, as we assign a score to individual neurons for pruning.

2.3. Importance-Based Pruning

Pruning can also be categorized based on the importance assigned to the weights, filters, and neurons of the network. These techniques induce sparsity by removing connections or filters based on criteria such as the weight magnitude [4,5,12] or sensitivity scores [19,20]. The sparse architectures are then retrained to regain accuracy. Zhu and Gupta [8] explored model pruning as a means of model compression by implementing magnitude-based pruning, where the weights with the smallest absolute values are pruned. In activation-based pruning, importance is assigned to the neurons of the network by assigning a score for each neuron, which guides the decision regarding connections to prune.

2.4. Iterative/One-Shot Pruning

Pruning can also be categorized as either iterative [2,6,7,21] or one-shot [20]. Iterative pruning is a multi-step process of assigning a score, pruning the network, and retraining. Han et al. [2] proposed a three-step pipeline to prune redundant connections in neural networks without affecting accuracy. They first trained the dense network, then pruned low-weight connections below a threshold to obtain a sparse network, and finally, retrained the sparse network to learn the weight parameters. Guo et al. [6] introduced a two-step process called pruning and splicing, where weight connections can be removed and added back, based on solving a constrained optimization problem for each layer. Han et al. [7] used a three-pronged approach of pruning, quantization, and Huffman coding, to achieve substantial compression of the network. Yuan et al. [21] demonstrated how networks can be grown and pruned dynamically during the training phase using structured continuous sparsification. Growing and pruning of the network are often performed by introducing a regularization parameter in the cost function and relaxing the initial optimization problem. Liu et al. [20] implemented one-shot pruning to prune weights in a single step. Activation-based pruning is an iterative method. Initially, a score is assigned to each neuron, and subsequently, the neurons with the lowest score are pruned. Afterward, the network is retrained, and this cycle is repeated multiple times.

2.5. When to Prune

Pruning can also be classified based on when the pruning occurs, before [2,6,19,22], during [5,17,21,23], or after training [7,8]. The motivation for pruning before training is to eliminate the cost of pretraining. Pruning during training iteratively prunes and retrains the network to induce sparsity by updating the weight magnitudes or filters and channels, and pruning after training generally takes a pretrained network and, subsequently, prunes and retrains multiple times. Activation-based pruning comes under pruning during training. We set a predefined training accuracy for the network to achieve before initiating the pruning and retraining cycles during training.

With the advent of large language models (LLMs), established pruning methods have failed to scale successfully. Frantar et al. [24] demonstrated that magnitude-based pruning [2] fails in LLMs with relatively low levels of sparsity. As activation-based pruning uses a small network and small dataset, the scalability of our approach belongs to a future scope of work. Hence, we refrained from including pruning techniques related to bigger architectures and larger datasets. However, we mention [25] to clarify that the methodology is different than our implementation. Sun et al. [25] introduced pruning for LLMs based on the product of network weights and input activation of neurons. Our method differs in principle as we did not consider the activation value of the neurons, but the number of times the neuron is activated during the training phase of the neural network.

3. Methodology

Activation-based pruning is based on the number of times each neuron is activated during the forward pass of the training phase of a neural network. In the forward pass, data are processed at each layer and passed on to the next layer until the processed data reach the output layer. In a fully connected network, each neuron is connected with incoming

weights from all the neurons of the previous layer. The output of a neuron is calculated as $\sigma(W^T X)$, where W is the vector of incoming weights, X is the vector of processed data from the previous layer, and σ is the non-linear activation function. Activation of a neuron means that its output is not zero, $\sigma(W^T X) \neq 0$. The number of times a neuron is activated is stored as a parameter called activation counter. Each neuron has a corresponding activation counter. During training, we chose a random set of data as the input to the network before each cycle of pruning, which resulted in the neurons either being activated ($\sigma(W^T X) \neq 0$) or not activated ($\sigma(W^T X) = 0$). Every time a neuron is activated, we incremented the value of the activation counter by 1. The values of the activation counter of all the neurons are used to perform pruning. After every pruning cycle, the network suffers some loss in training accuracy. Hence, we trained the network to reach a predetermined training accuracy before we initiated the next pruning cycle. The random set of data used to compute the activation counter can belong either to the training set or can be separated before the start of the training phase. The data can either be labeled or unlabeled, provided they belong to the same distribution used to train the neural network. We carried out empirical comparison of two ways of pruning the network using activation-based pruning. The first was global activation-based pruning, where at each stage of pruning, we considered neurons from all hidden layers of the network to decide which neurons to prune. The second was local activation-based pruning, where at each stage of pruning, we pruned one hidden layer.

3.1. Global Activation-Based Pruning

Algorithms 1–3 demonstrate the steps for pruning the network during training. Figure 1 illustrates pruning a fully connected feedforward network. The figure depicts that connection of neurons with activation counters 34 and 41 removed. This is a visual representation of activation-based pruning. We implemented the pruning by setting the incoming weights of the pruned neurons to 0.

Algorithm 1 Global activation-based pruning.

```

1: Input:  $X$ : image data,  $W$ : weight matrix.
2: Require:  $F_{ta}$ : final training accuracy,  $C_{ta}$ : current training accuracy,  $T_p$ : target prune
   percentage,  $C_p$ : current prune percentage,  $P_p$ : total pruned percentage,  $N_p$ : number of
   pruning cycles,  $P_l$ : pruning list in  $N_p$  cycles.
3: Procedure 1: Main
4: Calculate  $P_l$  for  $N_p$  pruning cycles.
5:  $P_p \leftarrow 0$ 
6: for  $C_p \in P_l$  do
7:   Call update neuron.
8:   if  $P_p < T_p$  then
9:     Retrieve  $C_{ta}$ .
10:    if  $C_{ta} \geq F_{ta}$  then
11:      Stop training.
12:      return{Exit the algorithm}
13:    else if accuracy reached for next pruning cycle then
14:       $W_{new} \leftarrow$  call prune.
15:       $W \leftarrow W_{new}$ 
16:       $P_p \leftarrow P_p + C_p$ 
17:    end if
18:  end if
19: end for

```

Algorithm 2 Update neuron.

```

1: Input:  $M$ : random batch of data from input  $X$ ,  $L$ : hidden layers of the network.
2: Require:  $A_l$ : activation counter of neurons for layer  $l$ .
3: for each  $m \in M$  do
4:   for  $l \in L$  do
5:     Calculate output:  $F_l \leftarrow \sigma(W_l^T F_{l-1})$ .
6:     {the for loop signifies a matrix operation}
7:     for  $F_l(r, c) \in F_l$  do
8:       if  $F_l(r, c) \neq 0$  then
9:          $A_l(r, c) \leftarrow A_l(r, c) + 1$ 
10:      end if
11:    end for
12:  end for
13: end for

```

Algorithm 3 Prune.

```

1: Input:  $L$ : hidden layers of the network.
2: Require:  $R$ : list for activation counter and associated weights,  $C_p$ : current prune
   percentage,  $P_{index}$ : prune index.
3: for  $l \in L$  do
4:    $R \leftarrow R + (A_l, W_l)$ . {add activation counters and associated weights to  $R$ }.
5: end for
6: if pruning type = neuron then
7:   Sort  $R$  on the activation counter in ascending order.
8: else
9:   Sort  $R$  on the product of the activation counter and weights in ascending order.
10: end if
11:  $P_{index} \leftarrow C_p \times \text{len}(R)$ . { $\text{len}(R)$ : No. of elements in  $R$ }.
12:  $R_W(0, P_{index}) \leftarrow 0$ . {Assign 0 value to weights}.
13:  $W_{new} \leftarrow R_W$ . {Set new weights for all hidden layers}.
14: return  $W_{new}$ 

```

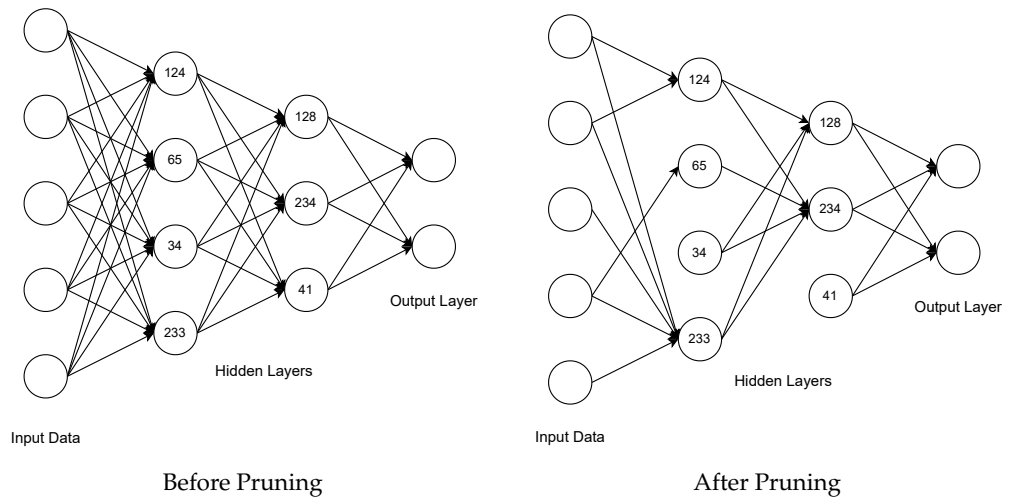


Figure 1. Network before and after pruning. Values in the circles indicate the activation counter of the respective neurons.

3.2. Local Activation-Based Pruning

The disadvantage of global activation-based pruning is the time and space complexity of the algorithm. Table 1 presents the time and space complexity for global and local activation-based pruning. We employed Big- $\mathcal{O}$ notation as a standard means to express

the time and space complexity of our proposed algorithm. For global activation-based pruning, the space complexity for storing activation counters for n neurons is $\mathcal{O}(n)$. Analyzing Algorithm 3, we find that the processing time for global activation-based pruning is $\mathcal{O}(n \log(n))$. As the network grows larger, it puts a considerable strain on the memory and processing requirement for each pruning cycle. We present an alternative implementation for activation-based pruning, where pruning is performed on one hidden layer per pruning cycle. We obtain significant performance improvements if the total number of neurons in a single layer l is much less than the total number of neurons in the entire network n , i.e., $l \ll n$. The space complexity to store the activation counters is $\mathcal{O}(l)$, and the time complexity for pruning is $\mathcal{O}(l \log(l))$. We initiated pruning with the first hidden layer of the network immediately following the input layer. We sequentially applied this procedure to each subsequent hidden layer in the ensuing pruning cycles. We stopped after pruning the hidden layer before the output layer. If the total pruning of the network was set at $x\%$, we pruned $x\%$ of a hidden layer during one pruning cycle. The total number of pruning cycles is equal to the number of hidden layers present in the feedforward network. Algorithms 4–6 demonstrate the steps for pruning the network during training.

Table 1. Time and space complexity for different types of pruning with $l \ll n$, where n represents the total number of neurons in the hidden layers and l denotes the number of neurons in a single hidden layer.

Pruning Type	Time	Space
Global	$\mathcal{O}(n \log(n))$	$\mathcal{O}(n)$
Local	$\mathcal{O}(l \log(l))$	$\mathcal{O}(l)$

Algorithm 4 Local activation-based pruning.

```

1: Input:  $X$ : image data,  $W$ : weight matrix.
2: Require:  $F_{ta}$ : final training accuracy,  $C_{ta}$ : current training accuracy,  $T_p$ : target prune
   percentage,  $C_p$ : current prune percentage,  $P_p$ : total pruned percentage,  $L$ : hidden layers
   of the network.
3: Procedure 1: Main
4:  $P_p \leftarrow 0$ 
5: for  $l \in L$  do
6:   if  $P_p < T_p$  then
7:     Retrieve  $C_{ta}$ .
8:     if  $C_{ta} \geq F_{ta}$  then
9:       Stop training.
10:      return{Exit the algorithm}
11:   else if accuracy reached for next pruning cycle then
12:     Call update neuron.
13:      $W_{l_{new}} \leftarrow$  Call prune.
14:      $W_l \leftarrow W_{l_{new}}$ 
15:      $P_p \leftarrow P_p + C_p$ 
16:   end if
17: end if
18: end for

```

Algorithm 5 Update neuron.

```

1: Input:  $M$ : random batch of data from input  $X$ ,  $l$ : Hidden layer to be pruned.
2: Require:  $A_l$ : activation counter of neurons for layer  $l$ .
3: for each  $m \in M$  do
4:   Calculate output:  $F_l \leftarrow \sigma(W_l^T F_{l-1})$ .
5:   {the for loop signifies a matrix operation}
6:   for  $F_l(r, c) \in F_l$  do
7:     if  $F_l(r, c) \neq 0$  then
8:        $A_l(r, c) \leftarrow A_l(r, c) + 1$ 
9:     end if
10:  end for
11: end for

```

Algorithm 6 Prune.

```

1: Input:  $l$ : hidden layer to be pruned,
2: Require:  $R_l$ : list for activation counter and associated weights for layer  $l$ ,  $C_p$ : current
   prune percentage,  $P_{index}$ : prune index.
3:  $R_l \leftarrow (A_l, W_l)$ . {assign activation counters and associated weights to  $R_l$ }.
4: if pruning type = neuron then
5:   Sort  $R_l$  on the activation counter in ascending order.
6: else
7:   Sort  $R_l$  on the product of the activation counter and weights in ascending order.
8: end if
9:  $P_{index} \leftarrow C_p \times \text{len}(R_l)$ . { $\text{len}(R_l)$ : No. of elements in  $R_l$ }.
10:  $R_{lW}(0, P_{index}) \leftarrow 0$ . {Assign 0 value to weights}.
11:  $W_{l_{new}} \leftarrow R_{lW}$ . {Set new weights for hidden layer  $l$ }.
12: return  $W_{l_{new}}$ 

```

3.3. Activation- vs. Magnitude-Based Pruning

Magnitude-based pruning as described in [8], is available in the TensorFlow library. We used the same experiment setup mentioned in Section 4.1. We compared activation- and magnitude-based pruning by pruning 80% of the network and, subsequently, attempting to retrain the pruned network to 98% accuracy. We observed the effect of retraining the pruned networks, on the training, validation, and test accuracy. We also observed the effect on the rank of each hidden layer of the pruned network. We wanted to determine the extent to which the networks were able to maintain sparsity and low-rank approximation after retraining.

3.4. Principal Component Analysis of Hidden Layers

Activation-based pruning generates low-rank approximation of each matrix representing the hidden layers of the neural network. Our hypothesis posited that activation-based pruning offers a better rank- k approximation of network layers while preserving significant levels of training, validation, and test accuracy. To substantiate this hypothesis, we ascertained the ranks of each pruned model and applied PCA to a fully trained model, resulting in the creation of rank-reduced models. Subsequently, we compared the test accuracy of the PCA-generated rank-reduced models with those derived from activation-based pruning.

3.5. Effect of Pruning on Singular Values of Matrices of Hidden Layers

Every hidden layer of a feedforward neural network can be represented in the form of a matrix. The singular values of a matrix can be computed using singular-value decomposition (SVD). The magnitudes of the singular values capture information about the data. The larger the singular value, the greater the information captured by the corresponding basis vector is. We observed the change in singular values of the weight matrix of each hid-

den layer before and after pruning. Notably, activation-based pruning leads to a low-rank representation of the hidden layer by selectively pruning the less important singular values.

Consider a feedforward neural network for the image classification problem formulated as an optimization problem. Given a set of images with corresponding labels, the goal is to learn a function f that maps each image X to its correct label y . The optimization problem is to find the optimal set of weights and biases for the neural network to minimize the classification error on the training data. This can be formalized as:

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^n L(f(x_i; w, b), y_i) \quad (1)$$

where w and b are the weights and biases of the neural network, n is the number of images in the training set, x_i is the i th image in the set, y_i is the corresponding label, $f(x_i; w, b)$ is the predicted label for image x_i with weights w and biases b , and L is the loss function that measures the difference between the predicted and true label.

The nuclear norm $\|X\|_*$ of a matrix X is defined as the sum of its singular values:

$$\|X\|_* = \sum_i |\lambda_i(X)|_1 \quad (2)$$

where $\lambda_i(X)$ is the i -th singular value of the matrix X .

The weighted nuclear norm of a matrix X is defined as:

$$\|X\|_{w,*} = \sum_i |w_i \lambda_i(X)|_1, \quad (3)$$

where $w = [w_1, \dots, w_n]$ and $w_i \geq 0$ is a non-negative weight assigned to $\lambda_i(X)$.

The weighted nuclear norm of a layer of the feedforward network is defined as $\|X_j\|_{w,*}$ where X_j is the matrix of the j -th hidden layer of the network.

We will show that activation-based pruning is equivalent to adding the sum of weighted nuclear norm as a regularization parameter to the image classification problem:

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^n L(f(x_i; w, b), y_i) + \sum_{j=1}^l \|X_j\|_{w,*} \quad (4)$$

where parameters for the first part of the equation are as described in the explanation followed by (1). Including the sum of the weighted nuclear norm as a regularization parameter encourages each matrix of the neural network to have a low rank.

4. Experiment Settings

4.1. Common Setting

We implemented activation-based pruning on a 4-layer feedforward network with the total number of neurons in each layer as 300, 200, 100, and 50, respectively. We used the Rectified Linear Unit (ReLU) as the nonlinear activation function for all neurons. We used Stochastic Gradient Descent (SGD) as the optimization method, where the loss function is sparse categorical cross-entropy. To train the network, we used the Fashion MNIST database, publicly available through the Keras framework. The target training accuracy for each run was set to 98%. We allocated a part of the data for validation testing. We implemented early stopping by monitoring the validation loss so that the model did not overfit to the training data. If validation and training accuracy are improving, but validation loss is increasing, it signifies that the model might be overfitting to the training data. The stopping criterion was set at 0.1% so that training would stop when the validation loss exceeded 10% of the minimum validation loss for three consecutive times.

In our study on activation-based pruning, we conducted an in-depth analysis of importance-based scoring. This evaluation was carried out using two distinct methods for assigning importance, which we termed *Neurons* and *Neurons & Weights*. In the first

case, we pruned $k\%$ of the neurons with the lowest activation counters. In the second case, we computed the product of the activation counter of each neuron and the absolute value of the corresponding incoming weights and pruned $k\%$ of the weights with the smallest product values. More precisely, consider a neuron with an associated activation counter a . If the incoming weights associated with the neuron are given by the vector $\mathbf{w}$, we computed the product of the absolute value of the weight vector with the activation counter, $|\mathbf{w}|a$. We performed this operation for each neuron in the network and pruned $k\%$ of the weights with the smallest product values.

4.2. Global Activation-Based Pruning Setting

We started pruning the network after it reached 80% training accuracy. The final achievable training accuracy was set to 98%. This was the best-case scenario as our early stopping criterion might stop the training of the network so that the model does not overfit to the training data. We experimented with two pruning percentages of 80% and 85%. The total number of pruning cycles set for both pruning percentages was 5 and 10. We implemented a form of gradual pruning. We started by pruning a larger percentage of the network and gradually reduced the percentage for subsequent pruning cycles. More precisely, the percentage of the network pruned after n pruning cycles was set to b^n , where $b \in (0, 1)$ represents a decay factor. The network loses training accuracy after each pruning cycle. The next pruning cycle is initiated after achieving a predetermined percentage of training accuracy. This provides the network sufficient time to recover training accuracy between successive pruning.

4.3. Local Activation-Based Pruning Setting

Due to the structure of feedforward networks, where each successive hidden layer has fewer neurons than the previous hidden layer, local pruning is a natural way of implementing gradual pruning. The number of times we pruned was based on the number of hidden layers in the network. As we had 4 hidden layers in our network, we pruned 4 times. As the value of the pruning percentage was set at 80%, we pruned 80% of a hidden layer during each pruning cycle. The hyperparameters used were as mentioned in Section 4.1.

5. Results and Discussion

5.1. Global Activation-Based Pruning

Figures 2 and 3 demonstrate the training and validation accuracy for, Neurons and Neurons & Weights. The target pruning percentages were 80% and 85% with the number of pruning cycles specified as 5 and 10. Subsequently, Figures 4 and 5 demonstrate the training and validation accuracy for pruning percentages of 80% and 85% providing a different perspective of the same test run. The drop in accuracy at certain epochs in Figure 4 was due to the affect of pruning. Eventually, the network trains and recovers the loss in accuracy. Figure 2 illustrates that a higher percentage of pruning can be achieved by increasing the number of pruning cycles, particularly when employing importance score to Neurons. When the total pruning was specified as 85%, we achieved 79.25% pruning in 10 pruning cycles and 74.64% pruning in 5 pruning cycles. We did not see evidence for a similar result when the importance score was Neurons & Weights. Figure 4 illustrates that using Neurons as the importance score resulted in a pronounced decrease in accuracy compared to Neurons & Weights. For the same number of pruning cycles, the pruning percentage attained for Neurons & Weights exceeded that achieved for Neurons. For 10 pruning cycles, the left figure achieved 77.59% pruning for Neurons & Weights and 74.59% for Neurons. We see the same results in the right-hand figure where total pruning specified was 85%.

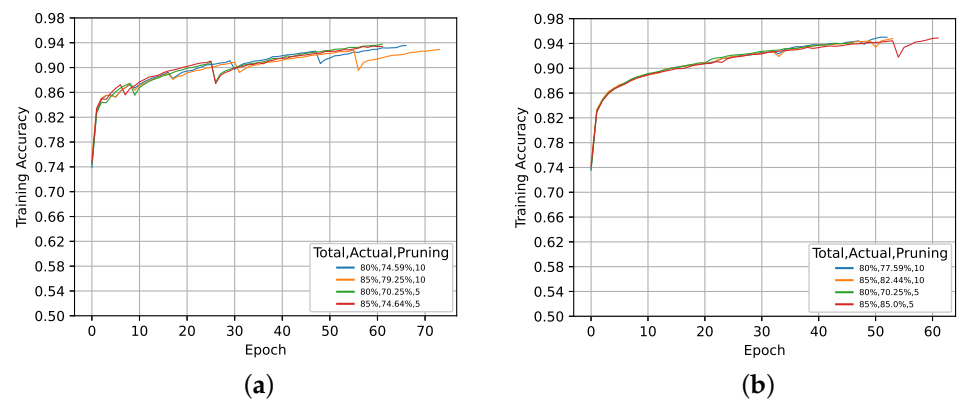


Figure 2. Training accuracy. (a) Neurons. (b) Neurons & Weights.

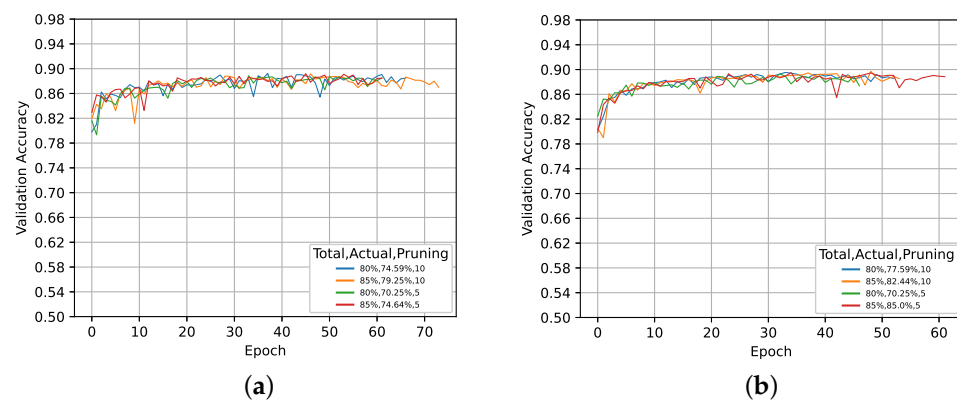


Figure 3. Validation accuracy. (a) Neurons. (b) Neurons & Weights.

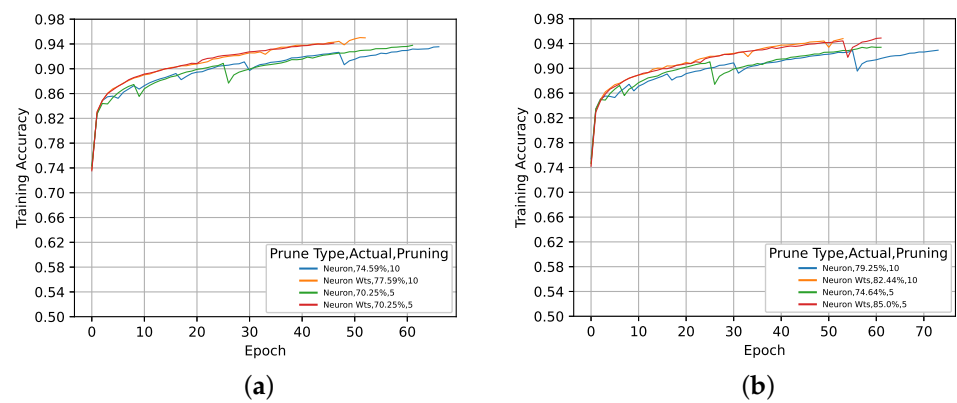


Figure 4. Training accuracy for different pruning percentages: (a) 80%; (b) 85%.

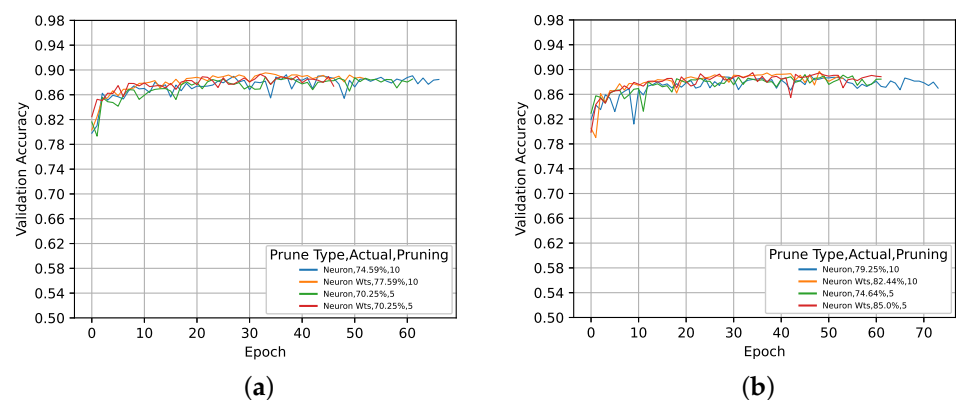


Figure 5. Validation accuracy for different pruning percentages: (a) 80%; (b) 85%.

Table 2 demonstrates the training and pruning of models with different sets of hyperparameters. The first row illustrates the results of training the network without any pruning. All subsequent rows specify the details of training and pruning the network based on the pruning number, pruning type, and target pruning percentage. The epoch column demonstrates that it took more epochs to prune and train the network compared to training the network without any pruning. After every subsequent pruning cycle, a greater number of epochs was required to recover the network's lost accuracy. All pruned models achieved training accuracy comparable to the standard model for top-1% accuracy and perform better in terms of validation and test accuracy. For top-5% accuracy, all pruned models demonstrated better results, except G1 and G6, which corresponded to the importance score of Neurons. For the same number of pruning and target prune percentages, we observed better results for the importance score of Neurons & Weights. Overall, G2, G3, G5, G7, and G8 performed better than the standard run in terms of pruning and accuracy achieved. However, the top-three results were G5, G7, and G8, which achieved considerable pruning and accuracy results. This demonstrated that the probability of achieving improved performance increased with more pruning iterations.

Table 2. Global activation-based pruning.

No.	Pruning Iterations	Pruning Type	Epochs	Target Prune %	Pruning Achieved	Top 1% Accuracy			Top 5% Accuracy		
						Training	Validation	Test	Training	Validation	Test
S1	NA	Standard	46	NA	NA	10.19	8.17	8.22	69.10	68.72	69.40
G1	5	Neurons	62	80	70.25	10.24	9.75	10.16	70.45	66.47	65.69
G2	5	Neurons	62	85	74.64	10.13	9.23	9.54	72.23	71.47	71.67
G3	5	Neurons & Weights	47	80	70.25	10.13	9.35	9.66	81.30	81.50	81.83
G4	5	Neurons & Weights	62	85	85.00	10.15	9.00	9.46	66.59	66.65	66.77
G5	10	Neurons	67	80	74.59	10.25	9.32	9.57	80.99	80.57	80.24
G6	10	Neurons	74	85	79.25	10.17	9.62	9.69	71.44	65.10	64.99
G7	10	Neurons & Weights	53	80	77.59	10.16	12.17	12.30	77.19	79.60	79.23
G8	10	Neurons & Weights	54	85	82.44	10.15	11.42	11.58	78.35	78.45	79.31

While activation-based pruning prolonged the network training duration, it consistently led to better validation and test accuracy in almost all scenarios. Pruning resulted in a loss of accuracy, which was recovered by retraining the network. The pruned model can be used for prediction in place of the original model. As discussed in Section 3.2, a limitation of global activation-based pruning is its memory- and time-intensive nature, which may lead to poor scalability for larger networks.

5.2. Local Activation-Based Pruning

Table 3 demonstrates the results of local activation-based pruning. Compared to Table 2 for global pruning, local pruning demonstrated a superior pruning percentage, while maintaining the training, validation, and test accuracy at a comparable level. Table 3 also demonstrates that the validation and test accuracy for local pruning was better than the standard run. For top-1% accuracy, nearly all pruning runs had better results than the standard run, except for the validation accuracy for Neurons & Weights in the fourth row, where the percentage of pruning achieved was 79.88%. For top-5% accuracy, all results were better than the standard run, except L1. In summary, it was evident that L2, L3, and L4 yielded significantly improved results compared to the standard run.

Table 1 demonstrates that local pruning exhibited superior space and time complexity compared to global pruning. A comparison of Tables 2 and 3 demonstrates that local pruning was much closer to achieving the target pruning percentage while maintaining comparable training, validation, and test accuracy. Local pruning delivered equivalent accuracy and pruning percentages to global activation-based pruning, while significantly enhancing processing time efficiency and reducing memory consumption.

Table 3. Local activation-based pruning.

No.	Pruning Type	Epochs	Pruning Achieved	Training Accuracy	Validation Accuracy	Test Accuracy	Top 1% Accuracy			Top 5% Accuracy		
							Training	Validation	Test	Training	Validation	Test
S	Standard	46	NA	95.64	85.68	84.35	10.19	8.17	8.22	69.10	68.72	69.40
L1	Neurons	88	79.88	93.12	88.02	87.56	10.22	11.08	10.83	64.33	63.37	64.09
L2	Neurons	88	84.85	92.29	86.73	86.79	10.18	12.12	11.99	72.81	73.55	74.67
L3	Neurons & Weights	75	79.88	93.29	87.92	87.11	10.19	7.73	8.01	86.36	84.92	85.29
L4	Neurons & Weights	73	84.87	92.83	87.62	86.94	10.23	12.10	12.10	74.26	75.92	75.64

5.3. Activation- vs. Magnitude-Based Pruning

In this section, we will conduct a comprehensive comparison between activation-based and magnitude-based pruning across various evaluation metrics. We will first analyze the impact of these pruning techniques on training, validation, and test accuracy. Furthermore, we will assess the computational efficiency by comparing the FLOPS requirements after network pruning. Additionally, we will examine the test results obtained after retraining the pruned network. Lastly, we will evaluate the rank of each hidden layer both before and after retraining the pruned networks.

Table 4 demonstrates that activation-based pruning was able to generate equivalent results to magnitude-based pruning. Activation-based pruning yielded better training accuracy compared to magnitude-based pruning, while demonstrating slightly lower validation accuracy and equivalent test accuracy. Activation-based pruning achieved network pruning comparable to magnitude-based pruning while preserving equivalent accuracy.

Table 4. Test result after pruning.

Model	Activation Type	Pruning Percentage	Training Accuracy	Validation Accuracy	Test Accuracy
Magnitude		80.00	91.79	92.35	88.72
Activation-based	Neurons	79.88	93.12	88.02	87.56
Activation-based	Neurons & Weights	79.88	93.29	87.92	87.11

In Tables 5–7, model M represents magnitude-based pruning, while the other entries correspond to activation-based pruning. Within activation-based pruning, models G1 to G8 underwent global pruning, while models L1 to L4 underwent local pruning.

Table 5 demonstrates the floating point operation per second (FLOPS) for each model. The explanation for the model symbols can be obtained from Tables 2 and 3. We demonstrate the FLOPS using kFLOPS, denoted as 1/1000th of FLOPS, commonly referred to as KiloFLOPS. For a fully connected feedforward network, the number of floating-point operations considering only non-zero weights was computed as $\sum_{i=1}^l (2 \times W_{nz}^i)$, where W_{nz}^i is the number of non-zero weights of the i -th layer and l is the total number of hidden layers of the network. We multiplied W_{nz}^i by 2 inside the summation, as each weight will be involved in one multiplication and one addition with the input. The values G4, G8, L1 to L4 in the Pct. Reduction column signify better performance relative to magnitude-based pruning. Local activation-based pruning outperformed global activation-based pruning and magnitude-based pruning, in reducing the computational cost of the network, as assessed through the FLOPS computation.

Table 6 demonstrates the training, validation, and testing accuracy achieved after retraining the pruned network. The training accuracy for models G1 to G8 was comparable to model M. However, the training accuracy of models L1 to L4 was lower than that of model M. The validation and test accuracy of models G1 to G8 and L1 to L4 was comparable to model M. For top-1% accuracy, the validation and test accuracy for model M were much higher than for the other models. For top-5% accuracy, models G3, G4, L2, L3, and L4 performed much better than model M. Our observations indicated that, in certain instances, magnitude-based pruning may outperform activation-based pruning. This disparity in performance is primarily attributed to the fact that retraining a network

subjected to magnitude-based pruning tends to utilize the full spectrum of network weights. In contrast, networks pruned using the activation-based method strive to preserve the low-rank structure of the hidden layers. A more-detailed explanation of this phenomenon is provided in the following paragraph.

Table 5. FLOPS comparison. kFLOPS before pruning ≈ 641 .

Pruning Type	$\approx$ kFLOPS	Pct. Reduction
M	129	79.87
G1	191	70.25
G2	163	74.64
G3	191	70.25
G4	96	85.00
G5	163	74.59
G6	133	79.25
G7	144	77.59
G8	113	82.44
L1	129	79.88
L2	97	84.85
L3	129	79.88
L4	97	84.87

Table 6. Test results after retraining pruned networks.

Model	Pruning Type	Epochs	Training Accuracy	Validation Accuracy	Test Accuracy	Top 1% Accuracy			Top 5% Accuracy		
						Training	Validation	Test	Training	Validation	Test
M	Magnitude	9	95.01	87.37	87.01	10.23	14.10	14.09	66.62	74.13	74.85
G1	Neuron	21	95.24	86.55	85.99	10.10	7.57	7.82	70.90	62.45	62.52
G2	Neuron	24	95.34	86.97	86.08	10.10	7.58	7.73	71.26	70.43	70.68
G3	Neuron	24	95.29	88.73	88.50	10.18	10.17	10.51	79.96	84.55	84.22
G4	Neuron	23	94.69	87.78	87.41	10.13	9.28	9.93	76.62	78.15	78.40
G5	Neuron & Weights	8	93.72	87.65	86.94	10.11	9.15	9.23	82.72	73.02	73.29
G6	Neuron & Weights	17	95.22	86.35	85.79	10.15	7.70	8.23	67.68	66.60	66.59
G7	Neuron & Weights	7	93.65	88.38	87.94	10.14	11.17	11.24	76.49	78.18	77.92
G8	Neuron & Weights	25	95.97	89.42	88.87	10.17	9.00	9.28	77.17	73.78	74.41
L1	Neuron	29	94.73	87.05	86.30	10.20	9.32	9.20	63.53	61.12	61.67
L2	Neuron	31	93.67	86.93	86.54	10.10	8.73	8.88	74.43	75.52	76.16
L3	Neuron & Weights	5	92.39	84.57	84.25	10.13	10.10	10.39	87.58	85.90	85.86
L4	Neuron & Weights	20	93.95	86.67	85.99	10.16	11.82	11.56	74.90	79.88	79.78

Table 7. Ranks before and after retraining pruned networks.

Model	Pruning Type	Percentage of Pruning		Rank of Pruned Network				Rank after Retraining Pruned Network			
		Before Retraining	After Retraining	Layer 1	Layer 2	Layer 3	Layer 4	Layer 1	Layer 2	Layer 3	Layer 4
M	Magnitude	80	0.11	300	200	100	50	300	200	100	50
G1	Neuron	70.25	43.57	78	80	40	21	170	132	66	34
G2	Neuron	74.64	40.74	60	68	56	29	184	114	72	37
G3	Neuron	74.59	37.16	61	74	50	25	199	113	66	37
G4	Neuron	79.25	41.02	48	64	38	26	185	122	59	43
G5	Neuron & Weights	70.25	20.37	225	163	74	36	244	167	77	36
G6	Neuron & Weights	85	24.36	207	143	83	38	237	150	84	40
G7	Neuron & Weights	77.59	17.39	243	161	79	36	257	163	80	36
G8	Neuron & Weights	82.44	25.19	205	156	84	43	227	163	87	46
L1	Neuron	79.88	50.43	60	40	20	10	151	130	50	32
L2	Neuron	84.85	58.72	45	30	15	8	132	86	42	26
L3	Neuron & Weights	79.88	22.76	148	98	46	26	235	160	74	39
L4	Neuron & Weights	84.87	26.67	151	80	45	24	225	166	84	38

In Table 7, we demonstrate the pruning percentage before and after the retraining of pruned networks. We specify the rank of each hidden layer before and after retraining. The percentage of pruning of model M before training was 80%, which implies that 80% of the matrix weights were initialized to a 0 value. Hence, the matrix of weights for model M before training was sparse. The column Rank Before Training illustrates the rank of each hidden layer of model M to be full rank. Networks pruned using activation-based pruning and, subsequently, retrained produced weight matrices that were sparse and low rank. Model M recovered a training accuracy close to 95%, but utilized the complete network weights, resulting in the pruning percentage being 0.11%. This is illustrated under the heading Percentage of Pruning with sub-heading After Training. However, retraining networks pruned with activation-based pruning was able to achieve comparable training accuracy while maintaining a certain level of sparsity and low-rank approximation for each hidden layer. The networks that were pruned using activation-based pruning and, subsequently, retrained contained 20–50% of the weights with zero value. This demonstrated that, if we trained a network pruned using activation-based pruning, it will try to maintain sparsity and a low-rank structure for each hidden layer in the network. We observed that, after retraining, pruning based on Neurons maintained a larger percentage of sparsity and a lower rank for each hidden layer compared to pruning based on Neurons & Weights.

Figure 6 illustrates the heatmap representing the weight matrix of the first hidden layer of the network. The darker regions represent the non-zero weight, and lighter regions represent zero weight values, respectively. The figure contains two sets of heatmaps. The set on the left represents weight matrices for pruned networks, and the set on the right represents weight matrices for networks retrained after pruning. Figures A9–A11 in the Appendix B correspond to heatmaps for the remaining hidden layers of the network.

The figures on the left, titled Pruned network, demonstrate that magnitude-based pruning occurred uniformly across the entire matrix weights, while activation-based pruning occurred along the columns of the matrix. This is evident in the way the lighter region is scattered in the heatmap of the respective prunings. The figures on the right, titled Retrained after pruning, demonstrate that, after retraining the pruned network, activation-based pruning still contained lighter regions across the columns of the matrix, while magnitude-based pruning utilized almost all the weights of the matrix uniformly.

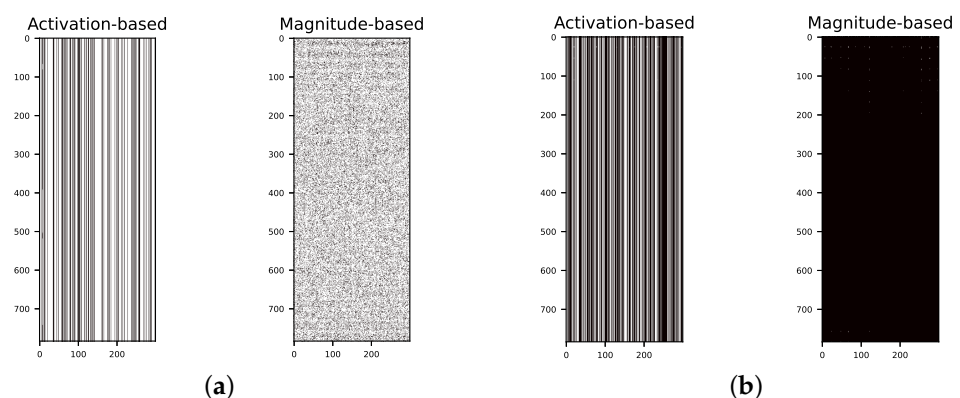


Figure 6. Heatmap for matrices of layer 1. (a) Pruned network. (b) Retrained after pruning.

Our empirical findings led us to hypothesize that activation-based pruning selectively targets weights that contribute minimally to the network’s training process. This pruning strategy resulted in the formation of sparse, low-rank matrix approximations, effectively reducing the full-rank matrices of a trained network. When such pruned networks underwent retraining, they tended to preserve the low-rank characteristics of these matrices, especially when previously pruned (zeroed) weights were allowed to be reutilized. In contrast, our observations with magnitude-based pruning indicated a different behavior: during the retraining phase, this method tended to engage nearly the entire spectrum of weights

within the network. This distinction highlighted the unique impact of activation-based pruning on the network's weight optimization during retraining.

5.4. Comparative Analysis

A broad range of metrics are utilized to assess pruning techniques, including accuracy [2,8], the compression rate [7,21], and parameter reduction [17,18]. Pruning evaluation results are also sensitive to the properties of the network architecture and dataset complexity. Moreover, many pruning algorithms contain hyperparameters, which influence the performance; hence, comparisons under fixed values cannot fully characterize the respective method potential. In response to variations in the evaluation metrics, we conducted two forms of comparative analysis—first, an in-depth ablation of activation-based pruning techniques and, second, under a fixed neural architecture and dataset. We provide assessments between our method and related the prior state-of-the-art, facilitating a fair contrast under similar experimental configurations.

5.4.1. Activation-Based Pruning Methods

We analyzed activation-based pruning methods including both global and local pruning, as well as the influence of importance scores assigned to Neurons and Neurons & Weights. A thorough discussion of these results is presented in Sections 5.1 and 5.2.

Tables 8 and 9 demonstrate the superior performance of local pruning in achieving significantly higher pruning percentages while simultaneously preserving test accuracy at an equivalent level compared to global pruning. The importance score computed for Neurons & Weights consistently yielded better results compared to considering only Neurons in both global and local pruning scenarios. The combination of the local pruning strategy with the importance score Neurons & Weights significantly outperformed other combinations.

Table 8. Global vs. local pruning.

Pruning Type	Pruning Iterations	Pruning %	Top 5% Test Accuracy
Global	5	75.03	71.49
Global	10	78.50	75.94
Local	4	82.37	74.90

Table 9. Importance-based scoring.

Importance Type	Pruning Type	Pruning %	Top 5% Test Accuracy
Neurons	Global	74.68	70.64
Neurons & Weights	Global	78.82	76.79
Neurons	Local	82.37	69.38
Neurons & Weights	Local	82.38	80.47

5.4.2. Activation-Based Pruning against Similar Models

In Table 10, we present a comparative analysis of different pruning methods that were tested under similar settings. All reference results used magnitude-based pruning on the LeNet-300-100 model using the MNIST dataset. The compression rate was calculated using the formula $W_{\text{total}}/W_{\text{rem}}$, where W_{total} is the total number of weights of the model before pruning and W_{rem} is the remaining number of weights after pruning. Sparsity % and Remaining Weights % are complements of each other. We present both of these data as some tests refer to Sparsity % and others to Remaining Weights %. Remaining Weights % and Required FLOPS % are quantitatively equal in value, but render different meaning to the numbers. We included both as different pruning methods presented either or both of these results.

Table 10. Comparative analysis of different models.

Paper	Compression Rate	Sparsity %	Remaining Weights%	Required FLOPS%
Activation-based	94×	98.94	1.06	1.06
Han et al. [2]	12×	92	8	8
Han et al. [7]	40×	92	8	8
Guo et al. [6]	56×	98.2	1.8	1.8
Fankle and Carbin [5]	7×	86.5	13.5	13.5
Molchanov et al. [17]	7×	86.03	13.97	13.97

Han et al. [2] presented an unstructured, iterative pruning method that uses magnitude-based pruning to achieve a 12-fold reduction in model size. Han et al. [7] employed an iterative pruning technique, where pruning occurred after training and achieved a 40-fold reduction in model size. Guo et al. [6] presented an iterative pruning method where pruning occurred before training and achieved a 56-fold reduction in model size. Frankle and Carbin [5] used the lottery ticket hypothesis in the context of pruning small fully connected nets on MNIST. It is a structured pruning technique with importance scoring based on the weight magnitude. It achieved a seven-fold reduction in model size. Molchanov et al. [17] presented an unstructured pruning, where pruning occurred during training. It achieved a compression rate of seven-fold. Activation-based pruning is an unstructured, iterative-based pruning method. The results demonstrated that activation-based pruning achieved an impressive 94-fold reduction in the model size.

5.5. Principal Component Analysis of Hidden Layers

Table 11 demonstrates the description of each model, the percentage of pruning achieved, and the ranks for the hidden layers of each model after using activation-based pruning. Compared to assigning Neurons & Weights as the importance score, Neurons achieved a substantial reduction in the rank for the matrix corresponding to each layer of the neural network. Table 12 demonstrates the test results of a fully trained model (S). Table 13 demonstrates the test accuracy of models whose ranks were reduced using PCA and activation-based pruning, respectively. We applied PCA on the hidden layers of the fully trained models and reduced the ranks to the corresponding ranks of the hidden layers. The ranks of the hidden layers are specified in Table 11 under the column Rank of each layer. In Table 13, column Low-rank approximation using PCA, demonstrated the test accuracy of models whose ranks were reduced using PCA, and the column Activation-based pruning, demonstrates the test accuracy of models pruned using activation-based pruning. Models subjected to activation-based pruning demonstrated better test accuracy compared to those whose hidden layers were rank-reduced through PCA.

Table 11. PCA analysis: model description and ranks of each layer.

Model	Pruning Type	Target Prune %	Pruning Achieved	Rank of Each Layer			
				Layer 1	Layer 2	Layer 3	Layer 4
S				300	200	100	50
G1	Neurons	80.00	70.25	78	80	40	21
G2	Neurons	85.00	74.64	60	68	56	29
G3	Neurons & Weights	80.00	70.25	225	163	74	36
G4	Neurons & Weights	85.00	85.00	207	143	83	38
G5	Neurons	80.00	74.59	61	74	50	25
G6	Neurons	85.00	79.25	48	64	38	26
G7	Neurons & Weights	80.00	77.59	243	161	79	36
G8	Neurons & Weights	85.00	82.44	205	156	84	43

Table 12. Test results of the fully trained model.

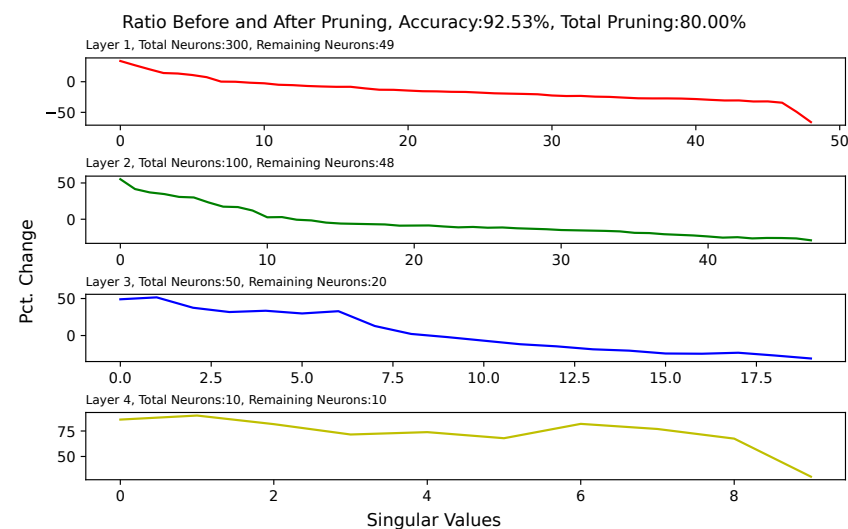
Model	Test Accuracy	Top-1% Accuracy	Top-5% Accuracy
S	84.35	8.22	69.40

Table 13. PCA analysis: test results

Model	Low-Rank Approximation Using PCA			Activation-Based Pruning		
	Test Accuracy	Top-1% Accuracy	Top 5% Accuracy	Test Accuracy	Top 1% Accuracy	Top 5% Accuracy
G1	81.79	8.85	63.73	87.90	10.16	65.69
G2	80.47	8.57	61.93	87.48	9.54	71.67
G3	84.75	8.24	68.75	87.12	9.66	81.83
G4	84.50	8.02	68.44	88.19	9.46	66.77
G5	78.01	9.24	65.08	87.83	9.57	80.24
G6	74.67	6.41	53.91	86.69	9.69	64.99
G7	84.52	8.26	69.05	88.07	12.30	79.23
G8	84.35	8.34	69.34	88.67	11.58	79.31

5.6. Analysis of Singular Value Changes in Each Layer

The experimental setup was similar to the setup outlined in Section 4.1. The final training accuracy was set at 98%; the total pruning to be achieved was set to 80%; the total number of pruning cycles was 10; the pruning was initiated when the network reached 70% training accuracy. At every stage of pruning, the singular values of the weight matrix were calculated before and after pruning. Figure 7 demonstrates the percentage variation in the singular values of the weight matrix, comparing values before the first pruning cycle to those after the completion of the final pruning cycle. Figures A1–A8 demonstrate the percentage variation in the singular values for each pruning cycle.

**Figure 7.** Ratio of singular values before and after pruning.

We observed that the singular values were reduced by a weighted amount. We also observed that, during the initial phase of pruning, the singular values with smaller magnitudes experienced a greater reduction in value than singular values with larger magnitudes. This corresponds to the notion that activation-based pruning results in removing the basis vectors that contain relatively little information about the data. As pruning progresses, the basis vectors that hold the least amount of information are removed successively. After a while, we are left with basis vectors that contain equally important amounts of information. At this stage, the rate of reduction in the magnitude of singular values shifts from a

weighted reduction to a uniform reduction. This is equivalent to stating that a reduction in the magnitude of the singular values shifts from a weighted nuclear norm minimization (WNNM) to nuclear norm minimization (NNM). Gu et al. [26] demonstrated that WNNM is better than NNM for low-rank matrix approximation problems. Zha et al. [27] demonstrated that NNM and WNNM are equivalent to L1-norm and weighted L1-norm minimization, respectively. Inspired by the superiority of weighted L1-norm minimization for sparse coding, they explained that WNNM is more effective than NNM for low-rank approximation. Their analysis provides a theoretical justification for WNNM's empirical effectiveness over NNM. To the best of our knowledge, our work is the first of its kind to analyze the effect of pruning on the singular values of hidden layers and provides empirical data to support the hypothesis that activation-based pruning results in a weighted nuclear norm minimization of the hidden layers. We hypothesize that, in a feedforward neural network using SGD as the optimization algorithm and ReLU as the activation function, activation-based pruning is equivalent to introducing the weighted nuclear norm as a regularization parameter to the original cost function.

6. Conclusions

We demonstrated the effectiveness of activation-based pruning in successfully reducing the size of a feedforward network. This pruning technique can be applied with either labeled or unlabeled data, as long as the data are drawn from the same distribution used to train the original feedforward network. Consequently, we hypothesized that activation-based pruning is adaptable to supervised, semi-supervised, or unsupervised learning algorithms. Furthermore, our results showed that each layer of the pruned network served as a sparse low-rank matrix representation of the fully trained original network. We provided empirical evidence supporting the hypothesis that activation-based pruning can be interpreted as introducing a regularization parameter of the weighted nuclear norm of the hidden layers. Additionally, considering the architectural and implementation characteristics of activation-based pruning, we proposed that this technique has the potential to be applied to various types of neural networks.

In our study, the focus was specifically on image classification tasks utilizing smaller datasets, namely FashionMNIST and MNIST, and employing compact network architectures, such as the fully connected feedforward model. This initial scope was chosen to facilitate a controlled analysis of the methods involved. Moving forward, we will aim to extend our investigation to encompass activation-based pruning applications on larger-scale models and more-extensive datasets. This planned expansion of our research is expected to provide a more-comprehensive understanding of the pruning method's efficacy across diverse neural network architectures.

The code can be found here: <https://github.com/tusharganguli/PrunedNetwork>, accessed on 18 December 2023. The code has the following dependencies: anaconda—2022.10, python—3.9.15, TensorFlow—2.10.0, pandas—1.5.1, matplotlib—3.5.3, seaborn—0.12.1, and openpyxl—3.0.10.

Author Contributions: Conceptualization, T.G.; methodology, T.G.; software, T.G.; validation, T.G.; formal analysis, T.G.; investigation, T.G.; resources, T.G.; data curation, T.G.; writing—original draft preparation, T.G.; writing—review and editing, E.K.P.C.; visualization, T.G.; supervision, E.K.P.C.; project administration, E.K.P.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. These data can be found here: https://www.tensorflow.org/datasets/catalog/fashion_mnist (accessed on 18 December 2023) and <https://www.tensorflow.org/datasets/catalog/mnist> (accessed on 18 December 2023).

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

Abbreviations

The following abbreviations are used in this manuscript:

SVD	singular-value decomposition
WNNM	weighted nuclear norm minimization
NNM	nuclear norm minimization
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
PCA	principal component analysis

Appendix A. Singular-Value Minimization

Figures A1–A8 show the relative percentage change in singular values before and after each pruning cycle.

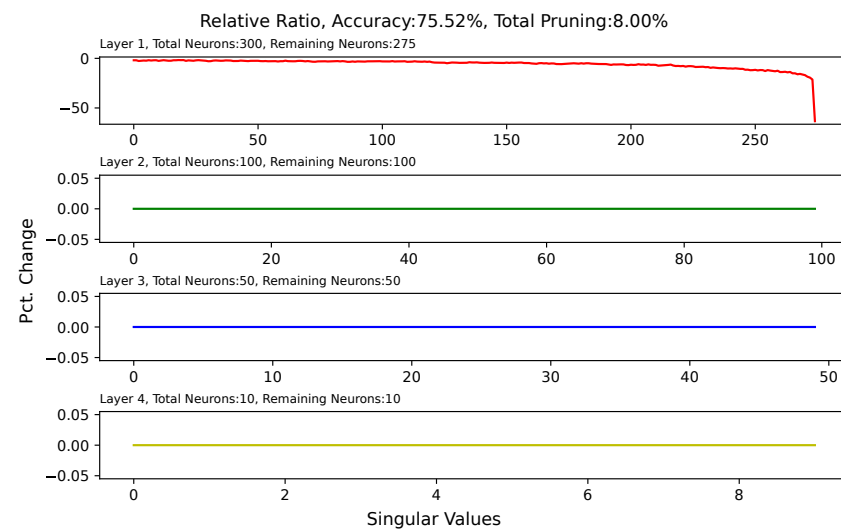


Figure A1. Relative ratio of singular values at 8% pruning.

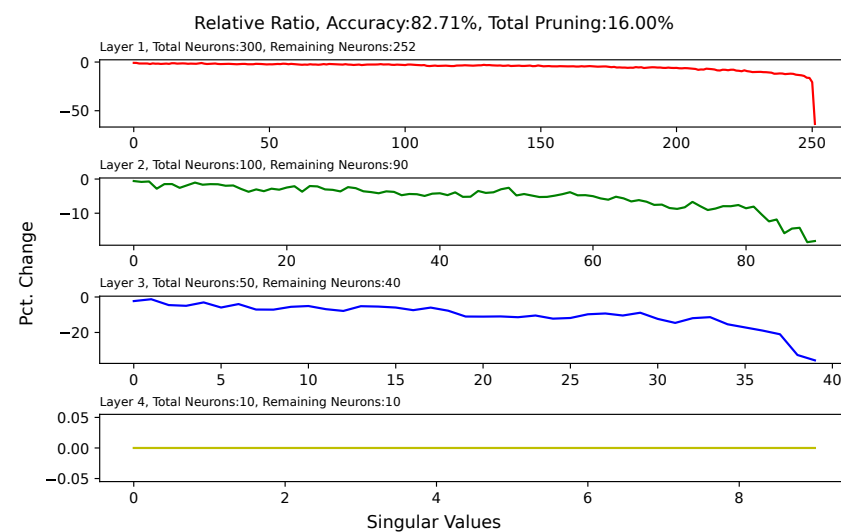


Figure A2. Relative ratio of singular values at 16% pruning.

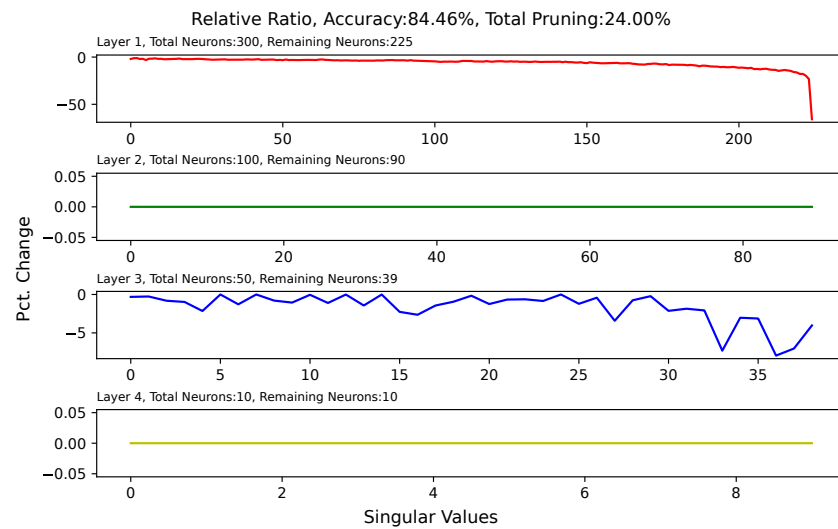


Figure A3. Relative ratio of singular values at 24% pruning.

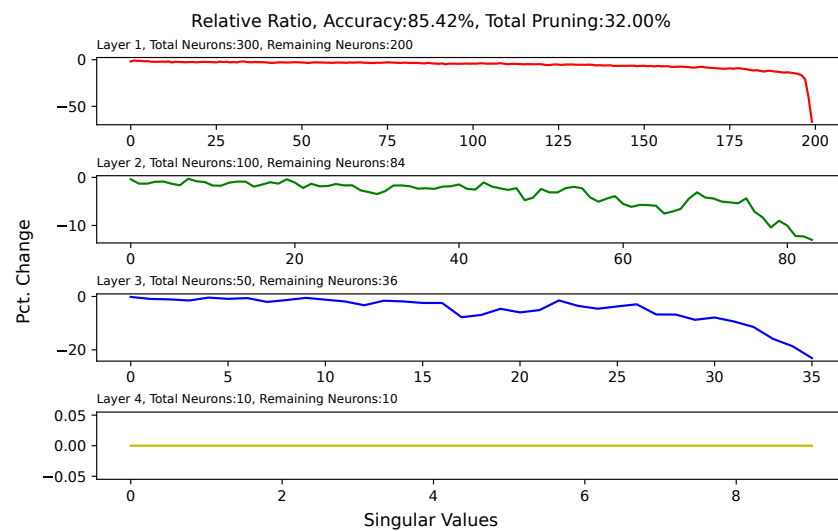


Figure A4. Relative ratio of singular values at 32% pruning.

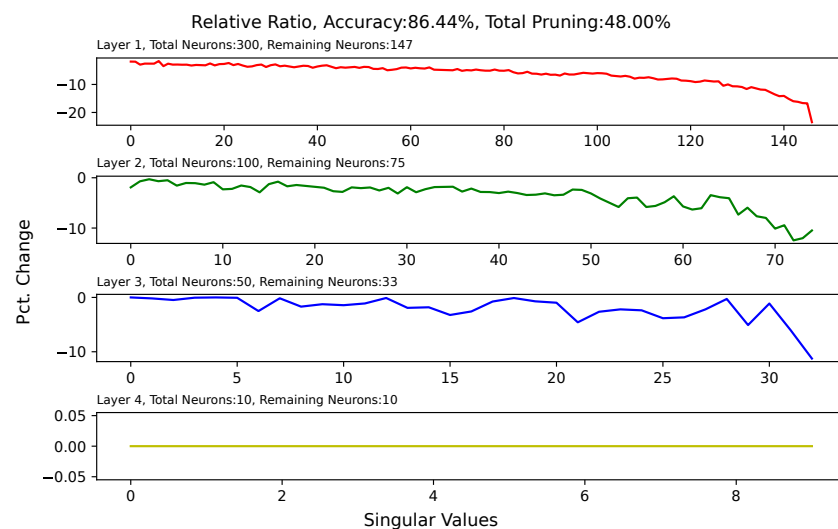


Figure A5. Relative ratio of singular values at 48% pruning.

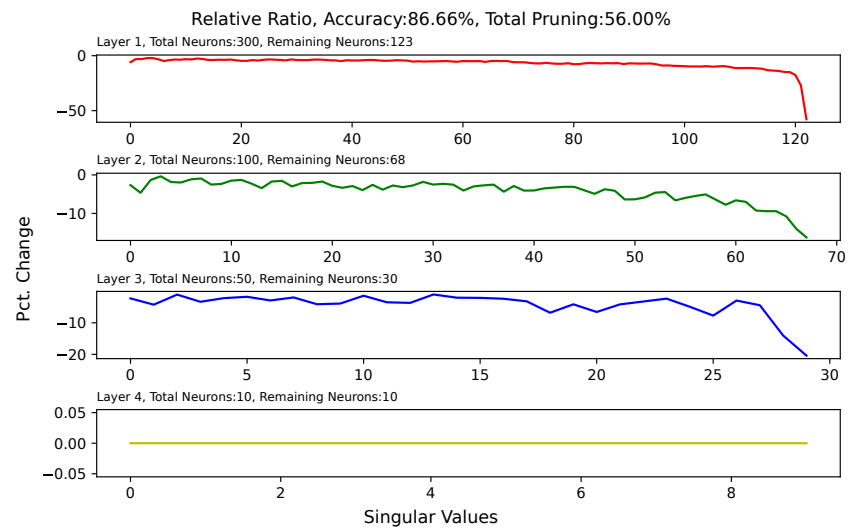


Figure A6. Relative ratio of singular values at 56% pruning.

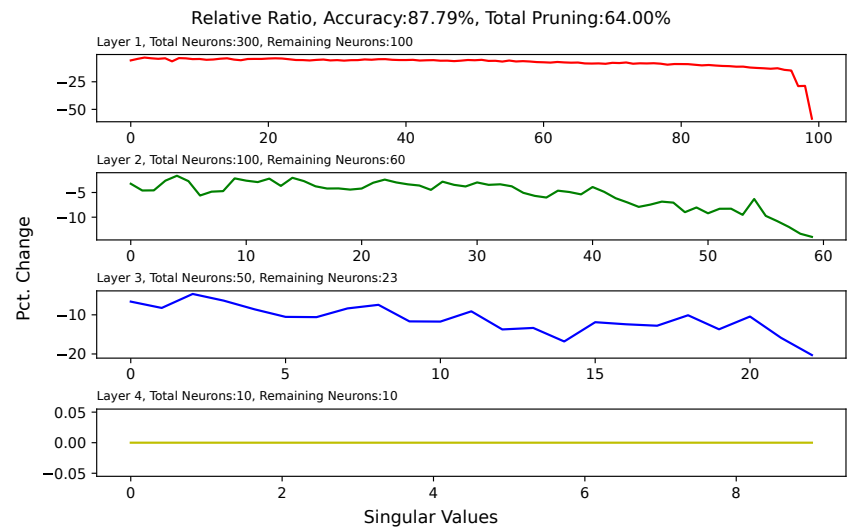


Figure A7. Relative ratio of singular values at 64% pruning.

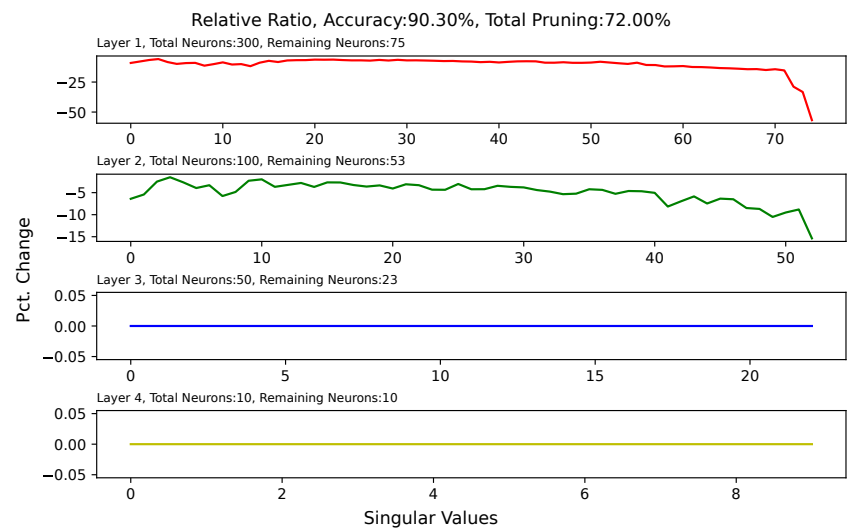


Figure A8. Relative ratio of singular values at 72% pruning.

Appendix B. Heat Map for Pruned and Retrained Network

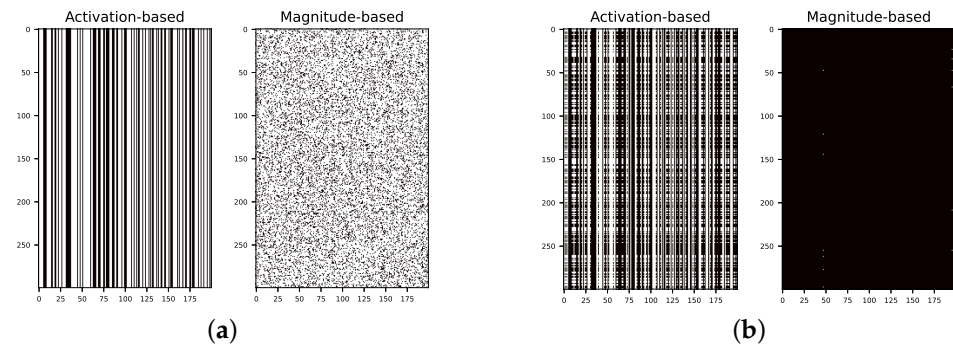


Figure A9. Heatmap for the matrices of layer 2. (a) Pruned network. (b) Retrained after pruning.

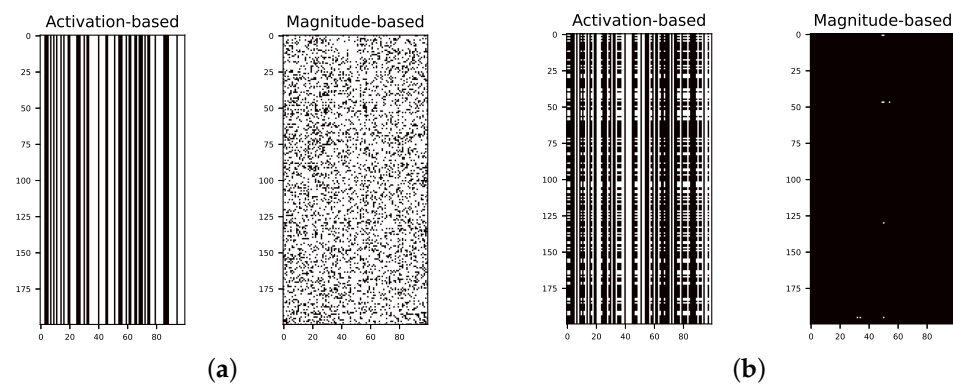


Figure A10. Heatmap for matrices of layer 3. (a) Pruned network. (b) Retrained after pruning.

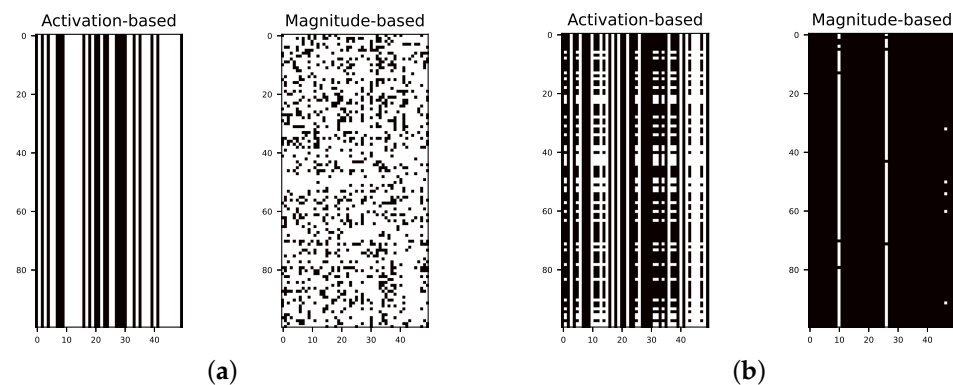


Figure A11. Heatmap for matrices of layer 4. (a) Pruned network. (b) Retrained after pruning.

References

1. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv* **2017**, arXiv:1611.06440.
2. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*; Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; Volume 28.
3. Hu, H.; Peng, R.; Tai, Y.W.; Tang, C.K. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv* **2016**, arXiv:1607.03250.
4. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning Filters for Efficient ConvNets. *arXiv* **2017**, arXiv:1608.08710.
5. Frankle, J.; Carbin, M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *Proceedings of the International Conference on Learning Representations*, New Orleans, LA, USA, 6–9 May 2019.
6. Guo, Y.; Yao, A.; Chen, Y. Dynamic Network Surgery for Efficient DNNs. In *Advances in Neural Information Processing Systems*; Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29.

7. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv* **2016**, arXiv:1510.00149.
8. Zhu, M.; Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. *arXiv* **2017**, arXiv:1710.01878.
9. Shlens, J. A Tutorial on Principal Component Analysis. *arXiv* **2014**, arXiv:1404.1100.
10. Swaminathan, S.; Garg, D.; Kannan, R.; Andres, F. Sparse low rank factorization for deep neural network compression. *Neurocomputing* **2020**, *398*, 185–196. <https://doi.org/10.1016/j.neucom.2020.02.035>.
11. Yang, H.; Tang, M.; Wen, W.; Yan, F.; Hu, D.; Li, A.; Li, H.; Chen, Y. Learning Low-Rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Virtually, 14–19 June 2020.
12. He, Y.; Zhang, X.; Sun, J. Channel Pruning for Accelerating Very Deep Neural Networks. In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 1398–1406. <https://doi.org/10.1109/ICCV.2017.155>.
13. Gray, S.; Radford, A.; Kingma, D.P. GPU Kernels for Block-Sparse Weights. *arXiv* **2017**, arXiv:1711.09224.
14. Kalchbrenner, N.; Elsen, E.; Simonyan, K.; Noury, S.; Casagrande, N.; Lockhart, E.; Stimberg, F.; van den Oord, A.; Dieleman, S.; Kavukcuoglu, K. Efficient Neural Audio Synthesis. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Dy, J., Krause, A., Eds.; Proceedings of Machine Learning Research; Volume 80, pp. 2410–2419.
15. Frankle, J.; Dziugaite, G.K.; Roy, D.M.; Carbin, M. Stabilizing the Lottery Ticket Hypothesis. *arXiv* **2020**, arXiv:1903.01611.
16. 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 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 9194–9203. <https://doi.org/10.1109/CVPR.2018.00958>.
17. Molchanov, D.; Ashukha, A.; Vetrov, D. Variational Dropout Sparsifies Deep Neural Networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; Precup, D., Teh, Y.W., Eds.; Proceedings of Machine Learning Research; Volume 70, pp. 2498–2507.
18. Mariet, Z.; Sra, S. Diversity Networks: Neural Network Compression Using Determinantal Point Processes. *arXiv* **2017**, arXiv:1511.05077.
19. Lee, N.; Ajanthan, T.; Torr, P. SNIP: Single-Shot Network Pruning Based on Connection Sensitivity. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
20. Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; Darrell, T. Rethinking the Value of Network Pruning. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
21. Yuan, X.; Savarese, P.; Maire, M. Growing Efficient Deep Networks by Structured Continuous Sparsification. *arXiv* **2020**, arXiv:2007.15353.
22. Wang, C.; Zhang, G.; Grosse, R. Picking Winning Tickets Before Training by Preserving Gradient Flow. *arXiv* **2020**, arXiv:2002.07376.
23. Tanaka, H.; Kunin, D.; Yamins, D.L.; Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 6377–6389.
24. Frantar, E.; Alistarh, D. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *arXiv* **2023**, arXiv:2301.00774.
25. Sun, M.; Liu, Z.; Bair, A.; Kolter, J.Z. A Simple and Effective Pruning Approach for Large Language Models. *arXiv* **2023**, arXiv:2306.11695.
26. Gu, S.; Zhang, L.; Zuo, W.; Feng, X. Weighted Nuclear Norm Minimization with Application to Image Denoising. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 2862–2869. <https://doi.org/10.1109/CVPR.2014.366>.
27. Zha, Z.; Wen, B.; Zhang, J.; Zhou, J.; Zhu, C. A Comparative Study for the Nuclear Norms Minimization Methods. In Proceedings of the 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 22–25 September 2019; pp. 2050–2054. <https://doi.org/10.1109/ICIP.2019.8803145>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.