Channel Pruning Based on Mean Gradient for Accelerating Convolutional Neural Networks

Congcong Liu, Huaming Wu*

Center for Applied Mathematics, Tianjin University, Tianjin 300072, P.R. China

Abstract

Convolutional Neural Networks (CNNs) are getting deeper and wider to improve their performance and thus increase their computational complexity. We apply channel pruning methods to accelerate CNNs and reduce its computational consumption. A new pruning criterion is proposed based on the mean gradient for convolutional kernels. To significantly reduce Float Point Operations (FLOPs) of CNNs, a hierarchical global pruning strategy is introduced. In each pruning step, the importance of convolutional kernels is evaluated by the mean gradient criterion. Hierarchical global pruning strategy is adopted to remove less important kernels, and get a smaller CNN model. Finally we fine-tune the model to restore network performance. Experimental results show that VGG-16 network pruned by channel pruning on CIFAR-10 achieves 5.64× reduction in FLOPs with less than 1% decrease in accuracy. Meanwhile ResNet-110 network pruned on CIFAR-10 achieves 2.48× reduction in FLOPs and parameters with only 0.08% decrease in accuracy.

Keywords: channel pruning, Convolutional Neural Networks, mean gradient, hierarchical global pruning, acceleration

1. Introduction

Convolution neural networks (CNNs) have achieved remarkable success in various recognition tasks [1][2][3], especially in computer vision [4][5][6]. CNNs have achieved state-of-the-art performance in these fields compared with traditional methods based on manually designed visual features [7]. However, these deep neural networks have a huge number of parameters. For example, AlexNet [4] network contains about 6×10^6 parameters, while a better performance network such as VGG

*Corresponding author
Email address: whming@tju.edu.cn (Huaming Wu)

[6] network contains about 1.44×10^8 parameters, which causes higher memory and computational costs. For instance, VGG-16 model takes up more than 500MB storage space and needs 1.56×10^{10} Float Point Operations (FLOPs) to classify a single image. The huge memory and high computational costs of CNNs restrict the application of deep learning on mobile devices with limited resources [8]. What's more, deep learning models are known to be over-parameterized [9]. Denil *et al. [10]* pointed out that deep neural networks can be reconstructed by a subset of network parameters without affecting network performance, which means that there are a huge number of redundant connections in neural network models and we can reduce the memory and computational costs by pruning and compressing such connections [11][12].

The huge memory consumption and high computational complexity of deep neural networks drive the research of compression [13][14] and acceleration algorithms [15][16], and pruning [17] is one of effective methods. In the 1990s, LeCun *et al.* [18] introduced the Optimal Brain Damage pruning strategy, they had observed that several unimportant weight connections could be safely removed from a well-trained network with negligible impact on network performance. Hassibi *et al.* [19] proposed a similar Optimal Brain Surgeon pruning strategy and pointed out that the importance of weight was determined by the second derivative. However, these two methods needed to calculate Hessian matrix, which increased the memory consumption and computational complexity of network model. Recently, Han *et al.* [20][21] reported impressive compression rates and effective decrease of the number of parameters on AlexNet network and VGG Network by pruning weight connections with small magnitudes and then retraining without hurting overall accuracy. The decrease of parameters was mainly concentrated in full connection layers, which achieved $3 \sim 4\times$ speedup in full connection layers during inference time. However, this pruning operation had generated an unstructured [22] sparse model, which additionally required sparse BLAS libraries [23] or even specialized hardware to achieve its acceleration [16]. Similar to our study, Li *et al.* [24] measured the relative importance of a convolution kernel in each layer by calculating the sum of its absolute weights, i.e., its l_1 norm. Compared to the minimum weight criterion[24], our criterion is based on mean gradient of feature maps in each layer, which more intuitively reflects the importance of feature extracted from convolutional kernels. Another pruning criterion obtained the sparsity of activations after a non-linear ReLU [25] mapping. Hu *et al.* [26] believed that if most outputs after these non-linear neurons are zero, the probability of neuronal redundancy should be bigger. This criterion measured importance score of a neuron by calculating its Average Percentage of Zeros

(APoZ). However, APoZ pruning criterion requires the introduction of threshold parameters, which will vary from layer to layer. These two criteria simply and intuitively reflect the importance of channels for convolutional kernels or feature maps, but do not directly consider the final loss after pruning. In this paper, pruning algorithm was based on the importance of feature maps in each channel, and considered the effect on network performance after pruning a channel. Meanwhile hierarchical global pruning strategy and FLOPs constraint were introduced to significantly reduce the network FLOPs.

Firstly, channel pruning for CNNs with different structures will be achieved in Section 2. Secondly pruning criterion based on the mean gradient and hierarchical global pruning strategy will be proposed in Section 3. Effectiveness of the algorithm will be presented by experimental comparisons in Section 4. Finally, the paper will be concluded in Section 5.

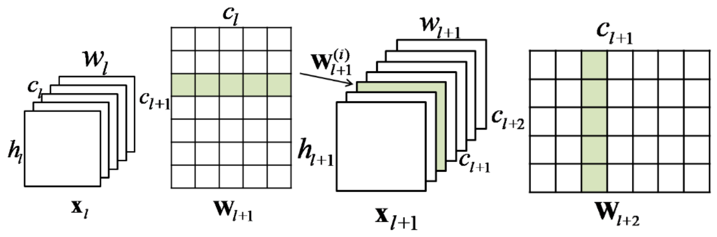
2. Pruning channels and corresponding feature maps

The paper mainly studies the effect of channel pruning on reducing network FLOPs. Convolutional layers accounts for more than 90% [27] FLOPs of common CNNs. Therefore, we only prune convolutional layers, Section 2.1 and 2.2 implement specific pruning on channels and their corresponding feature maps for different networks, respectively.

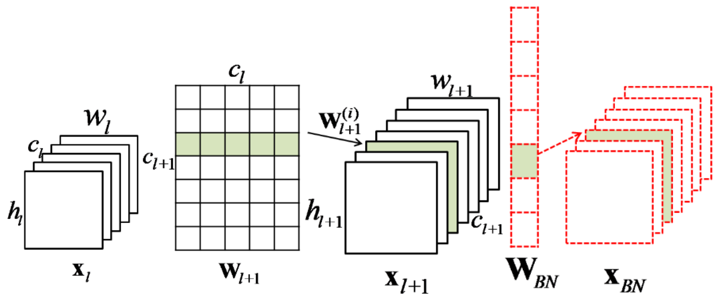
2.1. Channel pruning for networks without shortcut connections

For a CNN structure without shortcut connections [6], such as VGG network or AlexNet network, the process of channel pruning is shown in Fig.1. Input feature maps $\mathbf{x}_l \in \mathbb{R}^{c_l \times h_l \times w_l}$ are transformed to output feature maps $\mathbf{x}_{l+1} \in \mathbb{R}^{c_{l+1} \times h_{l+1} \times w_{l+1}}$ by convolution operation, where $\mathbf{x}_{l+1}$ are used as input feature maps for next convolutional layer, c_l and c_{l+1} represent the number of channels for feature maps $\mathbf{x}_l$ and $\mathbf{x}_{l+1}$, respectively, h_l and w_l represent height and width of feature maps $\mathbf{x}_l$, respectively. Convolution operation is implemented by convolution kernel matrix $\mathbf{W}_{l+1} \in \mathbb{R}^{c_{l+1} \times c_l \times k \times k}$, where k is size of convolutional kernel, such as $k = 3$ for VGG network. $\mathbf{W}_{l+1}$ is composed by c_{l+1} 3D kernel $\mathbf{W}_{l+1}^{(i)} \in \mathbb{R}^{c_l \times k \times k}$, $i \in (1, 2, \cdots, c_{l+1})$, each kernel $\mathbf{W}_{l+1}^{(i)}$ is convolved with input feature maps $\mathbf{x}_l$ to generate one output feature map. Therefore, when the ith output channel for the convolutional kernel matrix $\mathbf{W}_{l+1}$ is removed, the ith output feature map (marked as green in the output feature maps $\mathbf{x}_{l+1}$ in Fig.1(a)) will be removed, and the ith input channel for the next convolutional kernel $\mathbf{W}_{l+2}$ (marked as green in $\mathbf{W}_{l+2}$ in Fig.1(a)) will also be removed.

Furthermore, there are batch normalization (BN) layers in CNNs shown in Fig.1(b), similar to the process of pruning between convolutional layers, when the ith output channel for convolutional kernel matrix $\mathbf{W}_{l+1}$ is removed, the ith channel for the subsequent BN layer and its corresponding output feature map (marked as green in $\mathbf{W}_{BN}$ and $\mathbf{x}_{BN}$) will be removed.



(a) Channel pruning between convolutional layers [24]



(b) Channel pruning with batch normalization layers

Figure 1: Channel pruning for networks without shortcut connections. (a) Channel pruning between convolutional layers. Pruning a channel and its corresponding feature maps between convolutional layers. (b) Channel pruning with batch normalization layers. Pruning a channel and its corresponding batch normalization layer.

Channel pruning for non-tensor BN layers cannot reduce the computational complexity of CNNs, the reduction in network FLOPs by channel pruning only needs to consider the change of FOLPs for convolutional layers. Tab.1 shows the change of FLOPs on the $l+1$th and $l+2$th convolutional layers when m output channels are removed for the $l+1$th convolutional kernel matrix. It can be observed that FLOPs on the $l+1$th and $l+2$th convolutional layers are both reduced by m/c_{l+1}.

Table 1: The change of FLOPs on convolutional layers.

convolutional layer	FLOPs	FLOPs pruned	rate
$l+1$	$c_{l+1}c_l k^2 h_l w_l$	$m c_l k^2 h_l w_l$	m/c_{l+1}
$l+2$	$c_{l+2}c_{l+1} k^2 h_{l+1} w_{l+1}$	$c_{l+2} m k^2 h_{l+1} w_{l+1}$	m/c_{l+1}

2.2. Channel pruning for residual networks

The architectures of Residual Networks (ResNets) [6] are more complex than plain CNNs[6]. Shortcut connections are inserted in the ResNets, which makes channel pruning more complicated. When channels are pruned in a residual block, we should consider whether the corresponding channels need to be pruned in down-sample layers or not. Fig.2 illustrates the channel pruning process for ResNets.

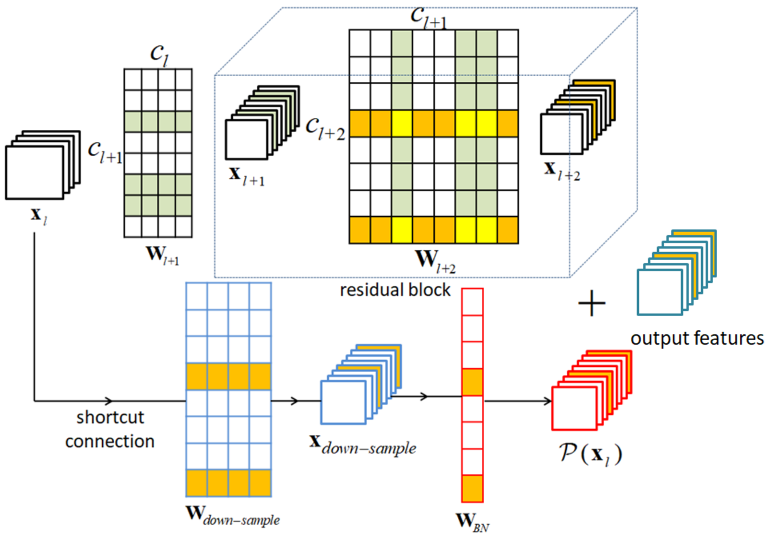


Figure 2: Channel pruning for ResNets [24]. The channels to be pruned for down-sample layer (marked as orange in $\mathbf{W}_{down-sample}$) are determined by the pruned channels of the last convolutional layer of residual block.

It can be seen that channel pruning inside a residual block is the same as channel pruning for networks in Section 2.1. Since the last feature maps $\mathbf{x}_{l+2}$ in residual block and feature maps generated by shortcut connection are of the same dimension, we should remove same output channels of convolution kernel matrix $\mathbf{W}_{l+2}$ and $\mathbf{W}_{down-sample}$ simultaneously. The channels to be pruned for down-sample layer(marked as orange in shortcut connection) are determined by the corresponding channels pruned for the last convolutional layer of residual block, which ensures that the dimension of feature maps $\mathcal{P}(\mathbf{x}_l)$ is consistent with the dimension of feature maps $\mathbf{x}_{l+2}$.

FLOPs for ResNets is the same as that of corresponding plain networks. Shortcut connections introduce neither extra parameter nor computation complexity. Therefore reduction in FLOPs for ResNets by channel pruning is the same as Tab.1. When m output channels are removed for the $l+1$th convolutional layer in ResNets, FLOPs on the convolutional layer and the next convolutional layer are both reduced by m/c_{l+1}.

3. Channel pruning strategy

The proposed strategy for channel pruning consists of the following steps: (1) Given a pre-trained network model; (2) Evaluating the importance of feature map on each channel by mean gradient criterion; (3) Adopting a hierarchical global pruning strategy to prune less important channels and corresponding feature maps; (4) Alternate iterations of pruning and further fine-tuning; (5) Stopping pruning until the desired pruning target is achieved. The flow chart is depicted in Fig.3. Our desired pruning target is to reduce FLOPs for network models as much as possible without compromising original accuracy.

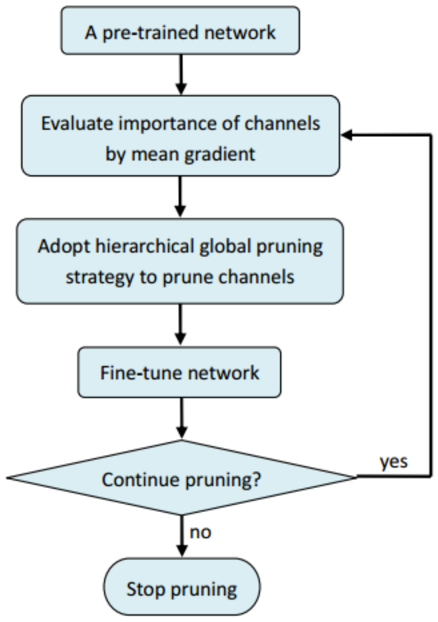

A pre-trained network
Evaluate importance of channels by mean gradient
Adopt hierarchical global pruning strategy to prune channels
Fine-tune network
Continue pruning?
yes
no
Stop pruning

Figure 3: The flow chart of channel pruning

3.1. Mean gradient criterion for pruning

Channel pruning for convolutional layers reduces the number of network parameters, which inevitably leads to a decrease in network performance, therefore the choice of channels to be pruned is especially important. As shown in Fig.4, pruning channels for the lth convolutional layer reduces the number of output channels from c_l to desired number $\tilde{c}_l$, where $0 < \tilde{c}_l \leqslant c_l$, and the lth convolutional kernel matrix $\mathbf{W}_l$ is transformed to $\tilde{\mathbf{W}}_l$. For the $l+1$th convolutional layer, the number of input channels is also reduced from c_l to $\tilde{c}_l$, and the $l+1$th convolutional kernel matrix $\mathbf{W}_{l+1}$ is transformed to $\tilde{\mathbf{W}}_{l+1}$. The set of network parameters is denoted by

$$\mathcal{W} = \Big\{\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_l, \mathbf{W}_{l+1}, \cdots, \mathbf{W}_L\Big\},$$

where L represents the depth of convolutional layers in the network. For simplicity, bias terms are ignored. Fig.4 illustrates that channel pruning will lead to changes in the set of network parameters, which is transformed to

$$\tilde{\mathcal{W}} = \Big\{\mathbf{W}_1, \mathbf{W}_2, \cdots, \tilde{\mathbf{W}}_l, \tilde{\mathbf{W}}_{l+1}, \cdots, \mathbf{W}_L\Big\}.$$

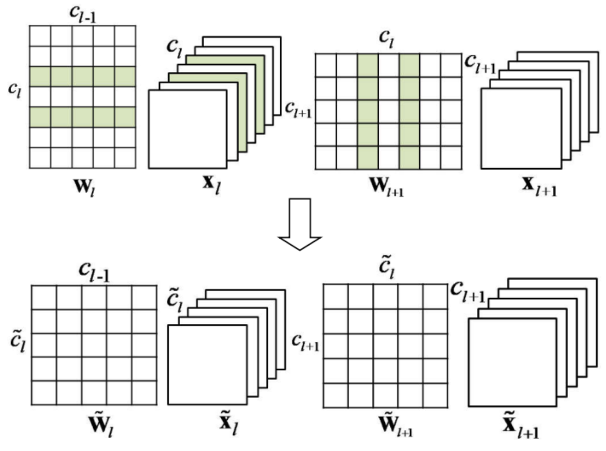


Figure 4: Transformation for network parameters. Output channels for lth convolutional layer are pruned (marked as green in $\mathbf{W}_l$), its corresponding feature maps and input channels for the next convolutional layer are also pruned (marked as green in $\mathbf{x}_l$ and $\mathbf{W}_{l+1}$).

Consider the set of network parameters converted from $\mathcal{W}$ to $\tilde{\mathcal{W}}$, which is optimized to minimize the loss function $\mathcal{C}(\,\cdot\,)$ of the network. A set of training examples is represented as

$$\mathcal{D} = \Big\{\mathcal{X} = \{x_0, x_1, \cdots\cdots, x_N\}, \mathcal{Y} = \{y_0, y_1, \cdots\cdots, y_N\}\Big\},$$

where $\mathcal{X}$ and $\mathcal{Y}$ represent an input and a target output of the neural network, respectively; N is the number of training examples. A good channel pruning method should maintain network performance when the set of network parameters changes, i.e.,

$$\mathcal{C}(\mathcal{D}|\tilde{\mathcal{W}}) \approx \mathcal{C}(\mathcal{D}|\mathcal{W}). \tag{1}$$

According to Eq.(1), the problem of channels selection can be converted into a combinatorial optimization:

$$\min_{\mathcal{W}'} \Big|\mathcal{C}(\mathcal{D}|\tilde{\mathcal{W}}) - \mathcal{C}(\mathcal{D}|\mathcal{W})\Big|, \quad s.t. \quad \begin{cases} \big\|\tilde{\mathcal{W}}_l\big\|_0 = \tilde{c}_l \\ \big\|\tilde{\mathcal{W}}_j\big\|_0 = c_j, \quad j \neq l \end{cases} \tag{2}$$

where the l_0 norm in $\big\|\tilde{\mathcal{W}}_l\big\|_0$ represents the number of non-zero parameters for the lth element $\tilde{\mathbf{W}}_l$ in $\tilde{\mathcal{W}}$, which means that the number of output channels for convolutional kernel matrix $\tilde{\mathbf{W}}_l$ changes to $\tilde{c}_l$, and the number of output channels of other convolution kernel matrix remains unchanged.

A new pruning criterion based on mean gradient of feature maps is introduced, this criterion prunes a particular channel that has an almost flat gradient of loss function $\mathcal{C}(\,\cdot\,)$ with respect to feature maps $\mathbf{x}_l$, i.e. it prunes the channel and its corresponding feature map with the minimum mean gradient. Let the lth feature maps be denoted as:

$$\mathbf{x}_l = \Big\{\mathbf{x}_l^{(1)}, \mathbf{x}_l^{(2)}, \cdots\cdots, \mathbf{x}_l^{(c_l)}\Big\},$$

mean gradient Θ_{MG} of the kth channel for feature maps $\mathbf{x}_l$ is calculated as follows:

$$\Theta_{MG}\big(\mathbf{x}_l^{(k)}\big) = \left|\frac{1}{M}\sum_m \frac{\partial \mathcal{C}}{\partial \mathbf{x}_{l,m}^{(k)}}\right|, \tag{3}$$

where M is the length of vectorized feature map, $\mathbf{x}_{l,m}^{(k)}$ denotes any element of $\mathbf{x}_l^{(k)}$. The gradient terms in Eq.(3) are easily computed from the same computations for back-propagation, Θ_{MG} gives an expectation of the change magnitude on loss function with respect to output feature map on each channel. For training examples $N > 1$, Θ_{MG} is computed for each example separately, and then the final result is obtained by averaging all examples.

3.2. Hierarchical global pruning strategy

Combining with the mean gradient criterion, this section proposes hierarchical global pruning strategy: according to each convolutional layer's sensitivity to pruning, the global pruning strategy is adopted between the layers with similar sensitivity. For networks without shortcut connection, such as VGG-16 network on CIFAR-10 dataset [28], network architecture and various parameters information are shown in Tab.2. And the pruned model is named as VGG16-pruned-A, which is described in Tab.3.

Table 2: VGG-16 and the pruned model on CIFAR-10.

Layer	$h_l \times w_l$	Channels	FLOPs	FLOPs/channels	channels	pruned%
Conv1	224×224	64	8.67×10^7	1.35×10^6	5	92.2%
Conv2	224×224	64	1.85×10^9	2.89×10^7	6	90.6%
Conv3	112×112	128	9.25×10^8	7.23×10^6	7	94.5%
Conv4	112×112	128	1.85×10^9	1.45×10^7	2	98.4%
Conv5	56×56	256	9.25×10^8	3.61×10^6	72	71.9%
Conv6	56×56	256	1.85×10^9	7.23×10^6	68	73.4%
Conv7	56×56	256	1.85×10^9	7.23×10^6	61	76.2%
Conv8	28×28	512	9.25×10^8	1.81×10^6	328	35.9%
Conv9	28×28	512	1.85×10^9	3.61×10^6	348	32.0%
Conv10	28×28	512	1.85×10^9	3.61×10^6	345	32.6%
Conv11	14×14	512	4.62×10^8	9.03×10^5	329	35.7%
Conv12	14×14	512	4.62×10^8	9.03×10^5	335	34.6%
Conv13	14×14	512	4.62×10^8	9.03×10^5	318	37.9%
Linear1	1	4096	1.03×10^8	2.50×10^4	4096	0%
Linear2	1	4096	1.68×10^7	4.10×10^3	4096	0%
Linear3	1	10	4.60×10^4	4.10×10^3	10	0%
Total	—	—	1.55×10^{10}	—	—	48%

The CIFAR-10 dataset consists of 60, 000 images, whose size is 32×32, and the number of images in each category is 6, 000, with 10 categories. During training, images are converted to 256×256 and then randomly cropped to 224×224 for network input, flip horizontal is applied to implement data augmentation. During testing, images are converted to 256×256 and then scaled to 224×224 using a center crop for network input. Since layers with the same size of feature maps have similar sensitivities to pruning [24], global pruning strategy is applied to the layers with similar sensitivities. For VGG-16 network, the strategy is applied to the first four convolutional layers: Conv1, Conv2, Conv3, Conv4, the three middle convolutional layers: Conv5, Conv6, Conv7, and the last six convolutional layers: Conv8, Conv9, Conv10, Conv11, Conv12, Conv13. Pruning ratio for each hierarchy is determined by the total number of output channels for convolutional layers.

Compared with layer-by-layer pruning and retraining, the pruning ratio of each convolutional layer does not need to be determined in advance according to the hierarchical global pruning s-trategy. Channel and its corresponding feature map with small Θ_{MG} are removed in Eq.(3), the sensitivity of convolutional layers is studied actively during each pruning process, and a reasonable ratio for each convolutional layer can be reached. What's more, pruning on layer-by-layer is extremely time-consuming and does not give a holistic view of network robustness resulting in a smaller network model. On the other hand, compared with global pruning strategy, our strategy makes pruning ratio for each convolutional layer closer to the ratio for the whole network, and it is impossible to obtain a significant reduction in network FLOPs in the case that there is a large difference in pruning ratio for convolutional layers.

FLOPs constraint on CNNs is introduced to further reduce computational complexity of networks. FLOPs for different layers require different amounts of computation due to the channles and sizes of input feature maps and convolution kernels. The pruning ratio of each hierarchy is determined by the FLOPs ratio of all convolutional layers within the hierarchy to the whole network. The pruning ratio varies with the number of channels for convolutional layers during each pruning.

3.3. Fine-tuning of the pruned networks

After each pruning, performance degradation would be compensated by fine-tuning the networks. *Pruning and fine-tuning iteratively* strategy is adopted, channel pruning is firstly performed, then the network is fine-tuned to enhance performance, and a smaller network model is obtained by updating parameters before next pruning. In addition, the mean gradient Θ_{MG} of each channel is standardized to get $\hat{\Theta}_{MG}$ during pruning. Its formula is shown as follows:

$$\hat{\Theta}_{MG}\left(\mathrm{x}_l^{(k)}\right) = \frac{\Theta_{MG}\left(\mathrm{x}_l^{(k)}\right)}{\sqrt{\sum_j \left(\Theta_{MG}\left(\mathrm{x}_l^{(j)}\right)\right)^2}}, \tag{4}$$

l_2-normalization can avoid the influence of the depth of convolutional layers on Θ_{MG}, which ensures the reasonability of sorting mean gradient across multiple layers. In conclusion, our channel pruning algorithm is described in **Algorithm 1**.

Algorithm 1 Channel pruning algorithm based on mean gradient

```
Input: A pre-trained Model: $M$
       Given the number of channels for each pruning: $n$
       Given a desired pruning ratio: $R$
       Training set: $\mathcal{D}$
       Testing set: $T$
Output: A pruned model: $M_{pruned}$
 1: Compute performance $P_M$ of $M$ on $T$
 2: Calculate the number of pruning iterations $I$ using $R$ and $n$
 3: for $i$ in range($I$) do
 4:     Obtain the number of channels $n_h$ for each hierarchy in each iteration according to the
        corresponding constraint condition
 5:     Calculate mean gradient $\Theta_{MG}$ of each channel using Eq.(3)
 6:     Apply $l_2$-normalization that rescale $\Theta_{MG}$ to $\hat{\Theta}_{MG}$ by Eq.(4)
 7:     Adopt global pruning strategy in each hierarchy
 8:     Update the set of network parameters to get new model $M_{new}$, update $M$ by $M_{new}$
 9:     Fine-tune $M$ on $\mathcal{D}$ using $SGD$ algorithm
10: end for
11: Fine-tune the network until the model converges and $M$ is saved as $M_{pruned}$
```

4. Experiments

To verify the validity of our algorithm, the following experiments are conducted. Effect of removing channels with different order for mean gradient on network accuracy is considered in

Section 4.1, which indicates that channels with larger mean gradient are more important in network performance. The comparison results of our strategy and global pruning strategy are shown in Section 4.2. Comparisons of different pruning criteria are given in Section 4.3, which shows that our algorithm can reduce FLOPs for networks effectively. Finally, the pruned models for VGG-16 and ResNet-110 are given in Section 4.4 and 4.5 respectively.

4.1. Effect of mean gradient on network performance

We compare the sensitivity of pruning channels for VGG-16 on CIFAR-10 with minimum mean gradient, maximum mean gradient and random channels. We show reduction in accuracy of several convolution layers, there are big differences between the three methods. This section mainly considers the effect of mean gradient on network performance, and thus, there is no fine-tuning using *SGD* algorithm.

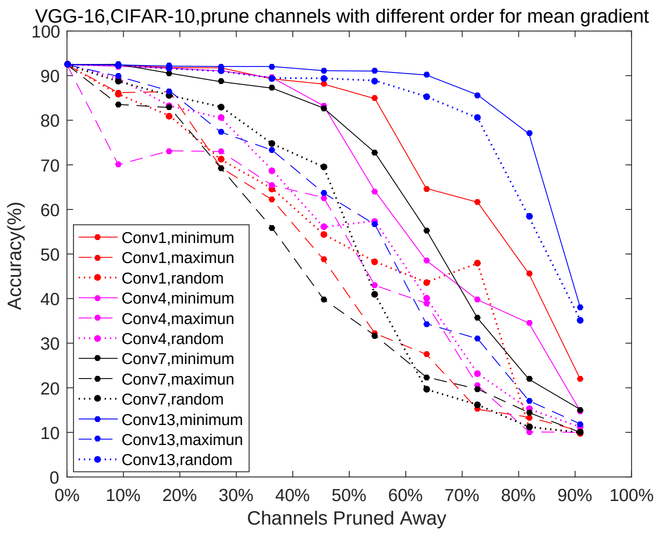


Figure 5: Comparison of three pruning methods for VGG-16 on CIFAR-10. The importance of channels is evaluated by its mean gradient. And pruning channels randomly is compared with our pruning criterion.

As shown in Fig.5, the accuracy of pruning channels with the maximum mean gradient drops quickly as pruning ratio increases, which indicates the importance of channels with larger mean gradient. What's more, from the comparison between pruning with minimum mean gradient and random channels, we can see that accuracy of pruned network with minimum gradient maintains better when pruning ratio of convolutional layers is less than 40%, while accuracy of pruning network randomly drops quickly when pruning ratio is low to 20%.

4.2. Comparison with global pruning strategy

According to **Algorithm 1**, the parameters of VGG-16 on CIFAR-10 are set as follows: we remove 100 channels at each pruning iteration, i.e. $n = 100$, subsequently, we perform 5 epochs SGD updates with batch-size 32, momentum 0.9, learning rate 10^{-4}, and weight decay 10^{-4}. Fig.6 shows the curve of model FOLPs changing with different pruning ratios, while our strategy and global pruning strategy are applied for VGG-16 on CIFRA-10 respectively, and neither of these strategies introduce FLOPs constraint condition. It can be seen that reduction in FLOPs is more obvious with hierarchical global pruning strategy.

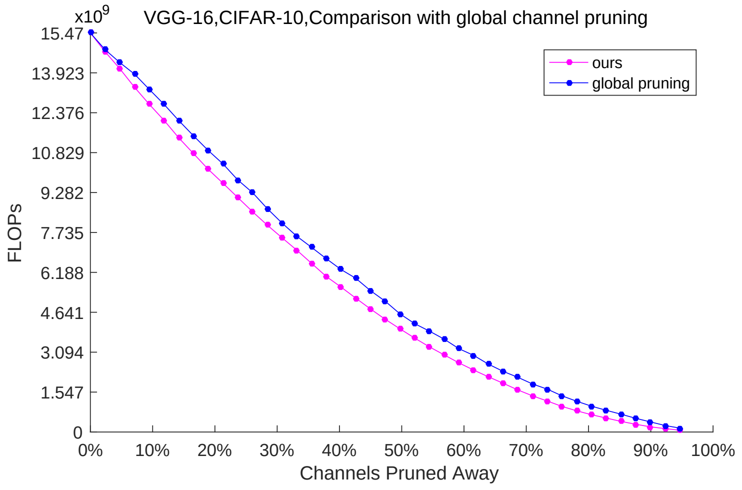
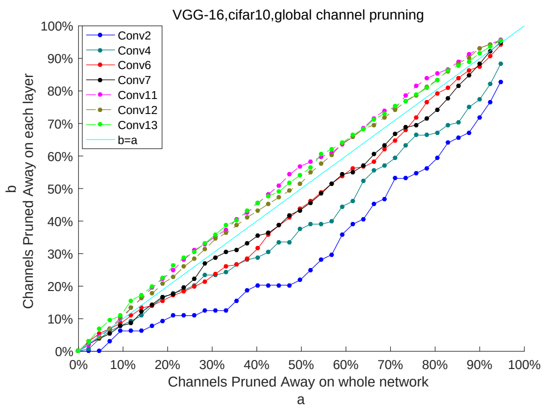
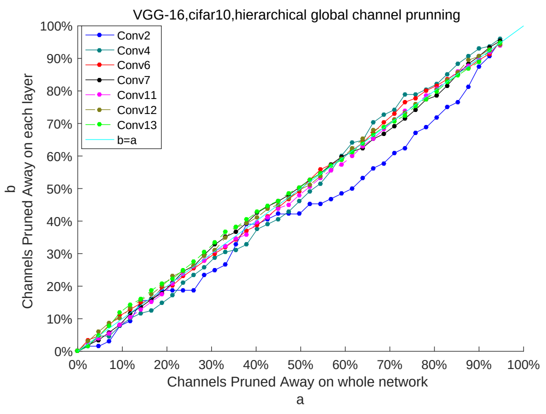


Figure 6: Comparison with global pruning strategy.

To further analyze the difference of reduction in network FLOPs with above two strategies, We compare pruning ratios of convolutional layers with large difference in value $FLOPs/Channel$, which is described in Tab.2, e.g. convolutional layers with large $FLOPs/Channel$: Conv2, Conv4, Conv6, Conv7; and convolutional layers with small $FLOPs/Channel$: Conv11, Conv12, Conv13.



(a)



(b)

Figure 7: Pruning ratio curves on convolutional layers. (a) Global pruning strategy. (b) Hierarchical global pruning strategy.

As shown in Fig.7, horizontal axis a represents pruning ratio on whole network and vertical axis b represents pruning ratio on each layer, Fig.7(a) illustrates that the relationship between the horizontal and vertical axis for convolutional layers with small $FLOPs/Channel$ (Conv11, Conv12, Conv13) is $b > a$, which means pruning ratios of these layers are all bigger than the ratio of whole network, in contrast, pruning ratio of convolutional layers with large $FLOPs/Channel$ (Conv2, Conv4, Conv6, Conv7) is less than the ratio of whole network. That is main reason that global pruning strategy cannot implement a significant reduction in network FLOPs. In Fig.7(b), the curves are all concentrated near the diagonal line $b = a$, which means that pruning ratio of each layer is similar to the ratio of the whole network.

Fig.7 further explains the experimental result in Fig.6 that reduction in FLOPs is more obvious with the hierarchical global pruning strategy when the pruning ratio of the whole network is equal to these two strategies. The hierarchical global pruning strategy can maintain a similar pruning ratio on each convolutional layer and achieve a more significant reduction in network FLOPs.

4.3. Comparison with other pruning criteria

There are many pruning criteria which evaluate the importance of a feature map or convolutional kernel. Common criteria include:

1. **Minimum weight [24]**:

$$\Theta_{MW}\big(\mathbf{W}_l^{(k)}\big) = \frac{1}{N}\sum\nolimits_j \big|\mathbf{W}_{l,j}^{(k)}\big|, \tag{5}$$

   where N is the dimensionality of convolutional kernel $\mathbf{W}_l^{(k)}$ after vectorization, and $\mathbf{W}_{l,j}^{(k)}$ denotes any element of $\mathbf{W}_l^{(k)}$.

2. **Mean Activation [29]**:

$$\Theta_{MA}\big(\mathbf{x}_l^{(k)}\big) = \frac{1}{M}\sum\nolimits_m \mathbf{x}_{l,m}^{(k)}, \tag{6}$$

   where M is the length of vectorized feature map $\mathbf{x}_l^{(k)}$.

3. **Std Activation [29]**:

$$\Theta_{std}\big(\mathbf{x}_l^{(k)}\big) = \sqrt{\frac{1}{M}\sum\nolimits_m \big(\mathbf{x}_{l,m}^{(k)} - \mu_{\mathbf{x}_l^{(k)}}\big)^2}, \tag{7}$$

   where $\mu_{\mathbf{x}_l^{(k)}}$ is the mean of feature map $\mathbf{x}_l^{(k)}$. This criterion is similar to the Mean Activation criterion, calculating feature maps generated after convolution operations.

4. **APoZ [26]**:

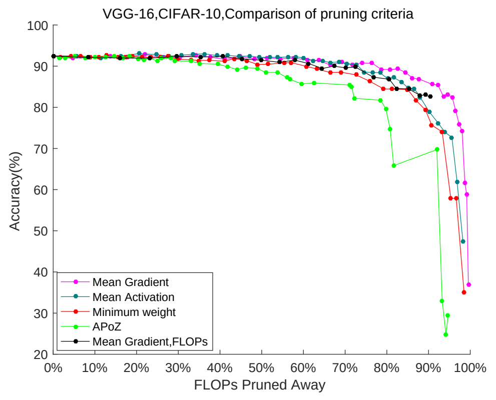
$$APoZ(\mathbf{o}_l^{(k)}) = \frac{1}{M}\sum\nolimits_m f(\mathbf{o}_{l,m}^{(k)} = 0), \tag{8}$$

   where $\mathbf{o}_l$ is the lth layer activation neurons produced by nonlinear ReLU neurons, $\mathbf{o}_{l,m}^{(k)}$ denotes any element of $\mathbf{o}_l^{(k)}$, $f(\cdot)$ is indicator function, i.e.

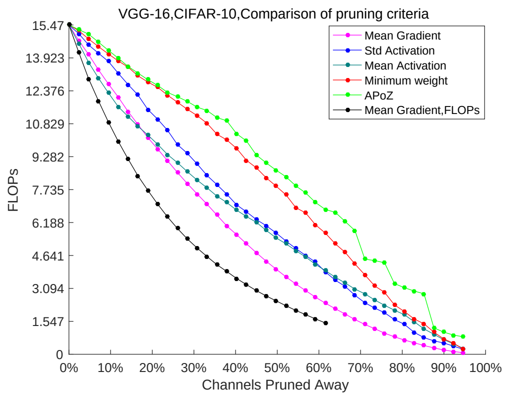
$$f(condition) = \begin{cases} 1, & if \quad condition \quad is \quad true \\ 0, & else \end{cases} \tag{9}$$

   activation neurons $\mathbf{o}_l$ are different from feature maps $\mathbf{x}_l^{(k)}$ in Eq.(3).

Fig.8 shows comparison results of pruned VGG-16 models on CIFAR-10 with different pruning criteria, and its parameters setting is consistent with Section 4.2. Fig.8(a) illustrates that the accuracy of the pruned model with mean gradient criterion is relatively stable as model FLOPs reduces, especially when the floating point calculation is reduced to over 70%, our pruning criterion achieves the highest network accuracy. On the other hand, mean gradient criterion obtains an effective reduction in network FLOPs with the same pruning ratio in Fig.8(b), and our algorithm achieves the best result in reducing FLOPs after adding FLOPs constraint.



(a)



(b)

Figure 8: Comparison of pruning criteria. (a) The curve between model FLOPs and network accuracy for VGG-16 on CIFAR-10. (b) The curve between pruning ratio and model FLOPs for VGG-16 on CIFAR-10.

4.4. Pruning VGG-16 on CIFAR-10

The pruned models for VGG-16 on CIFAR-10 are shown in Tab.3, VGG16-pruned-A indicates that network pruning ratio is 48%, which means the number of pruning iterations $I = 20$ in **Algorithm 1**, and the number of output channels for VGG16-pruned-A is shown in the last two columns of Tab.2. VGG16-pruned-B indicates that network pruning ratio is 34%, i.e. the number of pruning iterations $I = 14$. Inference time is not only affected by network FLOPs but also by specific convolution operations, parallel algorithms, hardware and other factors, therefore inference time for these two pruned models is measured to compare the actual acceleration with VGG-16 network.

Table 3: The pruned model for VGG-16 on CIFAR-10. Inference time in the last column is tested on Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz with batch size 32.

Model	Error	FLOPs	Pruned	Params	Pruned	Time(speed up)
VGG16	7.53%	1.55×10^{10}	—	1.34×10^{8}	—	26.0
VGG16-pruned-A	8.25%	2.74×10^{9}	**82.3%**	8.60×10^{7}	**36.0%**	**5.4(4.8×)**
VGG16-pruned-B	**7.49%**	4.59×10^{9}	70.3%	1.03×10^{8}	23.3%	9.0(2.9×)

We compare above pruned models with model VGG16-MW [24] and its pruning ratio is 37.1%. It notes that the architecture of VGG16-MW and the size of the input image for VGG16-MW are different from those described in Section 2.2, which affect the accuracy of pruned models. We can see that pruning ratio of VGG16-pruned-B is smaller than VGG16-MW from Tab.4, but its FLOPs has achieved a sharper drop. Moreover, VGG16-pruned-A obtains the best results in FLOPs reduction and actual speed up with less than 1% decrease in accuracy.

Table 4: Comparison of pruned models for VGG-16 on CIFAR-10.

Model	Accuracy	Params	Flops
VGG16-MW[24]	+0.93%	2.78×	1.52×
VGG16-pruned-A	-0.72%	1.56×	**5.64**×
VGG16-pruned-B	+0.04%	1.30×	**3.37**×

4.5. Pruning ResNet-110 on CIFAR-10

ResNet-110 is divided into three hierarchies by residual blocks, the size of its corresponding feature maps are 32×32, 16×16, 8×8, respectively. According to the process of pruning for ResNets in Section 2.2, we obtain the pruned model for ResNet-110 on CIFAR-10. During training, images are randomly cropped to 32×32 with padding 4 for network input, flip horizontal is applied to implement data augmentation. And images are not cropped and flipped horizontally during testing. The number of pruning iteration is $I = 15$ in **Algorithm 1** for ResNet110-pruned-A, and $I = 23$ for ResNet110-pruned-B. 1/64 of output channels for the last convolutional layer will be removed during each pruning in each residual block. At the same time, hierarchical global pruning strategy is applied in pruning the first two convolutional layers in each residual block. We prune 1/64 of output channels for the first two convolutional layers in each hierarchy during each pruning. We firstly perform 5 epochs *SGD* updates with batch-size 128, momentum 0.9, learning rate 10^{-2}, and we continue fine-tuning 5 epochs with learning rate 10^{-3}. FLOPs for ResNets is so low that the FLOPs constraint does not apply for ResNets.

Table 5: The pruned model for ResNet-110 on CIFAR-10. Inference time in the last column is tested on Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz with batch size 128.

Model	Error	FLOPs	Pruned	Params	Pruned	Time(speed up)
ResNet110	6.47%	2.53×10^8	—	1.72×10^6	—	2.71
ResNet110-pruned-A	6.56%	1.46×10^8	42.3%	1.02×10^6	40.7%	2.02(1.34×)
ResNet110-pruned-B	**6.55%**	1.02×10^8	**59.7%**	0.72×10^6	**58.1%**	**1.66(1.63×)**

For Residual Networks with low FLOPs, Tab.5 shows that pruning ratios of FLOPs and parameters maintain high ratios, and the decrease of network accuracy is less than 0.1% with half of the parameters are pruned for ResNet-110. The comparison of our pruned models with model ResNet110-MW [24] is shown in Tab.6. Convolutional layers in residual blocks are all pruned with our pruning strategy, so parameters are reduced more significantly than ResNet110-MW [24], which only prune the first layer of residual blocks. The best pruned model achieves 2.48× reduction in FLOPs and parameters with 0.08% decrease in accuracy.

Table 6: Comparison of pruned models for ResNet-110 on CIFAR-10.

Model	Accuracy	Params	FLOPs
ResNet110-MW [24]	-0.23%	1.47×	1.63×
ResNet110-pruned-A	-0.09%	1.69×	1.72×
ResNet110-pruned-B	**-0.08%**	**2.39**×	**2.48**×

5. Conclusion

In this paper, we apply channel pruning to accelerate CNNs and introduce a new criterion based on the mean gradient of feature maps, we propose hierarchical global pruning strategy to effectively reduce network FLOPs. During each pruning, we measure the importance of feature maps on each channel by its mean gradient and use hierarchical global pruning strategy to remove lower important feature maps, and then we obtain a smaller network model. We focus on the effect of removing feature maps on reduction in network FLOPs. In order to accelerate CNNs effectively, we apply FLOPs constraint condition to determine pruning ratio of each hierarchy. Channel pruning for VGG-16 and ResNet-110 are implemented respectively with less than 1% decrease in accuracy. In the future, we would like to combine our pruning strategy with other pruning criteria, channel pruning for both convolutional layers and full connection layers will be achieved to simultaneously accelerate and compress CNNs.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant Number: 61801325), the Natural Science Foundation of Tianjin City (Grant Number: 18JCQNJC00600) and the Huawei Innovation Research Program (Grant Number: HO2018085138).

References

[1] R. Girshick, "Fast R-CNN," in *IEEE International Conference on Computer Vision*, pp. 1440–1448, 2015.

[2] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *IEEE International Conference on Computer Vision*, pp. 1520–1528, 2016.

[3] X. Jia, E. Gavves, B. Fernando, and T. Tuytelaars, "Guiding the long-short term memory model for image caption generation," in *IEEE International Conference on Computer Vision*, pp. 2407–2415, 2016.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, pp. 1097–1105, 2012.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[7] M. Kuhn and K. Johnson, "An introduction to feature selection," in *Applied Predictive Modeling*, pp. 487–519, Springer, 2013.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[9] Y. D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *Computer Science*, vol. 71, no. 2, pp. 576–584, 2015.

[10] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, pp. 2148–2156, 2013.

[11] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” in *Advances in neural information processing systems*, pp. 1269–1277, 2014.

[12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *Computer Science*, vol. 3, no. 4, pp. pgs. 212–223, 2012.

[13] H. Zhou, J. M. Alvarez, and F. Porikli, “Less is more: Towards compact cnns,” in *European Conference on Computer Vision*, pp. 662–677, Springer, 2016.

[14] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, “Tensorizing neural networks,” in *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

[15] V. Lebedev and V. Lempitsky, “Fast convnets using group-wise brain damage,” in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pp. 2554–2564, IEEE, 2016.

[16] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

[17] G. Castellano, A. M. Fanelli, and M. Pelillo, “An iterative pruning algorithm for feedforward neural networks,” *IEEE transactions on Neural networks*, vol. 8, no. 3, pp. 519–531, 1997.

[18] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in *Advances in neural information processing systems*, pp. 598–605, 1990.

[19] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” *Advances in Neural Information Processing Systems*, vol. 5, pp. 164–171, 1992.

[20] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, pp. 1135–1143, 2015.

[21] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *Fiber*, vol. 56, no. 4, pp. 3–7, 2015.

[22] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[24] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[25] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on International Conference on Machine Learning*, pp. 807–814, 2010.

[26] H. Hu, R. Peng, Y. W. Tai, and C. K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[27] T. J. Yang, Y. H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *arXiv preprint*, 2017.

[28] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[29] R. Reed, "Pruning algorithms-a survey," *IEEE transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.