Multi-threshold channel pruning method based on L1 regularization

Fu Xin Han¹, Yang Li¹* and Chen Song Wang¹

¹School of Electronic and Information Engineering, Changchun University of Science and Technology, Changchun, 130022, China

* Corresponding author’s e-mail: han@mails.cust.edu.cn

Abstract. Aiming at the problem of single detection scene and poor detection speed, a multi-threshold channel pruning method based on L1 regularization is proposed. L1 regularization is applied to the scaling factor of the BN layer to push the scaling factor of the BN layer to zero, thereby identifying unimportant channels, and determining the initial threshold of channel pruning according to the scaling factor after sparse training. The most suitable pruning threshold of the convolutional layer is obtained through minimum error reconstruction. The least square method and the method of weighting the channel are used to reconstruct the error before and after pruning, and the most suitable pruning threshold is obtained by minimizing the error. Through the data set simulation experiment, the YOLOv3 network is pruned, so that the network has a smaller network structure and faster detection speed, which is suitable for deployment on mobile devices.

1. Introduction

Convolutional Neural Networks (CNNs), as the hottest technology in the field of machine learning in recent years, have achieved very successful applications in many fields. The success of convolutional neural networks is due to its more parameters and larger and deeper network models. It is for this reason that the operation of these networks requires high-performance computers and a large amount of operating memory to maintain their good detection performance. However, how to reduce the amount of floating-point operations and the size of training parameters and accelerate network inference without reducing the detection accuracy has become an urgent problem to be solved. Model pruning is undoubtedly an effective way to achieve this goal.

Model pruning can be implemented at different levels, for example, weight level, kernel level, channel level or layer level [1]. Weight-level pruning has higher flexibility, versatility, and can obtain higher compression ratios, but it usually requires special software or hardware accelerators to quickly infer the sparse model [2]. On the contrary, the simplest hierarchical pruning does not require a dedicated library to speed up the inference speed, but because the hierarchical pruning often requires whole-layer pruning, the flexibility is poor, and the removal of layers is effective only when the network is deep enough [3]. In contrast, channel pruning provides a good trade-off between flexibility and ease of implementation. It can be applied to any typical CNN or fully connected network. After pruning, the model can be trained using existing hardware and computing libraries, and finally a higher compression rate and shorter calculation time can be obtained.

Li et al. [4] proposed a compression technique based on convolution kernel pruning. Hu et al. [5] proposed to use Average Percentage of Zeros to measure the zero activation percentage of neurons. Luo et al. [6] regarded filter pruning as an optimization problem, and proposed that the filter pruning needs
to be calculated based on the next layer to obtain statistical information instead of the current layer. He et al. [7] proposed soft filter pruning to accelerate the inference of deep neural networks. This method allows the clipped filter to continue to be updated during training and retains the expressive ability of the original network. Hu et al. [8] introduced an additional loss to encode the difference in feature map features and semantic distribution between the original model and the pruned model. Zhuang et al. [9] introduced additional discrimination perception loss into the network to improve the discrimination ability of the intermediate layer, and then selected the most discriminative channel for each layer considering the additional loss and reconstruction error.

Liu et al. [10] enhanced the channel-level sparsity of the convolutional layer by applying L1 regularization to the channel scaling factor, and pushed the value of the BN layer scaling factor to zero, thereby identifying unimportant channels and pruning them. This paper proposes a multi-threshold channel pruning method based on the literature [10], which can automatically obtain the pruning threshold suitable for each convolutional layer and complete the pruning at one time. While ensuring the detection accuracy, the pruning model has Narrower network structure and faster detection speed.

2. Multi-threshold channel pruning
This paper proposes a multi-threshold channel pruning based on regularization and least squares. It can be briefly described as follows: first introduce a scaling factor for each channel of the convolutional network, and then jointly train these scaling factors to achieve the sparsity of the network and identify the unimportant Channel, the initial threshold of channel pruning is determined according to the size of the scaling factor after sparse training; then, the least square method and the method of weighting the channel are used to reconstruct the error before and after pruning, and the most suitable one is obtained by minimizing the reconstruction error Pruning threshold and pruning the model. The pruning is completed at one time without repeated pruning. Finally, the same training parameters as the normal training are used to fine-tune the pruning network to restore the network accuracy.

2.1. Channel pruning based on threshold
The network channels are sparsified based on L1 regularization. First, a scaling factor is introduced for each channel of the convolutional network, and then these scaling factors are jointly trained to achieve the sparsity of the network. The objective function of the method can be defined as:

$$L = \sum_{(x,y)} l(f(x,W), y) + \lambda \sum_{\gamma \neq 0} g(\gamma)$$

Where (x, y) represents the input image and target, W represents the weight that can be trained, the first sum term is the normal training loss function of CNN, and λ is the penalty factor of the two sum terms. g(γ) is the penalty function caused by the sparsity of the scaling factor, that is, the selected L1 regularization:

$$g(s) = |s|$$

After sparse training (regularization), a sparse model can be obtained, in which many scaling factors are close to 0, which can be used to determine the importance of channels and pruning. Use the scaling factor as the criterion for evaluating the importance of the channel. Because the scaling factor is trained at the same time as the network weight, the network can automatically identify unimportant channels, and can safely delete these channels without causing a great impact on the performance of the network method. influences.

2.2. BN layer and scaling factor
As a normalization method, the BN layer can make the network converge quickly and obtain better performance. There are two hyperparameters γ and β in the BN layer, γ is also called a scale factor (scaling factor). Each scale factor All are associated with a specific CNN convolution channel (or a neuron in a fully connected layer) [11]. Assuming that Z_{in} and Z_{out} are the input and output of the BN layer, B refers to the current mini-batch, and the BN layer performs the following conversion:
\[
\hat{Z} = \frac{Z_{in} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}; \quad Z_{out} = \gamma \hat{Z} + \beta
\]  

Among them, \(\mu_B\) and \(\sigma_B\) are the average value and standard deviation of the input activation on B, and \(\hat{Z}\) is the input actually sent to the current layer. If \(\gamma\) is very small, then the input value sent to the next layer is very small, so small that it can be ignored; at the same time It also shows that in the previous convolutional layer, the output value of this channel contributes little to the network, and the size of the scale factor in the BN layer can be used to evaluate the contribution of the channel to the network. Therefore, when performing sparse training, there is no need to introduce additional scaling factors. The BN layer can be directly used as the scaling layer of the network, and the \(\gamma\) parameter can be used as the scaling factor for sparse training to sparse the convolutional network, which can not only solve the problem of inserting the scaling layer, but also It also saves additional network overhead.

2.3. Pruning threshold

Assuming that the dimension of the feature map (feature map) output by the convolutional layer is \(h \times w \times c\), \(h\) and \(w\) are the height and width of the feature map respectively, and \(c\) is the number of channels. Sending it to the BN layer will get normalization In the subsequent feature maps, each of the \(c\) feature maps corresponds to a set of \(\gamma\) and \(\lambda\). Subtracting the channel corresponding to the smaller scaling factor is essentially directly cutting off the convolution kernel corresponding to this feature map. As for how to choose a smaller scaling factor, it depends on a pruning threshold. For example, all scaling factors are sorted according to the size of the value, and the scaling factor value of 10% of the scaling factors sorted from small to large is taken as the pruning threshold. Branch threshold. The channel corresponding to the scaling factor smaller than this threshold, that is, the convolution kernel corresponding to this feature map, will be subtracted. As shown in Figure 1.

![Figure 1 Pruning diagram](image)

2.4. Minimize reconstruction error

After sparse training, based on manually setting an initial threshold of pruning, according to the set threshold, through the method of weighting the convolution kernel, each convolutional layer in the network is "simulated pruning", and the least square method is adopted. To construct the output error before and after pruning.

Assuming that \(x\) is the input of the convolutional layer, its dimension is \((h_x, w_x, n)\), \(n\) is the number of input channels; \(k\) is the unpruned convolution kernel, and its dimension is \((h_k, w_k, n, c)\), \(c\) is the output channel of the convolution kernel; \(y\) is the output channel of the convolution kernel. The output of the pruned convolutional layer has the dimension \((h_y, w_y, c)\). Then the convolution operation performed in the convolution layer can be expressed as follows:

\[
y_i^{h_y \times w_y \times n} = k_i^{h_k \times w_k \times n} x_i^{h_x \times w_x \times n} \quad i \in c
\]

\[
y = \{y_1, y_2, \cdots, y_c\}
\]
Regarding the $c$ scaling factors corresponding to the feature map after sparse training as a $c$-dimensional column vector $\gamma = [\gamma_1, \gamma_2, ..., \gamma_c]^T$, according to the initial threshold set before, set $\gamma_i (i \in c)$ greater than this threshold to 1, and less than this threshold to 0 to obtain a new $c$-dimensional column vector $\eta = [\eta_1, \eta_2, ..., \eta_c]$, which contains only 0 and 1. Use this vector as a weighting coefficient to weight the convolution kernel. It can be assumed that the convolution kernel multiplied by 0 will be clipped, and the convolution channel multiplied by 1 is retained, thereby simulating the channel pruning of the network and obtaining the pruning After the output. Based on the method of weighting the channels, the least square method is used to construct the output error before and after pruning. The total output error can be expressed as:

$$d = \sum_{i=1}^{c} (y_i - \eta_i k_i x)$$

Then, based on the initial threshold pruning, the output error $d$ is minimized, and the pruning threshold that is most suitable for the network is solved. Suppose a $c$-dimensional column vector $\Delta \eta = [\Delta \eta_1, \Delta \eta_2, ..., \Delta \eta_c]^T$ is an offset of the initial threshold vector, and its form is the same as $\eta$. The problem of solving the optimal pruning threshold is transformed into the following optimization problem:

$$\arg \min_{\Delta \eta} \sum_{i=1}^{c} (y_i - (\eta_i + \Delta \eta_i) k_i x)^2$$

Finally, the vector $\hat{\eta}$ can be converted into a new pruning threshold, and the network pruning is performed according to the optimal pruning threshold. The optimal threshold is determined layer by layer, and the vector obtained by each convolutional layer will be different, that is, there will be many pruning thresholds. The advantage of this pruning method is that different thresholds are used to prune the different convolutional layers, and when the method uses the initial threshold for calculation, the network will not be pruned, but only once when the final pruning threshold is obtained. Pruning.

2.5. fine-tuning

After the pruning is completed, the accuracy degradation of the model can be compensated by fine-tuning the pruned model. This paper uses the same parameters as the normal training of the model to retrain the pruned network, that is, fine-tune the pruning network. The fine-tuned network may obtain higher generalization accuracy. This is because the trimmed redundant part contributes little to the network and will not cause too much loss to the network accuracy, and pruning can effectively reduce the network's overshoot. Fitting phenomenon, after fine-tuning, the network accuracy will recover or even exceed the accuracy before pruning.

3. Experiment and analysis

The experiment first uses the darknet53 network to perform pruning on the MNIST data set and CIFAR-10 data set respectively to verify the feasibility and effectiveness of the multi-threshold channel pruning method. The MNIST data set contains 10 categories, namely numbers 0 to 9, with a total of 70,000 images. CIFAR-10 is composed of 10 types of $32 \times 32$ color images: cats, dogs, birds, horses, deer, frogs, boats, cars, trucks, and airplanes, with a total of 60,000 images. In the experiment, the proposed pruning method was applied to the YOLOv3 network, and the network was pruned on the manual detection data set [12] to obtain a lightweight YOLOv3 pruning network, and compared with other pruning methods and the original network. To verify the effectiveness of the YOLOv3 pruning network, at the same time, compare the results of multi-threshold pruning with the existing pruning methods. The experimental environment is shown in Table 1, and the experimental results are shown in Table 2.
Table 1. Experimental environment

| Operating system                  | Windows 10 |
|-----------------------------------|------------|
| GPU                               | NVIDIA Quadro P5000 |
| Experimental environment          | Python 3.6 |
| Dependent libraries               | Albumentations, Matplotlib, Pytorch Terminaltables, Tqdm, Torch, Random .... |

Table 2. Comparison of experimental results

| The model name    | mAP   | parameters | FLOPS     | Speed up | Model size  |
|-------------------|-------|------------|-----------|----------|-------------|
| Tiny-YOLOv3       | 68.4% | 8.7M       | 5.46B     | 1.0×     | 33.1MB      |
| YOLOv3            | 90.8% | 61.5M      | 32.8B     | 1.0×     | 246.4MB     |
| Multi-threshold pruning | 89.1% | 14.2M      | 12.3B     | 2.7×     | 57.1MB      |

4. Conclusion
Analyzing the experimental results, the multi-threshold pruning method used in this article can reduce the size of the YOLOv3 model and increase the detection speed of the model. At the same time, the detection accuracy is reduced by only 1.7%, so that the network has a smaller network structure and faster detection speed. Suitable for deployment on mobile devices.

References
[1] LI Jiang-yun, ZHAO Yi-kai, XUE Zhuo-er, et al. A survey of model compression for deep neural networks [J]. Chinese Journal of Engineering, 2019,41(10):1229-1239.
[2] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In ICLR, 2016.
[3] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In NIPS, 2016.
[4] Li H, Kadav A, Durdanovic I, et al. Pruning Filters for Efficient Convnets[J]. arXiv preprint arXiv:1608.08710, 2016.
[5] Hu H, Peng R, Tai Y W, et al. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures[J]. arXiv preprint arXiv:1607.03250, 2016.
[6] Luo J H, Wu J, Lin W. Thinet: A Filter Level Pruning Method for Deep Neural Network Compression. Proceedings of the IEEE international conference on computer vision[C]. Venice, Italy: IEEE, 2017: 5058-5066.
[7] He Y, Kang G, Dong X, et al. Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks[C]. Twenty-Seventh International Joint Conference on Artificial Intelligence IJCAI-18. 2018.
[8] Hu Y, Sun S, Li J, et al. Multi-loss-aware Channel Pruning of Deep Networks.2019 IEEE International Conference on Image Processing [C]. IEEE, 2019: 889-893.
[9] Zhuang Z, Tan M, Zhuang B, et al. Discrimination-aware Channel Pruning for Deep Neural Networks[J]. arXiv preprint arXiv:1810.11809, 2018.
[10] Liu Zhuang, Li Jianguo, Shen Zhiqiang, et al. Learning Efficient Convolutional Networks through Network Slimming[J]. In Proceedings of the IEEE International Conference on Computer Vision, 2017: 2736-2744.
[11] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[C]. International Conference on Machine Learning, PMLR, 2015: 448-456.
[12] Mittal A, Zisserman A, Torr P H S. Hand detection using multiple proposals. TorrProceedings of British Machine Vision Conference[C]. BMVA Press, 2011, 2(3): 5.