A Convolutional Neural Network Pruning Method Based On Attention Mechanism

XiaoJie Wang, WenBin Yao* and Huiyuan Fu
Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia
Beijing University of Posts and Telecommunications
Beijing, China
{then, yaowenbin, fhy}@bupt.edu.cn

***Abstract*—Pruning effectively reduces the size of neural networks, which facilitates deployment of neural networks in production environment, especially in embedded systems with limited computing resources. In this paper, we propose a convolutional neural network pruning method based on attention mechanism. We add a attention module to model to generate scaling factors for channels. The scaling factors are considered as channels' importance score, thus filters and convolution kernels corresponding to channels with lower importance score are removed. Our method has the ability to learn importance of channels during training, instead of considering only the direct impact of parameters like existing methods. Moreover, it does not depend on any dedicated libraries, so could be combined with other compression methods for better performance. In experiments, we prune about 90% parameters in VGGNet with 0.67% accuracy drop and prune about 50% parameters in ResNet-56 with 1.02% accuracy drop.**

***Index Terms*—convolutional neural network; pruning; attention mechanism**

I. Introduction

In recent years, with the rapid development of deep learning, convolutional neural networks have achieved excellent performance in many fields, such as computer vision, speech recognition and natural language processing, etc. However, these extraordinary performances are at the expense of high computational and storage demands. Thus neural network compression technique has great significance for deploying a deep convolutional neural network on embedded devices(like mobile phone and IOT device) with constrained resource.

Many works have been proposed to compress deep models, including network quantization [1], [2], matrix decomposition [3], [4], and knowledge distillation [5], [6]. Pruning is one of the most effective compression methods, it aims to remove redundant parameters in neural networks. Which parameters are redundant depends on importance measurement of parameters. Early studies measure the importance of parameters by calculating second derivative of parameters to loss function [7], [8]. In spite of their success, second derivative has expensive calculation and memory overhead. Thus, most recently proposed pruning methods are based on direct impact of parameters, such as parameter magnitude [9] or sparsity of output [10], which do not take correlation between parameters and loss function into consideration.

In this paper, we propose a filter-level pruning method based on attention mechanism. In our method, a attention module(SEBlock [11]) is added to network to generate scaling factors for channels, then we consider scaling factors as channel importance score to guide filter-level pruning. The attention module adjust scaling factors of channels according to loss function in backpropagation, which associates the importance score of parameters with loss function. By removing filters and convolution kernels corresponding to low importance score channels, we can effectively reduce size of deep model and maintain prediction accuracy.

We conduct experiments on CIFAR-10 dataset [12], results show that our method can remove about 90% parameters in VGGNet [13], with only 0.67% accuracy drop. Pruning on ResNet-56 [14] also be conducted to verify the effectiveness of our method on the network with shortcut connections. More serious decrease of accuracy appears on ResNet-56 since it is a compact network, but we still remove half of parameters with roughly 1% accuracy drop. Moreover, we compare our method with weight sum [9] and APoZ [10] on CIFAR-100 [12], the result shows that our method has better performance under the same pruning ratio.

II. Related Work

LeCun et al. [8] proposed that some of the neurons connections in neural networks could be removed without decreasing model accuracy. Similar to LeCun's work, Hassibi et al. [7] used hessian matrix to get the second derivative of parameters. However, the spatial complexity of hessian matrix is $O(n^2)$(n is the number of parameters), which has expensive memory cost on deep models. In order to avoid problem mentioned above, recent studies prefer to prune models according to direct impact of parameters. Han et al. [15] proposed that the magnitude of the parameters could reflect the importance of the parameters, they remove parameters below a certain threshold to get compact model. Deep compression [16] further compress deep neural networks with pruning, trained quantization and Huffman Coding. However, pruning neurons connections produces sparse matrices, which relies on specific operational libraries and hardware to exploit performance advantages [17].

Correspondence: yaowenbin@bupt.edu.cn
DOI reference number: 10.18293/SEKE2019-147

In order to avoid the limitation of pruning neurons connections, structural pruning method was proposed. Lebedev et al. [18] explored a structured sparsity learning method(SSL), which adds the regularization term of parameter group to loss function so that certain groups of parameters would shrink to zeros during training, eventually be removed safely. However, SSL still destroys the network structure and depends on dedicated libraries.

In recent researches [9], [10], [19], [20], filter-level pruning become an effective pruning method that completely avoid using dedicated libraries. Li et al. [9] calculated the sum of the absolute values(L1 paradigm) of weights in filters as their importance score. This method has strong limitations because L1 paradigm of filter does not reflect the feature extraction ability of filter. Hu et al. [10] observed the sparsity of the ReLU activation function and assumed that a neuron is unimportant if most outputs of the neuron with ReLU are zero. Although Hu's method considers more further effects of the filters, it still does not determine the importance of parameters according to loss function. The methods mentioned above are based on the direct impact the parameters, which does not well represent the effect of parameters on neural network's loss function. Meanwhile, they are artificially formulated, which leads to that human intervention is added in training process which violates the rules of end-to-end training in deep learning.

III. OUR METHOD

In this section, we would first introduce filter-level pruning, which determines the granularity of our pruning method. Then, the attention module used in our method would be described. Next, average scaling factor formula would be presented. Finally, we would show overall steps of our method and pruning strategy adjustment on special model.

A. Filter-level pruning

As shown in Fig. 1, each layer of neural network consists of several filters, one filter has a set of convolution kernels. Instead of generating sparse matrices or destroying network structures, filter-level pruning removes entire filter, which maintains the regularity of the network. For example, channel2 would disappear when the filter2 marked by dotted lines are removed. In this case, convolution kernels process channel2 in next layer could be removed as well.

In a convolution layer, if we suppose its original parameter matrix is expressed as $W_{<I,W,H,C>}$, where I is the number of input channels, W is the width of convolution kernels, H is the height of convolution kernels and C is the number of filters. We can calculate the number of parameters in this layer as:

$$|W_{<I,W,H,C>}| = I \times W \times H \times C \quad (1)$$

If we set the filter pruning ratio of this layer to q, $q \times C$ filters in this layer would be removed, so that the number of remaining parameters in this layer can be calculated as:

$$|W_{<I,W,H,(1-q)C>}| = I \times W \times H \times (1-q)C \quad (2)$$

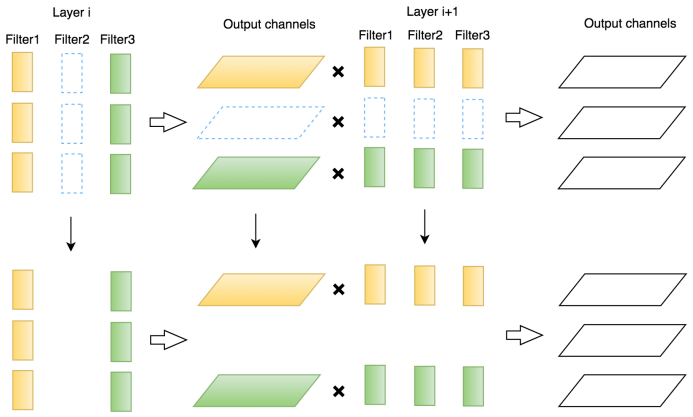


Fig. 1. Filter-level pruning.

If we set the filter pruning ratio of the front layer to p, $p \times I$ convolution kernels in this layer would be removed, so that the number of remaining parameters in this layer can be calculated as:

$$|W_{<(1-p)I,W,H,(1-q)C>}| = (1-p)I \times W \times H \times (1-q)C \quad (3)$$

Obviously, after pruning all layers, the total parameters of the convolution layer would be reduced to $(1-p) \times (1-q)$ of the original layer. In other words, $q+(1-q) \times p$ of the parameters would be removed.

B. Attention module

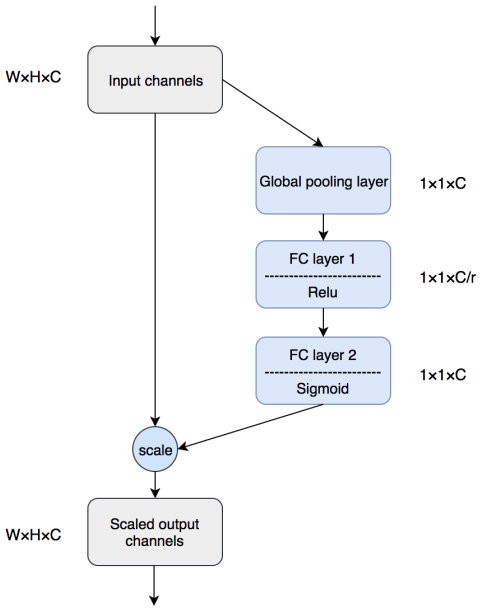


Fig. 2. SEBlock structure, the blue part is SEBlock.

As mentioned in related work, the filter-level pruning methods lack the consideration of correlation between parameters and loss functions due to computational complexity. To calculate importance score of filters according to loss function in a feasible computational complexity, we use attention module

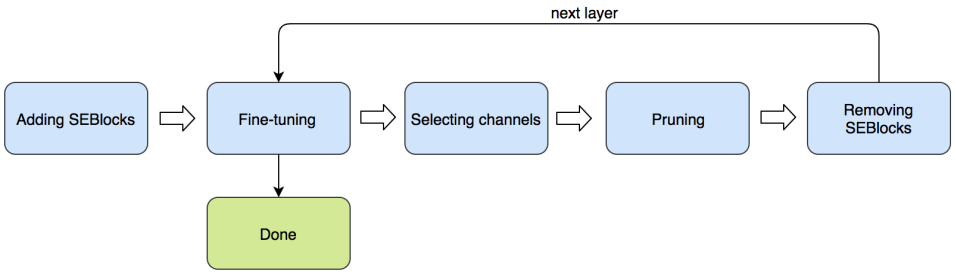


Fig. 3. The overall pruning steps of our method.

to generate scaling factors for channels and adjust scaling factors in backpropagation. In this case, which channel is enhanced or suppressed can be reflected by the scaling factors. Then we can rank channels by their scaling factors with the rule that channels with smaller scaling factors are less important. Because the one-to-one match between each filter and each channel, unimportant filters can be selected according to channel rank.

The attention module(namely SEBlock) used in our method comes from Squeeze-and-Excitation Networks(SENet) [11], which is designed for image classification tasks. SEBlock consists of a global average pooling layer, two fully connected layer, sigmoid activation function and ReLU activation function. SEBlock can be added after convolution layer, take the convolutional layer's output channels as input data, produce scaling factor for each channel.

As shown in Fig. 2, SEBlock take a C channels as input, obtain a vector with C elements after the global average pooling layer. The number of neurons in first fully connected layer is C/r, where r is a hyper-parameters for controlling module parameter number. Between the first and the second fully connected layer is a ReLU activation function. The second fully connected layer has C neurons in order to produce C output value. Finally, the C output values of are projected between 0-1 by sigmoid activation function to become scaling factors. The output channels of SEBlock can be obtained by multiplying each input channel with corresponding scaling factor.

Although SEBlock was originally designed for improving the accuracy of image classification, its scaling factor reflects the network's choice of feature, which is the embodiment of the channel importance. Channel-level weights introduce more scale features to the network and further enhance the expressive ability of the network. By explicitly describing the importance of inter-layer channels, model would eventually show the phenomenon of restraining or enhancing some channels. Therefore, our method uses the scaling factor produced by SEBlock as channel importance score, and prunes filters in the same layer according to channel rank. Since parameters in SEBlock are randomly initialized, the model would seriously lose its prediction accuracy at first. However, after 1 epoch fine-tuning, the model would quickly return to its original prediction accuracy level, after several epochs it would completely recover from adding SEBlock.

C. Average scaling factor

For SEBlock, scaling factors of channels are data-driven, so we collect a dataset for calculating average scaling factors. Given a collected subset with N images. We calculate average of scaling factor I_C as follows:

$$I_C = \frac{1}{N} F_{se}(X_{<n,H,W,C>}) \tag{4}$$

I_C is a vector having C elements, each element is average scaling factor of corresponding channel, $X_{<n,H,W,C>}$ is C input channels produced by n^{th} image in dataset, W,H is the shape of each channel and F_{se} is the operation of SEBlock.

If the training dataset is not too large, we can directly use the entire training dataset. However, in order to reduce the computational complexity on large datasets, a subset of training dataset is sufficient for calculating average scaling factor. The amount of samples depends on the total number of samples in the training dataset. In our experiments, 10%-20% of the training sample is enough for importance evaluation.

D. Steps of our method

Pruning steps are shown in Fig. 3, the specific steps are as follows:

1) **Adding SEBlock.** Given an original model, SEBlock needs to be added after each convolutional layer. Then, we would prune model layer by layer with a predefined pruning rate p, which means p and q in Eq. 3 is equal.
2) **Fine-Tuning.** After adding SEBlocks or pruning, the model's prediction accuracy would decrease. By fine-tuning(retraining) the model, it would recover from damage. Meanwhile, SEBlock would scale channels according to their contribution to the loss function.
3) **Selecting Channels.** We sort channels in current layer by importance score, and select $p \times C$ channels with small scaling factors as unimportant ones. In our experiment, we set the pruning rate p of each layer to the same.
4) **Pruning.** Aiming to remove low-score channels selected in step 3, the filters and convolution kernels corresponding to these channels would be removed as section A describes.
5) **Removing SEBlocks** After pruning, in order to compare pruning performance with other pruning methods, we

remove all SEBlocks. In fact, the SEBlock has few parameters(it is mentioned in [10] that SEBlock only leads to about 10% increase in parameter number), which means we could keep SEBlock in practical application.

6) **Pruning the Next Layer.** Go to step 2 until all layers are pruned.

E. Pruning Strategy Adjustment

The traditional architecture like AlexNet [21] and VGGNet [22] are often used to verify the effectiveness of pruning methods. In these models, pruning one layer would not change the input shape of other layers except the next layer. But in residual networks like ResNet [14], they have shortcut connections in residual module. Shortcut connection connects the first layer and the last layer in residual module, add them up as the output of residual module. The shortcut connection requires that the shapes of input and output be the same. If the last layer of residual module is pruned, we need to remove the corresponding filters in the first layer without considering their importance score. So it is difficult to prune the last layer of a residual module, in our method, we just prune the first few layers in a residual module, considering that most parameters of residual modules are in these layers.

IV. EXPERIMENTS

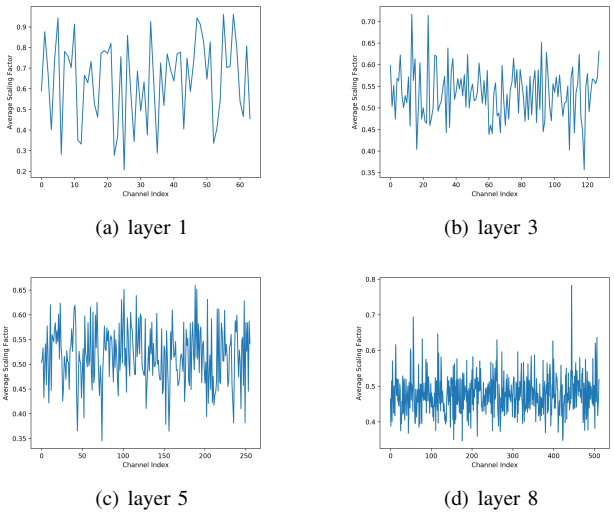
We conduct experiments on CIFAR-10 and CIFAR-100 dataset. The CIFAR-10 dataset consists of 60000 images in 10 classes, with 6000 images per class and resolution of each image is 32×32. The dataset is divided into a training set with 50000 images and a test set with 10000 images. The CIFAR-100 dataset consists of 100 classes, each class contains 500 images for training and 100 images for testing.

On CIFAR dataset, we evaluate our method on two convolutional neural network: VGGNet [13](a variant of vgg16 on the cifar dataset) and ResNet56. The hyper-parameters r mentioned in SEBlock is set to 8 in VGGNet and 4 in ResNet56. For training original model, batch size is set to 128, the learning rate used in first 50 epochs was 0.1, then reduced to 0.01. In pruning process, learning rate is set to 0.01. Weight decay was also used to overcome over-fitting with a coefficient of 0.0001. For calculating average scaling factor, we randomly pick 500 images in each category on CIFAR-10 and 50 images in each category on CIFAR-100. Padding, random cropping and horizontal flipping are applied for data augmentation.

A. Distribution of scaling factors

In this section, the distribution of scaling factors at each layer are are visualized. Experiments are conducted on VGGNet with attention module. The distribution of scaling factors are shown in Fig. 4.

The VGGNet contains four stages, distribution of scaling factors in the same stage are similar, so only scaling factors of layer 1,3,5,8 are shown. Fig. 4(a) is the result of the layer 1, which belongs to the first stage. Fig. 4(b) is the result of layer 3, which belongs to the second stage. Fig. 4(c) is the result of layer 5, which belongs to the third stage. Fig. 4(d) is the result of layer 8, which belongs to the fourth stage. The results show that the distribution of scaling factors in the same layer is quite different, especially the scaling factor between adjacent channels. This phenomenon is similar to the lateral inhibition of human visual neurons,which is beneficial for extracting the shape feature of objects. With the increase of depth, the distribution range of scaling factor decreases, which implies that the importance of channels near the end of neural network are similar.



(a) layer 1 (b) layer 3

(c) layer 5 (d) layer 8

Fig. 4. Distribution of scaling factors.

B. Results on CIFAR-10

On CIFAR-10 dataset, we evaluate our method on VGGNet and ResNet-56, the results are shown in Table I and Table II. The M in each table means million.

TABLE I
VGGNET RESULTS ON CIFAR-10 DATASET.

Model	Filters pruned	Accuracy	Params	FLOPs
Original VGGNet	0%	92.24%	14.98M	6.27×10^8
Pruned	30%	92.18%	7.43M	3.07×10^8
	50%	91.97%	3.82M	1.57×10^8
	70%	91.57%	1.42M	0.57×10^8

As shown in Table I, we prune 30% filters in VGGNet network without obvious accuracy drop, even get an increase in accuracy when we pruned the first few layers. When we prune 50% filters, we still could maintain the model's accuracy with 0.27% accuracy decrease. We conclude that our method correctly measures the importance of the channel which help model recover from pruning. When filter pruning ratio comes to 70%, accuracy decreases more obviously but still within 1% loss. It is worth noting that, by using filter-level pruning method, number of input channels and output channels for each layer would be reduced by 70%, so reduced parameters can be calculated as section III.D describes: $0.7 + (1 - 0.7) \times 0.7 = 91\%$.

TABLE II
RESNET-56 RESULTS ON CIFAR-10 DATASET.

Model	Filters pruned	Accuracy	Params	FLOPs
Original ResNet-56	0%	92.84%	0.85M	2.51×10^8
Pruned	30%	92.43%	0.58M	1.76×10^8
	50%	91.82%	0.42M	1.26×10^8
	70%	90.86%	0.25M	0.76×10^8

Table II shows the results on ResNet-56, different from VGGNet, pruning on ResNet-56 causes a relatively large decrease of accuracy. We prune ResNet-56 with 3 different compression rates as well: 30%, 50%, 70% filters in each layer respectively. Although it cause more serious accuracy decrease than VGGNet, we still prune more than half of the parameters with 1.02% accuracy drop.

According to our analysis, this phenomenon is reasonable because recent network architectures has an improvement of the utilization of parameters. For example, the shortcut connections of ResNet actually makes network structure in a shallow-deep state, which enhance feature fusion and feature delivery. Thus, the parameter utilization of ResNet is much higher than traditional models without shortcut connections, which leads to poor performance on a large percentage of pruning. This phenomenon reminds us that more attentions should be paid when pruning the networks with shortcut connections.

C. Results on CIFAR-100

Our method was compared with Weight Sum [9] and APoZ [10] on CIFAR-100 by pruning VGGNet.

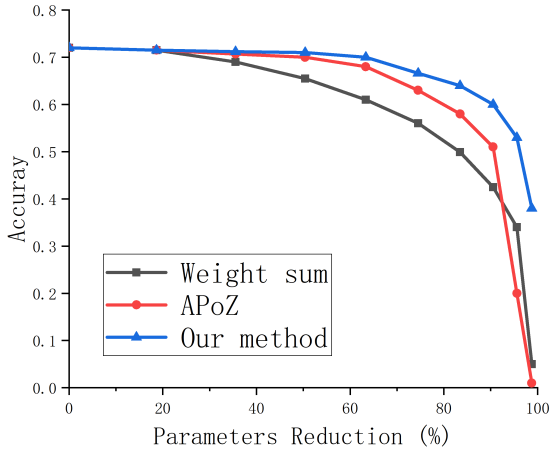


Fig. 5. Our method, Weight sum and APoZ's results on VGGNet.

As shown in Fig. 5, nine groups of experiments were conducted on all methods. Filter pruning ratio of each experiment range from 10% to 90% in step length of 10%. Weight sum has the fastest decline in model accuracy, the accuracy of model begins to decrease significantly when pruning ratio is higher than 10%. It is easy to understand that weight sum only takes parameter magnitude into consideration, which is not directly related to the model's loss function. APoZ has a better performance, when the pruning rate reaches 50% it begins to have a significant drop in accuracy. This result is reasonable since APoZ considers more further information, channels with more value of zero have less influence on the following layers. However, when pruning ratio is more than 80%, APoZ performs worse than weight sum.

Our method has the best performance among the three methods mentioned above. When pruning rate is less than 50%, the model has almost no drop of accuracy, even if the pruning rate is higher than 50%, the accuracy of the model still decreases slower than the other two methods. An interesting result is that when we prune 90% filters in each layer, our method still retain a certain degree of classification ability, unlike the other two methods whose ability to classify is completely lost. The result indicates that our method correlates the importance of channels with model loss function, which describes the importance of the channels naturally.

D. Results on Mobile Device

The goal of neural network pruning is to reduce the computational resource consumption of neural networks, so as to facilitate deployment on device with limited resources. Therefore, We deployed neural networks on mobile phones for verifying the performance improvement of our pruning method in real scenarios. We tested the performance of the original VGGNet and the VGGNet after removing 50% filters on mobile phones, the device information is: 4 GB Memory, Qualcomm Snapdragon 632 CPU, 2.0GHz basic frequency, training dataset is CIFAR-10 and the framework is Tensorflow. The results is shown in Table III.

TABLE III
RESULT ON MEIZU NOTE 6 MOBILE PHONE.

Parameter	Original	50% filters pruned
Inference time	1046ms	301ms
FLOPs	6.27×10^8	1.57×10^8
Model file size	59.9MB	15.3MB
Parameter Number	14.98M	3.82M
Parameter pruned	0%	75%
Accuracy	92.24%	91.97%

Due to the inference time is affected by hardware, we calculate the average inferences time of classifying a image. Obviously, the pruned model has a significant reduction in file size and inference time. It takes more than 1000 milliseconds for the original model to classify a image, which results in a significant pause of application. On the contrary, the pruned model only takes 301 milliseconds to classify a image, which greatly improves user experience. In addition, the endurance capability of device also benefits from the reduction of computational resource consumption.

V. CONCLUSION

In this paper, we have described a pruning method based on attention mechanism. Different from existing pruning method

based on direct impact of parameters, we use SEBlock to automatically learn the importance of channel during training. As low-score channels and corresponding parameters are removed, memory and computing cost of model would be effectively saved. We validated our method on different datasets, results show that it surpass existing methods under the same pruning ratio. In addition, our method does not require any dedicated libraries or hardwares, thus can be combined with other compression methods.

In future work, we would conduct our method on latest network and larger dataset to verify the generalization of our method. We tend to explore pruning method based on global importance score instead of pruning fixed percentage filters in each layer.

ACKNOWLEDGMENTS

This work was partly supported the NSFC-Guangdong Joint Found(U1501254) and China Information Security Special Fund (NDRC).

REFERENCES

[1] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[2] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[4] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. 2015.

[6] Guorui Zhou, Ying Fan, Runpeng Cui, Weijie Bian, Xiaoqiang Zhu, and Kun Gai. Rocket launching: A universal and efficient framework for training well-performing light net. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[7] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[8] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[9] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *The International Conference on Learning Representations*. MIT, 2017.

[10] H Hu, R Peng, YW Tai, CK Tang, and N Trimming. A data-driven neuron pruning approach towards efficient deep architectures. arxiv preprint. *arXiv preprint arXiv:1607.03250*, 2016.

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141. IEEE, 2017.

[12] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[13] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *The International Conference on Learning Representations*, pages 1—-14. MIT, 2016.

[17] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254. IEEE, 2016.

[18] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.

[19] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[20] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.