A new approach for Deep Neural Networks Compression

Ohad Poller
Dept. of Electrical and Computer Engineering
Ben-Gurion University of the Negev
Be'er Sheva, Israel
poller1992@gmail.com

Nathan Gadot
Dept. of Electrical and Computer Engineering
Ben-Gurion University of the Negev
Be'er Sheva, Israel
gadna@post.bgu.ac.il

Tomer Malach
Dept. of Electrical and Computer Engineering
Ben-Gurion University of the Negev
Be'er Sheva, Israel
malachto@post.bgu.ac.il

Dr. Shlomo Greenberg
Dept. of Electrical and Computer Engineering
Ben-Gurion University of the Negev
Be'er Sheva, Israel
shlomog@ee.bgu.ac.il

ABSTRACT

Neural networks are both computationally intensive and heavy memory users, making them difficult to deploy on IoT systems because those are usually limited with power and hardware sources. We want to approach this problem by finding a new compressing method that removes unnecessary parameters from the network. Our way of thinking is to focus on the neurons and filters outputs, means that if two neurons share almost the same outputs for majority of the samples out of a test-batch, we say that those neurons holds the same information and they are "Identical Neurons". The assumption is that "Identical Neurons" are redundant in the network. Using our algorithm, we managed to prune a significant percentage of the network's parameters without any accuracy drop.

Keywords: Neural Network, Compressing, Pruning, Identical, Neurons, Filters, Euclidian Distance, IoT Devices, CNN, DNN, Deep Learning.

I. INTRODUCTION

Modern computer vision tasks are large consumers of fully designed convolutional neural networks architectures [8, 9, 10, 11]. These CNNs usually consist of multiple convolutional layers with a large amount of parameters. They are computationally expensive and often contain many unnecessary parameters. Recently, network pruning has become a goal that many pursuit, it aims to simplify and accelerate large CNNs [12]. For example, the AlexNet model with CIFAR-10 dataset is over 2.2 million parameters, and VGG-19 model with CIFAR-10 dataset is over 38.9 million parameters which makes it difficult to deploy deep neural networks on mobile systems. Although having deep neural networks running on mobile has many great features such as better privacy, less network bandwidth and real time processing, the large storage overhead prevents deep neural networks from being incorporated into mobile devices.

Another issue is the energy consumption, running large neural networks require a lot of memory bandwidth in order to fetch the weights and for the large scale computations of matrices multiplications. Both mentioned operations consume considerable amount of energy. Mobile devices are battery constrained, making power hungry applications such as deep neural networks hard to deploy.

Our main goal is to reduce the storage and energy required to run these deep neural networks by removing unnecessary neurons so the neural networks can fit IoT devices. To achieve this goal, we need to define what is an unnecessary neuron, find those neurons and remove them from the network. Successful compress of neural networks' size could open new possibilities. We will be able to deploy much larger and powerful networks from what we can implement these days on various IoT devices, such as: mobile phones, drones, smart watches, and many more.

As for today, the research about networks pruning focuses on the weights, and looking to prune and decrease the weights amount [1-6, 12, 13, 15, 17-19]. Pruning weights produce pruned models that outperform retraining from scratch with the same sparsity pattern. Retraining from scratch in this context means training a fresh, randomly-initialized model with all weights clamped to zero throughout training, except those that are nonzero in the pruned model [16]. This approach, handling and deleting weights, results in weak connections between neurons and leave a sparsity network which must be retrained.

In our paper, we give a new approach and a new definition to networks pruning, we do not look on the connecting weights between neurons, instead, we observe the neurons and use the outputs that the activation functions return. By removing neurons or filters from the network together with their connecting feature maps, the computation costs and the memory bandwidth are reduced significantly. In contrast to pruning weights, our approach does not result in sparse connectivity patterns, if we choose to delete a neuron or a filter we also delete all of its previous together with its following connections, in that way, we save a lot of computations and prevent irrelevant parameters to be on the network. Figure 1 shows an example of what impact on the network gives a pruning of one neuron: all the red-colored weights will be deleted and the activation of the deleted neuron will be spared. Comparing to prune one weight which saves only the memory size of one weight.

In addition, the network does not need a be retrained because we delete neurons that holds the same information as other neurons. As a matter of fact, we do not lose any data by removing one out of two “Identical Neurons”. Therefore, the model can be deployed and get used straight after pruning process.

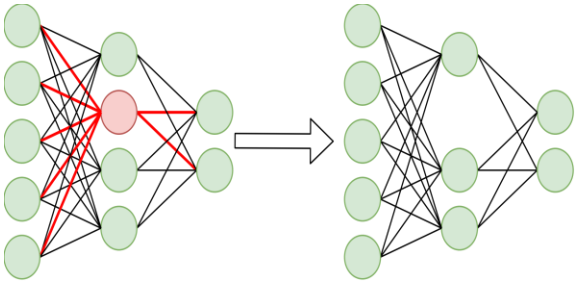


Fig. 1. Example of pruning one neuron out of a Fully Connected layer and its connections, deleting one single neuron coueses the deletion of 7 different weights.

II. Related Work

The best thing about Neural Network pruning, is that it works. Hundreds of articles have proven it already, more precisely, there are various methods that can significantly compress models with little or no loss of accuracy. In fact, pruning can sometimes increase accuracy [1]. We will discuss now several methods that are commonly used the field of compressing neural networks, and are well appreciated [20].

Pruning Weak Connections. This method focuses on finding low magnitude weights in the network and removing them, it was published in 2016 by S. Han et al. This method starts by learning the connectivity via normal network training. Next, pruning the small-weight connections: all connections with weights below a threshold are removed from the network. Finally, retrain the network to learn the final weights for the remaining sparse connections. If there is a neuron that is left with no connections to other neurons, it will be deleted [1]. This method showed promising compression rates on various CNNs. Nevertheless, it required additional computations in order to mask out pruned parameters and handle the sparsity.

Weight Sharing. Weight sharing focuses on saving less data by combining weights together and giving every neuron an index of the shared weight. This method further compresses the pruned network by reducing the number of bits required to represent each weight. The paper’s authors limit the number of effective weights that they need to store, by having multiple links share the same weight, and then fine-tune those shared weights [1]. In figure 2 we can see an example of weight sharing algorithm.

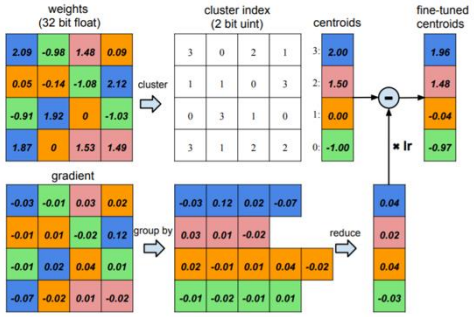


Fig. 2. Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom). Han Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding."(2015) [1].

Pruning filters. Hao Li et al. proposed a magnitude-based pruning method [20]. In each convolutional layer, they ranked the filters by the absolute sum of their feature maps elements and suggested to prune a certain percentage of filters that ranked the lowest. When deleting a filter’s kernel. Pruning a filter results in removal of its corresponding feature map and related kernels in the next layer. By that they save a lot of computations and memory usage. We implemented this method and found it efficient. When testing on VGG-19 model with CIFAR-10 dataset we managed to successfully reduce about 30% of the model’s parameters.

The last-mentioned method differs from our work in the way that it relays on the kernels’ values only, while our algorithm focus on the output of the layers instead. We will see in the experiments section a comparation between the two methods.

III. Method

Our way of thinking is to focus on the neurons and filters outputs, means that if two neurons or filters share almost the same outputs for majority of a test batch samples, we say that those share the same information and we define them “Identical”. The assumption is that “Identical Neurons” and "Identical Filters" are redundant in the network.

We define in this section the concepts of "Identical Neurons" and "Identical Filters":

Identical Neurons – We choose to define Identical Neurons by using "Euclidian Distance", we compare the activation values between every two neurons in the network for each sample of the test batch. "Identical Neurons" will be two neurons that have almost the same output value for the majority of the test batch samples.

Identical Filters – for Convolutional layers we want to find Filters that are redundant in that layer. To do so we compare the values of the elements in each feature map for every filter for that layer. "Identical Filters" will be two filters that share the same output feature maps for most of the test batch samples**.**

We propose a method for pruning a CNN while using those definitions. Assume we are working with a fully trained CNN f with L convolutional layers and K Fully-Connected layers. Firstly, we describe the algorithm for pruning the Fully-Connected layers of the network. Then, we discuss the details about the second algorithm that prunes the Convolutional layers from the network.

1. Pruning Fully Connected layers

Let N_k denote the number of neurons in the k^{th} Fully Connected layer of the Base-model CNN f. Let X_{test} be the test batch of the dataset and let S_X denote the number of samples in this test batch. Suppose the activation function's output for the i^{th} neuron in the layer is given by σ^i, let Th_k be a certain threshold for the k^{th} layer and let PTh be a certain percentage threshold.

Algorithm 1 Prune neurons from Fully Connected layers

Input: f, X_{test}
Output: A pruned network $\hat{f}$

Make a prediction on the network f with the test batch X_{test}
for $k = 1, 2, \ldots, K$ **do**
  for $s = 1, 2, \ldots, S_X$ **do**
    for $i = 1, 2, \ldots, N_k$ **do**
      Calculate the L_1 norm between the i^{th} *neuron's activation value and all other neurons activation values*
      if $\left(\left\|\sigma^i - \sigma^j\right\|_1 < Th_k\right)$ **then** i^{th} and j^{th} neurons will be considered as sample identical neurons (SIN) for the s^{th} sample of X_{test}
    end for
  end for
  for every pair of neurons i *and* j, $i \neq j$ in the k^{th} layer **do**
    if $\left(\frac{\#SIN}{S_X} > PTh\right)$ **then** i^{th} and j^{th} neurons will be considered as Identical Neurons
  end for
Delete one neuron out of every pair of Identical Neurons
end for

The result is the pruned network $\hat{f}$

---

2. Pruning Convolutional layers

Let F_l denote the number of filters and N_l be the dimension of the feature map in the l^{th} Convolutional layer of the base-model CNN f. Let X_{test} be the test batch of the dataset and let S_X denote the number of samples in this test batch. Suppose the filter's feature map for the i^{th} filter in the l^{th} layer is given by $M_{l\,N_l x N_l}^{i}$ and let Th_l be a certain threshold for the l^{th} convolutional layer and let PTh be a certain percentage threshold.

Algorithm 2 Prune filters from Convolutinal layers

Input: f, X_{test}
Output: A pruned network $\hat{f}$

Make a prediction on the network f with the test batch X_{test}
for $l = 1, 2, \ldots, L$ **do**
  for $s = 1, 2, \ldots, S_X$ **do**
    for $i = 1, 2, \ldots, N_l$ **do**
      Calculate the L_1 norm between the i^{th} *filter's feature map and all other filter's feature maps*
      if $\left(\sum_{n=1}^{N_l}\sum_{m=1}^{N_l}\left|M_l^i[n,m] - M_l^j[n,m]\right| < Th_l\right)$ **then**
        i^{th} and j^{th} filters will be considered as sample identical filters (SIF) for the s^{th} sample of X_{test}
    end for
  end for
  for every pair of filters i and j, $i \neq j$ in the l^{th} layer **do**
    if $\left(\frac{\#SIF}{S_X} > PTh\right)$ **then** i^{th} and j^{th} filters will be considered as Identical Filters
  end for
Delete one filter out of every pair of Identical Filters
end for

The result is the pruned network $\hat{f}$

Our method focuses on pruning the network offline after it is already fully trained. Similar to earlier work on ranking filters [12, 15, 20], in our method we rank the filters according to their sensitivity to pruning, after ranking all the convolutional layers of the network we use the algorithm on the layers that had the least impact on the networks performance. Figure 3(a),(b) shows the impact of each layer in the network.

The Percentage Threshold - *PTh* is chosen by the most efficient percentage of pruning, means the percentage that gives the best trade-off between the amount of pruning parameters and the network performance, *PTh* changes between different models. To optimize the trade-off, we do a percentage sweep and check all the possible options for pruning the network, according to the results we choose the value of *PTh* for that model. Unlike *PTh*, the Threshold – Th_l for deciding if two filters are SIF cannot be constant, because different layers generate different scale of feature maps, that is why the Threshold is determined for each layer individually, which happens after analysis of the feature maps for each layer.

Running through all the samples in the test batch X_{test} for each layer might be computational heavy and a big time consuming. We found out that it is not required to run through all of the test batch samples. 20% random samples from the batch is enough to determine if each two filters are SIF or not and all the results in this paper are based on this proven assumption.

As we mention before, in our method there is no need for retraining the network after the pruning process, that in contrary to other methods. The whole idea of the algorithm is do delete from the network only the parameters that holds the same information because those are redundant in the network.

IV. EXPERIMENTS

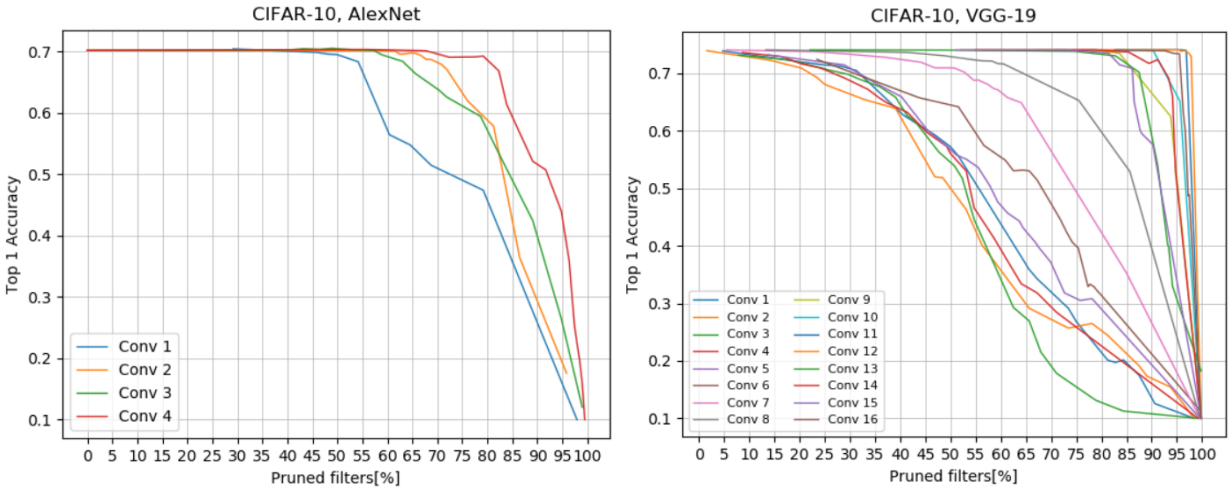
We experiment with our pruning algorithms on two models: AlexNet and VGG-19, both with CIFAR-10 dataset. CIFAR-10 dataset consists of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1,000 randomly selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5,000 images from each class [14].

AlexNet architecture was designed by Alex Krizhevsky in 2012. It was one of the first CNNs to win in the ImageNet challenge and consider to be one of the most influential papers published in computer vision[8]. In table 1 we can see the AlexNet architecture in details [14].

VGG-19 model is a Deep Convolutional Neural Network, from Visual Geometry Group, Department of Engineering Science, University of Oxford. The number 19 stands for the number of layers with trainable weights. 16 Convolutional layers and 3 Fully Connected layers. After the breakthrough provided by AlexNet in 2012 for the ImageNet Challenge competition, the next big breakthrough was by the VGG-19 architecture. In table 2 we see the VGG-19 architecture in details.

Our main focus was on pruning the Convolutional layers since the overall trend in today's CNNs is to reduce the usage of Fully Connected layers. That is because those usually contain a large number of parameters, and that's why we will focus on Algorithm 2, shown in the section above.

With our method we can analyze the redundancy in each layer of a given CNN. The pruning results at figure 3(b) shows that in VGG-19 model higher layers contain more unnecessary filters than the lower layers. In some of the layers we can remove more than 95% of their filters (for example layers 10, 11, 15) while those have relatively less impact on the performance. In fact, we can see clearly from figure 3(b) that the first eight layers of the model are much more sensitive to pruning comparing to the last eight layers, there is a clear separation between them. While the first layers impact the network's performance when pruned by over 20-30% the last layers can be pruned for at least 80% before any impact on the network's performance is seen.



(a) Ranking filters from AlexNet, CIFAR10

(b) Ranking filters from VGG19, CIFAR10

Fig. 3. Ranking filters for each network by the network's accuracy drop for each layer pruned on CIFAR10 dataset. The x axis is the percentage of filters pruned from a certain convolutional layer and the y axis is the accuracy of the network.

TABLE I. SUMMARY OF ALEXNET ARCHITECTURE ON CIFAR-10 DATASET

Layer		***#Parameters***	***Size***	***Activation***
Input	Image	0	32x32x3	-
1	Convolution	1344	32x32x48	relu
	Max Pooling	0	16x16x48	relu
2	Convolution	41568	16x16x96	relu
	Max Pooling	0	8x8x96	relu
3	Convolution	166080	8x8x192	relu
4	Convolution	331968	8x8x192	relu
	Max Pooling	0	4x4x192	relu
6	FC	1573376	512	tanh
7	FC	131328	256	tanh
Output	FC	2570	10	Softmax
Total parameters		2,248,234		

Firstly, we measured the sensitivity level to pruning of each Convolutional layer in the selected models. To do so, we use our algorithm on each layer at a time and measure the accuracy drop of the network after pruning from that specific layer. Then, we compared the accuracy drop of the network that each layer caused and ranked the layers by its effect on the network's performances.

To understand the sensitivity of each layer, we prune each layer independently and evaluate the resulting pruned network's accuracy on X_{test}. In figure 3(a) we can see that AlexNet's first layers are more sensitive to pruning than deeper layers. For example, pruning filters from the first layer caused a significant accuracy drop when pruned more than 38% of its filters, compared to the other three layers that started to affect the accuracy only when the pruning was over 55% of the filters in each layer. We pruned the AlexNet model according to these results, we focus pruning on the last 3 layers, that gave us the best pruning rate.

In our experiments we have pruned the AlexNet and VGG-19 models according to the layer ranking results we received. Sometimes pruning earlier layers from a network impact on the sensitivity of deeper layers since deleting any filter from earlier layer changes the structure of the next layers. For example, we discovered that when pruning from the first eight layers of VGG-19 model, the sensitivity of the last eight layers was damaged as well. That is why we pruned the VGG-19 only from the last eight layers which gave us the best results.

TABLE II. SUMMARY OF VGG-19 ARCHITECTURE ON CIFAR-10 DATASET

Layer		***#Parameters***	***Size***	***Activation***
Input	Image	0	32x32x3	-
1	Convolution	1792	32x32x64	relu
2	Convolution	36928	32x32x64	relu
	Max Pooling	0	16x16x64	relu
3	Convolution	73856	16x16x128	relu
4	Convolution	147584	16x16x128	relu
	Max Pooling	0	8x8x128	relu
5	Convolution	295168	8x8x256	relu
6	Convolution	590080	8x8x256	relu
7	Convolution	590080	8x8x256	
8	Convolution	590080	8x8x256	
	Max Pooling	0	4x4x256	relu
9	Convolution	1180160	4x4x512	relu
10	Convolution	2359808	4x4x512	relu
11	Convolution	2359808	4x4x512	
12	Convolution	2359808	4x4x512	
	Max Pooling	0	2x2x512	relu
13	Convolution	2359808	2x2x512	relu
14	Convolution	2359808	2x2x512	relu
15	Convolution	2359808	2x2x512	
16	Convolution	2359808	2x2x512	
	Max Pooling	0	1x1x512	relu
17	FC1	2101248	4096	relu
18	FC	16781312	4096	relu
Output	FC	40970	10	Softmax
Total parameters		38,947,914		

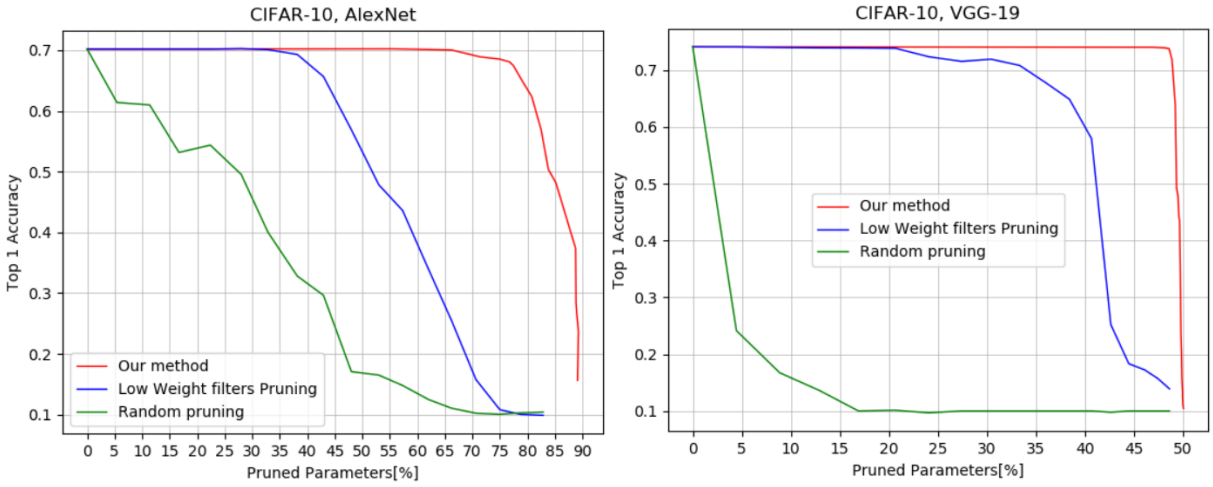
After trying our algorithm on AlexNet and VGG-19 with CIFAR10 dataset we managed to prune a significant percentage of the network's parameters without any affect on the accuracy while testing on a test batch. These results combine the usage of both algorithms, Algorithm 1 for the Fully Connected layers and Algorithm 2 for the Convolutional layers. Figure 4 shows the results of using our method on those two models. On AlexNet we have managed to prune over 67% of the network's parameters without any accuracy drop and about 75% with a minor accuracy drop (under 3%). On a bigger model such as VGG-19 we managed to prune with our method 48% of the network's parameters, as well, without any accuracy drop. While the other method [20] we compared with did not manage to prune this specific network by more than 30% of its parameters before the performance of the network was harmed. For both the models, pruning filters randomly from the same layers gave very poor results comparing to the two methods, that contrary to some previous publishes that suggested using random pruning [1].

Our experiments were by trial and error, after trying many different ways to prune our models, we came up with the best results we managed to receive, these results were obtained under the requirement that the decrease in the network's accuracy performance will not exceed 1%, or in other words,

TABLE III. OVERALL RESULTS. THE ACCURACY AND THE PRUNING RATE WHEN USING OUR METHOD ON ALEXNET AND VGG-19 MODELS.

Model	***Dataset***	***#Parameters***	***Memory size [MB]***	***Size out of the original size %***	***Top 1 Accuracy***	***Accuracy drop (%)***
AlexNet	CIFAR-10	2,248,234	8.82	100%	0.7018	-
AlexNet Prunned		720,832	2.84	32.2%	0.6997	0.30%
VGG-19		38,947,914	152.2	100%	0.7409	-
VGG-19 Pruned		20,029,575	78.3	51.44%	0.7361	0.64%

almost no accuracy drop at all. Table 3 shows in details the results obtained. For AlexNet we managed to reduce the network's size to 32.2% of its original size without significant damage to the network's accuracy. For VGG-19 model which is a much bigger network, we were able to reduce its size to 51.4% of its original size also with almost no accuracy drop at all.



(a) Prune AlexNet, CIFAR10 (b) Prune VGG19, CIFAR10

Fig. 4. (a) Pruning AlexNet once with our method, then with low weight filters pruning- the method from "Pruning Filters" in related work section[13] and lastly randomly pruning of the same layers. (b) Pruning VGG19 with our method the "Pruning Filters" method and randomly pruning. x axis is the percentage of the pruned parameters out of the network's parameters and y axis refers to the top 1 accuracy.

V. Conclusion

Modern CNNs may have over 50% redundancy parameters. In order to reduce the network's size, in this paper, we present a new method for pruning those parameters from the networks that share the same information, we define them as "Identical Neurons" and "Identical Filters" respectivly. We give a new approach for pruning - pruning the network by focusing on the Neurons and Filters outputs for a prediction on a test/validation-set instead of focusing on the network's weights values. Our method achieved about 48% reduction in memory size for VGG-19 model and 67% reduction for AlexNet model both on CIFAR-10 dataset and without significant loss of the original accuracy. By performing lesion studies on deep CNNs, we identify layers that are robust or sensitive to pruning, which can be useful for further understanding and improving future CNN architectures.

References

[1] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding." arXiv preprint arXiv:1510.00149 (2015).

[2] Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems. (2015).

[3] Molchanov, Pavlo, et al. "Pruning convolutional neural networks for resource efficient inference." arXiv preprint arXiv:1611.06440 (2016).

[4] Ullrich, Karen, Edward Meeds, and Max Welling. "Soft weight-sharing for neural network compression." arXiv preprint arXiv:1702.04008 (2017).

[5] Ding, Xiaohan, et al. "Approximated oracle filter pruning for destructive cnn width optimization." arXiv preprint arXiv:1905.04748 (2019).

[6] Suzuki, Kenji, Isao Horiba, and Noboru Sugie. "A simple neural network pruning algorithm with application to filter synthesis." Neural processing letters 13.1 (2001): 43-53.

[7] Qassim, Hussam, Abhishek Verma, and David Feinzimer. "Compressed residual-VGG16 CNN model for big data places image recognition." 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2018.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

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

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learn- ing for image recognition. In Proceedings of the IEEE Con- ference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[12] Huang, Qiangui, et al. "Learning to prune filters in convolutional neural networks." 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018.

[13] Augasta, M. Gethsiyal, and T. Kathirvalavakumar. "A novel pruning algorithm for optimizing feedforward neural network of classification problems." Neural processing letters 34.3 (2011): 241.Alex

[14] Krizhevsky's main websit: www.cs.toronto.edu/~kriz/index.html

[15] Molchanov, Pavlo, et al. "Importance estimation for neural network pruning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[16] Blalock, Davis, et al. "What is the state of neural network pruning?." arXiv preprint arXiv:2003.03033 (2020).

[17] Lin, Shaohui, et al. "Towards optimal structured cnn pruning via generative adversarial learning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[18] Luo, Jian-Hao, et al. "Thinet: pruning cnn filters for a thinner net." *IEEE transactions on pattern analysis and machine intelligence* 41.10 (2018): 2525-2538.

[19] Luo, Jian-Hao, and Jianxin Wu. "An entropy-based pruning method for cnn compression." arXiv preprint arXiv:1706.05791 (2017).

[20] Li, Hao, et al. "Pruning filters for efficient convnets." arXiv preprint arXiv:1608.08710 (2016).