D-Pruner: Filter-Based Pruning Method for Deep Convolutional Neural Network

Loc N. Huynh, Youngki Lee, Rajesh Krishna Balan
Singapore Management University
{nlhuynh.2014, youngkilee, rajesh}@smu.edu.sg

ABSTRACT

The emergence of augmented reality devices such as Google Glass and Microsoft Hololens has opened up a new class of vision sensing applications. Those applications often require the ability to continuously capture and analyze contextual information from video streams. They often adopt various deep learning algorithms such as convolutional neural networks (CNN) to achieve high recognition accuracy while facing severe challenges to run computationally intensive deep learning algorithms on resource-constrained mobile devices. In this paper, we propose and explore a new class of compression technique called *D-Pruner* to efficiently prune redundant parameters within a CNN model to run the model efficiently on mobile devices. *D-Pruner* removes redundancy by embedding a small additional network. This network evaluates the importance of filters and removes them during the fine-tuning phase to efficiently reduce the size of the model while maintaining the accuracy of the original model. We evaluated *D-Pruner* on various datasets such as CIFAR-10 and CIFAR-100 and showed that *D-Pruner* could reduce a significant amount of parameters up to 4.4 times on many existing models while maintaining accuracy drop less than 1%.

Keywords

Continuous Vision, Deep Learning, Compression;

1. INTRODUCTION

The appearance of augmented reality devices such as Google Glass and Microsoft Hololens has been opening up various new vision sensing applications. The core function of these applications is to continuously capture contexts of users and surroundings from streaming video data and enable situational interactions with users. For example, a virtual assistant system for dementia patients identifies objects and people near to the patient and provide the patient with the intelligent guidance in real-time [2]. Recently, deep learning algorithms such as a convolutional neural networks (CNN) have been actively adopted for various computer vision tasks such as image recognition, object detection, and identification tasks to achieve higher recognition accuracy [7, 20, 22].

EMDL'18, June 15, 2018, Munich, Germany

DOI: https://doi.org/10.1145/3212725.3212730

The key challenge to enable continuous vision applications is to run the state-of-the-art CNN models efficiently on resource-constrained mobile devices. Recent CNN models such as VGG-16 [20], ResNet [7], and Inception [22] often require a huge amount of computational resources regarding CPU/GPU cycles or memory usage, making their execution slow on mobile devices. For instance, VGG-16 and ResNet-152 require 15.3 GLOPS and 11.6 GLOPS to recognize a single image, which often takes at least hundreds of milliseconds on the commodity smartphones [9, 10, 14]. To address this problem, cloud offloading is often considered. However, the offloading approach has critical privacy concerns as it may expose a massive volume of private images and videos of users to the cloud.

Previous works [1, 5, 6, 11, 24] have shown that CNNs usually have a lot of redundancy in terms of filters and parameters. The problem is further aggravated since developers often leverage *transfer learning* [19] to fine-tune the state-of-the-art models on new datasets to increase recognition accuracy. For example, the first 13 convolutional layers in VGG-16 can be used to provide robust features for a variety of new tasks such as classifying different types of fruits or animals which are not available in the ImageNet dataset [4]. Developers can attach a few additional layers on top of the existing 13 layers to fine-tune the network on new datasets. In many cases, if we don't process it carefully, transfer learning makes the model unnecessarily large and redundant to run on mobile devices.

Compression of the neural networks has been actively studied for efficient execution of deep neural networks. Some works [1, 5] focus on approximating each layer separately via factorization techniques and fine-tune the whole network to restore accuracy. However, without global knowledge about relationships between lower and upper filters, independent pruning of filters might lead to significant loss in recognition accuracy. Other works [6] remove parameters based on their magnitudes during end-to-end fine-tuning. Unfortunately, the weight matrices become sparse and make it hard to leverage highly optimized libraries such as OpenBlas/ClBlast [18, 23] to perform inference efficiently on mobile devices. Recently, Yao et al. proposed *DeepIoT* system that used Recurrent Neural Network (RNN) to compress the models. However, RNN is prone to gradient vanishing problem and may not work well with very deep networks.

In this paper, we propose a general technique called *D-Pruner* to reduce the memory footprint and computational cost of many existing and transferred CNN models. *D-Pruner* automatically figures out redundant filters in convolutional layers and removes them to make the model smaller in terms of memory and computational requirements. Its key idea is to embed a small network called *masking layer* into every convolution layer to score how effectively each filter contributes to the outcome. *Masking layers* removes only low

scored filters and fine-tune the network to keep the accuracy while pruning out the unnecessary filters. By learning the extended network end-to-end, *D-Pruner* can figure out the relationship between filters and make a better pruning decision.

We conducted several experiments on two different datasets (CIFAR-10 and CIFAR-100 [12]) to evaluate *D-Pruner*. Our results show that *D-Pruner* can compress existing models to be 4.4× and 2.76× smaller in terms on memory footprint, 4.57× and 2.9× better in term of computational cost on CIFAR-10 and CIFAR-100 respectively. In our latency tests, pruned models on CIFAR-10 and CIFAR-100 achieve the speedup of 1.85× and 1.61× on Samsung Galaxy S7 device. Furthermore, *D-Pruner* achieves 8% smaller in size with accuracy of 90.48% comparing to pruned VGGNet with accuracy of 90.5% as proposed in *DeepIoT* [24] on CIFAR-10. We believe that mobile developers would be beneficial from *D-Pruner* to build small and efficient CNN models for many vision sensing tasks.

The contribution of our paper can be summarized as follows:

- We propose *D-Pruner* , a simple but effective compression technique to remove redundancy within existing and transferred CNN models. *D-Pruner* introduces a novel concept of the masking block to figure out redundant filters which have low impacts on final accuracy.
- We leverage the knowledge from the training set to effectively remove only a subset of redundant filters to maintain accuracy at the highest level.
- We conducted intensive experiments using two different datasets on two network architectures to demonstrate the usefulness of *D-Pruner*. Our results on CIFAR-10 and CIFAR-100 [12] show that *D-Pruner* can compress existing models to be 4.4× and 2.76× smaller in terms on memory footprint, 4.57× and 2.9× better in term of computational cost on CIFAR-10 and CIFAR-100 respectively. In our latency evaluation, pruned models on CIFAR-10 and CIFAR-100 achieve the speedup of 1.85× and 1.61× on Samsung Galaxy S7 device.

2. RELATED WORK

Mobile Deep Learning: There has been many prior works to migrate deep learning models onto mobile devices. *DeepEar* [15] was the first framework to support running deep neural network models on mobile CPUs and DSPs for audio sensing tasks. Afterwards, many other frameworks such as *DeepX* [14], *DeepSense* [9] and DeepMon [10] have been proposed with various optimizations for mobile vision applications. However, those focus primarily on optimizing the processing pipeline to make existing models run faster and more energy-efficient. For instance, *DeepX* tries to split computations between CPUs and low power DSPs to reduce energy consumption. *DeepSense* and *DeepMon* take another approach by doing low-level optimizations to utilize the powerful mobile GPUs to execute computations in parallel. Unlike those systems, *D-Pruner* focuses on optimizing the models to be smaller in terms of memory footprint and computational cost so that they can be run efficiently on any systems above.

Compression Techniques: General optimizations to compress deep learning models are widely studied for both servers and mobile devices. Denton et al. used matrix factorization to approximate the weights within CNN [5] to reduce its parameters. Similarly, Lane et el. also used matrix factorization to reduce the number of operations within dense layers to improve inference time [1]. In those approaches, each layer is approximated separately and combined for final fine-tuning step. However, there is no global information sharing between approximations to make sure that information loss in the first layer will have no effect on second layer. Hence, accuracy drop would be significant if we prune large amount of parameters. Han et al. proposed a method to prune network connection based on magnitude of parameters during fine-tuning phase [6]. However, weights matrix becomes sparse after pruning process and makes it hard to leverage optimized library to execute inference step efficiently.

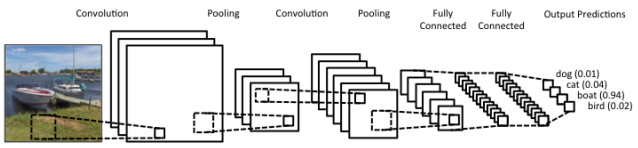

Figure 1: Convolutional Neural Network Architecture

Yao et al. proposed *DeepIot* system that leverages recurrent neural network (RNN) to learn the relationship between parameters across many layers and prune the redundancy automatically [24]. However, RNN is prone to gradient vanishing problem and may not work well if the input sequence is too large. In this case, *DeepIoT* may not work well with state-of-the-art models such as ResNet or Inception network. *D-Pruner* shares the similar concept by learning global relationship end-to-end and automatically pruning unnecessary parameters. However, *D-Pruner* solves the potential of gradient vanishing problem by integrating a novel masking blocks directly to CNN model and fine-tune the whole network without any needs of RNN.

3. CONVOLUTIONAL NEURAL NETWORK

Since AlexNet architecture was proposed in 2012 [13], there have been many significant changes in the first network architecture (Figure 1) to improve the capabilities of CNN on many computer vision tasks. One interesting change is the replacement of fully connected layers or dense layers by [1×1] convolutional layer and global average pooling in many state-of-the-art models such as ResNet [7], Inception network [22]. As dense layers consume the most parameters in CNN [9], this change significantly reduces the size (or memory footprint) of state-of-the-art models. However, as modern networks still rely heavily on convolutional layers to extract meaningful visual features, high computational cost is still an open problem [9].

There are two widely used methods to reduce computational cost in CNN. The first method is to use factorization techniques such as SVD (singular-value decomposition) to approximate the weights matrices during inference step to reduce the total processing operations. However, this approach tends to have high accuracy loss on very deep networks [1, 14]. The second method is to prune the redundant filters to achieve simpler but more efficient CNNs. As the computational cost is proportional to the number of filters, pruning unnecessary filters will result in improving both training and inference time. Many works have shown potential results using this approach [11, 24].

D-Pruner follows the latter approach by recognizing redundancy automatically during fine-tuning process. *D-Pruner* is designed as a general technique to compress any modern CNN models to be smaller and less resource-consuming to work efficiently on both servers and mobile devices.

4. D-PRUNER ALGORITHM

In this section, we first introduce briefly how the technique works. Secondly, we provide details about our novel masking block to determine removable filters. Finally, we show how the training

process takes place to prune unnecessary parameters based on the knowledge from masking blocks.

The algorithm works in multiple pruning iterations. In each iteration, we first expand all convolutional layers with extra layers called masking blocks to score how much each filter impacts on final accuracy. Each masking blocks will output a set of candidate filters to be removed for each particular image input. In order to prune only filters that have little impacts on the outcome, we leverage the all training images to collect the probability to be removed of each filter. We only remove those with high probability of being removed (e.g. over 95% on training set). We then fine-tune the new network to recover original accuracy and achieve a smaller model. Finally, we repeat the pruning process again until it converges (e.g. accuracy drop is above certain threshold.).

4.1 Masking Block

The goal of masking block is to determine removable filters during the pruning process. For example, fine-tuning ImageNet models such as VGG-16 or ResNet to detect multiple types of fruits might contain a lot of redundant filters to recognize animals, which can be removed to make the model smaller and simpler. By attaching masking block to convolutional layer, it will inspect the output of every filter and score how effectively they affect the final outcome. Hence, unnecessary filters may be removed if they have no or little impact on the final accuracy. Furthermore, masking block incurs very small computational overhead and should be easily fine-tuned.

Our masking block is inspired by SE block in Squeeze-and-Excitation network [8] which is used to measure the importance of each filter within a single convolutional layer. We leverage SE block and add masking function in order to filter out top-K unimportant filters.

Masking block as shown in Figure 2 consists of an average pooling followed by 2 dense layers and a softmax layer to compute the score of each particular filter. Afterwards, the masking layer takes the scores, a maximum number of filters K to be removed and outputs the binary masks which zero out top-K lowest scores. At the end, we multiply the masks and previous output of convolutional layer to remove all unnecessary outputs corresponding to removed filters. Hence, only remaining output will contribute to the final outcome during fine-tuning process.

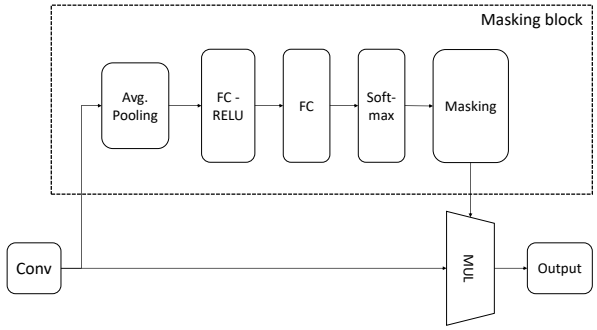

Figure 2: Masking Block

4.2 Pruning Method

The pruning process consists four main stages as described in Algorithm 1.

- Firstly, we attach masking blocks to original network as shown in Figure 2 and fine-tune the network on training set (line 4). We fix the original network and train only the masking blocks for first few epochs and then fine-tune the whole network for few more epochs afterwards. (line 5)
- Secondly, we predict which filters should be removed within the final network architecture. As the masking block outputs a set of removable filters for every single image input, one filter can be removed for a particular input but can be preserved for another. We collect the removal distribution of each filter on the all training images and only remove the filters that have removable probability higher than predefined threshold (e.g. 95%) (line 6-14). For instance, the first convolutional layer of VGG-16 has 64 filters. If we use K=10 filters, during the training phase, masking block will automatically zero-out all the output of 10 filters with lowest scores on each training image. Only 54 remaining filters will contribute to the final output. However, masking block does not always produce the same 10 filters for every training image. In order to make correct pruning decision, we use all training images to collect the probability to be removed of all 64 filters and remove only those have higher than a threshold T.
- Finally, we build a new network by removing masking blocks and removed filers from previous step (line 15). We transfer the learned parameters to the new network and fine-tune it for few epochs to recover original accuracy (line 16-17).
- We update the new model if validation accuracy is within affordable range (line 18-21) and repeat the pruning process until we satisfy with the result or final accuracy drops below a certain threshold (line 3).

Algorithm 1 Pruning Algorithm

Data: Network O, Dataset D, const K, theshold T, epochs N
Result: Network P

```
1  acc ← acc(O)
2  P ← Network O
3  while acc ≥ expected_accuracy do
4      P' ← Network P + {masking blocks}
5      Finetune P' on D for N epochs
6      R ← { }
7      for ∀ masking block l in P' do
8          for ∀ filter f in l do
9              Pr_f ← P(mask(f) == 1 | D)
10             if Pr_f ≤ T then
11                 R ← R ∪ {f}
12             end
13         end
14     end
15     P'' ← Network P - R
16     Transfer learned paraemters from P' to P''
17     Finetune P'' on D for N epochs
18     if acc(P'') ≥ acc then
19         acc ← acc(P'')
20         P ← P''
21     end
22 end
```

5. EXPERIMENTS

5.1 Experiment Setup

Datasets. We evaluated *D-Pruner* by compressing existing models on two datasets: CIFAR-10 and CIFAR-100 [12]. Each dataset consists of 60.000 32x32 color images (50.000 images for training and 10.000 images for validation). CIFAR-10 and CIFAR-100 contains images in 10 and 100 classes respectively.

	CIFAR-10						CIFAR-100		
	ALL-CNN-C	ALL-CNN-C(*)	Impr.	NIN	NIN(*)	Impr.	ALL-CNN-C	ALL-CNN-C(*)	Impr.
Accuracy(%)	90.19	89.34	-0.85	89.39	88.83	-0.44	61.71	61.08	-0.63
# Parameters	1.3M	310K	4.4×	966K	348K	2.77×	1.3M	501K	2.76×
# Mul-Add Operations	281M	61M	4.57×	222M	132M	1.68×	282M	97M	2.9×
Latency(ms)	211(±8)	113(±14)	1.85×	185(±25)	131(±11)	1.41×	208(±11)	129(±14)	1.61×

(*): pruned model
Impr.: Improvement

Table 1: Overall Performance of *D-Pruner*

	CIFAR-10				
	ALL-CNN-C	ALL-CNN-C(*)	ALL-CNN-C(**)	VGGNET	VGGNET-DEEPIOT
Accuracy(%)	90.19	**90.48**	89.34	90.5	**90.5**
Number of Parameters	1.3M	**664K**	310K	29.7M	**724K**

(*): Pruned model at 4th iteration (**): Final pruned model

Table 2: Comparison with *DeepIoT*

	Filter Shape	
Type / Stride - Activation	**ALL-CNN-C**	**NIN**
Conv1 / s1 - ReLU	3×3×3×96	5×5×3×192
Conv2 / s1 - ReLU	3×3×96×96	1×1×192×160
Conv3 / s2 - ReLU	3×3×96×96	1×1×160×96
Conv4 / s1 - ReLU	3×3×96×192	5×5×96×192
Conv5 / s1 - ReLU	3×3×192×192	1×1×192×192
Conv6 / s2 - ReLU	3×3×192×192	1×1×192×192
Conv7 / s1 - ReLU	3×3×192×192	3×3×192×192
Conv8 / s1 - ReLU	1×1×192×192	1×1×192×192
Conv9 / s1 - ReLU	1×1×192×10	1×1×192×10
Global Average Pool / s1		
Softmax		

Table 3: Network Architectures

Models. We trained the ALL-CNN-C model from [21] which achieves accuracy of 90.19% on CIFAR-10 and 61.71% on CIFAR-100. In order to show the robustness of *D-Pruner* on variety of architectures, we also trained NIN (network in network) model from [16] which achieves accuracy of 89.39% on CIFAR-10 for further evaluations. Unlike other models such as VGGNet [24] that use dense layers for classification, both networks in our evaluations use only convolutional layers which results in fewer number of parameters while achieving similar accuracy. ALL-CNN-C and NIN uses approximately about 281M and 222M Mul-Add operations respectively. Network architectures of ALL-CNN-C and NIN models on CIFAR-10 are shown in Table 3.

Training process. We used Keras [3] in *D-Pruner's* implementation. For every pruning step, we tried to remove $K = 20\%$ of the filters and fine-tuned the network for $N = 35$ epochs (10% of number of epochs we used to train original network). We used Nesterov Gradient Descent [17] for fine-tuning with learning rate, momentum and decay set to 0.01, 0.9 and 0.000001 respectively. We also used threshold T of 0.95 to determine which filter will be removed. We repeated the pruning process for several iterations until there was no filter to be removed or the expected accuracy loss was larger than 1%.

Metrics. We use accuracy, number of parameters, amount of mul-add operations and processing latency as our key performance metrics. For latency evaluation, we evaluated pruned models using *DeepMon* framework [10] and report the average latency on Samsung Galaxy S7 (with Exynos 8890 processor and Mali-T880 GPU).

5.2 Overall Results

Overall, *D-Pruner* successfully compresses investigated models to be much smaller and less computational consuming. Table 1 shows the performance of pruned versions of ALL-CNN-C and NIN models on both CIFAR-10 and CIFAR-100.

On CIFAR-10, *D-Pruner* easily compress both ALL-CNN-C and NIN models to be 4.4× and 2.77× smaller in memory footprint (in terms of number of parameters). It also reduces 4.57× and 1.68× computational cost (in terms of the amount of require Mul-Add operations) in ALL-CNN-C and NIN models respectively. We notice that performance of *D-Pruner* on ALL-CNN-C model is significantly higher than on NIN network due to several reasons. First, original NIN model has 1.34× less number of parameters comparing to ALL-CNN-C model which makes reduction in memory footprint seem to be lower. Second, NIN network leverages [1×1] convolutional filter which results in significantly reduction in computational cost, which explains why computation cost is reduced only 1.68× while memory footprint is reduced 2.77×. In latency evaluations, pruned models from ALL-CNN-C and NIN networks improves inference time up to 1.85× and 1.41× respectively.

Similarly, pruned version of ALL-CNN-C achieves 2.76×, 2.9× and 1.61× reduction in memory footprint, computational cost and inference time on CIFAR-100.

5.3 Performance Breakdown

Next, we investigate how *D-Pruner* affects the models during each pruning iteration in terms of accuracy, amount of parameters, number of Mul-Add operations and the amount of filters in each convolutional layer using results from pruning ALL-CNN-C model. In general, giving the expected accuracy drop, *D-Pruner* gradually compresses the model by pruning unnecessary filters over various iterations and makes it smaller in terms of memory footprint and computational cost while trying its best to maintain the highest accuracy.

5.3.1 Impacts on Accuracy

We now investigate the impact of *D-Pruner* on the final accuracy. Figure 3 shows the accuracy of pruned models during multiple pruning iterations on both CIFAR-10 and CIFAR-100. We achieve accuracy of 89.34% and 61.08% comparing to 90.19% and

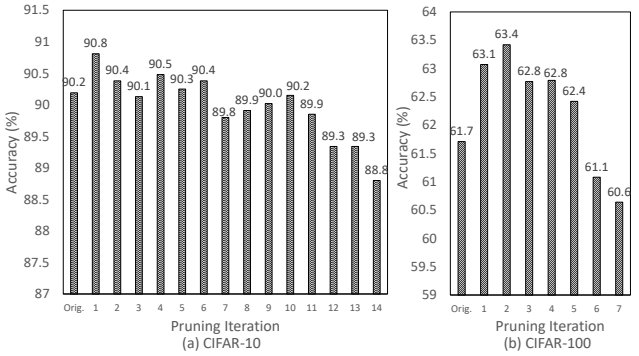

Figure 3: Accuracy

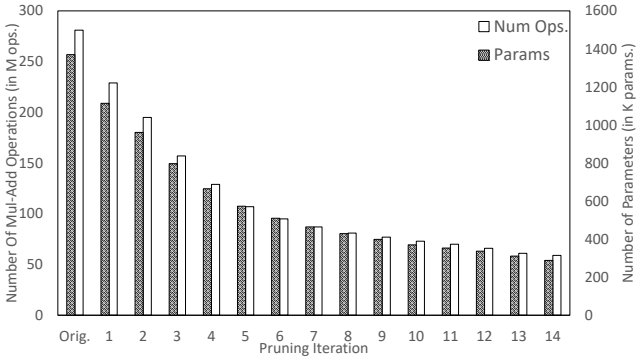

Figure 4: Parameters and Operations Reduction on CIFAR-10

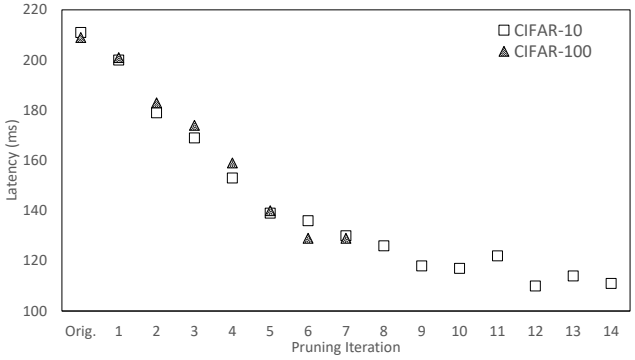

Figure 5: Latency

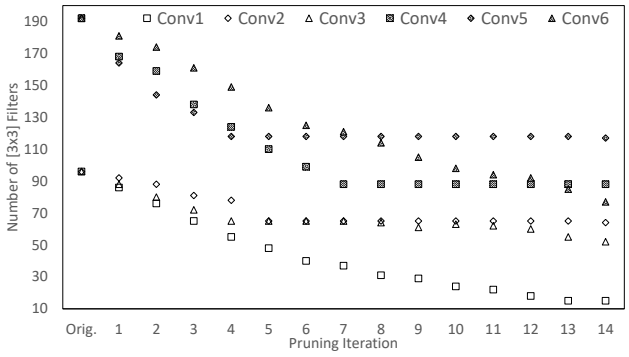

Figure 6: Number of Filters per Iteration on CIFAR-10

61.71% from original models after 13 and 6 pruning iterations on CIFAR-10 and CIFAR-100 respectively.

Firstly, we notice that it takes us 14 and 7 iterations to make the accuracy loss above 1% threshold on CIFAR-10 and CIFAR-100. This implies that the original models tend to have a lot of redundancy and *D-Pruner* can effectively prune them without significant loss in the final accuracy.

Secondly, we figure out that accuracy increases for the first few iterations which indicates that the some redundancy negatively affects the accuracy. Hence, *D-Pruner* can be used to slightly improve accuracy by eliminating most negatively redundant parameters.

Finally, we also want to note that the pruning process converges faster on CIFAR-100 than CIFAR-10. As we use same architecture on both tasks, it is understandable that classifying 100 classes requires more network capacity in terms of filters and parameters than classifying 10 classes.

5.3.2 Impacts on Parameters and Operations

Next, we investigate on how many parameters and number of operations *D-Pruner* can prune during each iteration. Figure 4 shows that both the number of parameters and operations gradually decrease during pruning process. At 13th iteration, we achieve 4.4× and 4.57× reduction in number of parameters and Mul-Add operations on CIFAR-10. As *D-Pruner's* optimization is to reduce the number of filters during each pruning iteration, both parameters and number of operations would always decrease during the pruning process.

Similarly, we also see the same trend on CIFAR-100 dataset which results in 2.76× and 2.9× improvement on model's parameters and number of Mul-Add operations.

5.3.3 Impacts on Latency

We also explore the performance of pruned models on existing mobile deep learning frameworks. Figure 5 shows the latency per pruning iteration on both datasets using DeepMon framework. We achieve the speedup of 1.85× and 1.61× on CIFAR-10 and CIFAR-100 respectively.

However, we notice that the latency slowly decreases after 9th iteration. One reason is that number of operations per layer become too small to make DeepMon utilize GPU resources efficiently. However, batching multiple images as input could improve the average inference time.

5.3.4 Impacts on Number of Filters

Finally, we investigate the reduction of filters during the pruning process on CIFAR-10. We plot the amount of filters within two first blocks in ANN-CNN-C model which consist the first 6 convolutional layers as shown in Figure 6. Each block consists of 3 convolutional layers which have 96 and 192 filters respectively. Both blocks end with spatial dimension reduction using convolutional layer with stride is set to 2 instead of using Max-Pooling.

Interestingly, two blocks share the same trend in filters reduction. The first and last convolutional layers within the block stop reducing after a certain threshold while the middle layer keeps reducing during pruning process. This shows some insights for us the build better network architecture where last layer inside a block should have few filters comparing to previous layers.

5.4 Comparisons with DeepIoT

We compare our pruned models with compressed VGGNet from *DeepIoT* [24] on CIFAR-10 dataset as shown in Table 2. At 4th iteration, *D-Pruner* provides a model with 8% less parameters than *DeepIoT's* model while achieving comparable accuracy (90.48% vs 90.5%), even though we start with less accurate model. If we are willing to sacrifice 1.16% (comparing to *DeepIoT*), we will achieve 2.33× smaller model.

We also notice that *DeepIoT* leverages recurrent neural network (RNN) to prune the parameters. However, RNN is prone to gradient vanishing problem and may not work well in very deep neural network such as ResNet or Inception network. Instead, *D-Pruner* 's masking blocks can be easily integrated into CNN and can be trained at ease.

6. DISCUSSION

Even though *D-Pruner* produces good preliminary results, there are still plenty of works that need to be done in the near future.

Number of Removable Filters: *D-Pruner* has to do pruning in several iterations to search for desirable model. Unfortunately, pruning steps might take long time to process. One approach to solve that problem is to increase the amount of removable filters K during each pruning process. However, if K is not chosen carefully, target model will lack of parameters to learn meaningful features which will result in great accuracy drop. An approach to automatically determine the number K during fine-tuning process should be taken in consideration to make *D-Pruner* work more efficiently.

Improving Accuracy: As CNN often has a lot of redundant parameters, some of them might have negative impact on the final outcome or make the model unstable. During our evaluations, we showed that accuracy can be improved by eliminating some parameters. This can be explored further to make pruned model not only smaller but also more accurate.

7. CONCLUSION

In this paper, we propose *D-Pruner* , a simple and efficient compression technique to simplify existing CNN models in order to make them work more efficiently on mobile devices without loss in accuracy. On CIFAR-10, our evaluations on existing CNN models show that *D-Pruner* can reduce 4.4× in terms of memory footprint and 4.57× in terms of computational cost with less than 1% accuracy drop. Furthermore, compressed model runs 1.85× faster on Samsung Galaxy S7 comparing to original model during our inference latency evaluations.

8. REFERENCES

[1] S. Bhattacharya and N. D. Lane. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 176–189. ACM, 2016.

[2] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, and A. Mihailidis. A planning system based on markov decision processes to guide people with dementia through activities of daily living. *IEEE Transactions on Information Technology in Biomedicine*, 10(2):323–333, 2006.

[3] F. Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 7:8, 2015.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[5] 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*, pages 1269–1277, 2014.

[6] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[9] L. N. Huynh, R. K. Balan, and Y. Lee. Deepsense: A gpu-based deep convolutional neural network framework on commodity mobile devices. In *Proceedings of the 2016 Workshop on Wearable Systems and Applications*, pages 25–30. ACM, 2016.

[10] L. N. Huynh, Y. Lee, and R. K. Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95. ACM, 2017.

[11] 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. *arXiv preprint arXiv:1511.06530*, 2015.

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

[13] 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.

[14] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, pages 1–12. IEEE, 2016.

[15] N. D. Lane, P. Georgiev, and L. Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 283–294. ACM, 2015.

[16] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[17] Y. Nesterov et al. Gradient methods for minimizing composite objective function, 2007.

[18] C. Nugteren. Clblast: A tuned opencl blas library. *arXiv preprint arXiv:1705.05249*, 2017.

[19] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

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

[21] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[22] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[23] Z. Xianyi, W. Qian, and Z. Chothia. Openblas, version 0.2. 8. *URL http://www. openblas. net/. Fe tched*, page 09, 2013.

[24] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems. ACM*, 2017.