Run Time Adaptive Network Slimming for Mobile Environments

Hong Ming Chiu, Kuan-Chih Lin, and Tian Sheuan Chang
Dept. Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan
h15731573@gmail.com, philip23582358@gmail.com, tschang@mail.nctu.edu.tw

***Abstract*— Modern convolutional neural network (CNN) models offer significant performance improvement over previous methods, but suffer from high computational complexity and are not able to adapt to different run-time needs. To solve above problem, this paper proposes an inference-stage pruning method that offers multiple operation points in a single model, which can provide computational power-accuracy modulation during run time. This method can perform on shallow CNN models as well as very deep networks such as Resnet101. Experimental results show that up to 50% savings in the FLOP are available by trading away less than 10% of the top-1 accuracy.**

I. Introduction

Many state-of–the-art convolutional neural network models have large structures for obtaining higher accuracy while doing tasks. These structures take lots of computation energy and occupy lots of memory space, which is not suitable for running on resource-limited portable devices. However, many artificial intelligence (AI) applications based on CNN models are designed for mobile applications, where the low power consumption and real time requirement are the two major constraints when running these large models.

One common way to solve above problem is the model compression through pruning on individual weights, channels, or filters[1][2] by eliminating those candidates smaller than a certain threshold. The individual weight pruning results in fine grained zero distribution, which is not good or even harmful for parallel computation datapaths in modern GPUs or CNN hardware accelerators. A more appropriate one is the channel or filter level pruning, which is more regular and easy for hardware speedup[3].

However, all these model compression methods decide the model at the design time, which needs original dataset to retrain models. The original dataset may not be available due to privacy or commercial interest concerns. Besides, the design time pruning cannot adapt to different run time requirements, especially at the mobile environments. For example, for face recognition, the device just needs a rough but low power face recognition to identify faces or not before locking the focus, and high accurate but also higher complexity face recognition to identify the person. Besides, the computing resource of the mobile system-on-chip (SoC) could change a lot due to different task allocation and priority or dynamic voltage and frequency scaling. In such case, how to meet real time constraints under these changing computing resource is a major concern for the execution of the CNN models. Thus, it would be convenient for the CNN model execution to provide power-accuracy modulation during runtime.

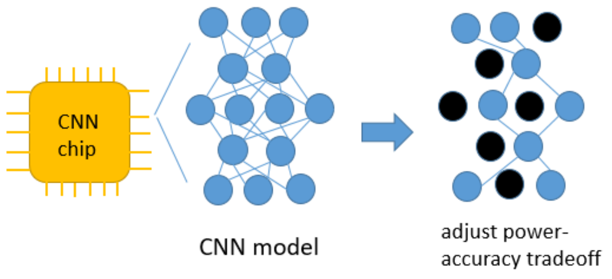

Fig. 1. Objective of our pruning method

The run time dynamic execution has been studied to explore the easy or hard input to decide how many layers shall be executed[4][5] without quality loss. The drawback of this approach is not able to adapt the model to application needs. Another approach to solve above problem is the DrowsyNet[6], which randomly drops out inference-stage convolutional neurons to support runtime power-accuracy tunability. However, such approach takes a long time to find the appropriate combinations due to its randomness and has significant performance drop for the Top-1 accuracy.

Motivated by above needs, this paper proposes a run time adaptive network slimming by a simple inference-stage model pruning to adapt to different execution needs as illustrated in Fig. 1. It reinforces CNN hardware with function that can adjust CNN models suitable for different circumstances. Our method is different from DrowsyNet. It sets a regulation for pruning instead of random probability. Furthermore, it has lower accuracy loss and can apply to deep CNN models without consuming too much computation power. Experimental results show that up to 50% savings in the number of FLOP are available by trading away less than 9% of top-1 accuracy.

II. Background and Related Works

Fig. 2 illustrates how convolution and weight pruning works in the CNN. There are nine weights in a 3×3 size filter, and the function of a filter is to extract a feature of a picture. A neuron in a convolution layer has nine synapses and each connects to a input pixel. The input will multiply the weight and sum together in the neuron. Pruning methods manage to mask the neuron, letting its output equals to zero, that is, sets a 3×3 mask matrix, which elements are all zero, multiplies with the filter, making the weights become zero. The output of a neuron indicates the similarity between image pixels and the feature which the filter extracts. Therefore, there are some filters whose weight values are small, making the output of the neuron not indicate the similarity apparently. Thus, we can treat these filters as unimportant and mask them in order to save computation effort.

There are different kinds of masking ways such as masking whole filter, masking a kernel, masking a vector of a kernel, or even masking an element of a kernel. Each way has pros and cons, for example, making whole filter can maintain regular sparsity for hardware to accelerate more easily, but has worse masking precision and may lose some important kernels. On the contrary, masking an element of a kernel can have better precision, but the irregular sparsity of kernel matrix can hinder the acceleration of hardware. In this paper, we adopt filter pruning, which is the most regular and hardware friendly.

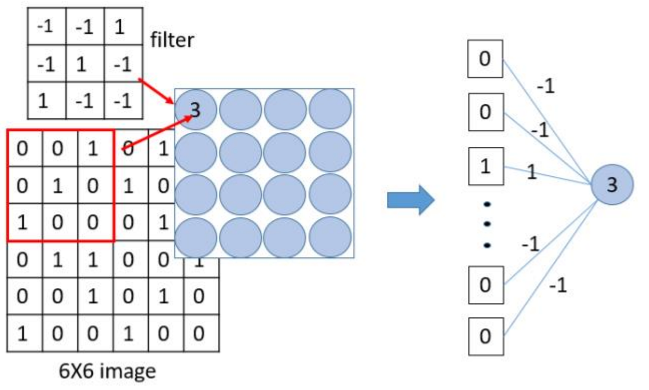

Fig. 2. Illustration of convolution

There are two different types of pruning method. One is performed during the training stage, with repeatedly retraining to recover the accuracy. The other is performed in the inference stage without retraining and just being performed once. The former has many prior strategies such as Molchanov's method [2] that proposes a criterion based on Taylor expansion, or Deep compression [1] that prunes weights that are close to zero and constructs a three stage pipeline: pruning, trained quantization and Huffman coding. These methods can achieve high compression ratio of CNN models with very small accuracy loss, but they cannot provide flexible modulation between power and accuracy during runtime. Besides, the real speedup is not as high as the compression ratio due to sparse weight distribution.

The inference-stage pruning can offer power-accuracy tunability during runtime, but has much more accuracy loss. DrowsyNet [6] adopts this method and uses random dropout during inference stage to prune neurons, and find optimal power-accuracy tradeoff point in a heuristic manner. Due to their randomness, they should run through all the possible combination to find the best result, which limits their applications to models with many layers due to extremely large computation.

III. Proposed Pruning Method

A. *Proposed approach*

The challenge of the inference stage pruning is that the model retraining is not available as in the training stage pruning or due to no available dataset. However, similar to the observations in the training stage pruning, not all filters are created equal. Filters that are less important shall be pruned first. Some common used indexes are L1 or L2 norm of the filter weights. In this paper, we choose L1 norm as our index.

Algorithm 1. Inference stage model pruning

```
Data:
  $S_i$ = filter size in ith layer
  $C_i$ = # channels in ith layer
  $F_i$ = # filters in ith layer
  $W_i$ = weights in ith layer, with dimension $(F_i,C_i,S_i,S_i)$
  $M_i$ = mask in ith layer, with dimension $(F_i,C_i,S_i,S_i)$

Evaluate Top 1 accuracy
Evaluate FLOP
For i = each conv, linear layer Do
  $P_i$ = sum values in $|W_i|$ along dimension 1, 2, 3 (zero indexing)
  Sort $P_i$ in order to find the proper threshold
  For j in length($P_i$) Do
    $M_i[j, :, :, :] = 0$, if $P_i[j] <$ threshold
  EndDo
  $W_i = W_i * M_i$
EndDo
Evaluate Top 1 accuracy after pruning
Evaluate FLOP after pruning
```

Algorithm 1 shows the proposed method for one complexity operating point evaluated by FLOP as shown in Algorithm 2. In algorithm 1, we sum all the elements in each filters respectively in weighted matrix, and sort these values to find the proper threshold. If some of these values are under the threshold, we set a mask matrix to record the according locations of the values with a zero. Finally, we multiply the weight matrices by this mask matrix for network slimming. In algorithm 2, to calculate FLOP in each layer, we first find the amount of neurons *n* in each filter by multiply the number of channel and filter size, and then obtain the FLOP per neuron (involve *n* multiplication and *n-1* addition) and time it with *Step* to get FLOP per filter. Note that *Step* depends on the size of input image, padding and stride. Finally, multiply FLOP per filter by the number of filters which

Algorithm 2. Evaluate FLOP

```
Data:
  $S_i$ = filter size in ith layer
  $C_i$ = # channels in ith layer
  $F_i$ = # filters in ith layer
  $W_i$ = weight in ith layer, with dimension $(F_i,C_i,S_i,S_i)$
  $Step_i$ = # step for which $F_i$ scan through the input
        image in ith layer
  $Skip_i$ = # filters pruned in layer i

For i = each conv, linear layer Do
  $n = C_i * S_i * S_i$;            // # neuron in filter
  $fpn = n + (n+1)$          // FLOP per neuron
  $fpf = n * Step_i$;           // FLOP per filter
  // multiply by  # filters didn't be pruned in layer i
  $FLOP += fpf * (F_i - Skip)$
EndDo
```

didn't be pruned. Note that *Skip* can be easily obtained by counting the number of elements in P_i such that $P_i[j] < threshold$ in algorithm 1. The pruning procedure is a layer-by-layer pruning procedure because filters in different layers have different relationship. We cannot simply prune filters by comparing their values. The comparisons are within the same layer. We prune the smaller one to meet the pruning rate. Note that the proposed algorithm focuses on reducing complexity to certain operating points, e.g. 25% reduction, instead of model size since the run time constraints are better matched with complexity instead of model size. With this algorithm, we can just use single model but operate at different run time complexity conditions. The only overhead is a small mask that stores the filter mask at different operating points, which is equal to the filter number and much smaller than the model size.

B. Pruning rate adjustment

A key aspect to determine the quality loss is the pruning rate. We have tried different combinations of pruning rate, including setting uniform or non-uniform rate (linearly increase pruning rate along layers) among all convolution layers. For modern CNN structure like Residual Net, the number of filters is gradually increased when layers going deeper. Thus, pruning fewer filters in the shallow layer and pruning more filters in the deep layers would prevent the error caused by pruning shallow layers from magnifying along the network, and keeps accuracy loss low with much more pruning numbers of filters.

Beside the plausible reasons above, performance of the uniform rate is actually better than that of the non-uniform rate. The reason is that pruning filter in deeper layer doesn't contribute much in term of FLOP saving because the input image size in the deeper layer is much more smaller than those in the shallower layer, which reduces the amount of convolution needs to be done in deeper layers. Experimental result has shown that choosing uniform pruning rate over nonuniform pruning rate will result in much higher performance as well as ease the process of setting proper pruning rate to achieve particular number of FLOP saving.

IV. EVALUATIONS

We evaluate our algorithm on a server with Intel Core i7-6850k CPU and NVIDIA GTX 1080, TITAN X graphic cards. We choose the ResNet[7] as our test bed over others popular CNN model because the ResNet has several versions with different number of layers, which allows us to compare the change of performance when the number of layers is gradually increased. Note that other network like VGG has similar performance as ResNet according to our experiment. The ResNet is trained with the CIFAR-10 dataset. We report the saving in FLOP as a metric of power saving and TOP1 score over 10000 images as a metric of accuracy.

Fig. 3 demonstrates the power-accuracy tunability achieved by our program using nonuniform pruning rate for ResNet at different depths. The nonuniform pruning rate is by setting an boundary pruning rate in the first and last layer and linearly increasing the pruning rate for the other layer according to their depth. The result shows that the deeper model tends to drop less accuracy than the shallower model when the FLOP saving is more then 25% but tends to drop more accuracy than the shallower model when FLOP saving is less than 25%. The deeper model can perform better for the higher pruning rate. Note that all these accuracy drop is less than 10% when saving 50% of FLOP, which allows models continually providing high reliability when running under limited resources. Moreover, this algorithm is performed during the inference stage and involves no retraining as well as no changes to CNN models and structure. Therefore, it can make CNN hardware support available power-accuracy tradeoffs after hardware is fabricated.

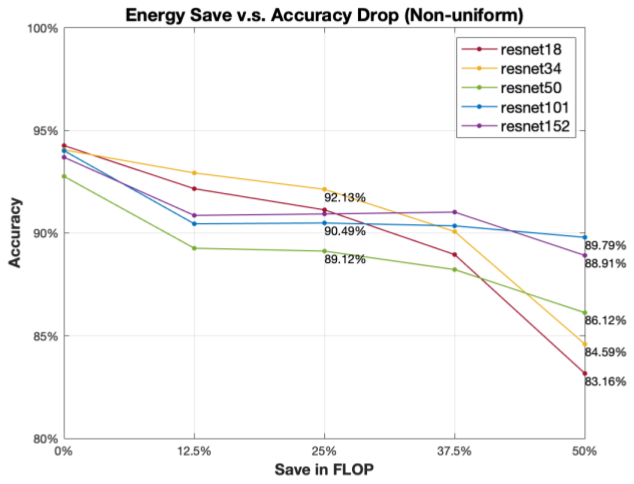

Fig. 3. Power-accuracy achieved by nonuniform pruning rate

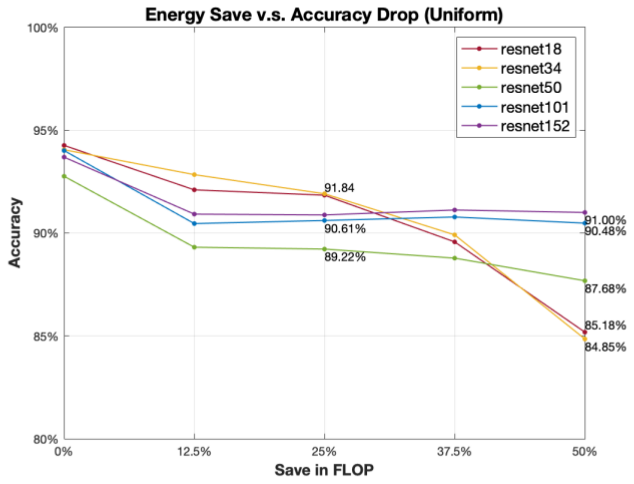

Fig. 4. Power-accuracy achieved by uniform pruning rate

Figure 4 shows the result with the uniform pruning rate for all layers instead. As illustrated, the performances are relatively better than those using the nonuniform pruning rate. Note that by using uniform pruning rate, we can just set all the pruning rate in each layer equal to the desire saving number of FLOP. For example, to achieve 25% of FLOP saving, simply just set pruning rate in each layer to 0.25, which is simple to do. Furthermore, this property has made the tunability between power and accuracy more easily when comparing with nonuniform pruning rate, whose pruning rate in the first layer and last layer has to be manually selected by trying out many different combinations and measure those combinations one by one in order to find out which combination is the best and meet the desire FLOP saving.

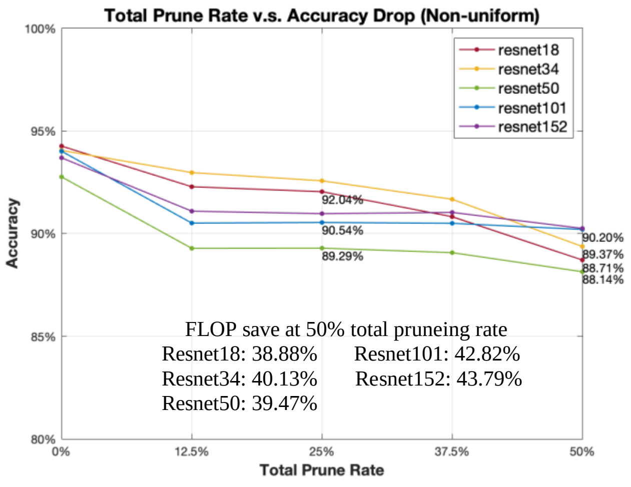

Fig. 5. Total pruning rate-accuracy achieved by the nonuniform pruning rate

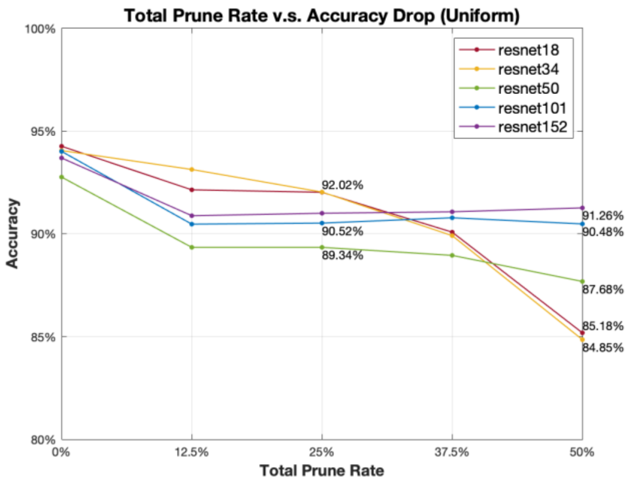

Fig. 6. Total pruning rate-accuracy achieve by the uniform pruning rate

Fig. 5 and Fig. 6 illustrate the total pruning rate of the model size v.s. accuracy drop with uniform pruning and nonuniform pruning, note that the total pruning rate is the fraction of pruned neuron in CNN model. As illustrated, the model size reduction does not translate to the same amount of FLOP reduction. Besides, the accuracy drops seem to be higher with respect to total pruning rate using nonuniform pruning rate than the one respect to FLOP saving. The resulted FLOP savings are far away from 50%, which has proved the fact that filters in the deeper layer contribute less FLOP than that in the shallower layer. Also, based on simple observation, Fig. 4 and Fig. 6 are extremely similiar, which means that we can just simply set all the pruning rates in each layer to be equal to the desire FLOP saving.

V. CONCLUSIONS

We propose an inference-stage model pruning method, which provides power-accuracy modulation during run time. This method prunes weights at the regular filters level instead of the individual weight level to fit parallel execution of GPUs or hardware accelerators. The experimental results show that we can have up to 50% of FLOP saving with less than 10% of accuracy drop.

Acknowledgements

This work was supported by Ministry of Science and Technology, Taiwan, under Grant 108-2634-F-009 -005, 107-2119-M-009-019, and Research of Excellence program 106-2633-E-009-001.

REFERENCES

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

[2] P. Molchanov, and et al., “Pruning convolutional neural networks for resource efficient inference,” in *International Conference on Learning Representations (ICLR)*, 2017.

[3] W. Wen, and et al., “Learning structured sparsity in deep neural networks”, arXiv:1608.03665, 2016

[4] T. Bolukbasi, and et al, “Adaptive Neural Networks for Efficient Inference, ” in International Conference on Machine Learning. 527–536. 2017

[5] Z. Wang, and et al, “SGAD: Soft-guided adaptively-dropped neural network,” arXiv:1807.01430, 2018

[6] R.-S. Liu, and et al, “DrowsyNet: Convolutional neural networks with runtime power-accuracy tunability using inference-stage dropout,” in *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2018.

[7] K. He, and et al, “Deep Residual Learning for Image Recognition”, arXiv:1512.03385, 2015