Learning Very Sparse Convolution Under a Pseudo-Tree Constraint

Hao-Tian Tang
Shanghai Jiaotong University

The convolution operation is very common in today's artificial intelligence tasks. Though the convolution operation takes advantage of parameter sharing and reduces the number of parameters by an order of magnitude comparing with the fully-connected matrix multiplication operation, there are still large redundancy in the convolution operation.

Inspired by the intuition of group convolution and the sparsity of the tree structure, we propose a novel idea in this paper which reduces the number of parameters involved in a convolution operation, in the circumstance of image classification with deep neural networks. We propose to group some feature maps in the input/output of a convolution layer, and find a maximum cardinality matching between the input nodes and output nodes of a convolution operation. Additionally, we constrain the final output network to be tree-structured to further reduce the number of parameters.

In terms of neural network pruning, our method gives a structured pruning approach, which achieves high compression rate comparable with weight-level pruning (which is not structured). In our experiments on CIFAR-10, we are capable of reduce the number of parameters in the VGG16 network by 24x, the number of parameters in the ResNet164 by 2.5x, and DenseNet by 3x, with small accuracy loss.

Categories and Subject Descriptors:

General Terms: Convolution, Compression

Additional Key Words and Phrases: Sparse convolution, deep neural networks, network compression

1. INTRODUCTION

The neural networks are very computation intensive. Despite the fact of convolution operations, the most common operation in today's deep neural networks, utilizes the property of parameter-sharing and thus reduce the number of parameters required to output the same feature map with matrix multiplication by an order of magnitude, it is still accepted by most people that the convolution operation is highly redundant.

If we examine the convolution operation as a graph like what is done in figure 1, we can very easily find out that the convolution operation can be viewed as a fully-connected bipartite graph. However, to make the graph connected, it is not necessary to maintain a maximal-sized bipartite graph. As is illustrated in figure 2, the tree-structured neural network in which one feature map is calculated through only one predecessor is sufficient to maintain the connectiveness of the whole computation graph.

Of course, the tree-structured neural network is seriously constrained in computational capability, as it's similar to a depthwise convolution operation without the pointwise convolution. With the presentation of a pointwise convolution, the computational capability of a depthwise convolution is limited, not to mention the version without it. So we don't consider to transform a traditional convolution operation to a tree-structured convolution, which will make it too difficult for the network to retain the computation capability.

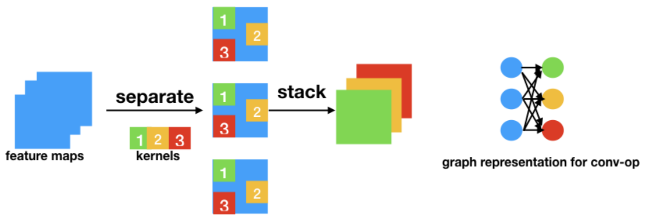

Fig. 1. The graph representation of a convolution operation. Each channel of the input and output feature map is viewed as a node in a graph, and the vertices between two nodes indicate that the right-hand-side node is calculated with the left-hand-side node.

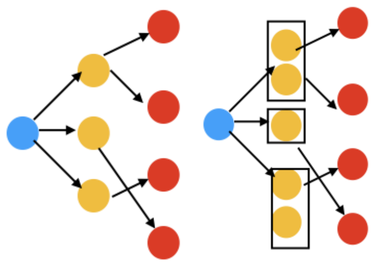

Fig. 2. Left: The minimal connection form: A tree-structured neural network. Right: Restoring some computation capability by grouping some nodes in a tree network together.

However, we can group some nodes in a layer together in the original computation graph as what is done in the right hand side part of 2. After the grouping, we allow the connection between two groups to be fully connected (i.e. for a feature map in each group, it is calculated with all the feature maps in its predecessor group). In order to still remove a large number of parameters in the neural network, we force each group to be calculated from only one predecessor group. Such constraint is called the **pseudo-tree constraint** in this paper, because the computation graph cannot be viewed as a tree in the level of feature maps, but it is indeed a tree in the level of feature-map group. In this case, the parameter reduction ratio of each convolution operation can be proved to be **at least the number of groups**. The term *at least* comes because some nodes can be cutoff from the computation graph because of the tree constraint. For example, if a layer $l+1$ has N nodes and its exact predecessor layer l has $N+k$ nodes, then at least k nodes in layer l is naturally disconnected from the graph. This effect can be viewed as automatic and parameter-free channel pruning.

Notice that we just group nodes together sequentially (The channels that are contiguous in memory are grouped together) in figure 2, but it is not a constraint in real implementations. Actually, grouping sequentially is just what *group convolution* does. In our proposed pseudo-tree constrained convolution scheme, it is allowed to form the groups arbitrarily, so that our framework is very suitable for the task of pruning a pretrained neural network. Obviously, under such scene, it is not reasonable to impose a strong prior of sequential grouping.

As we will point out later, by utilizing the sparsity of our proposed pseudo-tree constrained structure and the automatic channel pruning effect, we can compress some state-of-the-art neural networks including the VGG networks, the ResNets and the DenseNets by very impressive ratios. To point out, we achieve much higher compression rate (2.5x V.S 1.4x) comparing with the network slimming method proposed by [Liu et al. 2017], while retain the same accuracy to this channel-level pruning algorithm. For the VGG network and DenseNet, our proposed method can easily achieve very high compression ratio. For example, we prove that the last several convolution layers in VGG can be pruned by a factor of 120x, without any loss of accuracy on CIFAR-10. Such ratio exceeds the result in many network pruning works by an order of magnitude. Unfortunately, we failed to figure out the way to solve the accuracy drop, but we believe the setting of group numbers might be a reason, and we don't have enough time to tune these parameters.

2. RELATED WORK

Due to the popularity of the convolution operation, there are countless approaches to optimize it. Some focus on the mathematics behind convolution itself, others, like this paper, focus on optimizing the performance of the convolution operation in the concept of a neural network. Also, there're some works focused on tree-structured network (where the basic node in the computation graph might be a basic block like residual block). All of these works are related to this paper and will be introduced briefly.

2.1 Efficient Convolution Operation

There are many attempts to reduce the computational cost and memory footprint of the convolution operation. Some methods are equivalent to the original convolution operation, but saves computation through transformations and inverse transformations. For example, the FFT-based method proposed by [Mathieu et al. 2013], and the Winograd convolution algorithm proposed by [Selesnick and Burrus 1994].

There are also other works of *loss* convolution approximation. For example, previous works use SVD method to approximate the convolution operation, while [Denton et al. 2014] goes beyond SVD, and uses higher order approximation and filter clustering to reduce the memory consumption and computational requirement of the convolution operation.

2.2 Deep Neural Network Compression

Some papers also focus on the compression of deep neural networks. There are coarse-grained DNN pruning and fine-grained DNN pruning algorithms.

For the coarse-grained DNN pruning algorithms, the common objective is to prune some channels which are considered to be less important. For example, [Liu et al. 2017] considers to utilize the coefficient for the batchnorm operation as an indicator of the importance of each channel, and prune channels with small coefficients to compress the model. [He et al. 2017] uses LASSO regression to select the channels. Selecting the channels directly based on the norm of the channels is also possible in [He et al. 2018]. To point out, [He et al. 2018] also utilizes the deep reinforcement learning method to automatically determine the sparsity ratio of each layer.

For the fine-grained DNN pruning, the main concern is usually pruning the redundant weights. For example, [Han et al. 2015b] proposes to iteratively prune and retrain the deep neural networks, and achieves very high compression rates on multiple state-of-the-art neural networks. Following this work, [Han et al. 2015a] also puts forward trained quantization and Huffman coding to further reduce the memory cost of DNNs. However, the fine-grained pruning algorithms are always **unstructured**. To achieve real speedup, special hardware such as the EIE [Han et al. 2016] must be consulted.

2.3 Tree-Structured Neural Networks

Because of the sparsity of tree-structured connections, a lot of works focus on tree-structured neural networks. For example, [Huang et al. 2018] searches for tree-like structures for multi-attribute learning in facial attributes, and achieves state-of-the-art results with far less parameters than previous methods. [Cai et al. 2018] is also a paper in neural architecture search, a very efficient tree-structured basic block is introduced in this paper.

3. PROPOSED METHOD

```
Data: Convolutional Filters $K_1, K_2, ..., K_N$ in the network.
      Layer groupings $G_1, G_2, ..., G_N$.
Result: A set of masks $M_1, M_2, ..., M_N$ for the filters.
Initialize $M_i = 0 (\forall i)$.
for i=N; i≥0; i– do
    Initialize A $g_i \times g_{i+1}$ table $T$, where $g_i$ is the number of
    channels in layer i.
    for j = 0; j < $g_i$; j++ do
        Mask all $K_i$ parameters except for those for $G_j$.
        Train the neural network for 100 iterations.
        Update row $j$ of the $T$. $T_{jk}$ is the average descendent
        class accuracy of the $k^{th}$ group in layer $l + 1$.
    end
    Selection = argmax($T$, axis=0).
    Reconnect layer $i$ according to the Selection. Update $M_i$
    accordingly.
    Update the descendent class information for layer $i$.
    if $G_i$ has no descendent class then
        Remove $G_i$ from computation graph.
    end
end
```

Algorithm 1: Sparse Convolution Algorithm.

We have already introduced the intuition of pseudo-tree constraint in previous sections. We will mainly introduce in this section how to determine the predecessor of each feature-map group (which is a node in our computational graph), and also how to generate the groupings. Our connection selection algorithm is inspired by [Huang et al. 2018]. The major difference is that they use this idea mainly to search for connections between basic blocks, which is common in the scene of NAS. However, our focus is to search for connections between feature maps, within one convolution operation.

The pipeline of our algorithm can be summarized in algorithm 1. The algorithm sketch is too simple, and it is actually implemented in about 500 lines of Python code. For details, please refer to the github repository related to this paper.

3.1 Problem Formulation

We have illustrated our problem formulation through figure 2 informally in the introduction section. To make it more formal, we define the problem as follows:

For a neural network $\mathcal{N}$ with N layers, denote all its feature maps as $F_1, F_2, ..., F_N$. Each feature map group $F_i = [f_1^{(i)}, ..., f_C^{(i)}]$, if it has C channel groups. Also denote all the convolution kernels as $K_1, ..., K_{N-1}$. Notice that the fully connected layer is also viewed as 1×1 convolution. We have $F_{i+1} = Conv(K_i, F_i)$. Our target is to zero away most parameters in K_i.

For $f_k^{(i+1)}(\forall k)$, we have $f_k^{(i+1)} = h(f_1^{(i)}, f_2^{(i)}, ..., f_C^{(i)})$, where h is a linear combination, defined as part of the convolution operation. The coefficients of $f_i^{(k)}$ is related to the coefficients of K_i. To be specific, we want only one coefficient of $f_k^{(i)}$ to be nonzero, and others are all zero (pseudo-tree constraint). This can be formulated as introducing a mask M_i to K_i, in which the second dimension of K_i is only non-zero corresponding to one feature map group of F_i.

As long as the pseudo-tree constraint is satisfied, we want the overall accuracy of classification to be maximized.

3.2 Connection Selection for the Whole Network

We have already formulated the problem of connection selection in kernel coefficients K_i as determining the kernel mask M_i. Potentially, we need to decide $\{M_1, M_2, ..., M_{n-1}\}$ simultaneously. For each M_i, the potential choices of zeroing is the number of groups g_i for the corresponding layer. So the total complexity will be $O(g^n)$, where g is the maximal number of groups. The computation burden is not affordable, and we need to approximate the process of mask learning.

We introduce the Markovness assumption in the mask learning process. We assume that the selection of M_i is independent of $M_j, j \neq i$. We can understand this assumption in another way. Originally, we have $F_{i+1} = Conv(K_i, F_i)$. Now, we use $\hat{F}_{i+1} = Conv(K_i \circ M_i, F_i)$. If the approximation $\hat{F}_{i+1}$ is close enough to original feature map F_i, then the Markovness assumption should be very reliable, because introducing M_i will almost not change the output of each convolution operation (that means, for each i, having M_i or not doesn't influence the output, so the M_j doesn't influence M_i, if $j \neq i$), thus, M_i can be dealt with separately.

Although we don't explicitly force our masked convolutional filters to approximate the original feature map (by exerting some regression loss), such formulation will help justify our Markovness assumption, which easily simplify the computational cost of our proposed method to $O(gn)$, where the oracle operation is to determine a group-group connection in one convolution filter.

Here comes another problem. Now that we have decided to determine $M_i (i = 1, 2, ..., N)$ independently, the order of determining these masks matters. In this paper, we follow the conventions in [Huang et al. 2018] by selecting the masks M_i reversely, from $M_N, M_{N-1}, ...$ to M_1. Furthermore, we assure $M_{k+1}, M_{k+2}, ..., M_N$ already constrains convolution operations starting from layer $k+1$ to a pseudo-tree, when we are trying to determine M_k. The intuition is illustrated in figure 3. We will illustrate in next section that such **reverse order** is necessary for the single layer connection selection.

3.3 Single Layer Connection Selection

We have decided to determine masks M_i in the order $M_N, M_{N-1}, ..., M_1$. In this section, we will introduce the algorithm of determining single M_i. To determine M_i, we believe the most important criteria is the accuracy. Obviously, the global accuracy (i.e. the average accuracy of all classes) is indiscriminative and not helpful for our task of connection selection within a layer, because this statistic remains the same for all connections. So we propose to use a kind of **local accuracy** (this idea is similar to GNAS[Huang et al. 2018]).

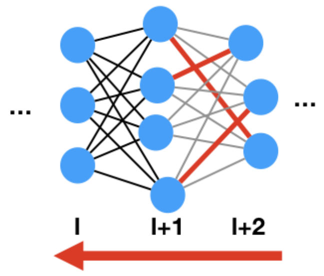

Fig. 3. Illustration of the idea of reverse mask selection. When we are going to determine the M_l, we assume the connection between layer $l+1$ and layer $l+2$ has already been determined, and the structure after layer $l+2$ has already been a forest. The nodes in this graph is also defined as feature map groups.

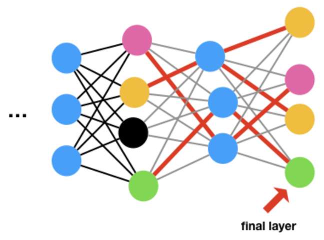

Fig. 4. Illustration of the idea of descendent classes. Let's focus on the second leftmost column in this computation graph. Each final class has a color. For each node in the left column, if their a path to a particular final class, then this node will share the color with the final class, and the final class is called its **descendent class**.

Recall that when we are determining M_l, the connections from layer $l+1$ to layer N is already a pseudo-tree, or a forest structure. In this case, we will see a situation like what is shown in figure 4. The leftmost layer can be viewed as layer l, and the second leftmost layer can be viewed as layer $l+1$. In this graph, we have visually defined the descendent classes of each node in layer $l+1$, and the formal definition is given in the caption. Having the information of **descendent classes**, we can always calculate a scalar for each node in layer $l+1$, which is the average descendent class accuracy for this given node. As we move on to layer $l, l-1, ..., 1$, this scalar will be used to select connection.

To be specific, let's consider the connections between layer l and layer $l+1$. For simplicity, we just illustrate how to select connection for one node in layer $l+1$, the whole process is just to iterate over all nodes in layer $l+1$. We consider the node 1 of layer $l+1$ in figure 5. As can be seen from the figure and caption, the connection with node 3 of layer l leads to the best descendent class average accuracy, which is 0.8, so this selection tends to be selected by our algorithm.

Of course, this process can be quite slow. The three connections in red in figure 5 cannot be evaluated in parallel. The reason is that the connection before layer l is still fully connected. If the three

Table I. Summary of the performance of different models after using the proposed algorithm.

Network	Parameters(before)	Parameters(after)	Accuracy(before)	Accuracy(after)
VGG16	20051530	825160	92.80	89.50
ResNet164	1727645	670237	93.45	92.90
DenseNet40	1078057	350572	93.86	90.00

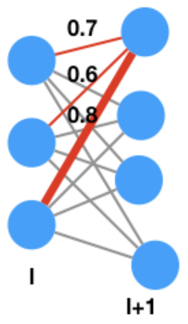

Fig. 5. Illustration of connection selection. We use the node 1 in layer $l+1$ as an example. We will try to connect node 1,2,3 of layer l to it respectively. After a connection is made, we can get a weight on the connection, which is the **average descendent class accuracy** define previously. Obviously, the connection to node 3 of layer l gives the highest accuracy, so this connection is selected (in bold).

Table II. Summary of the compression ratio. Compared with network slimming.

Network	Compression Ratio	Network Slimming
VGG16	24.30	8.28
ResNet164	2.58	1.41
DenseNet40	3.08	2.20

connections are evaluated in parallel, we can get only one average descendent class accuracy for the connection, and we don't know what the accuracy means. Hence, suppose the number of groups in layer l is g_l, number of groups in layer $l+1$ is g_{l+1}, the connections we have to try is $g_l g_{l+1}$. This quadratic complexity is intolerable for deep neural networks such as ResNet164.

The solution is quite simple. We cannot avoid filling in a descendent class accuracy table of size $g_l g_{l+1}$. However, instead of trying to fill in one column at a time (i.e. evaluate the connection from all nodes in layer l to one node in layer $l+1$), we want to determine one row at a time. That is, we can connect one node in layer l to all the nodes in layer $l+1$ simultaneously. Note that this is valid, because the intersection of descendent sets of any two nodes in layer $l+1$ will always be empty (because of the pseudo-tree constraint). By doing so, we successfully make the g_{l+1} term disappear (in parallel computation). The overall complexity of connection search is $O(g_l)$, which is affordable.

3.4 Dealing with Multi-Branch Networks and Skip Connections

Basically, the multi-branch neural networks and neural networks with skip connections cannot be pruned to a pseudo-tree structure. However, we can still relax the constraint a little bit to make it possible to use the previous algorithm. Notice that the algorithm is run from backward to forward, so the skip connection will never break the **descendent class set** of each node, so the criterion is still valid, we can still run the algorithm. Intuitively, the number of paths in multi-branch neural networks is exponential to the number of layers, so it is not possible to prune the multi-branch neural network in the path-level. i.e. It is not possible to consider the impaction on path on the other. However, the good news is that, the number of convolution operations in a multi-branch neural network is limited. Suppose the number of layers is N, the largest branch number is K, then the number of convolution operations is less than NK. So the overall complexity of the previous connection selection algorithm is just enlarged by a constant. For networks like the ResNets who have only two branches, the additional overhead is tolerable.

4. EXPERIMENTS

We conduct our experiments in the setting of image classification. Due to the limitation of computational resources, we chose CIFAR-10 [Krizhevsky and Hinton 2009] as our dataset. We implement the standard VGG16 network with batch normalization (with around 20.04M parameters), the standard ResNet164 network (with around 1.71M parameters) and the DenseNet40 network (with around 1.07M parameters) to illustrate the effectiveness of our algorithm in both single-branch and multi-branch networks with skip connections.

4.1 Dataset

The CIFAR-10 datasets [Krizhevsky and Hinton 2009] consist of natural images with resolution 32×32. There are 50000 images in the training set, we use 90% of them as our training data and leave the other 10% for validation. The test dataset contains 10000 images. In training, we use padding, random horizontal flipping, random clipping data augmentation. We remove all the data augmentation in the testing phase.

4.2 Experiment Results

We conduct our experiments using three popular neural networks. As can be seen from tables I and II[1], the method is able to give very high compression rate, while slightly lose some accuracy. Since the accuracy is measured after finetuning our network, we still have a long way to go before having some impactful results.

To point out, in the experiments, **the number of groups in each layer** has great impact on the final result. For example, for VGG16, if the number of groups in the last several layers is set to be very large (like 32), the accuracy is not influenced. However, for some critical layers (e.g. layer 7), if the group number is set to be large (say, 8), the accuracy drops dramatically. For all networks, the group number parameter for all layers can be found in our code. We just set first $\frac{1}{4}$ layers to have group number 1, last $\frac{1}{4}$ layers to have group number 4, and others to have group number 2 for ResNets and DenseNets. For the ResNet164, actually we even didn't prune any layer before layer 115, and the compression ratio is already impressive. For the VGG16 network, we set these parameters by hand, the largest group number can be 16 or 32.

[1]The compression rate becomes much higher than what is mentioned in mid-term presentation because we cut off the nodes who do not have descendent classes from the computation graph.

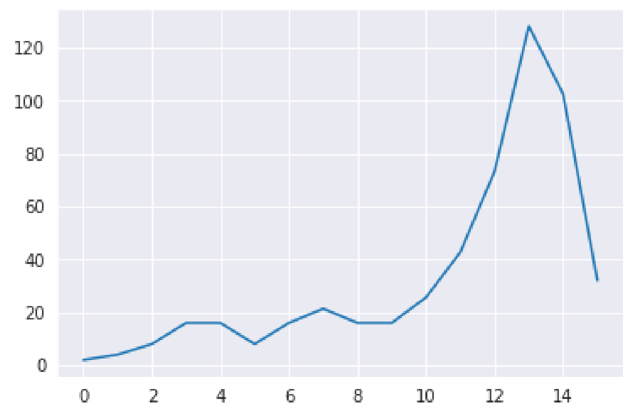

Fig. 6. The compression rate for the VGG16 network, shown layer by layer. The highest compression rate for a single **convolution** layer achieves 120x. Note that channel pruning (structured) methods rarely prune a convolutional filter by more than 10x.

For comparison with network slimming, we reproduced this work in PyTorch by ourselves. For VGG and DenseNet, we observed around 0.1% accuracy improvement in their work, while we observed exactly the same accuracy loss as our method when pruning ResNet164 using their proposed method. We believe our method will have better accuracy performance when the compression rate is lower, but we don't have time to tune our model carefully.

We also illustrate the compression rate of the VGG16 network layer by layer in figure 6. As can be seen from the figure, we are able to reduce the number of parameters in some last convolution layers by a factor of more than 100, which is quite astonishing. The 100x compression factor is contributed by both automatic channel pruning (cutting off the nodes which don't have descendent classes) and learned sparse connection. With only learned sparse connection, the compression rate for a layer can hardly exceed 32x. We believe that if the accuracy can be restored, this work will be quite valuable in terms of memory efficiency.

5. CONCLUSION

We demonstrate the value of tree-structured constraint in neural network pruning. Although we didn't retain the accuracy of the original neural networks after retraining, we found that some last convolution layers do have a redundancy of around 100x, which is astonishing. We also discovered in experiments that some layers in some network such as VGG are critical because they are very volatile to high compression rate.

There are lots of things not done in this project. For example, it is the current trend that the parameters like *sparsity ratio*, or *number of groups* (in this paper) are not determined by human-beings. The wide use of DDPG agents in design automation [He et al. 2018] have naturally inspired us to integrate reinforcement learning into this paper. However, channel selection is cheap, while our connection selection is expensive (with 100 iterations of retraining, though only one layer is updated). The usage of RL may not be so obvious under limited computation budget.

Another thing is to implement this algorithm for real hardware. Currently, what we have done is only zero away a large number of weights, instead of **really make them disappear**. We didn't do anything to **speedup** computation, either. Naturally, because of the structured pruning property, we expect our algorithm to work pretty well on FPGAs and ASICs, since we have smaller indexing overhead than previous weight-level pruning algorithms. However, whether our method can achieve latency performance comparable with some recent work tailored for target hardware (also using DDPG agents) is questionable. We have to admit that we break some spatial locality in our pruned models, which may make speedup negligible.

REFERENCES

Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. 2018. Path-Level Network Transformation for Efficient Architecture Search. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018.* 677–686.

Emily Denton, Wojciech Zaremba, Joan Bruna, Yann Lecun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. (2014), 1269–1277.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark Horowitz, and Bill Dally. 2016. Deep compression and EIE: Efficient inference engine on compressed deep neural network. In *2016 IEEE Hot Chips 28 Symposium (HCS), Cupertino, CA, USA, August 21-23, 2016.* 1–6.

Song Han, Huizi Mao, and William J. Dally. 2015a. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015). http://arxiv.org/abs/1510.00149

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015b. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada.* 1135–1143. http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *European Conference on Computer Vision (ECCV)*.

Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *The IEEE International Conference on Computer Vision (ICCV)*.

Siyu Huang, Xi Li, Zhiqi Cheng, Zhongfei Zhang, and Alexander G. Hauptmann. 2018. GNAS: A Greedy Neural Architecture Search Method for Multi-Attribute Learning. In *2018 ACM Multimedia Conference on Multimedia Conference, MM 2018, Seoul, Republic of Korea, October 22-26, 2018.* 2049–2057.

Alex Krizhevsky and Geoffery Hinton. 2009. Learning multiple layers of features from tiny images.. In *Tech Report*.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning Efficient Convolutional Networks Through Network Slimming. In *The IEEE International Conference on Computer Vision (ICCV)*.

Michaël Mathieu, Mikael Henaff, and Yann LeCun. 2013. Fast Training of Convolutional Networks through FFTs. *arXiv* (2013).

Ivan W. Selesnick and C. Sidney Burrus. 1994. Extending Winograd's Small Convolution Algorithm to Longer Lengths. In *1994 IEEE International Symposium on Circuits and Systems, ISCAS 1994, London, England, UK, May 30 - June 2, 1994.* 449–452.

Received September 2008; accepted March 2009