Structured Probabilistic Pruning for Convolutional Neural Network Acceleration

Huan Wang*
huanw@zju.edu.cn

Qiming Zhang*
qmzhang@zju.edu.cn

Yuehai Wang†
wyuehai@zju.edu.cn

Haoji Hu†
haoji_hu@zju.edu.cn

College of Information Science and
Electronic Engineering
Zhejiang University
Hangzhou, China

Abstract

In this paper, we propose a novel progressive parameter pruning method for Convolutional Neural Network acceleration, named *Structured Probabilistic Pruning (SPP)*, which effectively prunes weights of convolutional layers in a probabilistic manner. Unlike existing deterministic pruning approaches, where unimportant weights are permanently eliminated, SPP introduces a pruning probability for each weight, and pruning is guided by sampling from the pruning probabilities. A mechanism is designed to increase and decrease pruning probabilities based on importance criteria in the training process. Experiments show that, with $4\times$ speedup, SPP can accelerate AlexNet with only 0.3% loss of top-5 accuracy and VGG-16 with 0.8% loss of top-5 accuracy in ImageNet classification. Moreover, SPP can be directly applied to accelerate multi-branch CNN networks, such as ResNet, without specific adaptations. Our $2\times$ speedup ResNet-50 only suffers 0.8% loss of top-5 accuracy on ImageNet. We further show the effectiveness of SPP on transfer learning tasks.
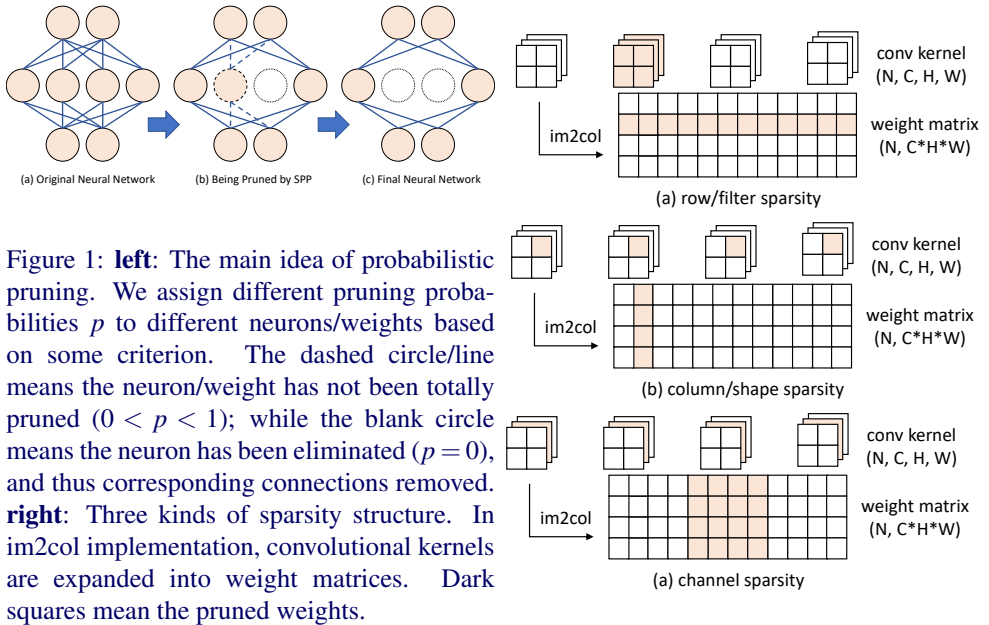
1 Introduction

Convolutional Neural Network (CNN) has obtained better performance in classification, detection and segmentation tasks than traditional methods in computer vision. However, CNN leads to massive computation and storage consumption, thus hindering its deployment on mobile and embedded devices.

Pruning is a promising way for CNN acceleration which aims at eliminating model parameters based on a performance loss function. However, unstructured pruning will lead to irregular sparsity, which is hard to implement for speedup on general hardware platforms [10]. Even with sparse matrix kernels, the speedup is very limited [37]. To solve this problem, many works focus on structured pruning, which can shrink a network into a thinner one so that the implementation of the pruned network is efficient [1, 34]. For example,

Figure 1: **left**: The main idea of probabilistic pruning. We assign different pruning probabilities p to different neurons/weights based on some criterion. The dashed circle/line means the neuron/weight has not been totally pruned ($0 < p < 1$); while the blank circle means the neuron has been eliminated ($p = 0$), and thus corresponding connections removed. **right**: Three kinds of sparsity structure. In im2col implementation, convolutional kernels are expanded into weight matrices. Dark squares mean the pruned weights.

many works were proposed to prune the filters, columns, or the channels of convolutional kernels [7, 25, 29, 37, 39], illustrated in Fig.1 (right).

However, existing importance-based pruning approaches mainly have a problem: they prune unimportant weights based on some importance criteria and never recover them in the following training process. Given the importance criteria are either simple, such as the commonly used L_1 norm and L_2 norm [9, 25], or derived under strong assumptions, such as the parameters need to be independent and identically distributed [29], it is likely that some pruned weights may become important later if they were kept through the whole training process. Therefore, it is necessary to design recovery mechanisms for pruned weights to correct the misjudgments during early training stages.

To solve this problem, we propose the *Structured Probabilistic Pruning (SPP)* for CNN acceleration, which prunes weights in a *probabilistic* manner, as shown in Fig.1 (left). We assign a pruning probability to each weight. When some weights are below the importance threshold and should have been pruned, we only increase its pruning probability rather than totally eliminate them. Only when p reaches 1 will the weights be permanently eliminated from the network. We also design a mechanism to decrease the pruning probability if the weights become more important during training, thus correcting the previous misjudgments. Moreover, SPP prunes the whole network at the same time instead of layer-wisely, so the time complexity is controllable when network becomes deeper.

Related Work. Intensive research has been carried out in CNN acceleration, which is normally categorized into the following five groups. (1) The direct way is to design a more compact network. For example, SqueezeNet [17], MobileNet [15], ShuffleNet [41] were proposed to target the mobile devices. (2) Parameter quantization reduces CNN storage by vector quantization in the parameter space. [8] and [38] used vector quantization over parameters to reduce redundancy. As the extreme form of quantization, binarized networks were proposed to learn binary value of weights or activation functions [4, 16, 26, 32]. (3)

Matrix decomposition modifies weights into smaller components to reduce computation. For example, several methods based on low-rank decomposition of convolutional kernel tensor were also proposed to accelerate the convolutional layer [6, 18, 23, 40]. (4) Parameter pruning was pioneered in the early development of neural networks. Optimal Brain Damage [24] leveraged a second-order Taylor expansion to select parameters for deletion. Deep Compression [8] removed close-to-zero connections and quantized the remained weights for further compression. Structured Sparsity Learning [37] used group LASSO regularization to prune weight rows or columns. Taylor Pruning [29] used a Taylor expansion based importance criteria to prune filters. Filter Pruning [25] was an one-shot pruning method, using the L_1 norm to prune filters. Recently, Channel Pruning [13] alternatively used LASSO regression based channel selection and feature map reconstruction to prune filters. Further, [12] employed Reinforcement Learning to automatically learn the pruning ratio for different layers. Besides, recent works [27, 28, 30] applied dropout to neural network sparsification by interpreting it in a Bayesian manner. (5) Some other works seek acceleration from the perspective of arithmetic complexity, since the acceleration of CNNs are strongly dependent on the implementation of convolution operation. The mainstream implementation is im2col, which transforms the convolution into matrix multiplication (Fig.3 right) [1, 3]. There are also other implementations like Winograd [22] and FFT [35, 36], which are faster theoretically but can cause side-effects such as bandwidth bottlenecks in practice.

2 The Proposed Method

Suppose that the we have a dataset D consisting of N inputs and their corresponding labels: $D = \{(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)\}$. The parameters of a CNN with K convolutional layers is represented by $\Omega = \{(W^1, b^1), (W^2, b^2), \ldots, (W^K, b^K)\}$, which are learned to minimize the discrepancy, i.e. the loss function $L(D|\Omega)$, between network outputs and labels. The common loss function for classification tasks is the negative log-likelihood of Softmax output q, which is defined as $L(D|\Omega) = -\frac{1}{N}\sum_{j=1}^{N} \log q_{y_j}$, where q_{y_j} represents the y_jth element of the Softmax output for the jth input.

The aim of parameter pruning is to find a simpler network Ω' with fewer convolutional parameters based on the original network Ω, in which the loss is minimized. This minimization problem is defined by Eqn.(1).

$$\min_{\Omega'} \quad L(D|\Omega') \quad \text{s.t.} \quad ||\Omega'||_0 < ||\Omega||_0 \tag{1}$$

Normally for CNN, an input tensor $Z^l \in \mathbb{R}^{c^l \cdot h^l \cdot w^l}$ of convolutional layer $l \in \{1, 2, \ldots, K\}$ is firstly convolved with the weight tensor $W^l \in \mathbb{R}^{c^{l+1} \cdot c^l \cdot \hat{h}^l \cdot \hat{w}^l}$, then a non-linear activation function $f(\cdot)$, usually Rectified Linear Units (ReLU), will be applied to it. Then the output will be passed as input to the next layer. For pruning, a mask $g \in \{0, 1\}$ is introduced for every weight, which indicates whether this weight is used in the network. Thus, the output of the lth layer is described as

$$f((g^l \odot W^l) * Z^l + b^l), \tag{2}$$

where $(\odot)$ denotes element-wise multiplication and $(*)$ denotes the convolution operation. Note that the masked weights are also not updated during back propagations. In the context of structured pruning, weights are divided into different groups. Technically, a weight group

can be defined in any way, but in practice weights are usually grouped in a way friendly to hardware implementation, *e.g.* the weights in the same row, column or channel are called a weight group (see Fig.1 right). Since SPP targets structured pruning of convolutional layers, we assign the same g to all weights in the same weight group, which are pruned or retained simultaneously in each iteration.

For traditional pruning methods, once weights are pruned, they will never be reused in the network, which ignores the interdependency and plasticity of the neural network, because during pruning some initially unimportant weights may become important later due to the missing of other dependent weights. We term this kind pf pruning as *deterministic pruning*. To address this problem, some buffering mechanism to postpone pruning is in demand so that weights can be evaluated more times before some of them are decidedly deemed as unimportant and get removed permanently. Inspired by dropout [14], we reinterpret pruning in a probabilistic manner, termed as *probabilistic pruning*. Specifically, a pruning probability p is introduced for each weight. For example, $p = 0.7$ means that there is 70% likelihood that the mask g of the corresponding weight is set to zero. During the training process, we increase or decrease all p's based on the importance criterion of weights. Only when p is increased to 1, its corresponding weight is permanently removed from the network. Obviously, deterministic pruning can be regarded a specific case of probabilistic pruning when we choose to increase p from 0 to 1 in a single iteration.

How to update p. Assume that a convolutional layer consists of N_c weight groups. Our aim is to prune RN_c among them, where $R \in (0, 1)$ is the pruning ratio, indicating the fraction of weight groups to be pruned at the final stage. SPP updates p by a competition mechanism based on some importance criterion. In this paper, we choose the importance criterion as the L_1 norm of each weight group: the bigger the L_1 norm, the more important that weight group is. In our experiments, we find that L_1 and L_2 norms have similar performance as pruning criterion, in line with previous works [9, 25]. There are also other importance criteria such as Taylor expansion [24, 29] to guide pruning. In this paper, we choose the L_1 norm for simplicity, but our method can be easily generalized to other criteria.

The increment Δ of p is a function of the rank $r \in \{0, 1, \ldots, N_c - 1\}$, where the rank r is obtained by sorting the L_1 norm of weight groups in ascending order. It may seem natural to use L_1 norm rather than the rank of L_1 norm as criterion, but in practice we find rank is more controllable (it plays a role like smoothing) and gives better results, so we tend to use the rank as criterion. The function $\Delta(r)$ should satisfy the following two properties: (1) $\Delta(r)$ is a strictly decreasing function because higher rank means greater L_1 norm. In this situation, the increment of pruning probability should be smaller since weights are more important based on the L_1 norm importance criterion. (2) $\Delta(r)$ should be zero when $r = RN_c$. Since we aim at pruning RN_c weight groups at the final stage, we need to increase the pruning probability of weights whose ranks are below RN_c, and decrease the pruning probability of weights whose ranks are above RN_c. By doing this, we can ensure that exactly RN_c weight groups are pruned at the final stage of the algorithm.

The simplest form satisfying these properties is a linear function, as shown in Fig.2 (left). However, our experiments show that it is not very good if we take $\Delta(r)$ as a linear function. The reason, we think, is that the L_1 norm is used to measure the weight importance, but the L_1 norm is not uniformly distributed. In Fig.2 (right), we plot the L_1 norm histogram of convolutional layers in AlexNet, VGG-16 and ResNet-50. It is observed that the L_1 norm of each layer shares a Gaussian-like distribution in which the vast majority of L_1 values are accumulated within a very small range. If we set $\Delta(r)$ linearly, the variations of increments would be huge for middle-ranked weight groups, but their actual L_1 values are very close.
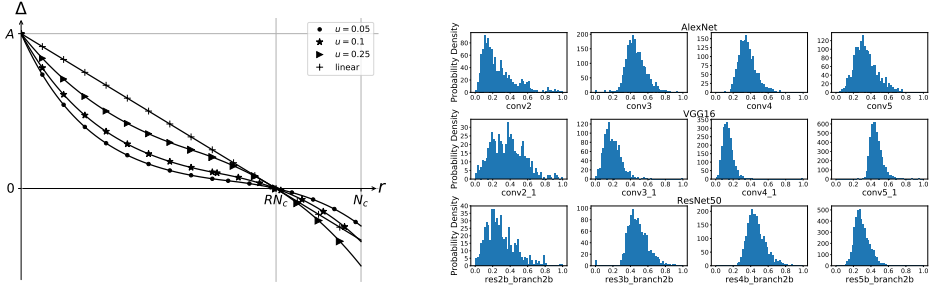
Figure 2: **left**: Functional relationship of the pruning probability increment Δ with regard to the rank r. Straight line is the linear function, the others comes from proposed center-symmetric exponential function. **right**: The L_1 norm histogram of convolutional layers in AlexNet, VGG-16 and ResNet-50. Four representative layers are shown here.

Intuitively, we need to set $\Delta(r)$ of middle-ranked columns similar and make $\Delta(r)$ steeper on both ends, making it in correspondence with the distribution of L_1 norms. So we propose a center-symmetric exponential function to achieve this goal, as shown in Eqn.(3).

$$\Delta(r) = \begin{cases} Ae^{-\alpha r} & , if\ r \leq N \\ 2uA - Ae^{-\alpha(2N-r)} & , if\ r > N \end{cases} \tag{3}$$

Here $u \in (0,1)$ is a hyper-parameter to control the flatness of the function, smaller u indicates that $\Delta(r)$ for middle-ranked weight group is flatter, as shown in Fig.2 (left). Note that α is a decaying parameter for the exponential function, and the function is center-symmetric on point (N, uA). Since we need to compel $\Delta(r)$ passing through (N, uA) and $(RN_c, 0)$, we can actually solve out α and N as $\alpha = \frac{\log(2) - \log(u)}{RN_c}$ and $N = -\frac{\log(u)}{\alpha}$.

For each weight group, its pruning probability is updated by

$$p_{k+1} = \max(\min(p_k + \Delta(r_k), 1), 0) \tag{4}$$

where k denotes the kth update of p, the min/max is to ensure that p is within the range of $[0, 1]$. During the kth update, for a specific weight group, if its rank r_k is less than RN_c, $\Delta(r_k)$ would be positive to make p_k increase; and if its rank r_k is greater than RN_c, $\Delta(r_k)$ would be negative to make p_k decrease.

When to update p. A common practice is to prune weights at a fixed interval [29]. In SPP, we follow this simple rule: p is updated every t iterations of training, where t is a fixed integer. For SPP training, after all the p's are updated, the masks g's are generated by Monte Carlo sampling according to $P(g = 0) = p$. After g is obtained for each weight, the pruning process is applied by Eqn.(2). Finally, after the pruning ratio R is reached, we stop pruning and retrain the pruned model for several epochs to compensate for accuracy. The whole algorithm of SPP is summarized in Algorithm 1. Note that convolutional layers own little percentage of parameters in the whole network [21, 33], so masks assigned to weights in convolutional layers will cause little burden to the memory. Also they are very easy to implement, since there are off-the-shelf optimized GPU multiplication kernels to help apply masks to weights.

Algorithm 1 The SPP Algorithm

1: Input the training set D, the original pre-trained CNN model Ω and target pruning ratio R.
2: Set hyper-parameters A, u and t (in default, $A = 0.05$, $u = 0.25$ and $t = 180$).
3: Set update number $k = 0$.
4: For each weight group in all conv layers, initialize its pruning probability $p_k = 0$.
5: Initialize iteration number $i = 0$.
6: **repeat**
7:     If   mod $(i, t) = 0$, then update p_k by Eqn.(3) and (4), and set $k = k + 1$.
8:     For each weight group, obtain g_i by Monte Carlo sampling based on p_k.
9:     Prune the network by Eqn.(2).
10:     Train the pruned network, updating weights by back propagations.
11:     $i = i + 1$.
12: **until** The weight group ratio of $p_k = 1$ equals to R.
13: Retrain the pruned CNN for several iterations.
14: Output the pruned CNN model Ω'.

---

3 Experiments

The hyper-parameters in SPP are A, u and t. Here we set $A = 0.05$, $u = 0.25$ and $t = 180$. In practice, we found the proposed algorithm is quite robust to these hyper-parameters across different neural network architectures and datasets, so actually the *same* hyper-parameter setting is used for *all* experiments. Other settings such as weight decay, momentum, and dropout are unchanged from the baseline model. In the pruning and retraining process, we only adjust the learning rate and batch size, which will be elaborated in the following experiments. As for sparsity structure, considering a filter typically contains much more weights than a column does, in this paper we choose to prune *columns* of the weight matrices (see Fig.1 right), which we believe will cause less severe damage to network in pruning.

On the small-scale dataset CIFAR-10 [20], we evaluate our method on a shallow model ConvNet [21]. Then on the large-scale dataset ImageNet-2012 [5], we evaluate our method with three popular models, i.e. AlexNet [21], VGG-16 [33] and ResNet-50 [11]. We use Caffe [19] for all of our experiments. Methods for comparison include four structured pruning approaches which were proposed in recent years: Structured Sparsity Learning (SSL) [37], Taylor Pruning (TP) [29], Filter Pruning (FP) [25] and Channel Pruning (CP) [13].

3.1 ConvNet on CIFAR-10

CIFAR-10 dataset contains 10 classes with $50,000$ images for training and $10,000$ for testing. We take $5,000$ images from the training set as the validation set. ConvNet is a tiny version of AlexNet for CIFAR-10 dataset, which is composed of 3 convolutional layers and 1 fully connected layer. The batch size is 256 and initial learning rate 0.001.

For simplicity, we set the pruning ratios of the 3 convolutional layers to the same value. The performance of different methods is shown in Tab.1. Consistently, SPP is much better than the other three methods, and especially when the speedup is small, such as $2\times$, SPP can even improve the performance, for which we argue that the modest pruning can take out the redundancy, which makes the model prone to overfitting, regularize the objective function and thus increase the accuracy. This phenomenon was also found by other pruning methods [37].

Method	Increased err. (%)				
	2×	4×	6×	8×	10×
TP [29] (our impl.)	1.0	3.4	5.4	7.1	8.7
FP [25] (our impl.)	1.6	3.6	5.0	7.5	8.5
SSL [37]	3.0	5.8	7.4	7.5	8.3
Ours	**-0.2**	**0.3**	**1.2**	**1.5**	**2.4**

Table 1: The increased error of different pruning methods when accelerating ConvNet on CIFAR-10. The baseline test accuracy is 81.5%. Minus means the test accuracy is improved.

Layer	2×	4×	6×	8×	10×
conv1	0	0	0	0	0
conv2	7.9%	14.0%	16.6%	25.2%	32.7%
conv3	11.4%	17.3%	16.9%	22.9%	30.8%

Table 2: Recovery ratios for the three convolutional layers of ConvNet, with different speedup ratios.

Recovery Analysis. To demonstrate the recovery effect of SPP, we studies the fraction of weights which are not important at the beginning, but become important at the final stage of training. We calculate the fraction of weights whose ranks are below RN_c at the beginning and then above RN_c at the final stage. These weights should be pruned by many one-shot pruning methods, but they are finally retained by SPP. Tab.2 shows that this fraction, termed as *recovery ratio*, for three convolutional layers of ConvNet. Because the first layer (conv1) is relatively small, consisting of only 75 columns, we think it may lack the dynamics which is necessary to recover misjudgments, thus none of them recovered. However, for conv2 and conv3, which consist of 800 columns each, the recovery ratios are very prominent, indicating that SPP could effectively utilize the dynamics of model parameters and achieve good performance. This may explain why the performance of SPP is more robust for large pruning ratios than the other three methods.

3.2 AlexNet on ImageNet

We further verify our method on ImageNet-2012, which is a large dataset of $1,000$ classes, containing 1.28M images for training, $100,000$ for testing and $50,000$ for validation. There are 5 convolutional layers and 3 fully connected layers in AlexNet [21]. Here we adopt the CaffeNet, an open re-implementation of AlexNet, as the pre-trained model. The baseline top-5 accuracy is 80.0% on ImageNet-2012 validation set.

Because AlexNet, VGG-16 and ResNet-50 are much deeper CNN models, where constant pruning ratio for all layers is not optimal. Usually pruning ratios of different layers are determined according to their sensitivity. There are mainly two ways to evaluate sensitivity: (1) Fix other layers and prune one layer, using the accuracy of the pruned model as the sensitivity measure for the pruned layer [25]; (2) Apply PCA to each layer and take the reconstruction error as the sensitivity measure [13, 37]. Here we take the PCA approach for simplicity. The PCA analysis of AlexNet is shown in Fig.3 (left). We plot the normalized reconstruction errors with different remaining principle component ratios. It can be seen that, under the same remaining principle component ratio, the normalized construction error

Method	2×	4×	5×
TP [29]	3.9	9.2	13.9
FP [25] (our impl.)	0.6	4.1	4.7
SSL [37]	1.3	4.3	5.3
Ours	**-0.7**	**0.3**	**0.9**

Table 3: Accelerating AlexNet on ImageNet. The baseline top-5 accuracy of the original network is 80.0%. We carefully tune the speedup ratio to be the same, and compare the top-5 error increase (%) of SPP, TP, FP and SSL.

of upper layers (like `conv5`) are greater than that of the bottom layers (like `conv1`), which means that the upper layers are less redundant. Thus, we set the proportion of the remaining ratios (one minus the pruning ratio) of these five layers to $1 : 1 : 1 : 1.5 : 1.5$.
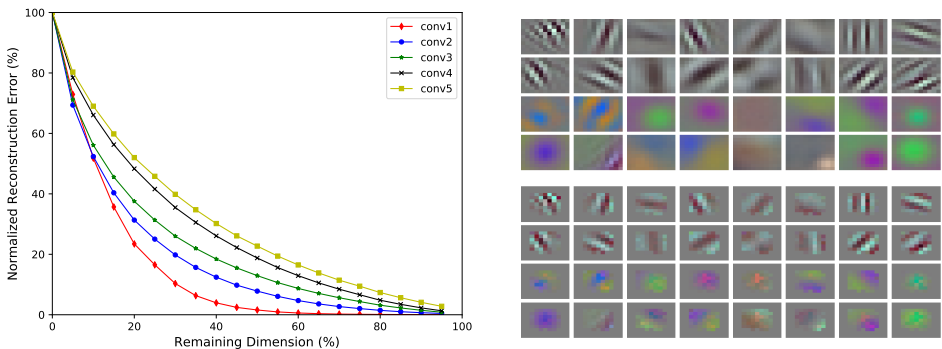

Figure 3: **left**: The normalized reconstruction errors with different remaining PCA ratios for the five convolutional layers in AlexNet. **right**: Filter visualization of first conv layer. The first four rows are the filters from the unpruned AlexNet, the following four rows are corresponding counterparts after SPP pruning with 4× speedup, where pruned weights are illustrated by the gray blocks in each filter (*better seen when in color and zoomed in*).

Accuracy comparison of TP, FP, CP and SPP is shown in Tab.3. SPP outperforms the other three methods by a large margin. SPP can accelerate AlexNet by 5× speedup with only 0.9% increase of top-5 error. And similar to the result of ConvNet on CIFAR-10, when the speedup ratio is small (2×), SPP can even improve the performance (by 0.7%). Previous work [37] reported similar improvements, while their improvement is relatively small (0.1%) and under less speedup settings (1.3×).

Visualization Analysis. To take a closer look at the effect of SPP, we visualize the filters of layer `conv1` of AlexNet. `conv1` contains 64 filters of size $11 \times 11 \times 3$. In Fig.3 (right), we randomly visualize 32 filters as 32 RGB images with size 11×11. Notably, SPP prunes the peripheral parts of filters, while the most salient parts are retained after pruning.

3.3 VGG-16 on ImageNet

VGG-16 is a deep single-branch convolutional neural network with 13 convolutional layers. We use the open pre-trained model, whose single-view top-5 accuracy is 89.6%.

Method	2×	4×	5×
TP [29]	–	4.8	–
FP [25] ([13]'s impl.)	0.8	8.6	14.6
CP [13]	**0**	1.0	**1.7**
SPP	**0**	**0.8**	2.0

Method	CPU time (ms)
VGG-16 baseline	1736.8
SPP (2×)	926.6(1.9×)
SPP (4×)	548.2(3.2×)
SPP (5×)	406.6(4.3×)

Table 4: Acceleration of VGG-16 on ImageNet. The baseline top-5 accuracy of the original network is 89.6%. The top-5 error increases (%) of TP, FP, CP and SPP are compared.

Table 5: Actual speedup of VGG-16. The CPU time is obtained by forward processing one image of size $224 \times 224 \times 3$. Experiments are conducted on Intel Xeon(R) CPU E5-2620 v4 @ 2.10GHz with single thread.

Like the experiment with AlexNet, we firstly use PCA to explore the redundancy of different layers. Similarly, we found that deeper layers have less redundancy, in line with previous work [13]. For convenience of comparison, we just follow [13] by setting the proportion of remaining ratios of shallow layers (conv1_x to conv3_x), middle layers (conv4_x) and top layers (conv5_x) to 1 : 1.5 : 2. Given the computation contribution of layer conv1_1 and conv5_3 to total computation is relatively small (0.6% and 3.0% respectively), we choose not to prune these two layers. We firstly prune the network with batch size 64 and learning rate 0.0005. Then we retrain the pruned model with batch size 256 and learning rate 0.005.

Tab.4 shows the accuracy comparison of TP, FP, CP and SPP. Generally, SPP and CP are much better than FP and TP. Under small speedup ratio (2×), both SPP and CP can achieve zero increase of top-5 error. When the speedup ratio is greater (4× and 5×), the performance of SPP is comparably with CP.

Actual Speedup Analysis. We further evaluate the actual speedup of pruned VGG-16 model on CPU with Caffe. Results are averaged from 50 runs with batch size 32. From Tab.5, we can see that the pruned model by SPP can achieve actual acceleration on general platform with off-the-shelf libraries. The discrepancy between theoretical speedup (measured by GFLOPs reduction) and actual speedup (measured by inference time) is mainly because of the influence of memory access, the unpruned fully-connected layers and the non-weight layers such as pooling and ReLU. Note that given the specific implementation differences, these values may vary with different platforms [29].

3.4 ResNet-50 on ImageNet

Unlike single-branch AlexNet and VGG-16, ResNet-50 is a more compact CNN with multi-branches, which is composed of 53 convolutional layers. We use the open pre-trained caffemodel, whose top-5 accuracy on ImageNet-2012 validation set is 91.2%. Considering ResNet-50 has many layers, we do not set different pruning ratios for different layers. Instead we set the pruning ratio of all convolutional layers to the *same* value in this experiment for simplicity. Pruning batch size is 64 and learning rate 0.0005. When retraining the batch size is 256 and initial learning rate 0.001.

The result is shown in Tab.6. It can be seen that our method achieves better result than CP under 2× speedup. Note that the implementation of SPP on ResNet-50 is very simple, just the same as previous experiments. However, given the multi-branch structure of ResNet, CP needs to add the multi-branch enhancement procedure to generalize their method to ResNet.

Method	Increased err. (%)
CP (enhanced) [13]	1.4
SPP	**0.8**

Table 6: Accelerating ResNet-50 on ImageNet. The baseline top-5 accuracy of the original network is 91.2%. Top-5 error increases of CP and SPP are compared under $2\times$ speedup.

Method	2×	4×	8×	16×
TP [29] (our impl.)	0.9	3.0	4.5	7.3
SPP	**0**	**1.2**	**3.9**	**5.5**

Table 7: Comparison of SPP and TP on the Flower-102 dataset. The baseline accuracy is 93.2% and the results are test error increases (%) under different speedup ratio.

3.5 Transfer Learning

Finally, we apply SPP to transfer learning, where a well-trained model is finetuned by the data from other knowledge domains. We use pre-trained AlexNet model to finetune the Oxford Flower-102 dataset [31]. The 102-class dataset is composed of $8,149$ images, among which $6,149$ is used for training, $1,020$ for validation and the other $1,020$ for testing. We use the open pre-trained caffemodel from Caffe Model Zoo as baseline, whose test accuracy is 93.2%. We compare the performance of our method with TP [29], which was reported very effective for transfer learning.

Like the above experiments, we firstly prune the pre-trained model with SPP, then finetune the model to regain accuracy. The learning rate is set to be 0.001 and batch size 50. For simplicity, we use a constant pruning ratio in this experiment. Tab.7 shows that SPP consistently outperforms TP at all speedup ratios. Note that in [29], the authors claim that Taylor-based criteria was significantly better than L_1 norm in pruning for the transfer learning task,while our result shows that **SPP+L_1 norm** outperforms **TP+Taylor-based criteria**, which proves the effectiveness of SPP in transfer learning tasks.

4 Conclusions

We proposed Structured Probabilistic Pruning (SPP) for CNN acceleration, which prunes weights of CNN in a probabilistic manner and able to correct misjudgment of importance in early training stages. The effectiveness of SPP is proved by comparison with state-of-the-art methods on popular CNN architectures. Further, the effectiveness of SPP is also demonstrated on transfer learning tasks.

The effectiveness of SPP shows a more dynamic and smoothed pruning process is worth exploring, but for now it is more heuristic than theoretically rigorous. In the future, we plan to introduce the Bayesian inference idea [27, 28, 30] into SPP to find a more solid foundation. In addition, it's also interesting to generalize our method to more neural network architectures such as Recurrent Neural Networks.

References

[1] S. Anwar and W. Sung. Compact deep convolutional neural networks with coarse pruning. *arXiv preprint*, arXiv:1610.09639, 2016.

[2] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *International Workshop on Frontiers in Handwriting Recognition*, 2006.

[3] S. Chetlur, C. Woolley, and P. Vandermersch. cuDNN: Efficient primitives for deep learning. *arXiv preprint*, arXiv:1410.0759, 2014.

[4] M. Courbariaux and Y. Bengio. BinaryNet: Training deep neural networks with weights and activations constrained to $+1$ or -1. *arXiv preprint*, arXiv:1602.02830, 2016.

[5] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Feifei. Imagenet: A large-scale hierarchical iage database. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition, CVPR'09*, pages 248–255, Miami, FL, 2009.

[6] E. 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, NIPS*, Montréal, Canada, 2014.

[7] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[8] 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.

[9] S. Han, J. Pool, and J. Tran. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems, NIPS*, pages 1135–1143, Montréal, Canada, 2015.

[10] S. Han, X. Liu, and H. Mao. EIE: efficient inference engine on compressed deep neural network. *ACM Sigarch Computer Architecture News*, 44(3):243–254, 2016.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, CVPR*, pages 770 – 778, Las Vegas, NV, 2016.

[12] Yihui He and Song Han. Adc: Automated deep compression and acceleration with reinforcement learning. *arXiv preprint arXiv:1802.03494*, 2018.

[13] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. 2017.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint*, arXiv:1207.0580, 2012.

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] Qinghao Hu, Peisong Wang, and Jian Cheng. From hashing to cnns: Training binary-weight networks via hashing. *arXiv preprint arXiv:1802.02733*, 2018.

[17] F. Iandola, M. Moskewicz, and K. Ashraf. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv preprint*, arXiv:1602.07360, 2016.

[18] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *Computer Science*, 4(4):1–13, 2014.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrel. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint*, arXiv:1408.5093, 2014.

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

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Information Processing Systems, NIPS*, pages 1097–1105, Lake Tahoe, CA, 2012.

[22] Andrew Lavin. Fast algorithms for convolutional neural networks. *CoRR*, abs/1509.09308, 2015. URL http://arxiv.org/abs/1509.09308.

[23] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *arXiv preprint*, arXiv:1510.03009, 2016.

[24] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Information Processing Systems, NIPS*, pages 598–605, Denver, CO, 1990.

[25] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations, ICLR*, 2017.

[26] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. *arXiv preprint*, arXiv:1510.03009, 2016.

[27] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3290–3300, 2017.

[28] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[29] P. Molchanov, S. Tyree, and T. Karras. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations, ICLR*, 2017.

[30] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6778–6787, 2017.

[31] M. E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, ICVGIP'08*, Bhubaneswar, India, 2008.

[32] M. Rastegari, V. Ordonez, J. Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision, ECCV*, pages 525–542, Amsterdam, Netherland, 2016.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Computer Science*, 2014.

[34] V. Sze, Y. H. Chen, T. J. Yang, and J. Emer. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint*, arXiv:1703.09039, 2017.

[35] Nicolas Vasilache, Jeff Johnson, Michaël Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A GPU performance evaluation. *CoRR*, abs/1412.7580, 2014. URL http://arxiv.org/abs/1412.7580.

[36] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

[37] W. Wen, C. Wu, and Y. Wang. Learning structured sparsity in deep neural networks. In *Advances in Information Processing Systems, NIPS*, pages 2074–2082, Barcelona, Spain, 2015.

[38] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 4820 – 4828, Las Vegas, NV, 2016.

[39] Niange Yu, Shi Qiu, Xiaolin Hu, and Jianmin Li. Accelerating convolutional neural networks by group-wise 2d-filter pruning. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 2502–2509. IEEE, 2017.

[40] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, 2016.

[41] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.