Accelerate CNN via Recursive Bayesian Pruning

Yuefu Zhou[1,2] Ya Zhang[1] ✉ Yanfeng Wang[1] Qi Tian[3]
[1]Cooperative Medianet Innovation Center, Shanghai Jiao Tong University
[2]MediaSmart Technology
[3]Huawei Noah's Ark Lab
{remicongee, ya_zhang, wangyanfeng}@sjtu.edu.cn, tian.qi1@huawei.com

Abstract

Channel Pruning, widely used for accelerating Convolutional Neural Networks, is an NP-hard problem due to the inter-layer dependency of channel redundancy. Existing methods generally ignored the above dependency for computation simplicity. To solve the problem, under the Bayesian framework, we here propose a layer-wise Recursive Bayesian Pruning method (RBP). A new dropout-based measurement of redundancy, which facilitate the computation of posterior assuming inter-layer dependency, is introduced. Specifically, we model the noise across layers as a Markov chain and target its posterior to reflect the inter-layer dependency. Considering the closed form solution for posterior is intractable, we derive a sparsity-inducing Dirac-like prior which regularizes the distribution of the designed noise to automatically approximate the posterior. Compared with the existing methods, no additional overhead is required when the inter-layer dependency assumed. The redundant channels can be simply identified by tiny dropout noise and directly pruned layer by layer. Experiments on popular CNN architectures have shown that the proposed method outperforms several state-of-the-arts. Particularly, we achieve up to $\mathbf{5.0}\times$*,* $\mathbf{2.2}\times$ *and* $\mathbf{1.7}\times$ *FLOPs reduction with little accuracy loss on the large scale dataset ILSVRC2012 for VGG16, ResNet50 and MobileNetV2, respectively.*

1. Introduction

Convolutional Neural Networks (CNNs) have recently achieved great success in computer vision and pattern recognition. However, this success is often accompanied by massive computation which makes the model difficult to deploy on resource-constrained devices. One popular solution, channel pruning [1, 35, 21], lowers computation cost by reducing the number of feature maps. The key challenge in channel pruning is to identify redundant channels. Recent Bayesian methods transform variational dropout noise [15] as a principled measurement for redundancy via Bayesian inference from sparsity-inducing prior [20, 23]. Redundant channels are considered either being multiplied by a noise of large variance [20] or with low Signal-to-Noise Ratio and thus less informative [23]. However, these methods assume that the channels in different layers are completely independent and simultaneously infers the redundancy of all layers, which leads to a sub-optimal solution. In fact, pruning certain channels of any layer is likely to change the distribution of input for the following layer, which may further incite the change of redundancy to fit new input there. This inter-layer dependency has been considered in heuristics and proved to make pruning more efficient [11, 22].

In this paper, we attempt to re-investigate the Bayesian pruning framework assuming the inter-layer dependency and propose a layer-wise Recursive Bayesian Pruning method (RBP). Similar to existing Bayesian methods [23, 20], a Gaussian dropout noise, an indicator of channel redundancy, is multiplied on each channel. To take the inter-layer dependency into consideration, we model the dropout noise across layers as a Markov chain. The inter-layer dependency is then reflected by the posterior of dropout noise given the dropout noise of the previous layer. However, the closed form solution for the posterior is intractable. We here derive a sparsity-inducing Dirac-like prior that regularizes the distribution of the dropout noise so as to automatically approximate the posterior. Compared to the existing Bayesian methods, with the Dirac-like prior, RBP requires no additional overhead when assuming the inter-layer dependency. In addition, the Dirac-like prior is shown to enforce the values of dropout noise to be close to 0 for redundant channels and close to 1 for important ones, a desired property of pruning. Thus, we only need to conduct Bayesian inference and prune the channels associated with tiny dropout noise layer by layer. Additionally, RBP is compatible with reparameterization tricks, which are proved to improve data fitness [15]. Hence as a bonus, the performance of CNNs pruned can be recovered fast after a few epochs of finetuning. In this way, RBP is designed as a completely data-driven approach, achieving a nice balance between data fitness and model acceleration.

We evaluate RBP on popular CNN architectures and benchmark data sets, showing superior performance to several state-of-the-arts in terms of acceleration. We achieve $\mathbf{5.0}\times$, $\mathbf{2.2}\times$ and $\mathbf{1.7}\times$ FLOPs reduction with little accuracy loss on large scale dataset ILSVRC2012 [4] for VGG16 [29], ResNet50 [9] and MobileNetV2 [28], respectively.

2. Related work

Over-parameterization in deep learning often raises huge computation cost, which incites the need for compact neural networks. Pruning is among the most popular solutions in this field and its main idea is removing redundant weights from the original networks. First introduced in [17, 8], measurement for redundancy is based on Hessian of the objective function. [7, 6] later propose to regard small-magnitude weights as less informative and should be pruned. However, these methods are unstructured and retain the format of weight matrix, thus the acceleration effect is limited unless Compressed Sparse Column (CSC) adopted.

Given that, recent trend is pruning whole channels or neurons. [35] proposes group sparsity regularization on weights. [11] combines l_1-norm regularization and reconstruction error. [22] prunes less informative channels layer by layer. [12] extends to select more general structures as residual blocks. Both [11, 22] also consider the influence for redundancy when the input is changed by pruning the previous layer. Particularly, they attempt to suppress this change via minimize a regression loss. By contrast, we propose to infer this change and guide it towards higher sparsity in each layer.

In line with these heuristics, under Bayesian framework, variational dropout [15] is adopted to infer the redundancy. [20] estimates redundancy from horseshoe prior. [23] proposes log-normal prior for regularization. Although the proposed method also adopts variational dropout for approximate inference over redundancy, we attempt to tackle the inter-layer dependency and the existing Bayesian methods solely suppose the channels are all independent in networks.

Alternative solutions for compact networks include: 1) Quantization [27, 2, 36] reduces bit number of weights stored. 2) Low-rank approximation [5, 30, 37] decomposes weight matrix by two stacked smaller ones. 3) Architecture learning [38] directly searches compact designs.

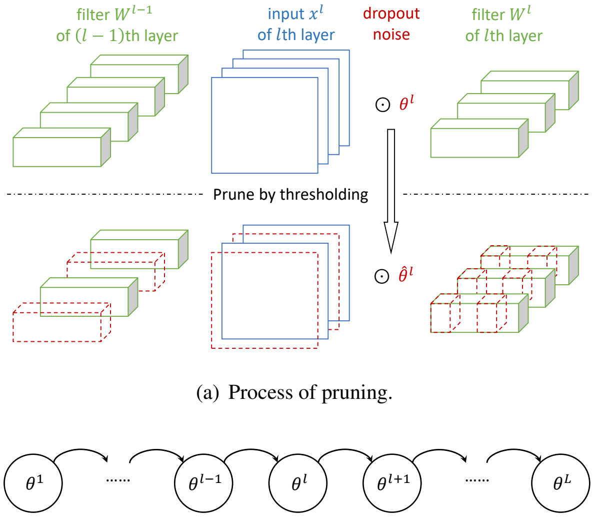
3. Recursive Bayesian Pruning

In this section, we provide a comprehensive introduction for the proposed Recursive Bayesian Pruning (RBP) method. Fig. 1 provides an illustration of RBP.

We first introduce the notation used in the rest of the paper as follows. x and y represent the data and label sampled from the dataset D, respectively. x^l is the input to lth layer and W^l is the filter weight of lth layer. $g(.)$ is the activation function. The output of lth layer is

$$x^{l+1} = W^l * g(x^l), \tag{1}$$

where $*$ is convolution, and bias is omitted for clarity.


filter W^{l-1} of $(l-1)$th layer
input x^l of lth layer
dropout noise
filter W^l of lth layer
$\odot\ \theta^l$
Prune by thresholding
$\odot\ \hat{\theta}^l$

(a) Process of pruning.

(b) CNN redundancy modeling.

Figure 1. Illustration for the proposed method. The pruning is conducted in layer-wise. a) For lth layer, we design a vector of dropout noise θ^l scaled on channels to indicate redundancy. The small noise will be assigned as 0 after estimation done, so that its associated channels and filters are pruned (dotted in red). b) While estimating redundancy, given the inter-layer dependency, we model the dropout noise as a Markov chain, thus pruning strategy depends on $p(\theta^l|\theta^{l-1})$.

3.1. Redundancy estimation

To indicate redundant channels in x^l, one intuitive choice is scaling a dropout noise sampled from Bernoulli $\mathcal{B}(1-r)$ [32] with dropout rate r (i.e. the probability of being dropped). Given the difficulty of training r under Bayesian framework, we adopt its Gaussian approximation $\mathcal{N}(1-r, r(1-r))$, which is actually the Lyapunov's Central Limit [34]. Let x^l contain C channels, then the Eqn. 1 can be rewritten as

$$\begin{aligned} x^{l+1} &= W^l * \left(g(x^l) \odot \theta^l\right), \\ \theta^l_c &\sim_q \mathcal{N}\left(1 - r^l_c, r^l_c(1 - r^l_c)\right), \end{aligned} \tag{2}$$

where $\theta^l = \left(\theta^l_1, ..., \theta^l_C\right)$, $r^l_c \in [0, 1]$ and $\odot$ is element-wise product on channels. In this case, for channels that are probably redundant, i.e. with dropout rate close to 1, they will be almost pruned since the noise scaling on them is near 0. Alternative choices such as log-normal distribution [23] for the dropout noise is much more complicated than the designed one, but lack an intuitive explanation for redundancy.

To estimate the redundancy with both data fitness and acceleration considered, we maximize the *variational lower bound* $w.r.t.$ $W = \{W^l\}$ and r_c^ls:

$$\mathcal{L} = \log \mathbb{P}(y|x, r^l, W) - D_{KL}\left(q(\theta^l)\,||\,p(\theta^l)\right), \quad (3)$$

where the first term is log-likelihood, the second term is the Kullback-Leibler (KL) divergence from the estimator q to the sparsity-inducing prior p. This training process is equivalent to conduct approximate Bayesian inference on θ^l, and for W, we leave it as the optimum for the log-likelihood.

3.2. Posterior of redundancy

Before choosing a sparsity-inducing prior p, we return to the core problem: the posterior of redundancy. Recall the observation that pruning channels of one layer may change the input of the following layer, which takes the risk of ruining data fitness. Thus it is preferred to continue pruning and retraining for adaptive weights when knowing how many channels are pruned in the previous layer. In our case, θ^l indicates redundancy, hence the posterior of redundancy is formed as $p(\theta^l|\theta^{l-1})$ for lth layer. Directly solving its closed form is difficult, because generally, we can only write the equation below

$$q(\theta^l) = \int p\left(\theta^l|\theta^{l-1}\right) q(\theta^{l-1}) d\theta^{l-1}. \quad (4)$$

While seeking for an efficient approximation, we note that once θ^{l-1} approaches Dirac distribution, the solution is immediate:

$$\begin{aligned} q(\theta^l) &\approx \int p\left(\theta^l|\theta^{l-1}\right) \delta(\theta^{l-1}) d\theta^{l-1} \\ &= p\left(\theta^l|\theta^{l-1} = \mathbb{E}\left[\theta^{l-1}\right]\right). \end{aligned} \quad (5)$$

This approximation is valid when the Gaussian noise θ^{l-1} has the dropout rate close to 0 or 1. This is intuitively true, because for a highly compact CNN, the channels left are supposed to be important and thus should have tiny probability of being dropped (i.e. $r^{l-1} \approx 0$), and for those pruned, once the accuracy is acceptable, there is no reason to keep them (i.e. $r^{l-1} \approx 1$). The experiments verify this conjecture, as seen in dropout noise analysis of Section 4.5.

Given that, we simply choose a Dirac-like prior $\mathcal{N}(0, \epsilon^2)$ as the sparsity-inducing prior, where ϵ is very tiny. Then the KL-divergence in Eqn. 3 can be developed as

$$\begin{aligned} & D_{KL}\left(q(\theta^l)\,||\,p(\theta^l)\right) \\ =& \sum_{c=1}^{C} D_{KL}\left(q(\theta_c^l)\,||\,p(\theta_c^l)\right) \\ =& \sum_{c=1}^{C} -\frac{1}{2} \log \frac{r_c^l(1-r_c^l)}{\epsilon^2} + \frac{1-r_c^l}{2\epsilon^2} - \frac{1}{2}. \end{aligned} \quad (6)$$

Here we adopt mean field theory [26] to ease the computation, which supposes the independence among channels within each layer.

Since maximizing the variational lower bound (Eqn. 3) partially minimizes D_{KL} (Eqn. 6), the sparsity will be induced by pushing dropout rates to 1. In fact, let the gradient of D_{KL} $w.r.t.$ r_c^l be zero, i.e. $\partial D_{KL}/\partial r_c^l = 0$, its optimum lies at

$$r_c^{l*} = \frac{1 - 4\epsilon^2 + \sqrt{1 + 16\epsilon^4}}{2} \approx 1 - 2\epsilon^2. \quad (7)$$

3.3. Data-driven pruning

To conduct pruning with data fitness considered, we adopt reparameterization tricks [15] by sampling the dropout noise as

$$\theta_c^l = 1 - r_c^l + \sqrt{r_c^l(1 - r_c^l)} \cdot \mathcal{N}(0, 1), \quad (8)$$

which will be scaled on the corresponding channels when forwarding. In this way, the dropout rates join the optimization of the log-likelihood (in Eqn. 3) and can be simply updated via gradient-based strategies. Since the log-likelihood indicates how well the networks fit data, the proposed pruning method is data-driven.

In this paper, we adopt mini-batch update strategy for training each layer. We summarize that on lth layer, the objective function to maximize for each batch is

$$\mathcal{L} = \mathcal{L}_D - D_{KL}\left(q(\theta^l)\,||\,p(\theta^l)\right), \quad (9)$$

where

$$\mathcal{L}_D = \frac{|D|}{|B|} \sum_{(x,y)\in B} \log \mathbb{P}\left(y|x, W, r^l\right), \quad (10)$$

with B a mini-batch. At the convergence of this objective, r^l is near 0 or 1 and thus θ^l approximately follow Dirac distribution. According to the deduction of section 3.2, this property will lead to $q(\theta^{l+1}) \approx p\left(\theta^{l+1}|\theta^l = \mathbb{E}[\theta^l]\right)$, which incites us to conduct Bayesian inference on θ^{l+1} with θ^l fixed as its expectation. Furthermore, given $\mathbb{E}[\theta^l] = 1 - r^l$ is already near 0 or 1, we are free to let large r_c^ls be 1 by thresholding without influencing much the output of this layer. An immediate benefit is that the channels and associated filters of the lth and $(l+1)$th layer are directly pruned. To avoid additional cost for storing parameters, we scale the dropout rates on filters

$$\begin{aligned} r_c^l &\leftarrow 1, \text{ if } \ r_c^l > T \\ W_c^l &\leftarrow W_c^l \odot (1 - r_c^l), \end{aligned} \quad (11)$$

where T is threshold value and W_c^l is the column for cth input channel. Note that θ^l can be discarded since then.

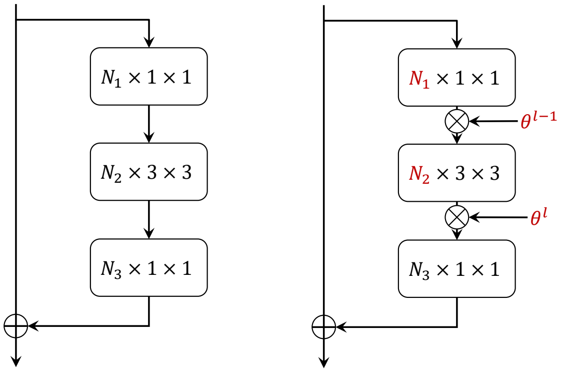


Figure 2. Illustration for pruning residual blocks. The dropout noise is only scaled on the input channels of the last two layers, and thus prunes the filters of the first two layers (in red). N_1 and N_2 are the number of filters.

3.4. Scale to ResNet

The proposed method can also be applied on residual networks [9]. As seen in Fig. 2, we scale dropout noise on the input of the last two stacked convolutional layers, thus only prune the filters of the first two layers. This pruning strategy is also adopted in [22, 19], because the output of the last layer is supposed to have the same channel numbers as the input of its residual block so that the sum operation can be valid.

4. Experiment

In this section, we validate the effectiveness of the proposed method RBP. The CNN architectures to prune include VGG16 [29] and ResNet50 [9]. We mainly report floating operations (FLOPs) to indicate acceleration effect. Inference-time and storage saving on GPU is measured for practical speed-up results. Compression rate (CR) is also revealed as another criterion for pruning. To have an insight on pruning result, we provide number of channels left.

4.1. Implementation

Given the layer-wise pattern adopted, the condition for moving to the next layer is important. One may choose the moment that the dropout rates are barely updated. In this paper, we observe that the number of epochs required for the convergence is almost the same for all the layers in one architecture. Thus, we simply set one "trigger epoch" as the number of epochs for training each layer. This value may vary from dataset or architectures, which will be specified later. There are two hyper-parameters left to be determined, threshold for pruning T and variance for prior ϵ^2. For the former, since the dropout rates are close to 0 or 1, any value in $[0.1, 0.9]$ works and does not differ much the results. In this paper, we adopt $T = 0.5$. For the later, a dropout rate above 0.95 almost prunes the corresponding channel, hence it is expected that $1 - 2\epsilon^2 = 0.95$ (Eqn. 7) and thus $\epsilon^2 = 0.025$. A smaller ϵ^2 can be tried for a higher dropout rate for redundant channels. We adopt Adam [14] as optimizer for RBP and SGD [33] for finetuning 30 epochs after pruning. Learning rate is always $1e-4$ during training and degraded by 0.95 every 2 epochs from $3e-4$ during finetuning. For stability of stochastic methods [31], we adopt pretrained models. On CIFAR10, we pretrain the models for 100 epochs. On ILSVRC2012, we adopt the pretrained models of ThiNet [22]. More details can be referred in Algorithm 1.

Algorithm 1 RBP for the whole network.

```
Input: Dataset $D$, $L$-layer CNN, trigger epoch $E$
Output: CNN pruned in channel level
while $l \leq L$ do
    $e = 0, r^l = 0.01, T = 0.5$;
    for $e$ in range($E$) do
        for batch $B$ in $D$ do
            sample $\theta^l$ (Eqn. 8);
            scale $\theta^l$ on input channels (Eqn. 1);
            compute objective $\mathcal{L}$ (Eqn. 9);
            update weight $W$ and $r^l$;
        end for
    end for
    $r^l \leftarrow 1$ if $r^l > T$ and scale on $W^l$ (Eqn. 11);
end while
```

4.2. VGG16 on CIFAR10

We first prune VGG16 network on CIFAR10 [16], which contains $50,000$ 32×32 images in training set and $10,000$ in test set. The model performance on CIFAR10 is qualified by the accuracy of classifying 10-class images. Considering that VGG16 is originally proposed for large scale dataset, the redundancy is very obvious, especially for the top three fully-connected (fc) layers (in 4096 dimension). Thus, we cut one fc layer and reduce the dimension of the rest to 512. To show that RBP can also be applied on fc-layers, we conduct pruning for the whole network, i.e. 13 convolutional layers and 2 fc-layers. The trigger epoch for each layer is set as 10 and the batch size is always 64.

We also duplicate BC [20] and SBP [23] for comparison. These two methods also adopt dropout noise and conduct Bayesian inference for pruning based on different sparsity-inducing prior. BC proposes horeshoe prior in hierarchical form and SBP does log-normal prior. However, both of them ignore the inter-layer dependency and prune all the channels at the same time.

As seen in Table 1, compared with the baseline model, RBP achieves 3.5× FLOPs reduction with only 0.6% error increased. SBP and BC control the error in the same level with RBP, but the acceleration effect is rather modest. In

Method	Architecture	CR	FLOPs	Err.
Baseline	**64-64-128-128-256-256-256-512-512-512-512-512-512**-512-512	1.0×	1.0×	8.4
SBP [23] (impl.)	**47-50-91-115-227-160-50-72-51-12-34-39-20**-20-272	17.0×	3.2×	9.0
BC [20]	**51-62-125-128-228-129-38-13-9-6-5-6-6**-6-20	18.6×	2.6×	9.0
RBC	**43-62-120-120-182-113-40-12-20-11-6-9-10**-10-22	25.7×	3.1×	9.5
IBP	**45-63-122-123-246-199-82-31-20-17-14-14-31**-21-21	13.3×	2.3×	8.3
RBP	**50-63-123-108-104-57-23-14-9-8-6-7-11**-11-12	**39.1×**	**3.5×**	**9.0**

Table 1. Comparison of pruning VGG16 on CIFAR10. Convolutional layers are in bold. "impl." denotes our implementation.

terms of compression rate, RBP exceeds SBP and BC by over 2 times.

To have an insight on pruning results, we report the channels left, as seen in Table 1. One may wonder that the superior effectiveness of RBP to BC and SBP stems from the layer-wise greedy strategy adopted, hence we implement, for ablation study, 1) IBP, pruning all layers independently at the same time with the proposed objective (Eqn. 9), 2) RBC, applying BC layer by layer. Comparing BC and RBP, we note that both BC and RBP prune `conv5` (last three convolutional layers) to very few channels (~ 10). However, RBP is able to prune more `conv1`-`conv4`, where lies about 90% FLOPs.

By adopting layer-wise strategy, RBC improves dramatically the compression rate and FLOPs reduction, but with more error increased. We suppose that in the theory of BC, the redundancy estimator and prior are neither designed for inter-layer dependency. Although pruning layer by layer, the distribution of BC's dropout noise can not fit the posterior of redundancy and thus may prune more in each layer but misunderstand the distribution of input.

Another interesting observation is that applying RBP for all the layers at the same time lets the pruning result approach BC. For instance, there is less difference between IBP and BC in `conv2`, i.e. $246\,v.s.\,228$ and $199\,v.s.\,129$. We ascribe the performance loss to the absence of layer-wise strategy. Pruning layer by layer in data-driven way can "inform" the following layers that data fitness can be retained with less filters. In this case, both BC and IBP keep most channels in `conv1` and `conv2` but still fail to reduce redundancy in the following layers. By contrast, RBP prunes to $104/256$ and $57/256$ channels in `conv3`.

4.3. VGG16 on ILSVRC

We now evaluate the performance of RBP for VGG16 on ILSVRC2012 [4]. ILSVRC2012 is a large-scale image classification dataset, which contains $1,000$ classes, more than 1.2 million images in training set and $5,0000$ in validation set. As input of VGG16, we sample 128 images as a batch and adopt data augmentation for each one when training: 1) resize to 256×256 and crop randomly a 224×224 part, 2) adopt random horizontal flip, 3) normalize with mean value and standard deviation pre-defined. During test, we almost apply the same data augmentation, except that a 224×224 part is extracted in the center. For VGG16 in this section, we return to the original architecture, i.e. 13 convolutional layers and 3 4096-d fc layers.

In terms of pruning strategy, we do not prune the whole network this time. Instead, the first 10 convolutional layers (`conv1`-`conv4`) are to be pruned. This strategy is commonly adopted for VGG16 on ILSVRC, because as mentioned before, more than 90% FLOPs is distributed on these layers. Given that, we apply RBP on the first 10 layers during 30 epochs and the trigger epoch is thus 3. And since we do not prune the fc layers, which contains most parameters, we focus on the FLOPs reduction.

We compare the results with DDS [12], ThiNet [22], CP [11] and AMC [10]. Similar with us, DDS also adopts a scale value to indicate redundancy, while the regularization is heuristic. ThiNet and CP also consider the influence for redundancy when the input is changed by pruning the previous layer. AMC applies reinforcement learning for network compression policies.

As shown in Table 2, RBP reduces FLOPs by 5× and still achieves competitive accuracy. Compared with CP, RBP does not only achieve lower FLOPs but also provides 1.4% more accurate classification result. Compared with DDS, improvement on accuracy is marginal, but that on FLOPs reduction is significant. CP outperforms ThiNet on FLOPs reduction but with 2% top-1 error increased. The gap mainly stems from the factitious pruning settings of CP, while ThiNet simply prunes half channels for each layer. We suppose that both these strategies are not effective enough. For CP, manually setting the remaining channel ratios introduces hyper-parameters and may need additional cost for tuning. Although this helps CP progressively prunes networks, the accuracy loss is also obvious because the hyper-parameters' setting is not data-driven. For ThiNet, pruning uniformly all the layers ignores the possibility that redundancy varies from depth. In fact, it has been widely known that deeper layers extract higher level semantic information, thus different functionality may require different numbers of filters for data fitness. By contrast, AMC shows a close performance to RBP without manual setting as CP or Thinet. However, the cost for reinforcement learning adopted in AMC is much higher and can not be ignored.

Method	FLOPs	Top-1 Err.	Top-5 Err.
Baseline	1.0×	27.5	9.2
DDS [12]	4.0×	31.5	11.8
ThiNet [22]	3.2×	30.2	10.5
CP [11]	4.4×	32.2	11.9
AMC [10]	5.0×	30.9	–
RBP	**5.0×**	**30.8**	**10.9**

Table 2. Comparison results of VGG16 on ILSVRC2012. Top-k Err. denotes the classification error for the first k predictions.

Layer	#Remained/#Original	Percent	FLOPs
224×224			
conv1_1	16/64	25%	1.1×
conv1_2	39/64	60%	1.2×
112×112			
conv2_1	45/128	35%	1.3×
conv2_2	81/128	63%	1.4×
56×56			
conv3_1	65/256	25%	1.6×
conv3_2	68/256	26%	2.0×
conv3_3	116/256	45%	2.2×
28×28			
conv4_1	132/512	26%	2.9×
conv4_2	135/512	26%	4.3×
conv4_3	257/512	50%	5.0×
Total	954/2688	35%	5.0×

Table 3. Remaining channels of VGG16 on ILSVRC2012. FLOPs reduction is reported in form of accumulation. Resolution of input channels is over each conv block.

We also report the number of remaining channels for conv1-conv4 in Table 3. Totally, we prune around two-thirds of channels. In channel level, one interesting observation is that the last layer of every conv block is pruned to around half channels, while the rest are reduced to around one quarter. Why is it harder to prune the former? We attribute it to the sensitivity raised by resolution reduction. In fact, the last layer of each block is stacked by a pooling layer to reduce size of feature maps. For instance, the feature maps generated by conv2_2 are sampled from 112×112 to 56×56. Thus, to maintain enough information, more channels may be required by the following block. This gives us a clue that pruning all the layers with the same remaining ratio, such as in ThiNet, is unwise, which may take the risk of pruning too much for sensitive layers or leaving much redundancy for others. Additionally, the FLOPs reduction accumulated in Table 3 may provide useful suggestions for pruning strategy. Note that FLOPs is reduced most in conv4 block. Especially on conv4_2 layer, the speed-up ratio grows from 2.9× to 4.3×. We suppose that there exits most redundancy in this block.

4.4. ResNet50 on ILSVRC

We now accelerate ResNet50 on ILSVRC2012. ResNet50 is a very deep CNN in the residual network family. It contains 16 residual blocks [9], where around 50 convolutional layers are stacked. Although the depth of ResNet50 is greater than VGG16, many filters of the former are of size 1×1 and hence already saves much FLOPs, i.e. 4.1 billion $v.s.$ 31.0 billion. However, reported in PyTorch model zoo [25], ResNet50 outperforms VGG16 by around 3% top-1 accuracy on ILSVRC2012. Given that, we suppose that ResNet50 is already a much more compact architecture than VGG16 and pruning should be more cautious.

In this section, we always follow the strategy proposed in Section 3.4, i.e. only prune the filters of the first two convolutional layers of each residual block. Furthermore, considering the following factors, we improve RBP to be more adaptive to residual network family:

1) Although we choose to only prune the filters of the first two convolutional layers of each residual block, there still exists 32 layers, which will be exhausting if the trigger epoch is large. Given that, we assume that the dependency across blocks is relatively weak and can be ignored. This is intuitively reasonable, because between two adjacent blocks, the layers to be pruned are separated by another convolutional layer and divided into two groups. Therefore, we are free to prune the first layers of all the blocks at the same time, and then move on all the second layers.

2) It has been found that the residual networks are very sensitive at the blocks with down-sampling layers and not robust to pruning [18]. In ResNet50, there are 4 residual blocks containing down-sampling layers. We propose to omit these blocks for better data fitness.

We name the variant RBP combined with the above two points ResNet-adaptive RBP (RRBP). The trigger epoch is respectively 3 and 7 for RBP and RRBP. Both sample 256 images as a batch with the same data augmentation as for VGG16.

The pruning results are shown in Table 4. For ThiNet, we cite ThiNet-50 which prunes 50% channels. And for DDS, we cite DDS(32) and DDS(26), where DDS respectively prunes ResNet50 to 32 and 26 residual blocks. We also report the performance of RBP, which simply conducts layer-wise pruning on ResNet50. It can be found that both RBP and RRBP achieve FLOPs reduction over 2×, while DDS and CP are rather conservative. In terms of classification accuracy, DDS(32) provides the lowest top-1 and top-5 error, however, the speed-up ratio is also the lowest. By contrast, DDS(26) prunes ResNet50 more progressively and outperforms DDS(32). Even so, RBP and RRBP show

Method	FLOPs	Top-1 Err.	Top-5 Err.
Baseline	1.0×	23.9	7.1
DDS(32) [12]	1.4×	25.8	8.1
DDS(26) [12]	1.7×	28.2	9.2
CP [11]	1.5×	27.7	9.2
ThiNet-50 [22]	2.3×	29.0	10.0
RBP	**2.3×**	**28.5**	**9.8**
RRBP	**2.2×**	**27.0**	**9.0**

Table 4. Comparison results of ResNet50 on ILSVRC2012. Top-k Err. denotes the classification error for the first k predictions.

Stage	#Remained/#Original	Percent	FLOPs
res2	20/256	9%	1.2×
res3	67/1024	7%	1.6×
res4	2408/3072	78%	1.8×
res5	1105/3072	36%	2.3×
Total	3600/7424	48%	2.3×

Table 5. **RBP** result. Remaining channels of ResNet50 on ILSVRC2012. FLOPs reduction is reported in form of accumulation. The #Remained and #Original only count the channels in the first two convolutional layers of each residual block. Stage groups residual blocks between two down-sampling layers.

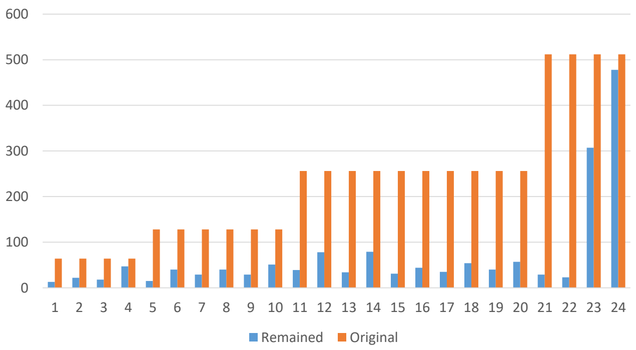


Figure 3. **RRBP** result. Columns for comparison between the number of remaining channels and original ones in each layer.

Model	Time	Storage	Top-5 Err.
VGG-RBP	2.6×	3.6×	+1.7
ResNet50-RBP	1.4×	1.5×	+2.7
ResNet50-RRBP	1.3×	1.4×	+1.8

Table 6. GPU acceleration for VGG and ResNet50 on ILSVRC.

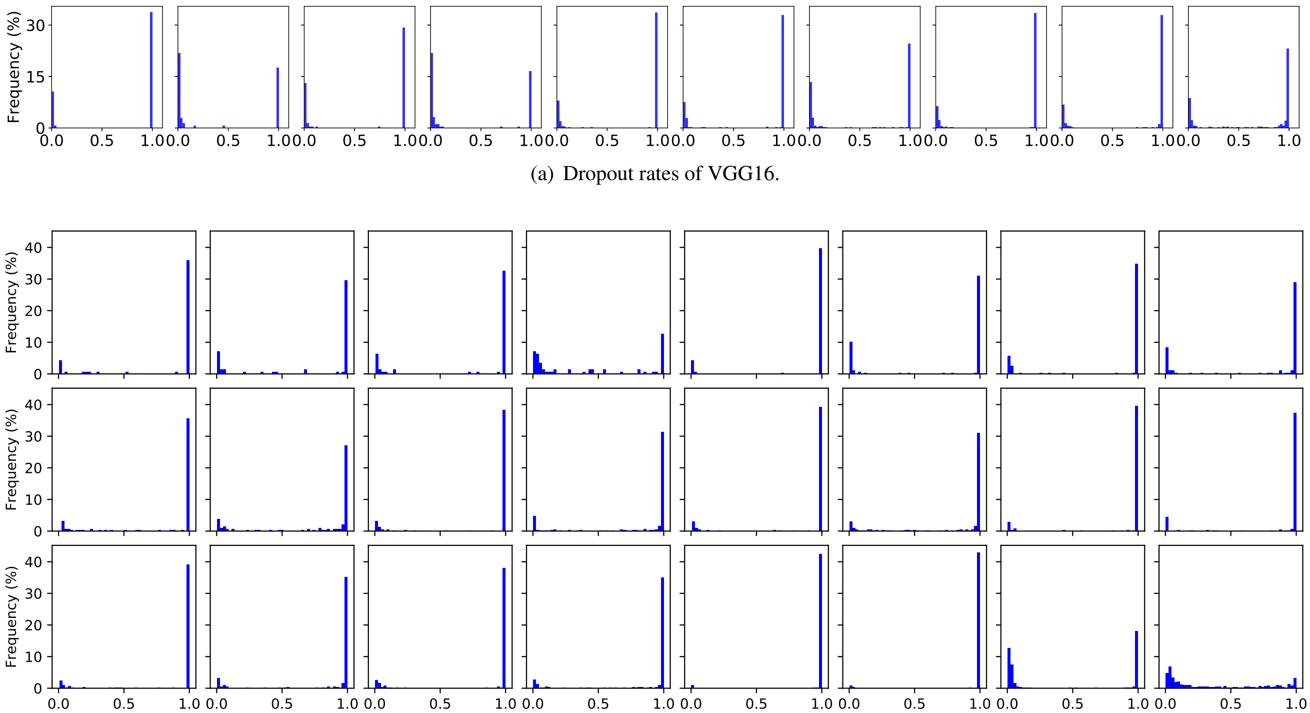
significant superiority to DDS(26). In fact, the former reduces almost 1× more FLOPs but keep competitive classification accuracy, i.e. RBP holds only 0.3% more top-1 error and RRBP even provides 1.2% less. Given this point, we conclude that RBP and RRBP are more effective on residual networks than DDS. Compared with ThiNet-50, RBP and RRBP achieve the same level of FLOPs reduction with competitive classification accuracy. Particularly, RRBP is even 2.0% and 1.0% better on Top-1 and Top-5 accuracy, respectively. Between RBP and RRBP, the later shows almost the same acceleration effect but with higher classification accuracy, which validates our idea that RRBP is more adaptive to residual networks.

Table 5 shows the remaining channels in each block when applying RBP. Totally, we prune more than half of channels. In stage level, over 90% channels are pruned in `res1` and `res2`, yielding most FLOPs reduction contribution. With a close look at `res2_2`, we find that only 1 filter is remained in the first two convolutional layers, which almost removes this residual block. Note that DDS is proposed to prune a more general structure rather than channels, such as residual blocks. In this case, RBP simulates block selection by pruning most channels there, which shows a similar generality with DDS. However, most channels of `res4` are kept, while DDS(32) prunes two residual blocks. This is mainly because simply adopting layer-wise strategy on ResNet50 may over-prune the first several residual blocks and thus requires the more filters in the following layers to ensure data fitness. Furthermore, RBP also progressively prunes the sensitive blocks with down-sampling layers, which explains why the classification error is higher.

Fig. 3 shows the channels remained after pruning by RRBP. It can be easily found that compared with RBP, RRBP prunes channels more uniformly and almost reduces the redundancy of each layer to a very low level. The only exceptions are the convolutional layer in the last residual block (23 and 24 in Fig. 3). We suppose that although this block contains no down-sampling layers, it is stacked by a pooling layer that reduces the resolution from 7×7 to 1×1. Despite the convenience for the following fc layer, it makes the last residual block more sensitive for channel pruning. This observation is consistent with the result of VGG16, where the layers before pooling retains more channels. For totality, RRBP removes $4,000$ channels, which is even better than RBP (3824 channels removed). Note that the pruning field of RRBP is smaller than RBP, because 4 residual blocks with down-sampling layers are ignored, however, it still shows superior performance for identifying redundant channels. The robustness of RRBP to residual networks is thus an immediate conclusion.

4.5. Further Analysis

Acceleration performance on GPU. For ILSVRC2012, We evaluate the acceleration performance on GPU (GeForce GTX 1080 Ti). All the models are run under Caffe [13] with CUDA8 [24] and cuDNN5 [3]. The inference time is averaged from 50 runs of batch size 32. As shown in Table 6, the proposed method achieves promising acceleration and lower storage with little accuracy drop on



(a) Dropout rates of VGG16.

(b) Dropout rates of ResNet50.

Figure 4. Histogram of dropout rates in each layer of VGG16 and ResNet50 (RRBP). Each sub-figure represents distribution of dropout rates in one specific convolutional layer.

Method	FLOPs	Top-1 Err.	Top-5 Err.
Baseline	1.0×	28.1	9.7
AMC [10]	1.4×	29.2	-
RBP-4×	1.4×	29.2	10.1
RBP	1.7×	30.3	10.9

Table 7. Comparison results of MobileNetV2 on ILSVRC2012. RBP-4× means in each inverted residual block, we stop pruning when the expansion rate is reduced lower than 4×.

VGG16 and ResNet50, respectively.

Distribution of dropout rates. Remind that in Section 3.2, the dropout rates r^l are supposed to be near 0 or 1 after optimization of Eqn. 9 done. This hypothesis is the precondition for the designed dropout noise to approach Dirac distribution. In this section, we provide experimental proofs that this hypothesis is generally valid. As shown in Fig. 4, almost all of the dropout rates are distributed near 0 or 1. Note that in the last block of ResNet50, some dropout rates does not approach 0 or 1, which is consistent with the proposition that this layer is sensitive to pruning (Section 4.4).

Lightweight Model. Besides VGG16 and ResNets, which are known to be redundant, the proposed method generalizes well on MobileNetV2 [28], a recently proposed lightweight model. As shown in Table 4.5, RBP reduces 10% more FLOPs than AMC with only 1.1% error increased. To show the performance on similar FLOPs level, we adopt pruning protocol RBP-4×, i.e. in each inverted residual block, we stop pruning when the expansion rate is reduced lower than 4× (6× originally). The Top-1 accuracy is as high as AMC.

5. Conclusion

In this paper, we extend the existing Bayesian pruning methods by embedding inter-layer dependency. By proposing RBP, the redundant channels are identified efficiently and directly pruned layer by layer. Given the data-driven pattern adopted, a nice balance between data fitness and model acceleration is found. The experiments on popular CNN architectures validate the effectiveness of the proposed method, also showing superior performance to several state-of-the-arts.

Acknowledgements

This work is supported by The High Technology Research and Development Program of China (2015AA015801), NSFC (61521062), and STCSM (18DZ2270700).

References

[1] Jose M. Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *NIPS*, 2016. 1

[2] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015. 2

[3] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. 7

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 2, 5

[5] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014. 2

[6] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2016. 2

[7] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *NIPS*, 2015. 2

[8] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 1992. 2

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 2, 4, 6

[10] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018. 5, 6, 8

[11] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017. 1, 2, 5, 6, 7

[12] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, 2018. 2, 5, 6, 7

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, 2014. 7

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 4

[15] Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Computer Science*, 2015. 1, 2, 3

[16] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 4

[17] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *NIPS*, 1989. 2

[18] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2017. 6

[19] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, 2018. 4

[20] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *NIPS*, 2017. 1, 2, 4, 5

[21] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. *CoRR*, abs/1712.01312, 2017. 1

[22] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5068–5076, 2017. 1, 2, 4, 5, 6, 7

[23] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *NIPS*, 2017. 1, 2, 4, 5

[24] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *2008 IEEE Hot Chips 20 Symposium (HCS)*, pages 1–2, 2008. 7

[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 6

[26] Carsten Peterson and James R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1, 1987. 3

[27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016. 2

[28] Mark B. Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 2, 8

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015. 2, 4

[30] Vikas Sindhwani, Tara N. Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *NIPS*, 2015. 2

[31] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *NIPS*, 2016. 4

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 2

[33] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013. 4

[34] Sida Wang and Christopher Manning. Fast dropout training. In *ICML*, 2013. 2

[35] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016. 1, 2

[36] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4820–4828, 2016. 2

[37] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:1943–1955, 2016. 2

[38] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2017. 2