Channel Pruning for Accelerating Very Deep Neural Networks

Yihui He[*]
Xi'an Jiaotong University
Xi'an, 710049, China
heyihui@stu.xjtu.edu.cn

Xiangyu Zhang
Megvii Inc.
Beijing, 100190, China
zhangxiangyu@megvii.com

Jian Sun
Megvii Inc.
Beijing, 100190, China
sunjian@megvii.com

Abstract

In this paper, we introduce a new channel pruning method to accelerate very deep convolutional neural networks. Given a trained CNN model, we propose an iterative two-step algorithm to effectively prune each layer, by a LASSO regression based channel selection and least square reconstruction. We further generalize this algorithm to multi-layer and multi-branch cases. Our method reduces the accumulated error and enhance the compatibility with various architectures. Our pruned VGG-16 achieves the state-of-the-art results by $5\times$ *speed-up along with only 0.3% increase of error. More importantly, our method is able to accelerate modern networks like ResNet, Xception and suffers only 1.4%, 1.0% accuracy loss under* $2\times$ *speed-up respectively, which is significant. Code has been made publicly available*[1].

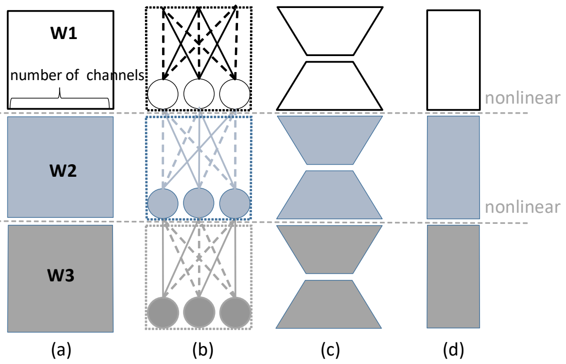

Figure 1. Structured simplification methods that accelerate CNNs: (a) a network with 3 conv layers. (b) sparse connection deactivates some connections between channels. (c) tensor factorization factorizes a convolutional layer into several pieces. (d) channel pruning reduces number of channels in each layer (focus of this paper).

1. Introduction

Recent CNN acceleration works fall into three categories: optimized implementation (*e.g.*, FFT [48]), quantization (*e.g.*, BinaryNet [8]), and structured simplification that convert a CNN into compact one [22]. This work focuses on the last one.

Structured simplification mainly involves: tensor factorization [22], sparse connection [17], and channel pruning [49]. Tensor factorization factorizes a convolutional layer into several efficient ones (Fig. 1(c)). However, feature map width (number of channels) could not be reduced, which makes it difficult to decompose 1×1 convolutional layer favored by modern networks (*e.g.*, GoogleNet [46], ResNet [18], Xception [7]). This type of method also introduces extra computation overhead. Sparse connection deactivates connections between neurons or channels (Fig. 1(b)). Though it is able to achieves high theoretical speed-up ratio, the sparse convolutional layers have an "irregular" shape which is not implementation friendly. In contrast, channel pruning directly reduces feature map width, which shrinks a network into thinner one, as shown in Fig. 1(d). It is efficient on both CPU and GPU because no special implementation is required.

Pruning channels is simple but challenging because removing channels in one layer might dramatically change the input of the following layer. Recently, *training-based* channel pruning works [1, 49] have focused on imposing sparse constrain on weights during training, which could adaptively determine hyper-parameters. However, training from scratch is very costly and results for very deep CNNs on ImageNet have been rarely reported. *Inference-time* attempts [31, 3] have focused on analysis of the importance of individual weight. The reported speed-up ratio is very limited.

In this paper, we propose a new inference-time approach for channel pruning, utilizing redundancy inter channels. Inspired by tensor factorization improvement by feature maps reconstruction [53], instead of analyzing filter weights [22, 31], we fully exploits redundancy inter feature maps. Specifically, given a trained CNN model, pruning each layer

[*]This work was done when Yihui He was an intern at Megvii Inc.
[1]https://github.com/yihui-he/channel-pruning

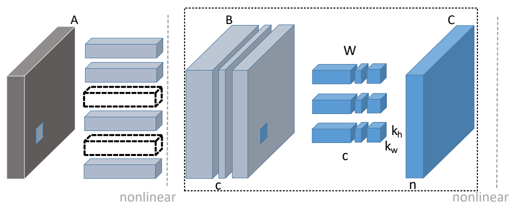

Figure 2. Channel pruning for accelerating a convolutional layer. We aim to reduce the number of channels of feature map B, while minimizing the reconstruction error on feature map C. Our optimization algorithm (Sec. 3.1) performs within the dotted box, which does not involve nonlinearity. This figure illustrates the situation that two channels are pruned for feature map B. Thus corresponding channels of filters W can be removed. Furthermore, even though not directly optimized by our algorithm, the corresponding filters in the previous layer can also be removed (marked by dotted filters). c, n: number of channels for feature maps B and C, $k_h \times k_w$: kernel size.

is achieved by minimizing reconstruction error on its output feature maps, as showned in Fig. 2. We solve this minimization problem by two alternative steps: channels selection and feature map reconstruction. In one step, we figure out the most representative channels, and prune redundant ones, based on LASSO regression. In the other step, we reconstruct the outputs with remaining channels with linear least squares. We alternatively take two steps. Further, we approximate the network layer-by-layer, with accumulated error accounted. We also discuss methodologies to prune multi-branch networks (*e.g.*, ResNet [18], Xception [7]).

For VGG-16, we achieve $4\times$ acceleration, with only **1.0%** increase of top-5 error. Combined with tensor factorization, we reach $5\times$ acceleration but merely suffer **0.3%** increase of error, which outperforms previous state-of-the-arts. We further speed up ResNet-50 and Xception-50 by $2\times$ with only **1.4%, 1.0%** accuracy loss respectively.

2. Related Work

There has been a significant amount of work on accelerating CNNs. Many of them fall into three categories: optimized implementation [4], quantization [41], and structured simplification [22].

Optimized implementation based methods [36, 48, 27, 4] accelerate convolution, with special convolution algorithms like FFT [48]. Quantization [8, 41] reduces floating point computational complexity.

Sparse connection eliminates connections between neurons [17, 33, 29, 15, 14]. [52] prunes connections based on weights magnitude. [16] could accelerate fully connected layers up to $50\times$. However, in practice, the actual speed-up maybe very related to implementation.

Tensor factorization [22, 28, 13, 24] decompose weights into several pieces. [51, 10, 12] accelerate fully connected layers with truncated SVD. [53] factorize a layer into 3×3 and 1×1 combination, driven by feature map redundancy.

Channel pruning removes redundant channels on feature maps. There are several training-based approaches. [1, 49, 54] regularize networks to improve accuracy. Channel-wise SSL [49] reaches high compression ratio for first few conv layers of LeNet [30] and AlexNet [26]. [54] could work well for fully connected layers. However, *training-based* approaches are more costly, and the effectiveness for very deep networks on large datasets is rarely exploited.

Inference-time channel pruning is challenging, as reported by previous works [2, 40]. Some works [45, 35, 19] focus on model size compression, which mainly operate the fully connected layers. Data-free approaches [31, 3] results for speed-up ratio (*e.g.*, $5\times$) have not been reported, and requires long retraining procedure. [3] select channels via over 100 random trials, however it need long time to evaluate each trial on a deep network, which makes it infeasible to work on very deep models and large datasets. [31] is even worse than naive solution from our observation sometimes (Sec. 4.1.1).

3. Approach

In this section, we first propose a channel pruning algorithm for a single layer, then generalize this approach to multiple layers or the whole model. Furthermore, we discuss variants of our approach for multi-branch networks.

3.1. Formulation

Fig. 2 illustrates our channel pruning algorithm for a single convolutional layer. We aim to reduce the number of channels of feature map B, while maintaining outputs in feature map C. Once channels are pruned, we can remove corresponding channels of the filters that take these channels as input. Also, filters that produce these channels can also be removed. It is clear that channel pruning involves two key points. The first is channel selection, since we need to select proper channel combination to maintain as much information. The second is reconstruction. We need to reconstruct the following feature maps using the selected channels.

Motivated by this, we propose an iterative two-step algorithm. In one step, we aim to select most representative channels. Since an exhaustive search is infeasible even for tiny networks, we come up with a LASSO regression based method to figure out representative channels and prune redundant ones. In the other step, we reconstruct the outputs with remaining channels with linear least squares. We alternatively take two steps.

Formally, to prune a feature map with c channels, we consider applying $n \times c \times k_h \times k_w$ convolutional filters W on $N \times c \times k_h \times k_w$ input volumes X sampled from this feature map, which produces $N \times n$ output matrix Y. Here, N is the number of samples, n is the number of output channels, and k_h, k_w are the kernel size. For simple representation, bias term is not included in our formulation. To prune the input channels from c to desired c' ($0 \leq c' \leq c$), while minimizing reconstruction error, we formulate our problem as follow:

$$\begin{aligned} &\underset{\boldsymbol{\beta}, \mathrm{W}}{\arg\min} \frac{1}{2N} \left\| \mathrm{Y} - \sum_{i=1}^{c} \beta_i \mathrm{X_i} \mathrm{W_i}^\top \right\|_F^2 \\ &\text{subject to } \|\boldsymbol{\beta}\|_0 \leq c' \end{aligned} \tag{1}$$

$\|\cdot\|_F$ is Frobenius norm. $\mathrm{X_i}$ is $N \times k_h k_w$ matrix sliced from ith channel of input volumes X, $i = 1, ..., c$. $\mathrm{W_i}$ is $n \times k_h k_w$ filter weights sliced from ith channel of W. $\boldsymbol{\beta}$ is coefficient vector of length c for channel selection, and β_i (ith entry of $\boldsymbol{\beta}$) is a scalar mask to ith channel (i.e. to drop the whole channel or not). Notice that, if $\beta_i = 0$, $\mathrm{X_i}$ will be no longer useful, which could be safely pruned from feature map. $\mathrm{W_i}$ could also be removed. c' is the number of retained channels, which is manually set as it can be calculated from the desired speed-up ratio. For whole-model speed-up (i.e. Section 4.1.2), given the overall speed-up, we first assign speed-up ratio for each layer then calculate each c'.

Optimization
Solving this ℓ_0 minimization problem in Eqn. 1 is NP-hard. Therefore, we relax the ℓ_0 to ℓ_1 regularization:

$$\begin{aligned} &\underset{\boldsymbol{\beta}, \mathrm{W}}{\arg\min} \frac{1}{2N} \left\| \mathrm{Y} - \sum_{i=1}^{c} \beta_i \mathrm{X_i} \mathrm{W_i}^\top \right\|_F^2 + \lambda \|\boldsymbol{\beta}\|_1 \\ &\text{subject to } \|\boldsymbol{\beta}\|_0 \leq c', \forall i \, \|\mathrm{W_i}\|_F = 1 \end{aligned} \tag{2}$$

λ is a penalty coefficient. By increasing λ, there will be more zero terms in $\boldsymbol{\beta}$ and one can get higher speed-up ratio. We also add a constrain $\forall i \, \|\mathrm{W_i}\|_F = 1$ to this formulation, which avoids trivial solution.

Now we solve this problem in two folds. First, we fix W, solve $\boldsymbol{\beta}$ for channel selection. Second, we fix $\boldsymbol{\beta}$, solve W to reconstruct error.

(i) The subproblem of $\boldsymbol{\beta}$. In this case, W is fixed. We solve $\boldsymbol{\beta}$ for channel selection. This problem can be solved by LASSO regression [47, 5], which is widely used for model selection.

$$\begin{aligned} &\hat{\boldsymbol{\beta}}^{LASSO}(\lambda) = \underset{\boldsymbol{\beta}}{\arg\min} \frac{1}{2N} \left\| \mathrm{Y} - \sum_{i=1}^{c} \beta_i \mathrm{Z_i} \right\|_F^2 + \lambda \|\boldsymbol{\beta}\|_1 \\ &\text{subject to } \|\boldsymbol{\beta}\|_0 \leq c' \end{aligned} \tag{3}$$

Here $\mathrm{Z_i} = \mathrm{X_i} \mathrm{W_i}^\top$ (size $N \times n$). We will ignore ith channels if $\beta_i = 0$.

(ii) The subproblem of W. In this case, $\boldsymbol{\beta}$ is fixed. We utilize the selected channels to minimize reconstruction error. We can find optimized solution by least squares:

$$\underset{\mathrm{W}'}{\arg\min} \left\| \mathrm{Y} - \mathrm{X}'(\mathrm{W}')^\top \right\|_F^2 \tag{4}$$

Here $\mathrm{X}' = [\beta_1 \mathrm{X_1} \;\; \beta_2 \mathrm{X_2} \;\; ... \;\; \beta_i \mathrm{X_i} \;\; ... \;\; \beta_c \mathrm{X_c}]$ (size $N \times c k_h k_w$). W' is $n \times c k_h k_w$ reshaped W, $\mathrm{W}' = [\mathrm{W_1} \;\; \mathrm{W_2} \;\; ... \;\; \mathrm{W_i} \;\; ... \;\; \mathrm{W_c}]$. After obtained result W', it is reshaped back to W. Then we assign $\beta_i \leftarrow \beta_i \|\mathrm{W_i}\|_F$, $\mathrm{W_i} \leftarrow \mathrm{W_i} / \|\mathrm{W_i}\|_F$. Constrain $\forall i \, \|\mathrm{W_i}\|_F = 1$ satisfies.

We alternatively optimize (i) and (ii). In the beginning, W is initialized from the trained model, $\lambda = 0$, namely no penalty, and $\|\boldsymbol{\beta}\|_0 = c$. We gradually increase λ. For each change of λ, we iterate these two steps until $\|\boldsymbol{\beta}\|_0$ is stable. After $\|\boldsymbol{\beta}\|_0 \leq c'$ satisfies, we obtain the final solution W from $\{\beta_i \mathrm{W_i}\}$. In practice, we found that the two steps iteration is time consuming. So we apply (i) multiple times, until $\|\boldsymbol{\beta}\|_0 \leq c'$ satisfies. Then apply (ii) just once, to obtain the final result. From our observation, this result is comparable with two steps iteration's. Therefore, in the following experiments, we adopt this approach for efficiency.

Discussion: Some recent works [49, 1, 17] (though training based) also introduce ℓ_1-norm or LASSO. However, we must emphasis that we use different formulations. Many of them introduced sparsity regularization into training loss, instead of explicitly solving LASSO. Other work [1] solved LASSO, while feature maps or data were not considered during optimization. Because of these differences, our approach could be applied at inference time.

3.2. Whole Model Pruning

Inspired by [53], we apply our approach layer by layer sequentially. For each layer, we obtain input volumes from the current input feature map, and output volumes from the output feature map of the un-pruned model. This could be formalized as:

$$\begin{aligned} &\underset{\boldsymbol{\beta}, \mathrm{W}}{\arg\min} \frac{1}{2N} \left\| \mathrm{Y}' - \sum_{i=1}^{c} \beta_i \mathrm{X_i} \mathrm{W_i}^\top \right\|_F^2 \\ &\text{subject to } \|\boldsymbol{\beta}\|_0 \leq c' \end{aligned} \tag{5}$$

Different from Eqn. 1, Y is replaced by Y', which is from feature map of the original model. Therefore, the accumulated error could be accounted during sequential pruning.

3.3. Pruning Multi-Branch Networks

The whole model pruning discussed above is enough for single-branch networks like LeNet [30], AlexNet [26] and VGG Nets [44]. However, it is insufficient for multi-branch networks like GoogLeNet [46] and ResNet [18]. We mainly focus on pruning the widely used residual structure (*e.g.*, ResNet [18], Xception [7]). Given a residual block shown in Fig. 3 (left), the input bifurcates into shortcut and residual

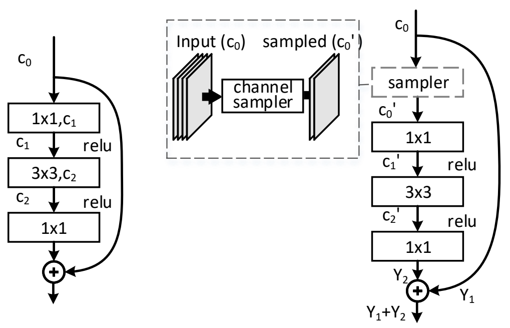

Figure 3. Illustration of multi-branch enhancement for residual block. **Left**: original residual block. **Right**: pruned residual block with enhancement, c_x denotes the feature map width. Input channels of the first convolutional layer are sampled, so that the large input feature map width could be reduced. As for the last layer, rather than approximate Y_2, we try to approximate $Y_1 + Y_2$ directly (Sec. 3.3 Last layer of residual branch).

branch. On the residual branch, there are several convolutional layers (*e.g.*, 3 convolutional layers which have spatial size of $1 \times 1, 3 \times 3, 1 \times 1$, Fig. 3, left). Other layers except the first and last layer can be pruned as is described previously. For the first layer, the challenge is that the large input feature map width (for ResNet, 4 times of its output) can't be easily pruned, since it's shared with shortcut. For the last layer, accumulated error from the shortcut is hard to be recovered, since there's no parameter on the shortcut. To address these challenges, we propose several variants of our approach as follows.

Last layer of residual branch: Shown in Fig. 3, the output layer of a residual block consists of two inputs: feature map Y_1 and Y_2 from the shortcut and residual branch. We aim to recover $Y_1 + Y_2$ for this block. Here, Y_1, Y_2 are the original feature maps before pruning. Y_2 could be approximated as in Eqn. 1. However, shortcut branch is parameter-free, then Y_1 could not be recovered directly. To compensate this error, the optimization goal of the last layer is changed from Y_2 to $Y_1 - Y'_1 + Y_2$, which does not change our optimization. Here, Y'_1 is the current feature map after previous layers pruned. When pruning, volumes should be sampled correspondingly from these two branches.

First layer of residual branch: Illustrated in Fig. 3(left), the input feature map of the residual block could not be pruned, since it is also shared with the shortcut branch. In this condition, we could perform *feature map sampling* before the first convolution to save computation. We still apply our algorithm as Eqn. 1. Differently, we sample the selected channels on the shared feature maps to construct a new input for the later convolution, shown in Fig. 3(right). Computational cost for this operation could be ignored. More importantly, after introducing *feature map sampling*, the convolution is still "regular".

Filter-wise pruning is another option for the first convolution on the residual branch. Since the input channels of parameter-free shortcut branch could not be pruned, we apply our Eqn. 1 to each filter independently (each filter chooses its own representative input channels). Under single layer acceleration, *filter-wise pruning* is more accurate than our original one. From our experiments, it improves 0.5% top-5 accuracy for 2× ResNet-50 (applied on the first layer of each residual branch) without fine-tuning. However, after fine-tuning, there's no noticeable improvement. In addition, it outputs "irregular" convolutional layers, which need special library implementation support. We do not adopt it in the following experiments.

4. Experiment

We evaluation our approach for the popular VGG Nets [44], ResNet [18], Xception [7] on ImageNet [9], CIFAR-10 [25] and PASCAL VOC 2007 [11].

For Batch Normalization [21], we first merge it into convolutional weights, which do not affect the outputs of the networks. So that each convolutional layer is followed by ReLU [37]. We use Caffe [23] for deep network evaluation, and scikit-learn [39] for solvers implementation. For channel pruning, we found that it is enough to extract 5000 images, and 10 samples per image, which is also efficient (i.e. several minutes for VGG-16 [2]). On ImageNet, we evaluate the top-5 accuracy with single view. Images are resized such that the shorter side is 256. The testing is on center crop of 224×224 pixels. We could gain more performance with fine-tuning. We use a batch size of 128 and learning rate $1e^{-5}$. We fine-tune our pruned models for 10 epoches (less than 1/10 iterations of training from scratch). The augmentation for fine-tuning is random crop of 224×224 and mirror.

4.1. Experiments with VGG-16

VGG-16 [44] is a 16 layers single-branch convolutional neural network, with 13 convolutional layers. It is widely used in recognition, detection and segmentation, *etc*. Single view top-5 accuracy for VGG-16 is 89.9%[3].

4.1.1 Single Layer Pruning

In this subsection, we evaluate single layer acceleration performance using our algorithm in Sec. 3.1. For better understanding, we compare our algorithm with two naive channel selection strategies. *first k* selects the first k channels. *max response* selects channels based on corresponding filters that have high absolute weights sum [31]. For fair comparison, we obtain the feature map indexes selected by each

[2]On Intel Xeon E5-2670 CPU

[3]http://www.vlfeat.org/matconvnet/pretrained/

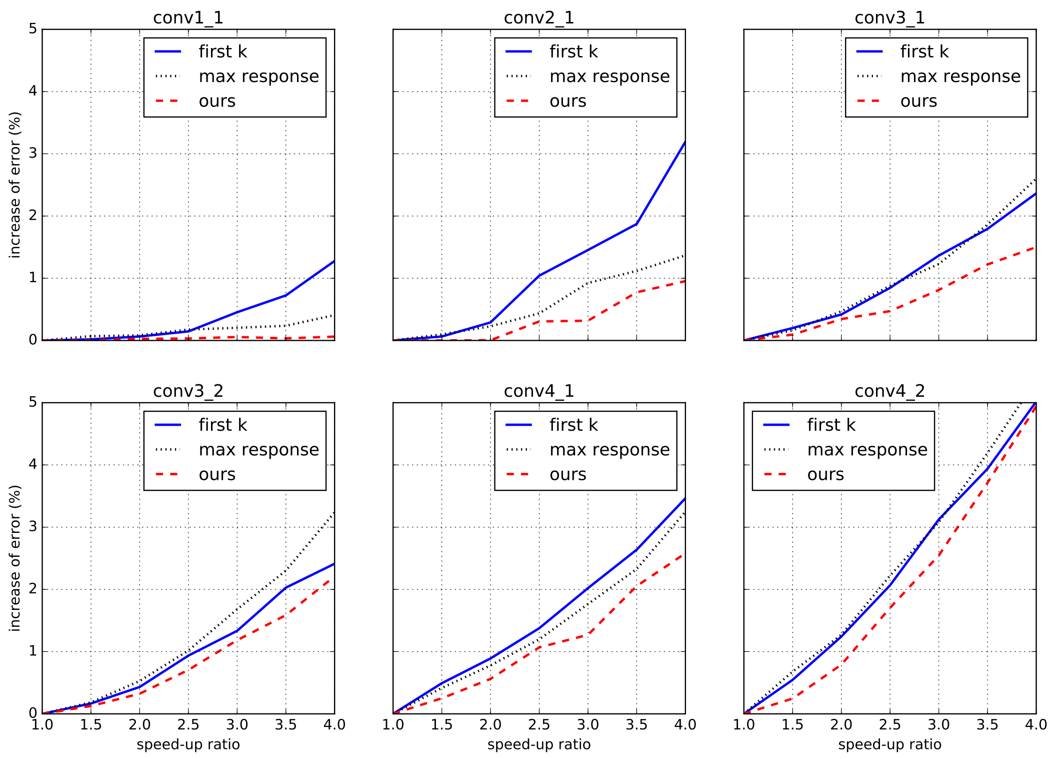

Figure 4. Single layer performance analysis under different speed-up ratios (without fine-tuning), measured by increase of error. To verify the importance of channel selection refered in Sec. 3.1, we considered two naive baselines. *first k* selects the first k feature maps. *max response* selects channels based on absolute sum of corresponding weights filter [31]. Our approach is consistently better (*smaller is better*).

of them, then perform reconstruction (Sec. 3.1 (ii)). We hope that this could demonstrate the importance of channel selection. Performance is measured by increase of error after a certain layer is pruned without fine-tuning, shown in Fig. 4.

As expected, error increases as speed-up ratio increases. Our approach is consistently better than other approaches in different convolutional layers under different speed-up ratio. Unexpectedly, sometimes *max response* is even worse than *first k*. We argue that *max response* ignores correlations between different filters. Filters with large absolute weight may have strong correlation. Thus selection based on filter weights is less meaningful. Correlation on feature maps is worth exploiting. We can find that channel selection affects reconstruction error a lot. Therefore, it is important for channel pruning.

Also notice that channel pruning gradually becomes hard, from shallower to deeper layers. It indicates that shallower layers have much more redundancy, which is consistent with [53]. We could prune more aggressively on shallower layers in whole model acceleration.

Increase of top-5 error (1-view, baseline 89.9%)			
Solution	$2\times$	$4\times$	$5\times$
Jaderberg *et al.* [22] ([53]'s impl.)	-	9.7	29.7
Asym. [53]	0.28	3.84	-
Filter pruning [31] (fine-tuned, our impl.)	0.8	8.6	14.6
Ours (without fine-tune)	2.7	7.9	22.0
Ours (fine-tuned)	0	1.0	1.7

Table 1. Accelerating the VGG-16 model [44] using a speedup ratio of $2\times$, $4\times$, or $5\times$ (*smaller is better*).

4.1.2 Whole Model Pruning

Shown in Table 1, whole model acceleration results under $2\times$, $4\times$, $5\times$ are demonstrated. We adopt whole model pruning proposed in Sec. 3.2. Guided by single layer experiments above, we prune more aggressive for shallower layers. Remaining channels ratios for shallow layers (`conv1_x` to `conv3_x`) and deep layers (`conv4_x`) is $1 : 1.5$. `conv5_x` are not pruned, since they only contribute 9% computation in total and are not redundant.

After fine-tuning, we could reach $2\times$ speed-up without losing accuracy. Under $4\times$, we only suffers 1.0% drops. Consistent with single layer analysis, our approach outper-

Increase of top-5 error (1-view, 89.9%)		
Solution	4×	5×
Asym. 3D [53]	0.9	2.0
Asym. 3D (fine-tuned) [53]	0.3	1.0
Our 3C	0.7	1.3
Our 3C (fine-tuned)	**0.0**	**0.3**

Table 2. Performance of combined methods on the VGG-16 model [44] using a speed-up ratio of $4\times$ or $5\times$. Our 3C solution outperforms previous approaches (*smaller is better*).

forms previous channel pruning approach (Li *et al*. [31]) by large margin. This is because we fully exploits channel redundancy within feature maps. Compared with tensor factorization algorithms, our approach is better than Jaderberg *et al*. [22], without fine-tuning. Though worse than Asym. [53], our combined model outperforms its combined Asym. 3D (Table 2). This may indicate that channel pruning is more challenging than tensor factorization, since removing channels in one layer might dramatically change the input of the following layer. However, channel pruning keeps the original model architecture, do not introduce additional layers, and the absolute speed-up ratio on GPU is much higher (Table 3).

Since our approach exploits a new cardinality, we further combine our channel pruning with spatial factorization [22] and channel factorization [53]. Demonstrated in Table 2, our 3 cardinalities acceleration (spatial, channel factorization, and channel pruning, denoted by 3C) outperforms previous state-of-the-arts. Asym. 3D [53] (spatial and channel factorization), factorizes a convolutional layer to three parts: $1 \times 3,\ 3 \times 1,\ 1 \times 1$.

We apply spatial factorization, channel factorization, and our channel pruning together sequentially layer-by-layer. We fine-tune the accelerated models for 20 epoches, since they are 3 times deeper than the original ones. After fine-tuning, our $4\times$ model suffers no degradation. Clearly, a combination of different acceleration techniques is better than any single one. This indicates that a model is redundant in each cardinality.

4.1.3 Comparisons of Absolute Performance

We further evaluate absolute performance of acceleration on GPU. Results in Table 3 are obtained under Caffe [23], CUDA8 [38] and cuDNN5 [6], with a mini-batch of 32 on a GPU (GeForce GTX TITAN X). Results are averaged from 50 runs. Tensor factorization approaches decompose weights into too many pieces, which heavily increase overhead. They could not gain much absolute speed-up. Though our approach also encountered performance decadence, it generalizes better on GPU than other approaches. Our results for tensor factorization differ from previous research [53, 22], maybe because current library and hardware prefer single large convolution instead of several small ones.

4.1.4 Comparisons with Training from Scratch

Though training a compact model from scratch is time-consuming (usually 120 epoches), it worths comparing our approach and from scratch counterparts. To be fair, we evaluated both from scratch counterpart, and normal setting network that has the same computational complexity and same architecture.

Shown in Table 4, we observed that it's difficult for from scratch counterparts to reach competitive accuracy. our model outperforms from scratch one. Our approach successfully picks out informative channels and constructs highly compact models. We can safely draw the conclusion that the same model is difficult to be obtained from scratch. This coincides with architecture design researches [20, 1] that the model could be easier to train if there are more channels in shallower layers. However, channel pruning favors shallower layers.

For from scratch (uniformed), the filters in each layers is reduced by half (eg. reduce `conv1_1` from 64 to 32). We can observe that normal setting networks of the same complexity couldn't reach same accuracy either. This consolidates our idea that there's much redundancy in networks while training. However, redundancy can be opt out at inference-time. This maybe an advantage of inference-time acceleration approaches over training-based approaches.

Notice that there's a 0.6% gap between the from scratch model and uniformed one, which indicates that there's room for model exploration. Adopting our approach is much faster than training a model from scratch, even for a thinner one. Further researches could alleviate our approach to do thin model exploring.

4.1.5 Acceleration for Detection

VGG-16 is popular among object detection tasks [43, 42, 34]. We evaluate transfer learning ability of our $2\times/4\times$ pruned VGG-16, for Faster R-CNN [43] object detections. PASCAL VOC 2007 object detection benchmark [11] contains 5k trainval images and 5k test images. The performance is evaluated by mean Average Precision (mAP) and mmAP (primary challenge metric of COCO [32]). In our experiments, we first perform channel pruning for VGG-16 on the ImageNet. Then we use the pruned model as the pre-trained model for Faster R-CNN.

The actual running time of Faster R-CNN is 220ms / image. The convolutional layers contributes about 64%. We got actual time of 94ms for $4\times$ acceleration. From Table 5, we observe 0.4% mAP drops of our $2\times$ model, which is not harmful for practice consideration. Observed from mmAP, For higher localization requirements our speedup model does not suffer from large degradation.

Model	Solution	Increased err.	GPU time/ms
VGG-16	-	0	8.144
VGG-16 (4×)	Jaderberg *et al.* [22] ([53]'s impl.)	9.7	8.051 (1.01×)
	Asym. [53]	3.8	5.244 (1.55×)
	Asym. 3D [53]	0.9	8.503 (0.96×)
	Asym. 3D (fine-tuned) [53]	**0.3**	8.503 (0.96×)
	Ours (fine-tuned)	1.0	**3.264** (2.50×)

Table 3. GPU acceleration comparison. We measure forward-pass time per image. Our approach generalizes well on GPU (*smaller is better*).

Original (acc. 89.9%)	Top-5 err.	Increased err.
From scratch	11.9	1.8
From scratch (uniformed)	12.5	2.4
Ours	18.0	7.9
Ours (fine-tuned)	11.1	**1.0**

Table 4. Comparisons with training from scratch, under 4× acceleration. Our fine-tuned model outperforms scratch trained counterparts (*smaller is better*).

Speedup	mAP	Δ mAP	mmAP	Δ mmAP
Baseline	68.7	-	36.7	-
2×	68.3	0.4	36.7	0.0
4×	66.9	1.8	35.1	1.6

Table 5. 2×, 4× acceleration for Faster R-CNN detection. mmAP is AP at IoU=.50:.05:.95 (primary challenge metric of COCO [32]).

Solution	Increased err.
Ours	8.0
Ours (enhanced)	4.0
Ours (enhanced, fine-tuned)	1.4

Table 6. 2× acceleration for ResNet-50 on ImageNet, the baseline network's top-5 accuracy is 92.2% (one view). We improve performance with multi-branch enhancement (Sec. 3.3, *smaller is better*).

4.2. Experiments with Residual Architecture Nets

For Multi-path networks [46, 18, 7], we further explore the popular ResNet [18] and latest Xception [7], on ImageNet and CIFAR-10. Pruning residual architecture nets is more challenging. These networks are designed for both efficiency and high accuracy. Tensor factorization algorithms [53, 22] have difficult to accelerate these model. Spatially, 1×1 convolution is favored, which could hardly be factorized.

4.2.1 ResNet Pruning

ResNet complexity uniformly drops on each residual block. Guided by single layer experiments (Sec. 4.1.1), we still prefer reducing shallower layers heavier than deeper ones.

Solution	Increased err.
Filter pruning [31] (our impl.)	92.8
Filter pruning [31] (fine-tuned, our impl.)	4.3
Ours	2.9
Ours (fine-tuned)	**1.0**

Table 7. Comparisons for Xception-50, under 2× acceleration ratio. The baseline network's top-5 accuracy is 92.8%. Our approach outperforms previous approaches. Most structured simplification methods are not effective on Xception architecture (*smaller is better*).

Following similar setting as Filter pruning [31], we keep 70% channels for sensitive residual blocks (`res5` and blocks close to the position where spatial size change, *e.g.* `res3a,res3d`). As for other blocks, we keep 30% channels. With multi-branch enhancement, we prune `branch2a` more aggressively within each residual block. The preserving channels ratios for `branch2a,branch2b,branch2c` is $2:4:3$ (*e.g.*, Given 30%, we keep 40%, 80%, 60% respectively).

We evaluate performance of multi-branch variants of our approach (Sec. 3.3). From Table 6, we improve 4.0% with our multi-branch enhancement. This is because we accounted the accumulated error from shortcut connection which could broadcast to every layer after it. And the large input feature map width at the entry of each residual block is well reduced by our *feature map sampling*.

4.2.2 Xception Pruning

Since computational complexity becomes important in model design, separable convolution has been payed much attention [50, 7]. Xception [7] is already spatially optimized and tensor factorization on 1×1 convolutional layer is destructive. Thanks to our approach, it could still be accelerated with graceful degradation. For the ease of comparison, we adopt Xception convolution on ResNet-50, denoted by Xception-50. Based on ResNet-50, we swap all convolutional layers with spatial conv blocks. To keep the same computational complexity, we increase the input channels of all `branch2b` layers by 2×. The baseline Xception-50 has a top-5 accuracy of 92.8% and complexity of 4450

Solution	Increased err.
Filter pruning [31] (fine-tuned, our impl.)	1.3
From scratch	1.9
Ours	2.0
Ours (fine-tuned)	**1.0**

Table 8. $2\times$ speed-up comparisons for ResNet-56 on CIFAR-10, the baseline accuracy is 92.8% (one view). We outperforms previous approaches and scratch trained counterpart (*smaller is better*).

MFLOPs.

We apply multi-branch variants of our approach as described in Sec. 3.3, and adopt the same pruning ratio setting as ResNet in previous section. Maybe because of Xception block is unstable, Batch Normalization layers must be maintained during pruning. Otherwise it becomes nontrivial to fine-tune the pruned model.

Shown in Table 7, after fine-tuning, we only suffer **1.0%** increase of error under $2\times$. Filter pruning [31] could also apply on Xception, though it is designed for small speed-up ratio. Without fine-tuning, top-5 error is 100%. After training 20 epochs which is like training from scratch, increased error reach 4.3%. Our results for Xception-50 are not as graceful as results for VGG-16, since modern networks tend to have less redundancy by design.

4.2.3 Experiments on CIFAR-10

Even though our approach is designed for large datasets, it could generalize well on small datasets. We perform experiments on CIFAR-10 dataset [25], which is favored by many acceleration researches. It consists of 50k images for training and 10k for testing in 10 classes.

We reproduce ResNet-56, which has accuracy of 92.8% (Serve as a reference, the official ResNet-56 [18] has accuracy of 93.0%). For $2\times$ acceleration, we follow similar setting as Sec. 4.2.1 (keep the final stage unchanged, where the spatial size is 8×8). Shown in Table 8, our approach is competitive with scratch trained one, without fine-tuning, under $2\times$ speed-up. After fine-tuning, our result is significantly better than Filter pruning [31] and scratch trained one.

5. Conclusion

To conclude, current deep CNNs are accurate with high inference costs. In this paper, we have presented an inference-time channel pruning method for very deep networks. The reduced CNNs are inference efficient networks while maintaining accuracy, and only require off-the-shelf libraries. Compelling speed-ups and accuracy are demonstrated for both VGG Net and ResNet-like networks on ImageNet, CIFAR-10 and PASCAL VOC.

In the future, we plan to involve our approaches into training time, instead of inference time only, which may also accelerate training procedure.

References

[1] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2262–2270, 2016. 1, 2, 3, 6

[2] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *arXiv preprint arXiv:1512.08571*, 2015. 2

[3] S. Anwar and W. Sung. Compact deep convolutional neural networks with coarse pruning. *arXiv preprint arXiv:1610.09639*, 2016. 1, 2

[4] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. *arXiv preprint arXiv:1611.06473*, 2016. 2

[5] L. Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373–384, 1995. 3

[6] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. 6

[7] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016. 1, 2, 3, 4, 7

[8] 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. 1, 2

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

[10] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014. 2

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 4, 6

[12] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015. 2

[13] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 2

[14] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016. 2

[15] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254. IEEE Press, 2016. 2

[16] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015. 2

[17] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 1, 2, 3

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1, 2, 3, 4, 7, 8

[19] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 2

[20] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016. 6

[21] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[22] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 1, 2, 5, 6, 7

[23] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 4, 6

[24] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 2

[25] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 4, 8

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2, 3

[27] A. Lavin. Fast algorithms for convolutional neural networks. *arXiv preprint arXiv:1509.09308*, 2015. 2

[28] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 2

[29] V. Lebedev and V. Lempitsky. Fast convnets using groupwise brain damage. *arXiv preprint arXiv:1506.02515*, 2015. 2

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2, 3

[31] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 1, 2, 4, 5, 7, 8

[32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, Cham, 2014. 6, 7

[33] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015. 2

[34] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. 6

[35] Z. Mariet and S. Sra. Diversity networks. *arXiv preprint arXiv:1511.05077*, 2015. 2

[36] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013. 2

[37] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 4

[38] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008. 6

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 4

[40] A. Polyak and L. Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015. 2

[41] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2

[42] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 6

[43] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. 6

[44] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3, 4, 5, 6

[45] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015. 2

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 1, 3, 7

[47] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. 3

[48] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014. 1, 2

[49] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances In*

Neural Information Processing Systems, pages 2074–2082, 2016. 1, 2, 3

[50] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016. 7

[51] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH*, pages 2365–2369, 2013. 2

[52] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint arXiv:1611.05128*, 2016. 2

[53] 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. 1, 2, 3, 5, 6, 7

[54] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer International Publishing, 2016. 2