Factorized Convolutional Neural Networks

Min Wang
University of Central Florida
Orlando, FL, 32816
mwang@cs.ucf.edu

Baoyuan Liu
Amazon.com
Seattle, WA, 98109
baoyuan@amazon.com

Hassan Foroosh
University of Central Florida
Orlando, FL, 32816
foroosh@cs.ucf.edu

Abstract

In this paper, we propose to factorize the convolutional layer to reduce its computation. The 3D convolution operation in a convolutional layer can be considered as performing spatial convolution in each channel and linear projection across channels simultaneously. By unravelling them and arranging the spatial convolutions sequentially, the proposed layer is composed of a low-cost single intra-channel convolution and a linear channel projection. When combined with residual connection, it can effectively preserve the spatial information and maintain the accuracy with significantly less computation. We also introduce a topological subdivisioning to reduce the connection between the input and output channels. Our experiments demonstrate that the proposed layers outperform the standard convolutional layers on performance/complexity ratio. Our models achieve similar performance to VGG-16, ResNet-34, ResNet-50, ResNet-101 while requiring ***42****×,****7.32****×,****4.38****×,****5.85****× less computation respectively.*

1. Introduction

Deep convolutional neural networks (CNN) have made significant improvement on solving visual recognition problems since 2012[13][20][9][19][6]. Thanks to their deep structure, vision oriented layer designs, and efficient training schemes, state-of-the-art CNN models [19][6] obtain better than human level accuracy on ImageNet LSVRC dataset [2].

The computational complexity for the state-of-the-art models for both training and inference are extremely high, requiring several GPUs to train for hundreds of hours. The most time-consuming building block of the CNN, the convolutional layer, is performed by convolving the 3D input data with a series of 3D kernels. The complexity of a convolutional layer is quadratic with three hyper-parameters: kernel size, number of channels, and spatial dimensions.On one hand, the recognition capability of the network is highly correlated with those hyper-parameters; On the other hand, a balance has to be cautiously controlled among them to keep the computation affordable. In practice, 1×1 and 3×3 kernels are widely used, and the increase of channels is often accompanied with the reduction of spatial dimensions.

Several attempts have been made to reduce the amount of computation in convolutional layers. Low rank decomposition has been extensively explored in various fashions [3][10][25][8][22][11][24] to obtain moderate efficiency improvement. Sparse decomposition based methods [16][5] achieve higher theoretical reduction of complexity, while the actual speedup is bounded by the efficiency of sparse multiplication implementations. Most of these decomposition-based methods start from a pre-trained model, and perform decomposition and fine-tuning while trying to maintain the accuracy. This essentially precludes the option of designing and training new efficient CNN models from scratch.

On the other hand, in recent state-of-the-art deep CNN models, several heuristics are adopted to alleviate the burden of heavy computation. In [20], the number of channels are reduced by a linear projection before the actual convolutional layer; In [6], the authors utilize a bottleneck structure, in which both the input and the output channels are reduced by linear projection; In [21], $1 \times n$ and $n \times 1$ asymmetric convolutions are adopted to achieve larger kernel sizes. While these strategies help to moderately trim down the computation of deep models in practice, they are not able to provide a comprehensive analysis of optimizing the efficiency of the convolutional layer.

In this work, we propose to factorize the standard convolutional layer to reduce the computational complexity. In standard convolutional layers, the 3D convolution can be considered as performing intra-channel spatial convolution and linear channel projection simultaneously, leading to highly redundant computation. Following [16], these two operations are first unraveled to 2D convolution with a set of bases in each channel and subsequent linear channel projection. Then, we make the further modification of performing the 2D convolutions **sequentially** rather than in parallel. In this way, we obtain a convolutional layer that involves only one 2D filter (basis) for each input channel, of which the computation is negligible, and linear channel projection, thus achieving significantly reduced complexity. Combined

with the residual connection, the missing spatial information from single 2D filter intra-channel convolution can be retrieved. We call this kind of single intra-channel convolutional (SIC) layer. By stacking multiple SIC layers, we can train models that have several times lower complexity, but achieve similar or higher accuracy than models based on standard convolutional layer. Additionally, the negligible complexity from the single 2D filter in the proposed SIC layer makes it possible to use larger the kernel size to further improve the classification accuracy without too much computation increase.

In an SIC layer, linear channel projection consumes the majority of the computation. To reduce its complexity, we propose a topological subdivisioning(TpS) framework between the input and output channels as follows: The input and output channels are rearranged into multi-dimensional tensors. Each output channel is only connected to the input channels that are within its local neighborhood. Such a framework leads to a regular sparsity pattern of the channel projection kernels, which is experimentally shown to possess a better performance/cost ratio than both the standard convolutional layer and the grouping strategy in [13].

The above two designs (SIC layer and topological subdivisioning) can improve the efficiency of traditional CNN models from different perspectives, and can be combined to achieve even lower complexity as demonstrated thoroughly in the remainder of this paper. The proposed methods will be explained in detail in Section 2, evaluated against traditional CNN models, and analyzed in Section 3.

2. Methods

In this section, we first review the standard convolutional layer, then introduce the proposed methods. In this work, all the input is padded with zeros so that the spatial dimensions of output is the same as input. We also assume that the residual learning technique is applied to each convolutional layer or channel-wise bottleneck structure that have same dimension of input and output.

2.1. Standard Convolutional Layer

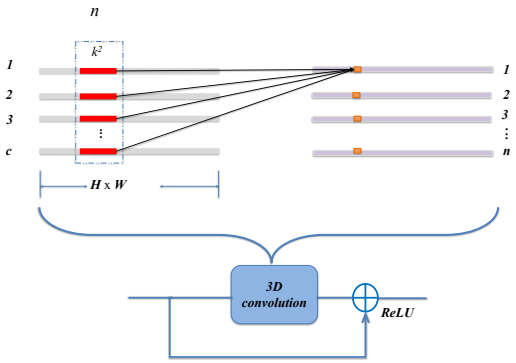
Consider the input data $\mathbf{I}$ in $\mathbb{R}^{h\times w\times c}$, where h, w and c are the height, width and the number of channels of the input feature maps, and the convolutional kernel $\mathbf{K}$ in $\mathbb{R}^{k\times k\times c\times n}$, where k is the size of the convolutional kernel and n is the number of kernels.The operation of a standard convolutional layer $\mathbf{O}\in\mathbb{R}^{h\times w\times n}=\mathbf{K}*\mathbf{I}$ is represented by the following formula.

$$\mathbf{O}(y,x,j)=\sum_{i=1}^{c}\sum_{u=1}^{k}\sum_{v=1}^{k}\mathbf{K}(u,v,i,j)\mathbf{I}(y+u-1,x+v-1,i) \tag{1}$$

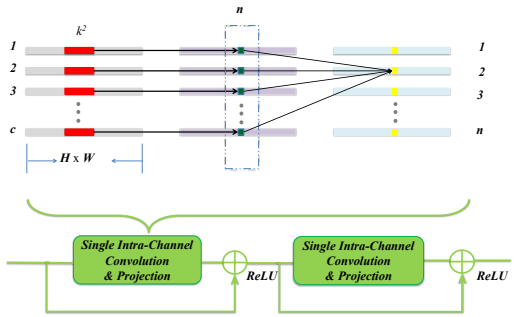
Where $1\le y\le h, 1\le x\le w, 1\le j\le n$.

The complexity of a convolutional layer measured by the number of multiplications is

$$nck^2hw \tag{2}$$



(a) Standard Convolutional Layer with residual connection



(b) Single Intra-Channel Convolutional Layer

Figure 1. Illustration of the convolution pipeline of standard convolutional layer and Single Intra-channel Convolutional(SIC) layer. In SIC layer, each input channel is first convolved with single 2D filter, then a linear projection (1×1 convolution) is applied to all the channels and followed by the residual connection from the input. Multiple SIC layers are stacked sequentially with ReLU.

Since the complexity is quadratic with the kernel size, in most recent CNN models, the kernel size is limited to 3×3 to control the overall running time.

2.2. Single Intra-Channel Convolutional Layer

2.2.1 Unravel 3D convolution

In standard convolutional layers, the output features are produced by convolving a group of 3D kernels with the input features along the spatial dimensions. Such a 3D convolution operation can be considered as a combination of 2D spatial convolution inside each channel (intra-channel convolution) and linear projection across channels. For each output channel, a spatial convolution is performed on each input channel. The spatial convolution is able to capture local structural information, while the linear projection transforms the feature space for learning the necessary non-linearity in the neuron layers. When the number of input and output channels is large, typically hundreds, such a 3D convolutional layer requires an exorbitant amount of computation. A natural idea is, the 2D spatial convolution and linear channel projection can be unraveled and performed separately.

As is described in [16], the 4D convolutional kernel $\mathbf{K}$

can be decomposed into the bases filters $\mathbf{B} \in \mathbb{R}^{k\times k\times b\times c}$, and the linear projection tensor $\mathbf{P} \in \mathbb{R}^{b\times c\times n}$, where b is the number of the bases, so that

$$\mathbf{K}(u,v,i,j) = \sum_{t=1}^{b} \mathbf{B}(u,v,t,i)\mathbf{P}(t,i,j) \tag{3}$$

For each input channel, the convolutional kernel is decomposed along the spatial dimensions. The operation of convolutional layer in Equation.1 is converted to

$$\mathbf{J}(y,x,t,i) = \sum_{u=1}^{k}\sum_{v=1}^{k} \mathbf{B}(u,v,t,i)\mathbf{I}(y+u-1,x+v-1,i) \tag{4}$$

$$\mathbf{O}(y,x,j) = \sum_{i=1}^{c}\sum_{t=1}^{b} \mathbf{P}(t,i,j)\mathbf{J}(y,x,t,i) \tag{5}$$

where $\mathbf{J} \in \mathbb{R}^{h\times w\times b\times c}$. Each channel of the input feature map is first convolved with b 2D filters(bases), generating an intermediate feature map $\mathbf{J}$, of which the number of channels is b times of the input. Then a linear channel projection is used to project the intermediate feature map onto the output.

Unravelling these two operations (2D spatial convolution and linear projection) provides us more freedom of model design by tuning both the number of 2D bases b and kernel size k. The complexity of such a layer is

$$bck^2hw + bcnhw \tag{6}$$

The first term corresponds to 2D spatial convolution and the second term corresponds to linear projection. Typically, k^2 is much smaller than n, and the majority of the computation is consumed by the linear projection, which is linear with the number of basis filter b. When $b = k^2$, this is equivalent to a full-rank decomposition of the convolutional kernel for each input channel along the spatial dimensions. The complexity of the linear projection is the same as the standard convolutional layer; By assigning $b < k^2$, the complexity is lower than the standard convolutional layer in a low-rank fashion.

2.2.2 Sequential Arrangement

Our key observation is that instead of convolving b 2D filters(bases) with each input channel simultaneously and perform linear projection over them altogether, we can arrange the convolutions sequentially, interlearving with linear projection. The above convolutional layer with b 2D filters can be transformed to a framework that has b layers. In each layer, each input channel is first convolved with **single** 2D filter(basis), then a linear projection is applied to all channels. In this manner, the number of channels are maintained the same as the input throughout all b layers. Note that such sequential arrangment keeps exactly the same computational complexity, with increased depth and learning capability. The complexity of each layer is thus

$$ck^2hw + cnhw \tag{7}$$

2.2.3 Residual Connection

When we consider each of the b layers, only one $k \times k$ kernel is convolved with each input channel. This is equivalent to a rank-1 decomposition and may seem to be a risky choice. Each convolutional layer in a CNN network learns higher abstract level of representation by transforming the input information into a different space[15]. In such a process, the useful information from the input is passed to the subsequent layers in a more discriminative structure. With single 2D filter for each input channel, we face the challenge of not being able to keep all the useful information from the input data. For instance, if one filter is learned to become a vertical edge detector, then all the horizontal information from the input channel will be lost after passing through this filter. The subsequent layers will not recover this information since this filter is the single path from the input channel. This limitation potentially constrains the 2D filters from learning discriminative local structure. However, the addition of residual connection helps to overcome this disadvantage.

For each layer, the residual connection is added from the input of 2D intra-channel convolution to the output of linear projection. The subsequent layers thus receive information from both the initial input and the output of preceding layers. We name such single intra-channel convolution and linear projection with residual connection structure as SIC layer. Figure.4 presents a graphical comparison between the standard convolutional layer and our SIC layer.

2.2.4 Increasing Kernel Size

One advantage of unravelled spatial convolution and linear projection is that, the complexity of the linear projection, which is the majority of the layer, is not dependent on the kernel size. While the complexity of standard convolutional layer is proportional to k^2, the complexity of the SIC layer is proportional to $1 + \frac{k^2}{n}$. With $k^2 << n$, increasing k will only lead to a fractional increase of overall complexity. This property provides us the flexibility of adopting larger convolutional kernel size. We will show in the experiment section that this is an effective way of improving the performance/complexity ratio.

2.3. Topological Subdivisioning

While SIC layer significantly reduces the complexity of spatial convolution in a convolutional layer, we make a further attempt to reduce the density of connection between the input channels and output channels. In a standard convolutional layer, every output channel is connected to every input channel. The complexity is proportional to the product

of input and output channels. We argue that such dense connection is redundant. The grouping strategy in Alexnet[13], although was proposed to reduce data transfering between multiple GPUs, can be considered as the simplest way of exploring such redundancy. In [16], the authors proved that extremely high sparsity of the connections could be accomplished without sacrificing accuracy. While the sparsity was obtained by fine-tuning and did not possess any structure, we study to build the sparsity with more regularity.

Inspired by the topological ICA framework in [7], we propose a s-dimensional topological subdivisioning between the input and output channels in the convolutional layers. Without loss of generality, we assume the number of input channels and output channels are both n. We first arrange the input and output channels as an s-dimensional tensor $[d_1, d_2, ..., d_s]$, so that $n = \prod_{i=1}^{s} d_i$.

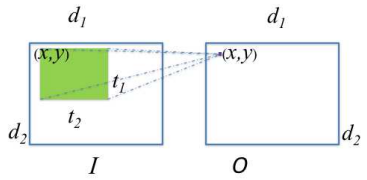
Each output channel is only connected to its local neighbors in the tensor space rather than all input channels. The size of the local neighborhood is defined by another s-dimensional tensor, $[t_1, t_2, ..., t_s]$. With input and output $\mathbf{I}, \mathbf{O} \in \mathbb{R}^{h \times w \times d_1 \times \cdots \times d_s}$, and kernel $\mathbf{K} \in \mathbb{R}^{k \times k \times t_1 \times \cdots \times t_s \times d_1 \times \cdots \times d_s}$, the topological subdivisioned convolutional layer can be formulated as:

$$\begin{aligned} &\mathbf{O}(y, x, j_1, \cdots, j_s) \qquad (8) \\ &= \sum_{u,v=1}^{k} \sum_{i_1=1}^{t_1} \cdots \sum_{i_s=1}^{t_s} \mathbf{K}(u, v, i_1, \cdots, i_s, j_1, \cdots, j_s) \cdot \\ &\quad \tilde{\mathbf{I}}(y + u - 1, x + v - 1, j_1 + i_1 - 1, \cdots, j_s + i_s - 1) \end{aligned}$$

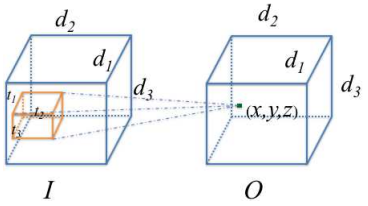
We use circular indexing to handle the boundary cases so that all the output channels are connected to same number of input channels. The complexity of the proposed topologically subdivisioned convolutional layers compared to the standard convolutional layers can be simply measured by $\frac{\prod_{i=1}^{s} t_i}{n}$. This layer can substitute a standard convolutional layer when $k > 1$, or the linear projection in SIC layer when $k = 1$. Figure. 2 illustrates the 2D and 3D topological subdivisioning between the input and output channels.

3. Experiments

We evaluate the performance of our methods on the ImageNet LSVRC 2012 dataset, which contains 1000 categories, with 1.2M training images, 50K validation images, and 100K test images. We use Torch to train the CNN models in our framework. Our method is implemented with CUDA and Lua based on the Torch platform. The images are first resized with its shorter edge equal to 256, then randomly cropped into 221×221 and flipped horizontally while training. Batch normalization [9] is placed after each convolutional layer and before the ReLU layer. We also adopt the dropout [18] strategy with a ratio of 0.2 during training. Standard stochastic gradient descent with mini-batch containing 256 images is used to train the model. We start the learning rate from 0.1 and divide it by a factor of 10 when training error reaches plateau. For batch normalization, we use exponential moving average to calculate the batch statistics as is implemented in cuDNN [1]. The code is run on a server with 4 Titan X GPU. For all the evaluated models, the top-1 and top-5 errors of validation set with center cropping are reported.



(a) 2D Topology



(b) 3D Topology

Figure 2. 2D and 3D topological subdivisioning. The input channels are re-arranged into 2D and 3D tensors. Each output channel is only connected with the input channels that are in its local neighborhood in the tensor space. The complexity is determined by the neighborhood size.

Baseline we compare the performance and complexity of our methods against a baseline CNN model that is built from standard convolutional layers. The details of the **baseline model A** are given in Table 1. The convolutional layers are divided into stages according to their spatial dimensions. Each stage includes 2 convolutional layers with kernel size 3×3. Inside each stage, the convolutional kernels are performed with paddings so that the output has the same spatial dimensions as the input. Across the stages, the spatial dimensions are reduced by max pooling and the number of channels are doubled by 1×1 convolutional layer. One fully connected layer with dropout is added before the logistic regression layer for final classification. Residual connection is added after every convolutional layer with same number of input and output channels. The baseline model **A** achieves 30.67% top-1 error rate with 1093M multiplications for each image. It should be noted that, with regard to performance/complexity ratio, this model compares favorably to GoogLeNet[20], VGG16[17] and ResNet-18[6] models.

We substitute the standard convolutional layers in the baseline model **A** with the proposed low complexity layers. The 7×7 convolutional layer in the first stage and the 1×1 convolutional layers across stages are left unchanged, and only the 3×3 convolutional layers are substituted. In the following sections, the relative complexity is also measured with regards to those layers. To make cross reference easier and help readers keep track of all the models, each model is indexed with a capital letter.

Stage	Output	**A**	**B**	**C**	**D**	**E**	**F**
1	108^2	$(7, 64)_2$					
2	36^2	3×3 max pooling , stride 3					
		(1, 128)					
		$(3, 128) \times 2$	$[3, 4, 128] \times 2$	$< 3, 128 > \times 2$	$< 3, 128 > \times 4$	$< 3, 128 > \times 6$	$< 5, 128 > \times 4$
3	18^2	2×2 max pooling , stride 2					
		(1, 256)					
		$(3, 256) \times 2$	$[3, 4, 256] \times 2$	$< 3, 256 > \times 2$	$< 3, 256 > \times 4$	$< 3, 256 > \times 6$	$< 5, 256 > \times 4$
4	6^2	3×3 max pooling , stride 3					
		(1, 512)					
		$(3, 512) \times 2$	$[3, 4, 512] \times 2$	$< 3, 512 > \times 2$	$< 3, 512 > \times 4$	$< 3, 512 > \times 6$	$< 5, 512 > \times 4$
		(1, 1024)					
	1^2	6×6 average pooling, stride 6 fully connected, 2048 fully connected, 1000 softmax					
FLOPs		1093M	593M	268M	376M	485M	393M

Table 1. Configurations of baseline models and models with proposed SIC layers . For each convolutional layer, we use numbers in brackets to represent its configuration. Each type of bracket correspond to one type of convolutional layer. With kernel size k, and n output channels, (k, n) stands for a standard convolutional layer; $[k, b, n]$ denotes an unraveled convolutional layer with b 2D filters(bases) for each input channel; $< k, n >$ represents an SIC layer. The number after the brackets indicates the times that the layer is repeated in the stage. Model **A** is the **baseline** model with standard convolutional layers. Model **B** replaces each standard convolutional layer in stage 2, 3, 4 of model **A** with an unraveled convolutional layer with 4 2D filters. Model **C**,**D** and **E** replace one standard convolutional layer in model **A** with 1, 2 and 3 SIC layers, respectively. The kernel size of models **A** to **E** is 3×3. Model **F** has the same architecture as model **D** except for larger kernel size (5×5).

3.1. Single Intra-Channel Convolutional Layer

	Method	Top-1	Top-5	Comp
A	Standard Convolution	30.67%	11.24%	1
B	Unraveled Convolution	30.69%	11.27%	~ 4/9
C	**SIC**, 2 layers / stage	32.00%	12.13%	~ 1/9
D	**SIC**, 4 layers / stage	29.78%	10.78%	~ 2/9
E	**SIC**, 6 layers / stage	28.83%	9.88%	~ 1/3

Table 2. Top-1 & Top-5 error and complexity per stage of model **A** to **E**. The kernel size in these models is 3×3. The models with SIC layers (model **C**, **D**, **E**)demonstrate significantly better performance / complexity ratio than the baseline model **A**.

We first substitute the standard convolutional layers in model **A** with unraveled convolution layers in model **B**. Each input channel is convolved with 4 2D filters(bases), so that the complexity of **B** is approximately $\frac{4}{9}$ of the baseline model **A**. We then substitute standard convolutional layers with the proposed SIC layers to form model **C**, **D** and **E**. In a typical SIC layer with 3×3 kernel and more than 100 channels, the 2D spatial convolution consumes less than 10% of total computation. The complexity of each SIC layer is approximatley $\frac{1}{9}$ of a standard 3×3 convolutional layer in model **A**. Due to the extremely low complexity of SIC layer, we can experiment with replacing one convolutional layer using one or more SIC layers. While model **A** has 2 convolutional layers per stage, model **C** **D** and **E** has 2, 4 and 6 SIC layers per stage, which correpond to $\frac{1}{9}$, $\frac{2}{9}$ and $\frac{1}{3}$ the complexity of model **A** respectively. Table 1 provides the detailed model configurations.

Table 2 lists the top-1 and top-5 errors and the complexity of models from **A** to **E**. With unravelled convolution, model **B** matches the error rate of model **A** with same number of layers but only $\frac{4}{9}$ amount of computation. When compared with model **A**, with the proposed SIC layer, model **C**, **D** and **E** achieves $+\mathbf{1.37}\%$, $-\mathbf{0.89}\%$, and $-\mathbf{1.84}\%$ relative top-1 error with $\frac{\mathbf{1}}{\mathbf{9}}$, $\frac{\mathbf{2}}{\mathbf{9}}$ and $\frac{\mathbf{1}}{\mathbf{3}}$ amount of computation. The comparison results demonstrate that our SIC layer based models are able to achieve remarkably higher performance/complexity ratio than standard convolutional layer based model.

	kernel size	Top-1	Top-5	FLOPs
D	3×3	29.78%	10.78%	376M
F	5×5	29.23%	10.48%	393M

Table 3. Compare the performance of SIC layer models with different kernel size. Increasing the kernel size from 3×3 in model **D** to 5×5 in model **F** will gain 0.5% better performance with less than 5%(17M) complexity increase.

Increasing kernel size

As is analyzed in Section 2.2.4, we can increase the kernel size of SIC layer with very little increase of complexity. To demonstrate this advantage, we design model **F** by increasing the kernel size of model **D** from 3×3 to 5×5. Table 3

compares the performance and error rate between model **D** and **F**. With 5×5 kernel size, model **F** obtains 0.5% performance gain with less than 5%(17M) increase of complexity. Increasing kernel size in the SIC layer provides us another choice of improving the performance/complexity ratio.

3.2. Topological Subdivisioning

We first evaluate the performance of two topological subdivisioning(TpS) configurations on the standard convolutional layers. The models with topological subdivisioning are modified from model **A** by replacing each convolutional layer with two convolutional layers with topological subdivisioning. Model **G** adopts 2D TpS with $t_i = d_i/2$ for both dimensions, which leads to a reduction of complexity by a factor of 4. In model **H**, we use 3D TpS and set t_i and d_i, so that the complexity is reduced by a factor of 4.27. The details of the network configuration are listed in Table 4. Since the number of layers with TpS is twice the number of convolutional layers in the baseline model **A**, the overall complexity per stage is approximately half of model **A**.

Stage	#Channels	2D TpS $d_1 \times d_2$ $t_1 \times t_2$	3D TpS $d_1 \times d_2 \times d_3$ $t_1 \times t_2 \times t_3$
2	128	8×16 4×8	$4 \times 8 \times 4$ $2 \times 5 \times 3$
3	256	16×16 8×8	$8 \times 8 \times 4$ $4 \times 5 \times 3$
4	512	16×32 8×16	$8 \times 8 \times 8$ $4 \times 5 \times 6$

Table 4. Configurations of model **G** and **H** that use 2D and 3D TpS. d_i and c_i stand for the tensor and neighbor dimensions. They are designed so that the complexity is reduced by (approximately for 3D) a factor of 4.

As a comparison, we also train a model **I** by applying the straightforward grouping strategy [13] to model **A**. Both the input and output channels are divided into 4 groups. The output channels in each group are only connected to the input channels in the corresponding group. The complexity of each layer is also reduced 4 times in this manner. Similarly, we double the number of layers per stage to make half overall complexity.

Table 5 lists the top-1 & top-5 error rate and complexities of model **G** to **I**.When compared with model **A**, while the grouping strategy(model **I**) results in 0.56% higher top-1 error, both the 2D and 3D TpS models(**G** and **H**) maintain the same or lower error rate with half complexity.

SIC layer with Topological Subdivisioning

Finally, we apply the Topological Subdivisioning to our SIC layer models to further reduce the complexity. Model **J** is modified from SIC layer based model **D** by doubling the number of SIC layers in each stage and applying 2D TpS to the linear projection in each SIC layer.

	Methods	Top-1	Top-5	Comp
A	standard convolution	30.67%	11.24%	1
G	2D TpS	30.53%	11.28%	$\sim$ 1/2
H	3D TpS	30.69%	11.38%	$\sim$ 15/32
I	Grouping[13]	31.23%	11.73%	$\sim$ 1/2

Table 5. Comparing the performace of the TpS layer to standard convolutional layer and grouping strategy. The 2D and 3D TpS models achieve similar performance to the baseline model **A** with half complexity per stage, and outperform the model with grouping that has similar complexity.

Table 6 evaluates the performance of SIC layer with 2D TpS structure by comparing with same complexity SIC layer model **C** and standard convolutional layer model **A**. Compared to the baseline model **A**, this model achieves similar error rate with $\frac{1}{9}$ complexity per stage. Compared to model **C** that uses SIC layer only, model **J** achieves 1.2% lower top-1 error rate with same complexity.

	Methods	Top-1	Top-5	Comp
A	Standard Convolution	30.67%	11.24%	1
C	SIC, 2 layers/stage	32.00%	12.13%	$\sim$ 1/9
J	SIC+2D TpS	30.78%	11.29%	$\sim$ 1/9

Table 6. Compare SIC layer with 2D TpS with SIC layer and standard convolutional layer. With $\frac{1}{9}$ complexity of standard convolutional layer model **A**, model **C** with only SIC layer has 1.2% higher top-1 error than model **A**, while model **J** that is built from SIC layer with 2D Tps can match the error rate of model **A**.

3.3. Comparison with standard CNN models

In this section, we increase the depth of our models to compare with recent state-of-the-art CNN models (VGG-16 and ResNet models). To go deeper without increasing too much complexity, we adopt the channel-wise bottleneck structure in a similar fashion to [6]. In each channel-wise bottleneck structure, the number of channels is first reduced by half by the first layer, then recovered by the second layer. Such a two-layer bottleneck structure has approximately the same complexity to single layer with equal number of input and output channels, thus increase the overall depth of the network.

In both SIC layer and topological subdivisioning, we substitute one high complexity layer with two low complexity layers. These scheme reduces the amount of computation while at the same time increases the depth of the network. When training deep models with state-of-the-art performance, we will not have sufficient GPU memory if both methods are applied. Therefore, only the SIC layer is used in the models evaluated in this section.

Memory Usage the training memory consumption of our models is similar to the corresponding ResNet models with comparable performance.

Models Setting We gradually increase the number of SIC

Model	Top-1 Err	Top-5 Err	FLOPs	# Parameters	Speedups	
					Theoretical	Actual
VGG-16	28.5%	9.9%	16000M	138M	×1	×1
Model L	**28.30%**	**9.9%**	**381M**	**6.3M**	**×42.0**	**×14.00**
ResNet-34	26.73%	8.74%	3600M	21.8M	×1	×1
Model M	**26.54%**	**8.77%**	**492M**	**7M**	**×7.32**	**×2.74**
ResNet-50	24.01%	7.02%	3800M	24M	×1	×1
Model N	**24.11%**	**7.15%**	**866M**	**10.3M**	**×4.38**	**×1.89**
ResNet-101	22.44%	6.21%	7600M	43M	×1	×1
Model O	23.33%	6.90%	**1300M**	**14.7M**	**×5.85**	**×2.19**

Table 7. Top-1 and Top-5 validation error rate, number of FLOPs, parameters of our model and several previous work, theoretical and actual speedups(measured by the average running time for the **forward pass of the whole network**). The numbers in this table are generated with single model and center-crop. For ResNet-34, ResNet-50, ResNet-101 we use the number with Facebook's implementation[4].

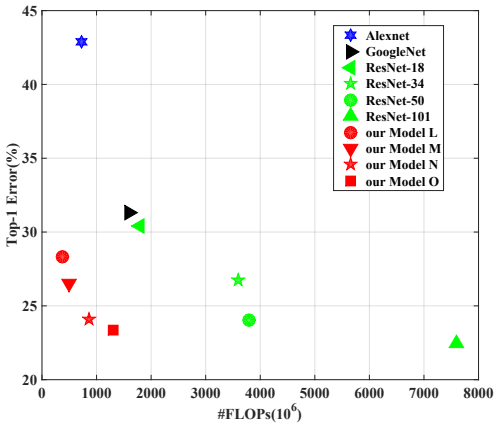


Figure 3. Comparing top-1 error and complexity between our models and several previous work. It demonstrates that our models(red marks) have significant performance/complexity ratio than previous work.

	Top-5(Val)	Top-5(Test)
Model **N**	7.15%	7.39%
Model **O**	6.90%	6.98%

Table 8. Top-5 error rate(single model, center-crop) for model **N** and **O** on validation and test set.

layers with channel-wise bottleneck structure in each stage and compare their complexity to recent CNN models with similar top-1 or top-5 error. Model **L** and **M** have 8 and 12 layers in each stage. While model **L** and **M** have the same spatial dimensions and stage structures as in Table 1, model **N** and **O** adopt the same structure as in [6]. They have different pooling strides and one more stages right after the first 7×7 convolutional layer. From the second to the fifth stage, model **N** has 20, 24, 24 and 32 layers, and model **O** has 20, 32, 56, and 56 layers respectively. The detailed model configurations are put in the supplemental materials.

Table 7 compares the performance and complexity of our model from **L** to **O** with previous work. Figure 3 provides a visual comparison in the form of scattered plot. The red marks in the figure represent our models. All of our models demonstrate remarkably lower complexity while being as accurate. Compared to VGG-16, Resnet-34, Resnet-50 and Resnet-101 models, our models consume $\mathbf{42}\times$, $\mathbf{7.32}\times$, $\mathbf{4.38}\times$, $\mathbf{5.85}\times$ less computation respectively with similar or lower top-1 or top-5 error. It is noteworthy that the number of parameters of our models are also $\mathbf{21}\times$, $\mathbf{3.1}\times$, $\mathbf{2.3}\times$, $\mathbf{2.9}\times$ less than VGG-16 and ResNet models. Table 8 lists the top-5 errors of model **N** and **O** on the test set.

Methods	Top-5 error	Actual speedup ratio
VGG-16	9.9%	1×
Low-rank[24]	10.4%	2.9×
Low-rank[22]	9.7%	2.05×
QCNN[23]	10.5%	4×
Tucker Decomposition[12]	10.6%	3.34×
our Model **L**	9.9%	14×

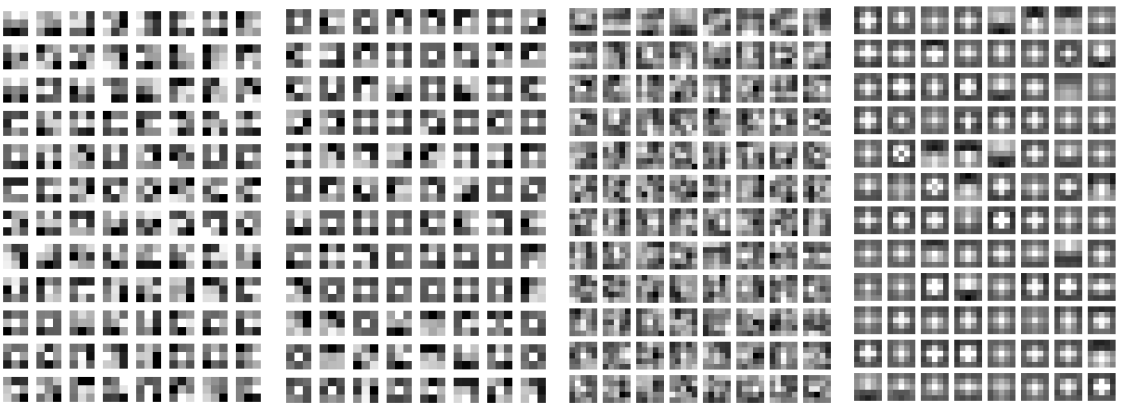
Table 9. Comparison with other acceleration methods based on VGG-16 model

3.4. Comparison with other acceleration methods

Most work on accelarating convolutional neural networks restructure and finetune from off-the-shelf pre-trained models (AlexNet or VGG), while our method redesignes the convolutional layer and trains from scratch. Table 9 compares the speedup ratio of the accelartion methods[24][23][12][22] that apply on VGG-16 with our model **L** that has similar top-5 error. While other methods can obtain no greater than $4\times$ speedup, some of which with higher error rate, our model achieves $14\times$ speed up with same top-5 error to orignial VGG-16 model.

3.5. Implementation details and running time

In SIC layer, most of the computation is consumed by the linear projection, which is equivalent to a matrix multiplication. We wrote a CUDA kernel to perform the 2D spatial convolution in each channel. Our current implementation of the TpS layer simply discards all the non-connected weights in a convolutional layer. Unlike the random connection patterns in sparsity based methods, the regular structure of TpS layer resembles the 2D and 3D convolution operations, and



(a) 3×3 standard convolutional layer (b) 3×3 single intra-channel convolutional layer (c) 5×5 standard convolutional layer (d) 5×5 single intra-channel convolutional layer

Figure 4. Visualization of convolutional kernels. We compare the 3×3 and 5×5 kernels that are learned by the proposed single intra-channel convolutional layer and the standard convolutional layer. The kernels from single intra-channel convolution exhibit a higher level of regularity in structure.

makes writing an efficient kernel possible.

We list the theoretical and actual speedups over VGG16 and ResNet models in the last two columns of Table 7. The actual speedups are calculated by measuring the average running time of the forward pass, with batch size equal to 128. The running time is measured on a workstation with Intel Core i7-3930K CPU and NVIDIA Titan X GPU. cuDNN v5.1 is used for acceleration. The difference between the theoretical and actual speedups is due to the following reasons: (i) With the use of Wingrad algorithm proposed in [14], 3×3 convolutional layer has higher efficiency than 1×1 convolutional layer; (ii) While our models have a few times smaller number of channels than the deep ResNet models, larger number of channels often results in higher efficiency in cuDNN's implementation; (iii) Although ReLU layers, residual layers, and the 2D in-channel convolution in our SIC layers have very low complexity, they are less optimized than the convolutional layers and take considerable time. We expect closer to theoretical efficiency can be attained with an expert-level GPU implementation.

3.6. Visualization of filters

Given the exceptionally good performance of the proposed methods, one might wonder what type of kernels are actually learned and how they compare with the ones in traditional convolutional layers. We randomly chose some 3×3 and 5×5 kernels in the SIC layers and the standard convolutional layers from each stage, and visualize them side by side in Figure 4.

We notice that many Gaussian or derivative of Gaussian patterns exist in 4(b) and 4(d), kernels from our SIC layers, especially for the 5×5 kernels in 4(d), while the kernels in standard convolutional layers exhibit more randomness. It demonstrates that the kernel learned by SIC layer can provide high level of regularized structure. We attribute this stronger regularization to the reduction of number of 2D filters (single 2D filter in SIC layer). In the SIC layer each input channel has one 2D filter, but is connected to all the output channels, as stated in 2.2.2, the filter is driven to learn the important information, showing more regularized shape, in some sense similar to the principal components in PCA .

4. Conclusion

This work introduces a novel design of low complexity convolutional layer in deep CNN that involves two specific improvements: (i) a single intra-channel convolutional (SIC) layer ; (ii) a topological subdivision scheme. As we demonstrated, they are both powerful schemes in different ways to yield a new design of the convolutional layer that has higher efficiency, while achieving equal or better accuracy compared to classical designs. While the numbers of input and output channels remain the same as in the classical models, both the convolutions and the number of connections can be optimized against accuracy in our model - (i) reduces complexity by unraveling convolution, (ii) uses topology to make connections in the convolutional layer sparse, while maintaining local regularity. Although the CNN have been exceptionally successful regarding the recognition accuracy, it is still not clear what architecture is optimal and learns the visual information most effectively. The methods presented herein attempt to answer this question by focusing on improving the efficiency of the convolutional layer. We believe this work will inspire more comprehensive studies in the direction of optimizing convolutional layers in deep CNN.

Acknowledgement

This work was supported in part by the National Science Foundation under grant # IIS-1212948.

References

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

[2] 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.

[3] 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*, 2014.

[4] S. Gross and M. Wilber. Resnet training in torch. `https://github.com/facebook/fb.resnet.torch`, 2016.

[5] 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.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[7] A. Hyvärinen, P. Hoyer, and M. Inki. Topographic independent component analysis. *Neural computation*, 13(7):1527–1558, 2001.

[8] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

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

[10] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proc. BMVC*, 2014.

[11] J. Jin, A. Dundar, and E. Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

[12] 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.

[13] 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.

[14] A. Lavin and S. Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

[15] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[16] 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.

[17] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[19] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[22] C. Tai, T. Xiao, X. Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

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

[24] 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.

[25] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1984–1992, 2015.