ESPACE: Accelerating Convolutional Neural Networks via Eliminating Spatial and Channel Redundancy

Shaohui Lin,[†‡] **Rongrong Ji,**[†‡*] **Chao Chen,**[†‡] **Feiyue Huang,**[♮]
[†]Fujian Key Laboratory of Sensing and Computing for Smart City, Xiamen University, 361005, China
[‡]School of Information Science and Engineering, Xiamen University, 361005, China
[♮]BestImage Lab, Tencent Technology (Shanghai) Co.,Ltd, China
shaohuilin007@gmail.com, rrji@xmu.edu.cn, silentcc@icloud.com, garyhuang@tencent.com

Abstract

Recent years have witnessed an extensive popularity of convolutional neural networks (CNNs) in various computer vision and artificial intelligence applications. However, the performance gains have come at a cost of substantially intensive computation complexity, which prohibits its usage in resource-limited applications like mobile or embedded devices. While increasing attention has been paid to the acceleration of internal network structure, the redundancy of visual input is rarely considered. In this paper, we make the first attempt of reducing spatial and channel redundancy *directly* from the visual input for CNNs acceleration. The proposed method, termed ESPACE (Elimination of SPAtial and Channel rEdundancy), works by the following three steps: First, the 3D channel redundancy of convolutional layers is reduced by a set of low-rank approximation of convolutional filters. Second, a novel mask based selective processing scheme is proposed, which further speedups the convolution operations via skipping unsalient spatial locations of the visual input. Third, the accelerated network is fine-tuned using the training data via back-propagation. The proposed method is evaluated on ImageNet 2012 with implementations on two widely-adopted CNNs, *i.e.* AlexNet and GoogLeNet. In comparison to several recent methods of CNN acceleration, the proposed scheme has demonstrated new state-of-the-art acceleration performance by a factor of $5.48\times$ and $4.12\times$ speedup on AlexNet and GoogLeNet, respectively, with a minimal decrease in classification accuracy.

Introduction

In recent years, convolutional neural networks (CNNs) have demonstrated impressive performance in various computer vision and artificial intelligence applications, such as object recognition (Krizhevsky, Sutskever, and Hinton 2012)(Simonyan and Zisserman 2014)(Lecun et al. 1998)(Szegedy et al. 2015)(He et al. 2015), object detection (Girshick et al. 2014)(Girshick 2015)(Ren et al. 2015), and image retrieval (Gong et al. 2014b). The cutting-edge CNNs are computationally intensive, in which the speed limitation mainly resorts to the convolution operations in the convolutional layers[1]. For example, an 8-layer AlexNet (Krizhevsky, Sutskever, and Hinton 2012) with about 600,000 nodes costs 240MB storage (including 61M parameters) and requires 729M FLOP[2] to classify one image with size 224×224. Such cost is further intensified in deeper CNNs, *e.g.* a 16-layer-VGGNet (Simonyan and Zisserman 2014) with 1.5M nodes costs 528MB storage (including 144M parameters) and requires about 15B FLOP to classify one image.

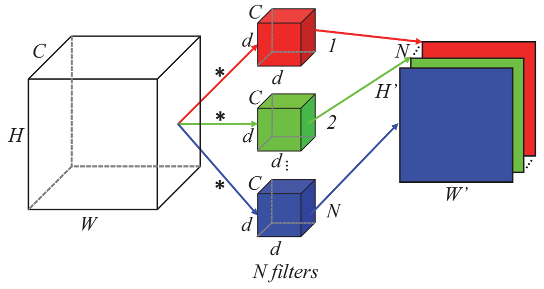
Under such circumstance, the existing CNNs cannot be directly deployed to scenarios that require fast processing and compact storage, such as streaming or real-time applications. On one hand, CNNs with million-scale parameters typically tend to be over parameterized and heavily computed (Denil et al. 2013). Therefore, *not all* parameters and operations (*e.g.* convolution or non-linear activation) are essentially necessary in producing a discriminative decision. On the other hand, it is quantitatively shown in (Ba and Caruana 2014) that, neither shallow nor simplified CNNs provide comparable performance to deep CNNs with billion-scale online operations. Therefore, to accelerate online CNNs predictions without significantly decreasing the decision accuracy, a natural thought is to discover and discard redundant parameters and operations in deep CNNs.

Accelerating CNNs has attracted a few research attention very recently, most of which focus on accelerating the convolutional layer, which is the most time-consuming part of CNNs. In the literature, the related works can be further categorized into four groups, *i.e.* designing compact convolutional filters, parameters quantization, parameters pruning and tensor decomposition.

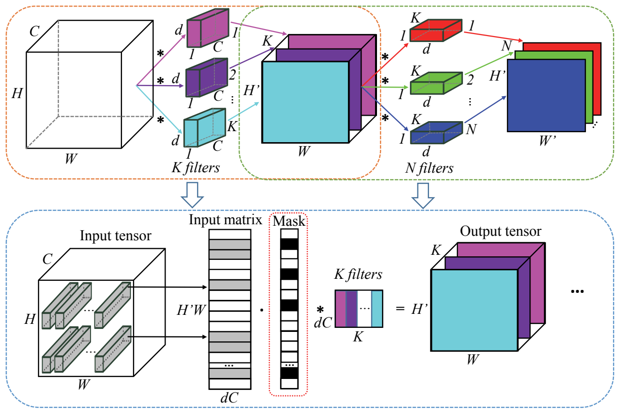
Designing compact convolutional filters. Using a compact filter for convolution can directly reduce the computation cost. The key idea is to replace the loose and over-parametric filters with compact blocks to improve the speed, which significantly accelerate CNNs like GoogLeNet (Szegedy et al. 2015), ResNet (He et al. 2015) on several benchmarks. Decomposing 3×3 convolution with two 1×1 convolutions was used in (Szegedy, Loffe, and Vanhoucke 2016), which achieved state-of-the-art acceleration performance on object recognition. SqueezeNet (Iandola, Moskewicz, and Ashraf 2016) was proposed to replace 3×3 convolution with 1×1 convolution, which created a com-

[1]In this paper, we focus on the acceleration of the convolutional layers, as it takes up over 80% running time in most existing CNNs, *i.e.* AlexNet, GoogLeNet and VGGNet.

[2]FLOP: The number of Floating-point operation to classify one image with CNNs.



(a) The original convolution



(b) ESPACE based acceleration

Figure 1: The framework of the proposed ESPACE convolutional computation for CNNs acceleration.

pact neural network with about 50× fewer parameters and comparable accuracy comparing with AlexNet. However, the above compact filters are specifically designed for respective models, which are less general for compressing and accelerating other deep models.

Parameter quantization. Both Gong *et al.* (Gong et al. 2014a) and Wu *et al.* (Wu et al. 2016) employed vector quantization over parameters to reduce the redundancy in the parameter space. Chen *et al.* (Chen et al. 2015) proposed a HashedNets model which used a low-cost hash function to group weights into hash buckets for parameter sharing. Gupta *et al.* (Gupta et al. 2015) used 16-bit fixed-point presentation to replace the full-precision point when using stochastic rounding based CNN training, which significantly reduced memory and point operations while only degenerating little accuracy in classification. Recently, direct training with binarized weights in CNNs was proposed, for instance, BinaryConnect (Courbariaux, Bengio, and David 2015), BinaryNet (Courbariaux and Bengio 2016) and XNOR-Networks (Rastegari et al. 2016). The main idea is to directly learn binary-value weights or activations during the model training. However, the accuracy of such binary nets are significantly degenerated when dealing with large CNNs such as GoogLeNet.

Parameters pruning. The third group is to prune redundant, non-informative weights in a pre-trained CNN model. For example, Optimal Brain Damage (LeCun et al. 1989) used Hessian based loss function to prune a network by reducing the number of connections. Srinivas and Babu (Srinivas and Babu 2015) explored the redundancy among neurons, and proposed a data-free pruning to remove redundant neurons. Han *et al.* (Han et al. 2015) proposed to reduce the total amount of parameters and operations in the entire network. The deep compression (Han, Mao, and Dally 2015) removed the redundant connections and quantized weights (so that multiple connections share the same weight), and then used Huffman coding to encode the quantized weights. It is worthy to note that, the above pruning schemes typically produce sparse CNNs with non-structured random connections, which typically causes irregular memory access, *i.e.* a complex data storage structure that adversely impacts practical implementations of CNNs in hardware platforms.

Tensor decomposition. Several recent works have been proposed to decompose generalized convolution into a sequence of tensor-based convolutions with fewer parameters (Denton et al. 2014)(Lebedev et al. 2014)(Kim et al. 2016). Such approaches typically adopt inexact low-rank factorizations, which can considerably reduce computations by selecting a lower rank to approximate the original convolution. However, these methods are potentially problematic due to the vanishing/exploding gradients in back-propagation, which is difficult to fine-tune the deeper CNN that was formed by decomposing a single convolutional layer into four layers, especially for GoogLeNet and ResNet.

However, all above methods still focus on reducing the redundancy of internal structure *inside* CNNs, which fully accepts all external visual input. In such a case, the redundancy *outside* CNNs is typically ignored. However, such visual input is by nature redundant, in which less discriminative and unsalient information widely exist either spatially or spectrally. The former refers to spatial redundancy, while the latter refers to channel redundancy. Therefore, it would be highly beneficial to remove such kinds of redundancy *directly* from the visual input.

In this paper, we make the first attempt towards eliminating redundancy of visual input for CNN acceleration. To this end, we propose a novel CNN acceleration approach to jointly eliminate spatial and channel redundancy in the convolutional layers, termed ESPACE (elimination of spatial and channel redundancy), which is shown in Fig. 1(b). In this model, the channel redundancy is firstly eliminated by a low rank basis of filters to approximate the convolutional filters. Subsequently, the spatial redundancy is eliminated by masking and removing unsalient spatial regions. Several convolutional masks are proposed to select salient responses in the convolutional layers to be sent layer-by-layer in a bottom-up manner. Finally, the CNN model with ESPACE layers is fine-tuned via stochastic gradient descent with back-propagation.

The proposed ESPACE acceleration scheme is evaluated on the large-scale ImageNet dataset (Deng et al. 2009) and implemented on the widely-used AlexNet (Krizhevsky, Sutskever, and Hinton 2012) and GoogLeNet (Szegedy et al. 2015). Comparing to the most recent CNN acceleration models, the proposed ESPACE model has the state-of-the-art rate-distortion[3] by a factor of $5.48\times$ and $4.12\times$ on AlexNet and GoogLeNet, respectively, with a minimal decrease in the classification accuracy.

Eliminating Spatial and Channel Redundancy

Preliminaries

Convolution is the most time-consuming operation in CNNs. From the perspective of tensor product, it transforms an input tensor $\mathcal{I}$ of size $H \times W \times C$ into an output tensor $\mathcal{O}$ of size $H' \times W' \times N$ using the following linear mapping:

$$\mathcal{O}_{h',w',n} = \sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{c=1}^{C} \mathcal{K}_{i,j,c,n}\mathcal{I}_{h_i,w_j,c}, \qquad (1)$$

where the set of convolutional kernels $\mathcal{K}$ is given by a tensor of size $d \times d \times C \times N$. Here, $d \times d$ corresponds to the spatial dimensions, while C and N are the number of input and output channels, respectively. The height and width of the input are denoted as $h_i = h' + i - 1$ and $w_j = w' + j - 1$. For simplicity, we assume a unit stride with no zero-padding and skip biases. This computation process is illustrated in Fig. 1(a). The cost for the convolutional layer with N filters of size $d \times d$ acting on C input channels is $\mathbf{O}(NCd^2H'W')$. In order to describe the channel redundancy discussed subsequently, Eq. 1 is rewritten as:

$$\mathcal{O}_n = \mathcal{K}_n * \mathcal{I} = \sum_{c=1}^{C} \mathbf{K}_n^c * \mathbf{I}^c, \;\; n = 1, 2, \cdots, N, \qquad (2)$$

where $\mathbf{K}_n^c \in \mathbb{R}^{d\times d}, c \in [1, 2, \cdots, C], n \in [1, 2, \cdots, N]$. Additionally, we define the set of all spatial positions of the output as $\Omega = \{1, \cdots, H'\} \times \{1, \cdots, W'\}$ and the set of indices of ESPACE mask as $I \subset \Omega$, in which the outputs are calculated exactly. We further denote $m = |I|$ as the number of positions that needs to be computed exactly, and $r = \frac{m}{|\Omega|}$ as the spatial convolutional rate.

In practice, many deep learning frameworks (*e.g.* caffe (Jia et al. 2014) and MatConvNet (Vedaldi and Lenc 2015)) compute tensor-based convolutional operator by relying on highly optimized matrix-by-matrix multiplication of the basic linear algebra packages, such as ATLAS, Interl MKL and OpenBLAS. For example, we can transform an input tensor of size $H \times W \times C$ into an input matrix $\mathbf{M}$ of size $H'W' \times (d \times d \times C)$ using *im2row* operator. The rows of $\mathbf{M}$ are elements of patches of input tensor with size $d \times d \times C$. At the same time, we can transform convolutional filters into a filter matrix of size $(d \times d \times C) \times N$ using *reshaped* operator. Then, we can obtain the output tensor by reshaping the result matrix by multiplying data and filter matrices. In the latter case, we only consider specific spatial points decided by ESPACE mask, which would be introduced in depth in the following section.

[3]Rate-distortion evaluates the acceleration performance in which the rate and distortion represent the model speedup and classification precision, respectively.

ESPACE Layers

The ESPACE layer is an accelerated convolutional layer by reducing spatial and channel redundancy, which contains three steps to accelerate convolution. Fig. 1(b) illustrates the overall framework. First, channel redundancy is reduced by a low rank decomposition of 3D convolutional filters. Second, masks are introduced to skip convolution operation in certain spatial locations. Third, the network is fine-tuned to further eliminate the approximation errors accumulated layer-by-layer.

Removing channel redundancy. We reduce the channel redundancy by factorizing each convolutional layer as a sequence of two regular convolutional layers with rectangular filters. The first convolutional layer has K filters of spatial size $d \times 1$, resulting in a filter bank $\{\mathcal{V}_k \in \mathbb{R}^{d\times 1\times C} : k \in [1, 2, \cdots, K]\}$ to produce output feature maps $\mathcal{S} \in \mathbb{R}^{H'\times W\times K}$. The second convolutional layer has N filters of spatial size $1 \times d$, resulting in a filter bank $\{\mathcal{T}_n \in \mathbb{R}^{1\times d\times K} : n \in [1, 2, \cdots, N]\}$ to produce output feature maps $\mathcal{O} \in \mathbb{R}^{H'\times W'\times N}$ with the same size of the convolutional output. Therefore, the convolution by the original filters $\mathcal{O}_n = \sum\limits_{c=1}^{C} \mathbf{K}_n^c * \mathbf{I}^c$ in Eq. 2 is approximated by:

$$\begin{aligned} \mathcal{O}_n \approx \mathcal{T}_n * \mathcal{S} &= \sum_{k=1}^{K} \mathbf{T}_n^k * (\sum_{c=1}^{C} \mathbf{V}_k^c * \mathbf{I}^c) \\ &= \sum_{c=1}^{C} \Big(\sum_{k=1}^{K} \mathbf{T}_n^k * \mathbf{V}_k^c\Big) * \mathbf{I}^c, \end{aligned} \qquad (3)$$

where one of the original filters can be approximated by the sum of separable filters $\mathbf{T}_n^k * \mathbf{V}_k^c$. By far, we reduce the computation complexity from the original $\mathbf{O}(NCd^2H'W')$ to $\mathbf{O}\big(K(CW + NW')dH'\big)$. Assuming the image width $W \gg d$ and the same value of K, N, C, ESPACE speedups about d times compared with the original convolution.

We can obtain the approximated low-rank filter basis $\mathcal{T}_n, \mathcal{V}_k$ by reconstructing the outputs of the original convolutional layer. This is done by optimizing two separable basises given the training data, which amounts to:

$$\min_{\{\mathbf{T}_n^k, \mathbf{V}_k^c\}} \sum_{i=1}^{|X|}\sum_{n=1}^{N} \Big\| \mathcal{K}_n * f_{l-1}(\mathcal{X}_i) - \sum_{c=1}^{C}\sum_{k=1}^{K} \mathbf{T}_n^k * \mathbf{V}_k^c * f_{l-1}(\mathbf{X}_i^c) \Big\|_2^2, \qquad (4)$$

where $f_l(\mathcal{X}_i)$ is the input of the l-th layer (or the output of the $(l-1)$-th layer), $\mathcal{X}_i \in \mathcal{X}$ is one example in the training dataset. This problem can be solved by an alternative optimization layer-by-layer in a bottom-to-up manner (Jaderberg, Vedaldi, and Zisserman 2014). Moreover, we use rank selection (Zhang et al. 2015) to select a desired rank K of convolutional filters in each layer, which determines the speedup ratio.

Removing spatial redundancy. In order to reduce the spatial redundancy of convolutions, only outputs at a small fraction of spatial positions are computed, while the remainings are interpolated using nearest neighbors from the aforementioned set of positions.

Specifically, we define a mask $I \subset \Omega$, which determines spatial output that needs to be calculated exactly. The function $q : \Omega \to I, (h, w) \mapsto q(h, w)$ returns the index of the nearest neighbor in I according to the Euclidean distance as below:

$$q(h,w) = \underset{(h',w') \in I}{\arg\min} \sqrt{(h-h')^2 + (w-w')^2}. \tag{5}$$

Note that the function $q(h, w)$ may be calculated in advance. We can approximate the original $\mathcal{O}$ using $\hat{\mathcal{O}}$ as follows:

$$\hat{\mathcal{O}} = \mathcal{O}(q(h,w), n). \tag{6}$$

Therefore, we only need to calculate the values $\mathcal{O}(h, w, n)$ for spatial index $(h, w) \in I$ and it is further approximated efficiently by matrix multiplication which is described in Preliminaries. In this case, the input matrix $\mathbf{M}$ contains $m = |I|$ rows instead of the original $H'W' = |\Omega|$ rows (*i.e.* spatial convolutional rate $r = \frac{m}{|\Omega|}$). Therefore, the value of m is much smaller, the speedup ratio is higher, and the memory size required to store is lower.

Collaborative Removal of Spatial & Channel Redundancy. We further jointly remove both spatial and channel redundancy. Specifically, by using the low-rank separable filters, the original convolution is splitted into two separable (and computational light) convolutions. Subsequently, the spatial redundancy in each two separable convolution is further reduced by only computing output at a small fraction of spatial positions. In order to restore the network accuracy, we perform fine-tuning on the whole approximated network.

To that effect, we obtain the derivatives of the ESPACE layer by the chain rule, which calculates the derivatives $\frac{\partial \hat{\mathcal{O}}(q(h,w),n)}{\partial \mathcal{O}(h,w,n)}$ as follows:

$$\frac{\partial \hat{\mathcal{O}}(h',w',n')}{\partial \mathcal{O}(h,w,n)} = \begin{cases} 1, q(h,w) = (h',w') \ and \ n = n', \\ 0, otherwise. \end{cases} \tag{7}$$

It means the derivatives are summed over spatial regions that share the same interpolated values during back-propagation.

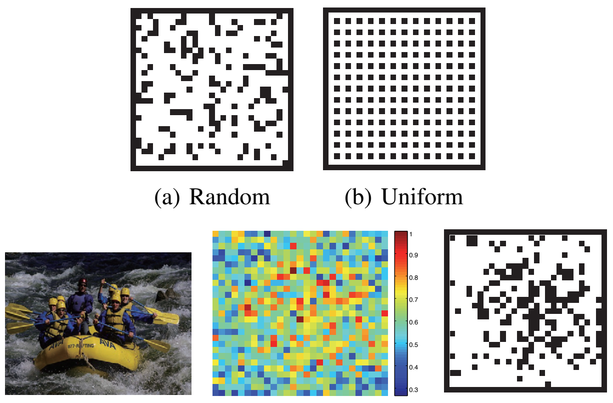
By far, we reduce the computation complexity from the original $\mathbf{O}\big(NCd^2H'W'\big)$ to $\mathbf{O}\big(Kd(Cm_1 + Nm_2)\big)$, where $m_1 = H'W, m_2 = H'W'$. In particular, assume $H \gg d, W \gg d, m_1 = m_2 = 0.5 \cdot H'W'$ and the same value of K, N, C, we can further speedup the ESPACE layer about $2d$ times comparing to the original convolution.

ESPACE Masks

Random ESPACE-Conv Mask: The $m = |I|$ spatial points are chosen randomly with probability $p = \frac{m}{|\Omega|}$ which is shown in Fig. 2(a). However, it is hard to estimate the spatial position of these points, and it is undesirable if the points tend to cluster for $m \ll |\Omega|$, since it leads to high interpolated error with the randomness.

Uniform ESPACE Mask: In such a way, a set of uniform scatter points is chosen, which is defined as follows:

$$I = \big\{H'_h \cdot 1, \cdots, H'_h \cdot \lfloor\sqrt{m}\rfloor\big\} \times \big\{W'_w \cdot 1, \cdots, W'_w \cdot \lfloor\sqrt{m}\rfloor\big\}, \tag{8}$$



(a) Random (b) Uniform

(c) ImageNet image (d) $E(h'_0, w'_0)$ (e) Impact

Figure 2: ESPACE masks in the second convolutional layer in AlexNet with r = 0.25. the black pixels in positions require to be calculated exactly. Best view in color.

where $H'_h = \lfloor\frac{H'}{\sqrt{m}}\rfloor, W'_w = \lfloor\frac{W'}{\sqrt{m}}\rfloor$. An example of the uniform mask is shown in Fig. 2(b). This mask is more scatter to produce richer feature representation than random ones. However, it is not robust when facing changed objects.

Impact ESPACE Mask: We propose an impact ESPACE mask to estimate the impact of each output position on the loss function of CNNs, from which we choose the most important positions to be computed. Recall that by removing the channel redundancy, two respective convolutional filters are obtained. Correspondingly, the first impact mask is deployed over the first convolutional computation, while the second mask is deployed over the second computation.

Let $L(\mathcal{O})$ be the loss function of CNN (*e.g.* negative posterior log-likelihood), which is a function of the convolutional layer outputs $\mathcal{O}$. Suppose $\hat{\mathcal{O}}$ is obtained from $\mathcal{O}$ by replacing one element (h'_0, w'_0, n_0) with zero element. Therefore, we can estimate the impact of the position (h'_0, w'_0, n_0) by using the first-order Taylor expansion:

$$|L(\mathcal{O}) - L(\hat{\mathcal{O}})| \approx \left| \frac{\partial L(\mathcal{O})}{\partial \mathcal{O}(h'_0, w'_0, n_0)} \mathcal{O}(h'_0, w'_0, n_0) \right|, \tag{9}$$

where the value $\frac{\partial L(\mathcal{O})}{\partial \mathcal{O}(h'_0, w'_0, n_0)}$ is obtained via back-propagation. In order to calculate the total impact at a spatial position $(h'_0, w'_0) \in \Omega$, we sum over all the channels and average this estimation over the training dataset as below:

$$G(h'_0, w'_0; \mathcal{O}) = \sum_{n=1}^{N} \left| \frac{\partial L(\mathcal{O})}{\partial \mathcal{O}(h'_0, w'_0, n)} \mathcal{O}(h'_0, w'_0, n) \right|. \tag{10}$$

$$E(h'_0, w'_0) = \mathbb{E}_{\mathcal{O} \sim P(\mathcal{X})} G(h'_0, w'_0; \mathcal{O}). \tag{11}$$

We take the top-m positions with the highest values of $E(h'_0, w'_0)$ to form the impact ESPACE mask. An example of impact ESPACE mask is shown in Fig. 2(e). The value $E(h'_0, w'_0)$ tends to be higher in the center, which is due to the fact that objects typically appear in centroid positions in the ImageNet dataset.

Table 1: The evaluations of computational time on CPU (ms), GPU (ms), and classification error rates (Top-1/5 Err.) of AlexNet and GoogLeNet with batch size 50. Values are averaged over 5 runs in all Figures and Tables of this paper.

Model	CPU (ms)	GPU (ms)	Top-1 Err.	Top-5 Err.
AlexNet	1,445	74	42.27%	19.11%
GoogLeNet	3,697	263	31.07%	10.86%

Experiments

Experimental Settings

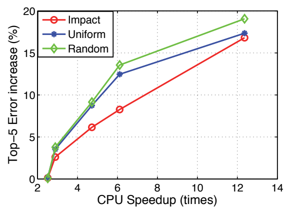
Datasets. We evaluate on the ImageNet 2012 dataset (Deng et al. 2009), which contains more than 1 million training images from 1,000 object classes, and a validation set of 50,000 images. Each image is associated with one ground truth category. We train the proposed ESPACE model on the training set of ImageNet 2012, and test it on the validation set using single-view testing (central patch only).

The original convolutional neural networks. The GoogLeNet is a deeper and wider Inception network with 21 convolutional layers with filter sizes alternating among $1 \times 1, 3 \times 3$ and 5×5. The AlexNet contains 5 convolutional layers with larger filter sizes among 3×3, 5×5 and 11×11. More details of their architectures are given in (Szegedy et al. 2015)(Krizhevsky, Sutskever, and Hinton 2012). The networks are obtained from Caffe Model Zoo[4].

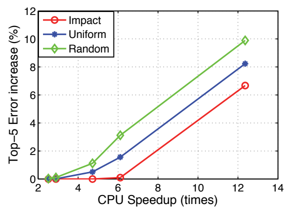
Baselines. We compare the proposed CNNs using the ESPACE layer to two groups and 4 state-of-the-art approaches, including: 1. tensor decomposition based acceleration (LRD (Jaderberg, Vedaldi, and Zisserman 2014): low rank decomposition, CPD (Lebedev et al. 2014): CP-Decomposition), and 2. parameter quantization (BC (Courbariaux, Bengio, and David 2015): BinaryConnect, Q-CNN (Wu et al. 2016): Quantized CNN). For all baselines, we use their reported results under the same setting for fair comparison. The proposed ESPACE model is trained (or fine-tuned) using Caffe and run on a 24-core Intel E5-2620 CPU, NVIDIA GTX TITAN X graphics card with 12GB and 32G RAM. The learning rate starts at 0.0001 and is halved every 10,000 iterations with batch size 100 for AlexNet and 32 for GoogLeNet. The weight decay is set to be 0.0005 and the momentum is set to be 0.95.

Evaluation Protocol. The classification error on the validation set is employed as the evaluation protocol. We use the top-5 classification error to evaluate the accuracy degeneration of different acceleration methods. Then we measure the acceleration performance in term of the rate-distortion, which reflects the balance between acceleration ratio and the increase of classification error. In our ESPACE method, rate-distortion is controlled by K (the number of output channels) and $r_1 = \frac{m_1}{H'W}, r_2 = \frac{m_2}{H'W'}$ (the spatial convolutional rate). We fix K using rank selection (Zhang et al. 2015) and set the same values of r_1, r_2 varying among $\frac{2}{3}, \frac{1}{2}, \frac{1}{4}, \frac{1}{5}$ and $\frac{1}{8}$ in order to better analyze the acceleration performance. We

[4]http://dl.caffe.berkeleyvision.org/

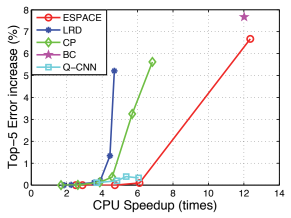


(a) CPU without fine-tuning

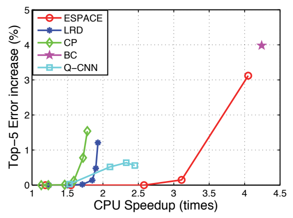


(b) CPU with fine-tuning

Figure 3: Results of different masks in Conv2 of AlexNet.



(a) Conv2



(b) Inception 3b (3×3)

Figure 4: Results of single layer in Conv2 of AlexNet and inception 3b (3×3).

compare the computation time of AlexNet and GoogLeNet in Tab. 1, together with their classification error on the ImageNet 2012.

Single Layer Results

We explore the best rate-distortion of the proposed ESPACE on the bottleneck convolutional layers of AlexNet (*i.e.* Conv2 which consumes 23.6% of the evaluation time.) and the most operational Inception layer 3b with 3×3 of GoogLeNet (*i.e.* Inception 3b with 304M operations). In order to select a best ESPACE mask, We first analyze the results of different ESPACE masks with several (K, r_1, r_2) parameter settings in the single convolutional layer (*i.e.* Conv2 in AlexNet), which is presented in Fig. 3. We can see that the impact ESPACE mask works best both with and without fine-tuning, while classification error is dropping substantially with high speedup rate.

We compare our results to 4 state-of-the-art approaches on acceleration in Conv2 of AlexNet and Inception 3b (3×3) of GoogLeNet. As for parameter setting in Conv2 of AlexNet, the parameter (*i.e.* rank) in LRD is set among 16, 24, 32, 48, 64 and 128, which is the same with CPD parameters (*i.e.* rank). The hyper-parameters (*i.e.* quantized parameters) of Q-CNN is set among 4 pairs (4,64), (6,128), (6,128) and (8,128). For BC, which has no parameter to tune, the speedup rate is fixed as 12.02. Based on the setting of spatial convolutional rate in the Evaluation Protocol, we only set the parameters of ESPACE (*i.e.* K) as 71 in Conv2 of AlexNet with ESPACE impact mask, and 59 in Inception 3b (3×3) of GoogLeNet with ESPACE uniform mask, respectively. As for parameters setting in Inception 3b (3×3) of GoogLeNet, we only change the parameters of LRD and

Table 2: Speeding-up all conv. layers of AlexNet.

Model	Method	Para.	Speed-up (CPU)	Top-5 Err. ↑
AlexNet	LRD	∼	1.33×	0.00%
			2.39×	0.00%
			3.13×	0.57%
			4.15×	1.37%
			4.67×	2.41%
	CPD	∼	1.67×	0.00%
			2.87×	0.31%
			3.87×	1.12%
			4.83×	3.73%
			5.44×	7.12%
	BC	∼	11.23×	10.23%
	Q-CNN	4/64	3.32×	0.94%
		6/64	4.32×	1.90%
		6/128	3.71×	0.36%
		8/128	4.27×	0.60%
	ESPACE	∼/0.667/0.667	1.89×	0.00%
		∼/0.5/0.5	2.27×	0.00%
		∼/0.25/0.25	4.15×	0.01%
		∼/0.2/0.2	**5.48×**	**0.97%**
		∼/0.125/0.125	8.12×	8.89%

Table 3: Speeding-up all conv. layers of GoogLeNet.

Model	Method	Para.	Speed-up (CPU)	Top-5 Err. ↑
GoogLeNet	LRD	∼	1.21×	0.00%
			2.31×	0.00%
			2.89×	0.51%
			3.57×	1.03%
			4.02×	5.16%
	CPD	∼	1.34×	0.00%
			2.42×	0.14%
			3.25×	1.37%
			3.97×	3.43%
			4.68×	5.57%
	BC	∼	8.35×	13.59%
	Q-CNN	∼	3.43×	0.63%
			3.96×	1.38%
			4.23×	1.97%
			4.56×	1.76%
	ESPACE	∼/0.667/0.667	2.05×	0.00%
		∼/0.5/0.5	2.78×	0.00%
		∼/0.25/0.25	3.76×	0.04%
		∼/0.2/0.2	**4.12×**	**0.68%**
		∼/0.125/0.125	4.63×	2.89%

CPD with the same value among 1, 4, 8, 16, 32 and 64, while keeping the remaining parameters of other methods with the same setting above.

Fig. 4 shows a consistent trend in rate-distortion except Q-CNN, which is due to the influence of codebook. To analyze the quantitative results on Conv2 of AlexNet, for tensor decomposition, CPD achieves a better classification error to LRD when the speedup rate is high. Instead, by adding spatial redundancy of external visual input, ESPACE greatly improves the rate-distortion comparing to CPD. ESPACE also performs much better than BC and Q-CNN ($4.72\times$ with no loss and $6.11\times$ with minor accuracy drop by 0.1%). To explain, BC can speedup the convolutional layer by a high factor (*e.g.* 12.02 for AlexNet), but the binarization of parameters leads to higher quantization error, which is hard to be adaptive when we want to control the speedup rate. Q-CNN achieves comparatively better performance ($6.06\times$ with 0.33% dropping in classification accuracy) comparing to those of LRD and CPD. However, Q-CNN is also hard to achieve high speedup rate, which might be due to the limited codebook size. In contrast, ESPACE achieves the best rate-distortion by comparing to other baselines. Second, for Inception 3b (3×3) with higher speedup rate, it is limited to increase the top-5 error instead. Especially, for tensor decomposition, it is noted that the speedup rate is very limited for LRD and CPD ($1.93\times$ and $1.78\times$, respectively), even if the rank is set to 1. It is interesting to find that LRD can achieve a better rate-distortion than CPD. As an explanation, the spatial size of filter and internal structure of inception leads to higher difficulty in fine-tuning multiple layers based on CP decomposition. By fixing the rank K by rank selection (*i.e.* 59), we remove the spatial redundancy of internal visual information, which leads to the best rate-distortion ($3.11\times$ with 0.15% dropping in classification accuracy) comparing to the baselines.

Whole Network Results

We further speedup all convolutional layers in AlexNet and GoogLeNet with ESPACE. The rank K of ESPACE in each layer is computed by rank selection, and the spatial convolutional rate r_1, r_2 in each layer is fixed among $\frac{2}{3}, \frac{1}{2}, \frac{1}{4}, \frac{1}{5}$ and $\frac{1}{8}$. We evaluate the quantitative results of the whole network by taking several layers with high computation complexity (*i.e.* the middle three convolutional layers of AlexNet and “Conv1”, “Conv2”, “Inception 3b” and “Inception 4e” of GoogLeNet) into account, since the acceleration rate is less significant for the remaining layers in these CNNs. The quantitative results of all methods are shown in Tab. 2 and Tab. 3. For AlexNet, ESPACE accelerates the computation of all the convolutional layers by a factor of $5.48\times$ with a dropping of 0.97% top-5 error, which is the best rate-distortion comparing to other methods. For GoogLeNet, ESPACE also achieves much better results by a factor of $4.12\times$ with a dropping of 0.68% top-5 error than other methods. We can see that it is hard to fine-tune the network using CPD due to the vanishing gradients in back-propagation.

Conclusion

In this paper, we present a novel ESPACE model for CNN acceleration. Differing from previous approaches that reside in reducing the internal redundancy of CNNs, ESPACE accelerates convolutional computations greatly by *directly* eliminating both spatial and channel redundancy from the visual input. First, the 3D channel redundancy of convolutional layers is reduced by a set of low rank approximation of filters. Second, the spatial redundancy is reduced by skipping unsalient spatial locations of the visual input, which is achieved via a novel mask based selective processing scheme. Finally, the entire ESPACAE network is further fine-tuned to improve classification accuracy. We have demonstrated that ESPACE can lead to state-of-the-art rate-distortion by a factor of 5.48× and 4.12×, with less than 1% accuracy drop comparing to several recent acceleration methods on AlexNet and GoogLeNet, respectively.

Acknowledgments

This work is supported by the National Key RD Program (No. 2016YFB1001503), the Special Fund for Earthquake Research in the Public Interest No.201508025, the Nature Science Foundation of China (No. 61402388, No. 61422210, No. 61373076, and No. 61572410), the CCF-Tencent Open Research Fund, the Open Projects Program of National Laboratory of Pattern Recognition, and the Xiamen Science and Technology Project (No. 3502Z20153003).

References

Ba, L., and Caruana, R. 2014. Do deep nets really need to be deep? In *NIPS*.

Chen, W.; Wilson, J.; Tyree, S.; Weinberger, K.; and Chen, Y. 2015. Compressing neural networks with the hashing trick. In *ICML*.

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

Courbariaux, M.; Bengio, Y.; and David, J. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*.

Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; and Li, F. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*.

Denil, M.; Shakibi, B.; Dinh, L.; Ranzato, M.; and Freitas, N. 2013. Predicting parameters in deep learning. In *NIPS*.

Denton, E.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

Girshick, R. 2015. Fast r-cnn. In *CVPR*.

Gong, Y.; Liu, L.; Yang, M.; and Bourdev, L. 2014a. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Gong, Y.; Wang, L.; Guo, R.; and Lazebnik, S. 2014b. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*.

Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; and Narayanan, P. 2015. Deep learning with limited numerical precision. In *ICML*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *NIPS*.

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

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

Iandola, F.; Moskewicz, M.; and Ashraf, K. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 1mb model size. *arXiv preprint arXiv:1602.07360*.

Jaderberg, M.; Vedaldi, A.; and Zisserman, A. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*.

Kim, Y.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *ICLR*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

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

LeCun, Y.; Denker, J.; Solla, S.; Howard, R.; and Jackel, L. 1989. Optimal brain damage. In *NIPS*.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

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

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*.

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

Vedaldi, A., and Lenc, K. 2015. Matconvnet: Convolutional neural networks for matlab. In *ACM Multimedia*. ACM.

Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; and Cheng, J. 2016. Quantized convolutional neural networks for mobile devices. In *CVPR*.

Zhang, X.; Zou, J.; He, K.; and Sun, J. 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP(99):1–14.