Accelerating Convolutional Neural Networks with Dominant Convolutional Kernel and Knowledge Pre-regression

Zhenyang Wang, Zhidong Deng(✉), and Shiyao Wang

State Key Laboratory of Intelligent Technology and Systems,
Tsinghua National Laboratory for Information Science and Technology,
Department of Computer Science, Tsinghua University, Beijing 100084, China
crazycry2010@gmail.com, michael@tsinghua.edu.cn,
sy-wang14@mails.tsinghua.edu.cn

Abstract. Aiming at accelerating the test time of deep convolutional neural networks (CNNs), we propose a model compression method that contains a novel dominant kernel (DK) and a new training method called knowledge pre-regression (KP). In the combined model DK^{2}PNet, DK is presented to significantly accomplish a low-rank decomposition of convolutional kernels, while KP is employed to transfer knowledge of intermediate hidden layers from a larger teacher network to its compressed student network on the basis of a cross entropy loss function instead of previous Euclidean distance. Compared to the latest results, the experimental results achieved on CIFAR-10, CIFAR-100, MNIST, and SVHN benchmarks show that our DK^{2}PNet method has the best performance in the light of being close to the state of the art accuracy and requiring dramatically fewer number of model parameters.

Keywords: Dominant convolutional kernel · Knowledge pre-regression · Model compression · Knowledge distilling

1 Introduction

In recent years, deep convolutional neural network (CNN) has made impressive success in several computer vision tasks such as image classification [1], object detection and localization [2,3]. On many benchmark challenges [1,4–6], records have been being consecutively broken with CNNs since 2012 [1] on. Surprising performance, however, usually comes with a heavy computational burden due to the use of deeper and/or wider architectures. Complicated models with numerous parameters may lead to an unacceptable test or inference time consuming for a variety of real applications. To resolve such challenging problem, there was an early interest in hardware-specific optimization [1,7–10]. But it is very likely to be unable to meet increasingly demands for model acceleration in an era of mobile Internet. The reason is that huge amounts of portable devices such as

B. Leibe et al. (Eds.): ECCV 2016, Part VIII, LNCS 9912, pp. 533–548, 2016.
DOI: 10.1007/978-3-319-46484-8_32

smart phones and tablets are often equipped with low-end CPUs and GPUs as well as limited memory.

To speed up cumbersome CNNs, one is to directly compress existing large models or ensembles into small and fast-to-execute models [11–15]. Another is to employ deep and wide top-performing teacher networks to train shallow and/or thin student networks [16–19]. Based on low rank expansions, convolutional operator is decomposed into two procedures: feature extraction and feature combination (Sect. 3.1). Initially inspired from low rank approximation of responses proposed by Zhang *et al.* [15], this paper proposes a novel dominant convolutional kernel (DK) for greatly compressing filter banks. To deal with performance degradation problems caused by model compression, we present a new knowledge pre-regression (KP) training method for compressed CNN architectures, which expands the FitNet training method [19] to make it much easier to converge and implement. Such KP-based training method fills the intermediate representation gap between original teacher network and compressed student.

In this paper, our CNN models with DK, together with KP-based training method, are referred to as DK^{2}PNet. It can be viewed as combination of two model acceleration methods mentioned above. For compact DK^{2}PNet, we conduct extensive experiments on different benchmark datasets, including CIFAR-10 [4], CIFAR-100 [4], MNIST [5], and SVHN [6]. When compared to several recently published CNN models, our experimental results show that the proposed DK^{2}PNet method has the best performance that is close to state of the art test errors, while using remarkable less number of parameters.

2 Related Work

In general, there are three types of model acceleration methods for CNN architecture, *i.e.* model compression approach, knowledge distillation based training method, and hardware-specific acceleration.

Model compression approaches are exploited to directly compress parameters of large CNN models layer by layer. They generally include two components: decomposition that are adopted to reduce computational cost, and optimization for such decomposition. Among optimization schemes, the most widely used method includes filter reconstruction, which rebuilds the filter weights, and data reconstruction , through which each layer's responses are remodeled [15]. Actually, decomposition method attracts more attention, among which low rank decomposition methods become increasingly hotspot of research. LeCun *et al.* [20] use optimal brain damage interactively to reduce the number of parameters in neural network by a factor of four. Denil *et al.* [11] demonstrate that there are usually heavy over-parameterization problems for deep networks. Thus a small subset of parameters should be employed to accurately predict remaining ones. Jaderberg *et al.* [12] further adopt low rank decomposition to speed up processing of convolutional layers. They present two rank-1 filter decomposition schemes to attain approximations for scene text character recognition. Similar work is concurrently investigated by Denton *et al.* [13], who extend low rank

tensor approximation methods to a remarkable larger model. But they only illustrate effectiveness of their methods on the first two convolutional layers. Lebedev *et al.* [14] decompose low rank convolutional product of 4D convolutional kernel tensor into a sum of four small number of rank-1 tensors. Instead of approximating linear filters or linear responses, Zhang *et al.* [15] minimize the reconstruction error of nonlinear responses and exploit channel level decomposition to reduce complexity. Chen *et al.* [21] further propose HashedNets to exploit inherent redundancy in neural networks.

The second approaches for accelerating test time computation are knowledge distilling based training methods, which aim to teach small student network with large teacher network. Knowledge distilling is proposed by Hinton *et al.* [16] so as to speed up inference time. They are viewed as a different kind of training method, where knowledge is compressed and transferred from cumbersome model into compact one. In fact, similar ideas could date back to [17], where large complex ensembles are compressed into single fast model without any significant loss of accuracy. Similar to knowledge distillation, a learning method with teacher-student architecture is presented by Ba and Caruana [18], which demonstrates that shallow network can also learn complicated nonlinear mappings from deep network and then achieve accuracy similar to deep network. Following such work, a deep and thin network FitNet [19] is trained . In addition, knowledge is transferred from intermediate hidden layers of teacher network (hint) to that of student network (guided) in hint/guided-based training.

In addition to the above-mentioned two methods, hardware-specific optimization can also reduce test time complexity. Vanhoucke *et al.* [7] make efforts to lower computational cost on x86 CPUs, where data layout, computation batching, and fixed-point SIMD operations can be utilized to effectively optimize large matrix computations. Farabet *et al.* [8] design a large scale FPGA-based architecture for convolutional networks. A new way to parallelize training of CNNs across multiple GPUs is presented in [9]. As major open source frameworks, both cuda-convnet [1] and caffe [10] provide effective GPU acceleration for CNN computation. Besides that, based on FFT, Mathieu *et al.* [22] transform convolutional operation to the Fourier domain, where feedforward convolutional operation with speed-up of approximately 2 times is done.

3 Method

3.1 CNN Architecture with Dominant Convolutional Kernels

Convolutional layer, which has a set of learnable parameters, plays a critical role in CNNs. In fact, most of parameters and computational burden lie in such convolutional layers. To compress regular CNN architecture, convolutional layer is the most profitable target. In this paper, we propose a new dominant convolutional kernel method to accomplish compression of convolutional layers.

Suppose that c_i and c_o indicate the number of input and output channels, respectively, and k_h and k_w the height and width of convolutional kernel, respectively. For regular convolution (Fig. 1(a)), we have convolutional kernels of $c_i \times c_o$

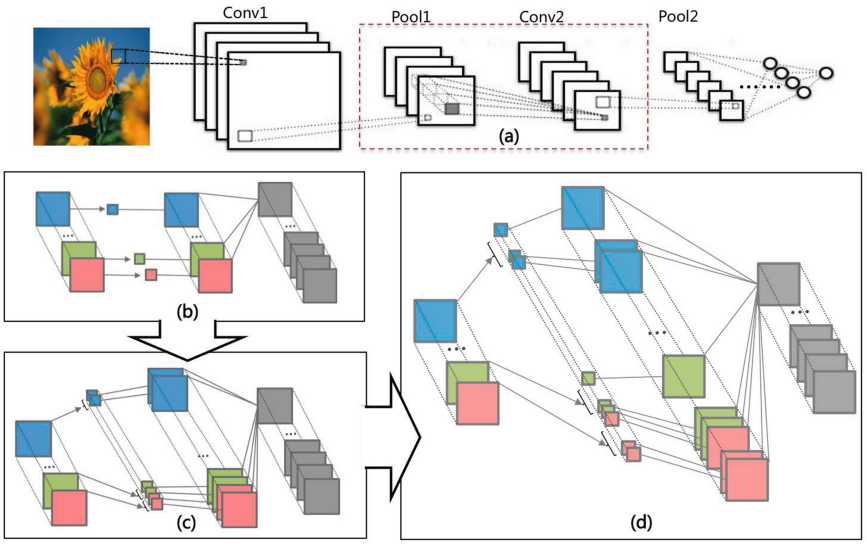

Fig. 1. Dominant Convolutional Kernel (DK). (a) Regular convolutional kernel; (b) DK of 1×1; (c) DK of 1×2; (d) DK of $1 \times n$.

in total. In fact, convolutional operations on input feature maps are regarded as feature extraction, while the summation of corresponding temporarily convolved feature maps could be considered as feature combination.

As for our dominant convolutional kernels shown in Figs. 1(b) and 3(d), feature is extracted only for single incoming feature map, while feature combination is done across temporarily convolved maps through 1×1 weights. Specifically, instead of convolving input feature maps with $c_i \times c_o$ convolutional kernels in regular convolutional layer, we merely employ c_i convolutional kernels, called dominant kernel (DK) in this paper. Apparently, there is 1-to-1 correspondence between inputs and temporarily convolved feature maps. This can be extended to the 1-to-n case, where $1 \leq n \leq k_h \times k_w$ as shown in Fig. 1(c) and (d). As a result, we have a total of parameters of $n \times c_i \times k_h \times k_w$ in such feature extraction phase. During feature combination, each output channel is a weighted sum of all the temporarily convolved maps. Note that weight vectors of $n \times c_i$ are exploited for combination of different maps. Thus there are totally $n \times c_i \times c_o$ parameters in such combination procedure. As n reaches up to the maximum of $k_h \times k_w$, our dominant convolutional layer may have capabilities of approximation similar as regular convolutional layer. It is because linear independence vectors of $k_h \times k_w$ determine a set of basis vectors for $k_h \times k_w$-dimensional feature spaces.

Our dominant convolutional layer uses parameters of $n \times c_i \times k_h \times k_w + n \times c_i \times c_o$ in total, which is only as many as $n/c_o + n/(k_h \times k_w)$ times of regular convolutional layer. For example, let us consider a commonly used convolutional configuration with a receptive field of 3×3 and 96×96 convolutional kernels. In this case, a dominant convolutional layer merely contains 12 % (1-to-1) and 24 %

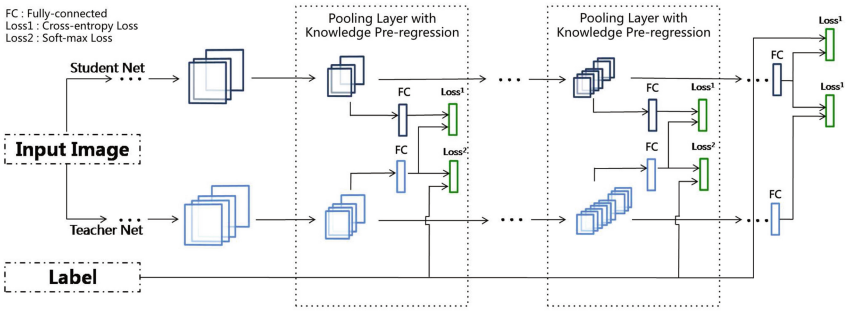

Fig. 2. Knowledge Pre-regression training method.

(1-to-2) of the parameters. Owing to the fact that computational compl exity is proportional to the number of parameters in a convolutional layer, time consuming is dramatically reduced. Moreover, the hyper-parameter n can be exploited to balance approximation capability and computational cost of a dominant convolutional layer.

3.2 Knowledge Pre-regression Training Method

Knowledge distillation lately proposed by Hinton *et al.* [16] aims to transfer knowledge from teacher network to student network. Knowledge of teacher network is usually stored in a form of parameters. Convolutional kernel parameters seem like too abstract, which makes it even hard to carry out knowledge transfer. In general, knowledge is defined as nonlinear mappings that map an input image to a target probability distribution (*i.e.* the output of softmax layer). Furthermore, such probability distribution of teacher network is transferred to student network. Romero *et al.* [19] extend such idea to transfer knowledge of intermediate hidden layers of teacher network to student network. Following this, we propose a knowledge pre-regression (KP) method to perform such knowledge transfer between teacher and student networks. In addition, we take advantage of the cross entropy loss function as our objective function, rather than Euclidean distance utilized in [19]. Actually, it is demonstrated to be more stable and robust.

Second, we choose the hint and guided layer pairs [19] for teacher network and student network, respectively. Considering that student network is often much simpler than its teacher network, a layer-by-layer pre-regression may lower recognition capabilities of student network. Based on our empirical data, the selection of pooling layers as hint/guided layers, which are further employed for performing knowledge pre-regression, is verified to be effective enough.

The hint/guided layer pairs, however, usually have different dimensions. Romero *et al.* [19] deal with this problem by introducing a convolutional regressor to match their dimensions, where Euclidean loss is utilized to optimize them. Although this method is simple, it is hard to converge. In this paper, we propose

a KP-based training method to tackle such problem, as shown in Fig. 2. An auxiliary regressor with a fully-connected layer R_t and the ground truth label l is introduced to the hint layer to generate a target probability distribution b_t. The probability distribution is then transferred to the guided layer. Meanwhile, we add another auxiliary regressor with a fully-connected layer R_s for the guided layer of student network. Furthermore, we adopt the cross entropy loss function to connect the hint and the guided layers. The loss function in our KP-based training method can be written as Eqs. (1), (2), and (3),

$$\mathcal{L}_{KB}(W_s, R_t, R_s) = \lambda \mathcal{H}(b_t^\tau, b_s^\tau) + \mathcal{S}(l, b_t) \tag{1}$$

$$b_t^\tau = softmax\left(\frac{h(x; W_t, R_t)}{\tau}\right) \tag{2}$$

$$b_s^\tau = softmax\left(\frac{g(x; W_s, R_s)}{\tau}\right) \tag{3}$$

where $\mathcal{S}$ and $\mathcal{H}$ represent the loss function of softmax and cross entropy, respectively, h and g are two functions that map an input image x to the hint and the guided layers, respectively, W_t and W_s stand for parameters in teacher network and student network, respectively, a temperature parameter τ [16] is utilized to soften the probability distribution over classes, and λ is a hyper-parameter used to balance these two loss functions.

We train our student network by minimizing the loss function below,

$$\mathcal{L}(W_s) = \lambda \mathcal{H}(p_t^\tau, p_s^\tau) + \mathcal{S}(l, p_s) + \sum_{i=1\cdots l} \lambda_i \mathcal{H}(b_t^{\tau^{(i)}}, b_s^{\tau^{(i)}}) + \mathcal{S}(l, b_t^{(i)}) \tag{4}$$

where p_t and p_s are the output probability of the teacher and student networks, respectively, and l indicates the hint/guided layers that we select. In our experiments, only the first two pooling layers are chosen as our hint/guided layers, which are further used for knowledge pre-regression.

3.3 Discussion

As described above, we present a model compression method using dominant convolutional layers, which can greatly decrease the number of parameters and computational burden as well. Although our method is also based on a low rank decomposition method, it is different from Jaderberg *et al.*'s [12] and Zhang *et al.*'s[15] methods, as shown in Fig. 3. Jaderberg *et al.* (Fig. 3(b)) approximate regular convolution as a composition of two linear mappings with intermediate maps. Each of two mappings convolves its input with a receptive field of $1 \times d$ or $d \times 1$. Zhang *et al.* (Fig. 3(c)) also adopt a two-step decomposition with intermediate maps. The decomposition is only conducted on the channel. Our dominant convolutional kernel (Fig. 3(d)) seems very similar to Zhang *et al.*'s method.

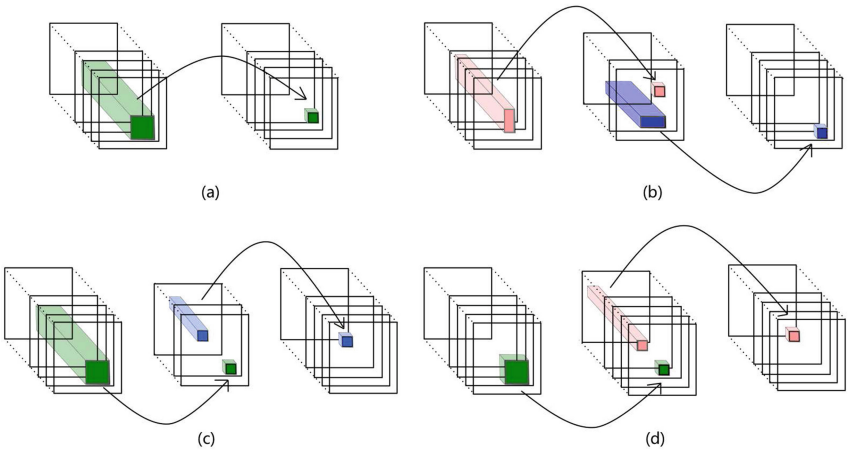

Fig. 3. Comparison to existing model compression methods. (a) Regular convolution operation; (b) Jaderberg *et al.*'s [12]; (c) Zhang *et al.*'s [15]; (d) The proposed DK decomposition.

But the difference is on the first decomposition step, instead of using a regular convolution, we use single or multiple dominant convolutional kernels to separately convolve each incoming feature map in order to further reduce computational complexity.

In order to improve performance of compressed model, we introduce a KP-based training method. We follow the work of Hinton *et al.* [16] and Romero *et al.* [19] and extend such work to intermediate layers and make it easy to converge through reconstructing objective function. As shown in Fig. 2, this architecture looks like two DSN models [23], although they have different meanings. DSN pays attention to improving classification performance of single model by introducing so-called companion objective to individual hidden layers, while our KP-based method focuses on knowledge transfer. Knowledge pre-regression enables our student model to have possibility of retaining a good balance between recognition and generalization capabilities.

4 Experimental Results

4.1 Overall Settings

We conduct our experiments based on the framework of caffe [10], running on two GPUs with data parallelism. We test the DK^{2}PNet models on several different benchmark datasets, including CIFAR-10 [4], CIFAR-100 [4], MNIST [5], and SVHN [6]. As an exceptional case, we complete more experiments on CIFAR-10 so as to specifically demonstrate effectiveness and efficiency of the proposed method.

For comparison, our DK^{2}PNet models are trained only by replacing KP-based training method with stochastic gradient descent (SGD) algorithm, which is denoted as DK^{2}PNet (SGD). In all the experiments, we set a mini-batch of 96. The learning rate is initially assigned to 0.01. As the loss begins to reach an apparent plateau, we drop this learning rate by a constant factor of 10 throughout training, repeatedly decreasing it three times until it arrives to 1e−5. In regular convolutional layers and dominant convolutional layers, the learning rate for bias is selected to twice as big as for weights. A momentum of 0.9 is adopted in the entire training process to make SGD stable and fast. We also exploit both weight decay and dropout for preventing over-fitting. All the learnable parameters make use of the same weight decay of 0.004. Dropout with a small ratio of 0.1 is introduced after each convolutional layer. Hyper-parameters such as weight factors of λ, λ_1 and λ_2 are carefully adjusted so as to yield better performance. In our experiment, we make fine-tuning of λ, λ_1, and λ_2 in interval of [0.0001, 1], where $\lambda \geq \lambda_1 \geq \lambda_2$. In fact, $\lambda = 0.1$ for the softmax layer, $\lambda_1 = 0.01$ for the second pooling layer, and $\lambda_2 = 0.001$ for the first pooling layer are a set of workable parameters. For all the datasets, image samples are preprocessed with removing the pre-pixel mean over their entire training dataset.

4.2 Setup of DK^{2}PNet Architecture

We design a couple of DK^{2}PNets with similar architecture but different parameters. Assume that there are K incoming feature maps for all the convolutional layers, whatever they are either regular or dominant. We adopt the same notation as [24] for convenience, *i.e.* DK^{2}PNet-K or CNN-K, which stands for the teacher network in our experiments, means that there are K feature maps in each layer.

The CNN-K is composed of 11 layers, including ten convolutional layers and one fully-connected layer. Three pooling layers succeed the 2nd, the 6th, and the 10th convolutional layers, respectively. Finally, a softmax layer is utilized to produce the final classification.

As the student network, DK^{2}PNet-K is structurally similar as its teacher network CNN-K. By substituting the second convolutional layer to the last one of CNN-K with dominant convolutional layers, it gives rise to our DK^{2}PNet-K. Note that the first convolutional layer in DK^{2}PNet-K is just a regular convolutional layer as warm-up.

In all regular and dominant convolutional layers, a reception field of 3×3 with stride and padding of 1 is employed for preserving spatial resolution. For the first two average pooling layers, we use a pooling area of 2×2 with stride of 2. A global average pooling is exploited as the last pooling layer. Additionally, all convolutional layers are combined with regularization of batch normalization [25], ReLU non-linearity, and dropout with a small drop ratio of 0.1. As a result, the DK2PNet-K (1-to-1) has a total of $(9K^2 + 108K)$ parameters, while regular teacher CNN-K has $(81K^2 + 27K)$ parameters.

4.3 CIFAR-10

CIFAR-10 [4] is a labeled subset of 80 million tiny image dataset. It contains 60,000 color images with size of 32×32. These images are categorized into 10 classes with 6,000 images per class. Such dataset is divided into five training batches and one test batch. Consequently, there are 50,000 training image samples and 10,000 test ones.

Table 1. Approximation capabilities of dominant convolutional kernel.

Model	CNN-96	DK^{2}PNet-96								
		1-to-1	1-to-2	1-to-3	1-to-4	1-to-5	1-to-6	1-to-7	1-to-8	1-to-9
#param	0.75 M	0.09 M	0.18 M	0.28 M	0.37 M	0.46 M	0.55 M	0.64 M	0.73 M	0.82M
Error (%)	**7.82**	9.59	8.98	8.55	8.26	**7.98**	8.25	8.56	8.21	8.03

To gain insight into approximation capabilities of DK^{2}PNet, nine DK^{2}PNet-96 models with different compression ratio are trained for comparison. As shown in Table 1, the teacher network CNN-96 is the base line model, while the student network DK^{2}PNet-96 is the compressed model produced by replacing the second to the last convolutional layers with the dominant convolutional layers. All these 10 models are embedded with BN and Dropout, and trained with SGD algorithm for the sake of comparison. The experimental results obtained are rather encouraging, the DK^{2}PNet-96 (1-to-1) with 0. 09 million parameters can achieve a test error of 9.59 %. Although it is not the best model on CIFAR-10, it is most likely to be the smallest model with test error of less than 10 %. By decreasing the compression ratio, we are able to improve performance of DK^{2}PNet-96, and the best compression performance is achieved by the DK^{2}PNet-96 (1-to-5). Unexpectedly, the DK^{2}PNet-96 (1-to-9), which has a slightly larger number of parameters compared to its teacher, cannot perform as good as its teacher. Such performance degradation is probably attributed to the training difficulty caused by the increase in depths and free parameters.

Furthermore, we complete comparative study to verify effectiveness of our knowledge pre-regression (KP) training method given in Sect. 3.2. From Table 2, the four DK^{2}PNets are trained with KP-based method and SGD algorithm, respectively. For the KP-based training method, we choose two teacher networks with different recognition performance to teach our DK^{2}PNet. Compared to SGD algorithm, KP-based training method is able to improve performance of all the four models. There is only a slight improvement for DK^{2}PNet-96 (1-to-1). The reason is that it seems like too small to optimize. Using KP-based training method, performance of DK^{2}PNet-128 (1-to-2) with more parameters is made better from a test error rate of 8.31 % to that of 7. 90 %. Even given that a teacher network achieves relatively poor performance, the use of it to train student networks also helps improve accuracy. By contrast, a good teacher could be more valuable.

Table 2. Comparison of our KP training method with SGD algorithm.

Model	#parameter	Test error (%)
SGD training		
DK^{2}PNet-96 (1-to-1, BN)	0.09 M	9.59
DK^{2}PNet-96 (1-to-2, BN)	0.18 M	8.98
DK^{2}PNet-128 (1-to-2, BN)	0.32 M	8.31
DK^{2}PNet-160 (1-to-2, BN)	0.49 M	7.91
KB-based training		
Good teacher	0.75 M	7.82
DK^{2}PNet-96 (1-to-1, BN)	0.09 M	9.52
DK^{2}PNet-96 (1-to-2, BN)	0.18 M	8.73
DK^{2}PNet-128 (1-to-2, BN)	0.32 M	7.90
DK^{2}PNet-160 (1-to-2, BN)	0.49 M	7.60
Poor teacher	0.75 M	12.18
DK^{2}PNet-96 (1-to-1, BN)	0.09 M	9.57
DK^{2}PNet-96 (1-to-2, BN)	0.18 M	8.78
DK^{2}PNet-128 (1-to-2, BN)	0.32 M	7.96
DK^{2}PNet-160 (1-to-2, BN)	0.49 M	7.70

Table 3. Comparison between different normalizations on CIFAR-10.

Model	#parameter	Test error (%)
CNN-96 (LRN)	0.75 M	12.18
DK^{2}PNet-96 (1-to-1, LRN)	0.09 M	26.88
DK^{2}PNet-96 (1-to-2, LRN)	0.18 M	26.52
DK^{2}PNet-128 (1-to-2, LRN)	0.32 M	25.01
CNN-96 (BN)	0.75 M	7.82
DK^{2}PNet-96 (1-to-1, BN)	0.09 M	9.59
DK^{2}PNet-96 (1-to-2, BN)	0.18 M	8.98
DK^{2}PNet-128 (1-to-2, BN)	0.32 M	8.31

Excellent performance of DK^{2}PNet should be partially attributed to regularization of batch normalization (BN) [25]. Comparative study of this issue is conducted on regular CNN and DK^{2}PNet. We train the following two sets of models: one is included with BN layers and another with LRN layers, instead of. The experimental results obtained are provided in Table 3. Apparently, the networks with LRN layers have very poor performance. This illustrates that normalization is able to significantly enhance approximation capabilities of CNN models, which gives us more confidence to take advantage of our KP-based

training method presented in Sect. 3.2. In fact, the KP-based training method can also be seen as a kind of regularization.

Finally, we compare our methods with the state of the art models without any data augmentation. First, the teacher network CNN-96 with 0.75 million parameters is trained as our base line model. Second, we compress CNN-96 to give rise to the small model of DK^{2}PNet-96 (1-to-2) with 0.18 million parameters and then achieve a test error of 8.73 %. We expand this model with more hidden units. The generated DK^{2}PNet-160 (1-to-2) with 0.49 million parameters has a test error of 7.60 % and outperforms its teacher network, which becomes the best record on such dataset if any data augmentation is not adopted (Table 4).

To keep consistent with previous work, we also test our models with data augmentation of translation and horizontal flipping on CIFAR-10. We randomly crop a portion of 24×24 pixels from original images and flip them horizontal randomly. In the test phase, five 24×24 crops from four corners and one center with their horizontal flipping crops are employed for testing. The final test results are found by an average of all the outputs of ten crops. With SGD training, the DK^{2}PNet-96 (1-to-1, SGD) with 0.09 million parameters yields a test error of 8.78 %, while a test error of 6.44 % of the DK^{2}PNet-256 (1-to-1, SGD) almost approaches the state of the art result of 6. 05 % provided in [29]. But our DK^{2}PNet-256 (1-to-1) model merely exploits 34 % of the parameters in comparison with [29].

4.4 CIFAR-100

Like CIFAR-10, CIFAR-100 [4] has the same training and test sizes but contains 100 classes. Using the same procedure as CIFAR-10, we train a couple of DK^{2}PNet models on such dataset and compare them with the state of the art models (as listed in Table 5). It is easy to see that the DK^{2}PNet-160 (1-to-2) achieves a test error of 31.39 % without any data augmentation. Note that it contains only 0.49 million parameters, which is less than its teacher network CNN-96, and reaches roughly 26 % of the state of the art lightweight model [24].

4.5 MNIST

MNIST [5] is a handwritten digits dataset with digits from 0 to 9. Each image sample in MNIST has been centered and size-normalized to 28×28 grayscale image. It contains 60,000 training images and 10,000 test images. We exploit the small DK^{2}PNet-96 (1-to-2) model with 0.18 million parameters for such relatively simple benchmark problem. It is readily observed from Table 6 that we achieve a test error rate of 0.31 % that is very close to [29], while requiring the least number of parameters, which is reduced by 0.15 million *w.r.t.* its teacher network CNN-64 and by 1.67 million compared to Tree+Max+Avg [29].

4.6 SVHN

SVHN [6] is a real image dataset, which is collected from house numbers in Google street view images. It includes 73,257 training images, 26,032 test images,

Table 4. Comparison with existing models on CIFAR-10.

Model	#parameter	Test error (%)
Without data augmentation		
Maxout [26]	>5 M	11.68
NIN [27]	0.97 M	10.41
DSN [23]	0.97 M	9.69
RCNN [24]	1.86 M	8.69
ALL-CNN [28]	1.4 M	9.08
Tree+Max-Avg [29]	1.85M	**7.62**
Teacher (CNN-96)	0.75 M	7.82
DK^{2}PNet-96 (1-to-2)	0.18 M	8.73
DK^{2}PNet-128 (1-to-2)	0.32 M	7.90
DK^{2}PNet-160 (1-to-2)	0.49 M	**7.60**
With data augmentation		
Maxout [26]	>5 M	9.38
DropConnect [30]	-	9.32
NIN [27]	0.97 M	8.81
DSN [23]	0.97 M	8.22
RCNN [24]	1.86 M	7.09
Highway Network [31]	2.3 M	7.54 (7.72 ± 0.16)
ALL-CNN [28]	1.4 M	7.25
ResNet [32]	1.7M	6.43 (6.61 ± 0.16)
Fitnet4-LSUV [33]	2.5M	6.06
Tree+Max-Avg [29]	1.85M	**6.05**
Tuned CNN [34]	1.29M	6.37
DK^{2}PNet-96 (1-to-1, SGD)	0.09 M	8.78
DK^{2}PNet-96 (1-to-2, SGD)	0.18 M	7.68
DK^{2}PNet-128 (1-to-2, SGD)	0.32 M	7.06
DK^{2}PNet-160 (1-to-2, SGD)	0.49 M	7.06
DK^{2}PNet-256 (1-to-1, SGD)	0.62 M	**6.44**

and an extra 531,131 additional samples of less difficulty for training. Among two formats for such dataset, we adopt the second format. When multiple digits simultaneously appear in single image, we only need to recognize the centering one. The experimental results show that our DK^{2}PNet-160 (1-to-2) with 0.49 million parameters yields a test error of 1.83 %, as listed in Table 7. Note that the number of parameters of DK^{2}PNet-160 (1-to-2) is significantly decreased compared to RCNN [24] and Tree+Max+Avg [29].

Table 5. Comparison with existing models on CIFAR-100.

Model	#parameter	Test error (%)
Without data augmentation		
Maxout [26]	>5 M	38.57
Tree based priors [35]	-	36.85
NIN [27]	0.98 M	35.68
DSN [23]	0.98 M	34.57
RCNN [24]	1.86 M	**31.75**
ALL-CNN [28]	1.3 M	33.71
Highway Network [31]	2.3 M	32.24
Tree+Max-Avg [29]	1.76 M	32.37
ELU-Network [36]	39.32 M	**24.28**
Teacher (CNN-96)	0.75 M	33.63
DK^{2}PNet-96 (1-to-2)	0.18 M	35.24
DK^{2}PNet-128 (1-to-2)	0.32 M	34.48
DK^{2}PNet-160 (1-to-2)	0.49 M	**31.39**
With data augmentation		
Fitnet4-LSUV [33]	2.5 M	27.66
Tuned CNN [34]	1.29 M	27.40

Table 6. Comparison with existing models on MNIST.

Model	#parameter	Test error (%)
Without data augmentation		
Maxout [26]	0.42 M	0.45
NIN [27]	0.35 M	0.47
DSN [23]	0.35 M	0.39
RCNN [24]	0.67 M	0.31
Tree+Max-Avg [29]	1.85 M	0.31
FitNet-LSUV-SVM [33]	0.03 M	0.38
Tree+Max+Avg [29]	1.85 M	**0.29**
Teacher (CNN-64)	0.33 M	0.33
DK^{2}PNet-64 (1-to-2)	0.09 M	0.38
DK^{2}PNet-96 (1-to-1)	0.09 M	0.36
DK^{2}PNet-96 (1-to-2)	0.18 M	**0.31**
With data augmentation		
Dropconnect [30]	-	0.21
MCDNN[37]	-	0.23

Table 7. Comparison with existing models on SVHN.

Model	#parameter	Test error (%)
The state of the art models		
Maxout [26]	>5 M	2.47
NIN [27]	1.98 M	2.35
DSN [23]	1.98 M	1.92
RCNN [24]	2.67 M	**1.77**
Tree+Max+Avg [29]	4.00M	**1.69**
Teacher (CNN-96)	0.75 M	1.82
DK^{2}PNet-96 (1-to-2)	0.18 M	2.04
DK^{2}PNet-128 (1-to-2)	0.32 M	1.95
DK^{2}PNet-160 (1-to-2)	0.49 M	**1.83**

5 Conclusion

In this paper, we propose a model acceleration method using dominant convolutional kernel and knowledge pre-regression. First, by replacing regular convolutional layers with dominant convolutional layers, CNN architecture can be simplified significantly, resulting in efficient model acceleration. To tackle performance degradation problems caused by compression, a new knowledge pre-regression training method is further presented to transfer knowledge of intermediate hidden layers from original teacher network to its compressed student network. It makes student network quickly learn and generalize well. Finally, our experimental results show that the proposed DK^{2}PNet provides near state of the art test errors, while requiring notably fewer parameters than regular CNN models. For example, without any data augmentation, the DK^{2}PNet-160 yields the best performance of 7.60 % on CIFAR-10 using almost 3.8 times less parameters, when compared to existing state of the art results of 7.62 % [29]. On CIFAR-100, while the DK^{2}PNet-160 achieves better performance with a test error of 31.39 % that is close to [36], it only requires roughly 80.2 times fewer parameters. On MNIST without any data augmentation, the DK^{2}PNet-96 with 0.18 million parameters, which is the least number of parameters adopted, obtains a test error of 0.31 %. Our DK^{2}PNet-160 receives near state of the art result of 1.83 % on SVHN benchmark with roughly 12 % of the parameters in comparison with [29], dramatically reducing the number of parameters by a factor of 8.

Acknowledgements. This work was supported in part by the National Science Foundation of China (NSFC) under Grant Nos. 91420106, 90820305, and 60775040, and by the National High-Tech R&D Program of China under Grant No. 2012AA041402.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
2. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
3. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)
4. Krizhevsky, A.: Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto (2009)
5. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
6. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Granada, Spain, vol. 2011, p. 4 (2011)
7. Vanhoucke, V., Senior, A., Mao, M.Z.: Improving the speed of neural networks on CPUs. In: Proceedings of Deep Learning and Unsupervised Feature Learning NIPS Workshop, vol. 1 (2011)
8. Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., Talay, S.: Large-scale FPGA-based convolutional networks. In: Scaling up Machine Learning: Parallel and Distributed Approaches, pp. 399–419 (2011)
9. Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks. arXiv preprint arXiv:1404.5997 (2014)
10. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the ACM International Conference on Multimedia, pp. 675–678. ACM (2014)
11. Denil, M., Shakibi, B., Dinh, L., de Freitas, N., et al.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems, pp. 2148–2156 (2013)
12. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. In: Proceedings of the British Machine Vision Conference. BMVA Press (2014)
13. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems, pp. 1269–1277(2014)
14. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553 (2014)
15. Zhang, X., Zou, J., Ming, X., He, K., Sun, J.: Efficient and accurate approximations of nonlinear convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1984–1992 (2015)
16. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
17. Buciluǎ, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 535–541. ACM (2006)

18. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Advances in neural information processing systems, pp. 2654–2662 (2014)
19. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: hints for thin deep nets. arXiv preprint arXiv:1412.6550 (2014)
20. LeCun, Y., Denker, J.S., Solla, S.A., Howard, R.E., Jackel, L.D.: Optimal brain damage. In: NIPS, pp. 598–605(1989)
21. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: ICML, pp. 2285–2294 (2015)
22. Mathieu, M., Henaff, M., LeCun, Y.: Fast training of convolutional networks through FFTs. arXiv preprint arXiv:1312.5851 (2013)
23. Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. In: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, pp. 562–570 (2015)
24. Liang, M., Hu, X.: Recurrent convolutional neural network for object recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3367–3375 (2015)
25. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning, pp. 448–456 (2015)
26. Goodfellow, I., Warde-farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: Proceedings of the 30th International Conference on Machine Learning (ICML 2013), pp. 1319–1327 (2013)
27. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
28. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
29. Lee, C.Y., Gallagher, P.W., Tu, Z.: Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. arXiv preprint arXiv:1509.08985 (2015)
30. Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. In: Proceedings of the 30th International Conference on Machine Learning (ICML 2013), pp. 1058–1066 (2013)
31. Srivastava, R.K., Greff, K., Schmidhuber, J.: Training very deep networks. In: Advances in Neural Information Processing Systems, pp. 2368–2376 (2015)
32. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
33. Mishkin, D., Matas, J.: All you need is a good init. arXiv preprint arXiv:1511.06422 (2015)
34. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable bayesian optimization using deep neural networks. In: Proceedings of the 32nd International Conference on Machine Learning (ICML 2015), pp. 2171–2180 (2015)
35. Srivastava, N., Salakhutdinov, R.R.: Discriminative transfer learning with tree-based priors. In: Advances in Neural Information Processing Systems, pp. 2094–2102 (2013)
36. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUs). arXiv preprint arXiv:1511.07289 (2015)
37. Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3642–3649. IEEE (2012)