RotateConv: Making Asymmetric Convolutional Kernels Rotatable

Jiabin Ma, Weiyu Guo, Wei Wang and Liang Wang
Center for Research on Intelligent Perception and Computing, CASIA, Beijing, China
National Laboratory of Pattern Recognition, CASIA, Beijing, China
University of Chinese Academy of Sciences, Beijing, China
Email: jiabin.ma@cripac.ia.ac.cn, weiyu.guo@ia.ac.cn, wangwei@nlpr.ia.ac.cn, wangliang@nlpr.ia.ac.cn

***Abstract*—In deep Convolutional Neural Networks(CNN), the design of kernel shapes influences a lot on the model size and performance. In this work, our proposed method, RotateConv, applies a novel kernel shape to massively reduce the number of parameters while maintaining considerable performance. The new shape is extremely simple as a line segment one, and we equip it with the rotatable ability which aims to learn diverse features with respect to different angles. The kernel weights and angles are learned simultaneously during end-to-end training via the standard back-propagation algorithm. There are two variants of RotateConv that only have 2 and 4 parameters respectively depending on whether using weight sharing, which are much compressed than the normal 3×3 kernel with 9 parameters. In experiments, we validate our RotateConv with two classical models, ResNet and DenseNet, on four image classification benchmark datasets, namely MNIST, CIFAR10, CIFAR100 and SVHN.**

I. Introduction

In recent years, CNN becomes a state-of-the-art technique for computer vision tasks like Image Classification [1], [2], Semantic Segmentation [3], [4] and Object Detection [5], [6], [7] since it effectively extracts deep and valuable features. Although deep models [2], [8], [9], [10], [11] are very powerful, the large number of learnable parameters consumes a lot of memory storage. For example, the parameter numbers of ResNet101 [10] and DenseNet100 [11] are 39M and 27M respectively, which imposes restrictions on device-limited applications such as mobile systems. And among these parameters, convolution weights are main parts.

Kernel factorization and weight pruning are two methods to reduce the model size of deep neural networks. Kernel factorization like Inception V3 [9] uses asymmetric kernels like $n \times 1$ and $1 \times n$, and perpendicularly intersects them to replace one normal square kernel. Because asymmetric kernels have line segment shapes, the two-layer solution is still 33% cheaper for the same number of output channels. Weight pruning [12], [13] is a post processing method which deletes the small weight parameters after pre-training. The thoughts of two methods are both to get even smaller kernels than original square kernels. But these two methods have their limitations. Asymmetric kernels of kernel factorization only have two kinds of angles as vertical and horizontal, which somehow restricts the model capability of convolution. Weight pruning can not be completed in one step as it needs to wait for the whole kernel pre-training and finetune it later.

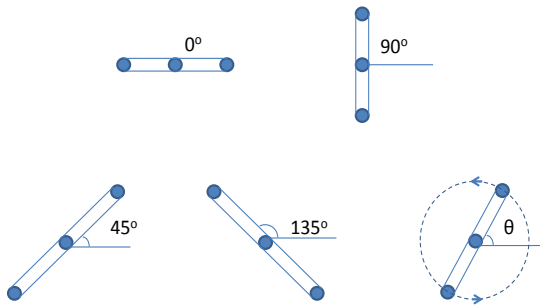
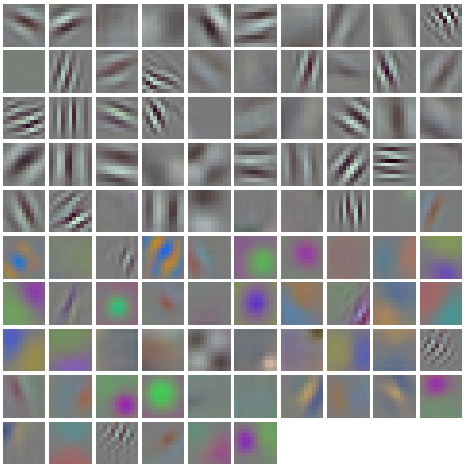


Fig. 1. Asymmetric kernels with different angles. In top-down and left-right order, the former four kernels have different but fixed angles as $0°$, $90°$, $45°$ and $135°$. The last one is an illustration of our RotateConv kernel that is asymmetric but rotatable, which means the angle θ is learnable during training.

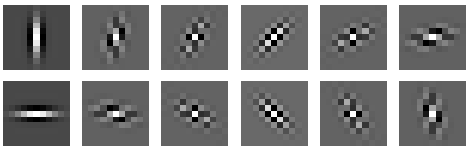
In this work, we propose a novel kernel deformation which makes up the deficiency of both above two methods. Compared with asymmetric kernels of kernel factorization, RotateConv kernels, shown as the last kind in Figure 1, are much more flexible considering that they can have any continuous angles from $0°$ to $180°$ instead of $0°$ and $90°$. Compared with weight pruning, RotateConv kernels do not need to cut the kernel first as they are born with line segment shape and have much less weights than the square one, so that we can train the network in only one step.

RotateConv is inspired by the kernels visualization of AlexNet [2] and Gabor [14]. As shown in Figure 2(a), AlexNet has learned a variety of frequency- and orientation-selective kernels. Simple as these kernels are, they can model much more complicated patterns in combination, which is similar to the thought of calculus. Figure 2(b) shows some Gabor filters with respect to different directions, which are very similar with the ones in AlexNet. We can infer from these figures that directions are important for both gray-like and colored kernels so as to extract abundant information.

Aiming at simulating the direction attributes in the kernel weights distribution, we propose a new kernel deformation method, RotateConv. RotateConv Kernel is shown as the last one in Figure 1. As we can see, the basic shape of RotateConv kernel is a line segment, which means that it only has 3 weights for convolution. But besides weights, it has an additional learnable variable, angle θ. Note that making



(a). Kernels visualization of AlexNet



(b). Kernels visualization of Gabor

Fig. 2. (a) 96 convolutional kernels of size $11 \times 11 \times 3$ learned in the first convolutional layer in AlexNet. (b) Gabor filters visualization with respect to 12 different angles.

the kernel rotatable is achieved by making the variable θ continuous and learnable. Due to the existence of θ, this 3-weights kernel can have a larger receptive field like 3×3, so its modeling capacity is guaranteed. Considering different emphases on the model size and performance, we propose two variants of our rotatable kernels. One has 4 parameters consisting of 3 weights and 1 angle as explained above, and the other has 2 parameters consisting of 1 weight and 1 angle because of the use of weight sharing. Accordingly the compress ratios are $4/9$ and $2/9$ respectively. Experiments of image classification are executed to validate the effectiveness of our proposed methods. And we find that, the kernel with 4 parameters gets comparable results with the 3×3 baseline kernel, and the kernel with 2 parameters can get acceptable results.

II. Related Work

Since RotateConv devotes to achieve a more compressed convolutional kernel, related work can be divided into two groups, i.e., kernel design and model compression.

A. Kernel design

The basic kernel design in deep CNN can be inferred from the series work of Inception VN [15], [16], [9]. Inception V1 uses multi-scale kernels to model scale-invariant features. Inception V2 uses more stacked small kernels to replace one bigger kernel so as to increase the depth of the network and reduce the number of parameters. Inception V3 further makes the kernel smaller as it uses asymmetric kernels like 1×3 and 3×1. As we mentioned above, though this asymmetric kernel reduces parameters a lot, its fixed angle as vertical and horizontal puts limitations on the capacity of modeling more orientation-flexible patterns. Dilated convolution [17] is another widely used kernel shape which aims at solving the resolution reduction problem of feature maps in forward propagation. It is a dilated variant of traditional compact kernels, which helps the kernel have a larger receptive field without increasing parameters.

Recently, there emerge some novel deformable kernel design works. Deformable Convolutional Networks (DCN) [18] is a recently proposed excellent work which learns irregular kernels. DCN has a similar thought with Region Proposal Network [6] as it applies a usual convolution on the input feature and outputs the new kernel shape for the following deformable convolution layer. Irregular Convolutional Neural Networks (ICNN) [19] is another work learning irregular kernels. Different from DCN, ICNN directly models the kernel's shape attributes as learnable variables and learns the shapes in the same way as kernel weights. These two methods aim to expand the kernel capacity utilizing all original 3×3 parameters, but our RotateConv devotes to maintain the capacity with less parameters.

B. Model compression

There has been growing interest in model compression due to the demands of device-limited applications. Weight pruning methods hold the point that small weights are less important, as S. Han et al. [12] remove the small weights and V. Lebedev et al. [20] remove the entire groups by introducing sparsity regularization on groups. Weight pruning is the most similar method to our approach, but RotateConv is more convenient for training as it can be trained in an end-to-end manner for only one step. Low bit network is another emerging approach which uses less bits to represent the filter values, such as Ternary Weight Networks [21] with 2 bit weights and Binary Weight Networks [22] with 1 bit weights. And the method of product quantization [23] uses clustering and weight sharing to compress the model size of neural networks. Additionally, many matrix factorization methods [24], [25] have also been proposed in the recent literature.

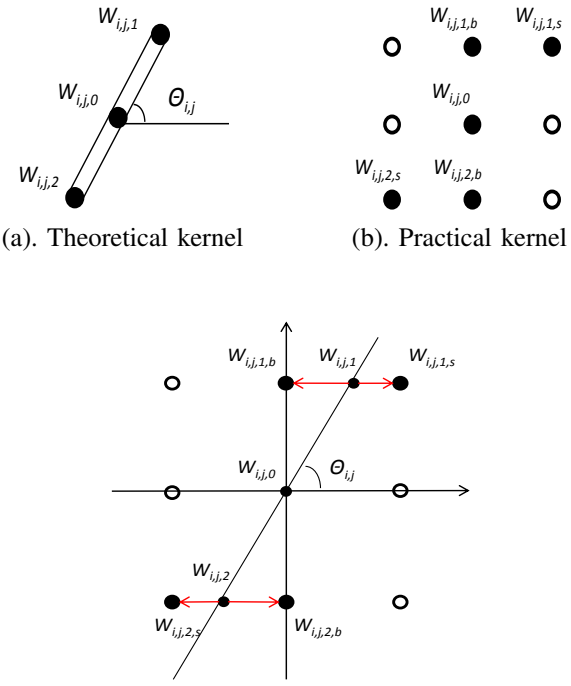
III. Mathematical Derivation

In this section, we will explain RotateConv's mathematical derivations for both 4 parameters and 2 parameters versions.

A. 4 parameters version

1) Data Structure: In our work, we firstly use the line segment kernel shape which consists of 3 weights placed on a straight line as shown in Figure 1. We secondly add a new angle attribute to the data structure of the convolutional kernel K:

$$
\begin{aligned}
&K = \{W, T\} \\
&W = \{w_{i,j,0}, w_{i,j,1}, w_{i,j,2} | i = 1, 2, ..., N, j = 1, 2, .., M\} \\
&T = \{\theta_{i,j} | i = 1, 2, ..., N, j = 1, 2, .., M\}
\end{aligned} \tag{1}
$$



(a). Theoretical kernel (b). Practical kernel

(c). Angle inverse-interpolation

Fig. 3. (a) Theoretical RotateConv kernel has a line segment shape, whose weights are w_0, w_1, w_2 and angle is θ. (b) Practical RotateConv kernel for computation after angle inverse-interpolation. (c) The angle inverse-interpolation process for RotateConv.

where M equals to the number of input channels and N equals to the number of output channels. W is the set of kernel weights and each feature map gets 3 weights. T is the set of kernel angles θ. θ is defined as the included angle between the horizontal line and the kernel line as shown in Figure 1. It is bigger than 0° and can not exceed 180°.

2) Inverse-Interpolation: For a RotateConv kernel shown in Figure 3(a), the output can be calculated as:

$$S_i = \sum_{j=1}^{M} (w_{i,j,0} I_{j,0} + w_{i,j,1} I_{j,1} + w_{i,j,2} I_{j,2}) \tag{2}$$

where S_i is the weighted summation in the ith output channel. Inputs $I_{j,0}$, $I_{j,1}$ and $I_{j,2}$ correspond to the weights $w_{i,j,0}$, $w_{i,j,1}$ and $w_{i,j,2}$ respectively.

Note that for RotateConv, $I_{j,1}$, $I_{j,2}$ are sampled by $\theta_{i,j}$. The problem is that, as $\theta_{i,j}$ is continuous, when $\theta_{i,j}$ does not equal to an integer multiple of 45°, $I_{j,1}$ and $I_{j,2}$ do not exist. To solve this problem, we split the weights[1], $w_{i,j,1}$ and $w_{i,j,2}$, to the adjacent positions which are the integer multiple of 45° as shown in Figure 3(c). And at last, we will get the practical 5 weights kernel like Figure 3(b). In this way, we can sample 5 existing inputs for weighted summation. The split process can be calculated as:

[1]Instead of spliting the weights to the adjacent positions, we can also combine the input pixels corresponding to $w_{i,j,1,b}$ and $w_{i,j,1,s}$. But this method would bring in a time consuming problem related with the practical convolution implementation $im2col$, so we do not adopt it.

$$\begin{aligned} w_{i,j,1,b} &= w_{i,j,1} \times f(\theta_{i,j}) \\ w_{i,j,1,s} &= w_{i,j,1} \times (1 - f(\theta_{i,j})) \\ w_{i,j,2,b} &= w_{i,j,2} \times f(\theta_{i,j}) \\ w_{i,j,2,s} &= w_{i,j,2} \times (1 - f(\theta_{i,j})) \end{aligned} \tag{3}$$

$$s.t. f(\theta_{i,j}) = \frac{\theta_{i,j}\%45}{45}, 0 \leq \theta_{i,j} < 180^\circ$$

where $w_{i,j,1,b}$ and $w_{i,j,1,s}$ mean the weights split by $w_{i,j,1}$, same as $w_{i,j,2,b}$ and $w_{i,j,2,s}$ split by $w_{i,j,2}$. The split sizes are determined by $f(\theta_{i,j})$, which is simply defined at the ratio between the include angle and 45°.

3) Convolution: The convolution is a weighted summation after angle inverse-interpolation:

$$\begin{aligned} &S_i = \sum_{j=1}^{M} (S_{i,j,0} + S_{i,j,1} + S_{i,j,2}) \\ &s.t. \begin{cases} S_{i,j,0} = w_{i,j,0} I_{i,j,0} \\ S_{i,j,1} = w_{i,j,1,b} I_{i,j,1,b} + w_{i,j,1,s} I_{i,j,1,s} \\ S_{i,j,2} = w_{i,j,2,b} I_{i,j,2,b} + w_{i,j,2,s} I_{i,j,2,s} \end{cases} \end{aligned} \tag{4}$$

4) Back Propagation: There are three kinds of learnable variables, inputs I , weights W and angles T. And there is one kind of intermediate variables W_{bs}, as called $w_{i,j,1,b}$, $w_{i,j,1,s}$, $w_{i,j,2,b}$ and $w_{i,j,2,s}$. The gradients for inputs I are same as the ones of traditional convolution. The gradients for W_{bs} and $W_{i,j,0}$ can be calculated in the traditional way, since W_{bs}, $W_{i,j,0}$ and zeros are combined to form a usual 3×3 kernel as shown in Figure 3(b). The key point is to compute the gradients for weights $W_{i,j,1}$, $W_{i,j,2}$, and angles $\theta_{i,j}$ with the intermediate variables W_{bs}.

$$\begin{aligned} \Delta w_{i,j,1} &= \Delta w_{i,j,1,b} f(\theta_{i,j}) + \Delta w_{i,j,1,s} (1 - f(\theta_{i,j})) \\ \Delta w_{i,j,2} &= \Delta w_{i,j,2,b} f(\theta_{i,j}) + \Delta w_{i,j,2,s} (1 - f(\theta_{i,j})) \\ \Delta \theta_{i,j} &= w_{i,j,1} (\Delta w_{i,j,1,b} - \Delta w_{i,j,1,s}) f'(\theta_{i,j}) \\ &\quad + w_{i,j,2} (\Delta w_{i,j,2,b} - \Delta w_{i,j,2,s}) f'(\theta_{i,j}) \end{aligned} \tag{5}$$

5) Initialization: Weights W and angles T can both be initialized from random distribution or pre-trained 1×3 and 3×1 kernels. Note that the initialization for angles T should be in the range from zero to 180°.

6) Update Values: The update mechanism is same as before for weights W, but not for angles T. For a certain angle θ, since backward gradients, are supplied from w_{bs}, the updated new θ should not excess the boundaries defined by w_b and w_s too much.

$$\begin{aligned} &\theta_{update} = (\theta_{last} + \Delta\theta)\%180 \\ &s.t. \begin{cases} \theta_{last\_small} - \epsilon < \theta_{update} < \theta_{last\_big} + \epsilon \\ \theta_{last\_small} = \theta_{last} - \theta_{last}\%45 \\ \theta_{last\_big} = \theta_{last} - \theta_{last}\%45 + 45 \end{cases} \end{aligned} \tag{6}$$

where ϵ is a small positive value to allow θ_{update} to get out of the last adjacent boundaries θ_{last_small} and θ_{last_big} but not too much.

B. 2 parameters version

The mathematical derivations are similar for 2 parameters version and 4 parameters version except whether using weight sharing among $w_{i,j,0}$, $w_{i,j,1}$ and $w_{i,j,2}$ to have a single weight $w_{i,j}$. In this subsection, the meanings of all variables (except the shared weight $w_{i,j}$) are identical to the ones of 4 parameters version. So for simplicity, there will be little annotation for derivation. Please refer to the part of 4 parameters version for detailed explanation.

1) Data Structure:

$$\begin{aligned} K &= \{W, T\} \\ W &= \{w_{i,j} | i = 1, 2, ..., N, j = 1, 2, .., M\} \\ T &= \{\theta_{i,j} | i = 1, 2, ..., N, j = 1, 2, .., M\} \end{aligned} \tag{7}$$

2) Inverse-Interpolation:

$$\begin{aligned} w_{i,j,1,b} &= w_{i,j} \times f(\theta_{i,j}) \\ w_{i,j,1,s} &= w_{i,j} \times (1 - f(\theta_{i,j})) \\ w_{i,j,2,b} &= w_{i,j} \times f(\theta_{i,j}) \\ w_{i,j,2,s} &= w_{i,j} \times (1 - f(\theta_{i,j})) \\ w_{i,j,0} &= w_{i,j} \end{aligned} \tag{8}$$

$$s.t. f(\theta_{i,j}) = \frac{\theta_{i,j}\%45}{45}, 0 \leq \theta_{i,j} < 180^\circ$$

3) Convolution: Same as 4 parameters version.

4) Back Propagation:

$$\Delta w_{i,j} = \Delta w_{i,j,1} + \Delta w_{i,j,2} + \Delta w_{i,j,0} \tag{9}$$

Calculations for $\Delta w_{i,j,1}$, $\Delta w_{i,j,2}$ and $\Delta w_{i,j,0}$, and gradients for θ, please refer to the ones in 4 parameters version.

5) Initialization: Same as 4 parameters version.

6) Update Values: Same as 4 parameters version.

IV. Experiments

This section consists of four subsections. The first two subsections introduce datasets and deep models respectively. In the third subsection, we evaluate the two versions of RotateConv on the image classification task. last, we analyze the kernel angle distribution for better understanding.

A. Datasets

The MNIST dataset [26] of handwritten digits with 10 classes has a training set of 60,000 examples and a test set of 10,000 examples. Each digit is centered at a 28×28 gray image. MNIST is an ideal dataset for researchers who need quick verification for their first thought, since its pictures are small enough and it has been applied enough preprocessing as normalization, formatting and centering.

There are exactly two kinds of CIFAR datasets, CIFAR10 and CIFAR100 [27], according to different samples and label degrees. Both of them are object classification datasets chosen from real world. Each of them has 50,000 images for training and 10,000 images for testing, and each image is coloured and resized to 32×32 for normalization. CIFAR10 has 10 classes. And for each class, it has 5,000 images for training and 1,000 images for testing. CIFAR100 is finer as it has 100 classes. And for each class, it has 500 images for training and 100 images for testing.

SVHN [28] is a real-world digit image dataset for developing machine learning and object recognition algorithms. It is obtained from house numbers in Google Street View images. The task is to classify the digit centered in image. It has 10 classes, 73,257 digits for training, 26,032 digits for testing, and 531,131 digits for additional but somewhat less difficult samples which are used as extra training data. We use the MNIST-like version for experiments, each image has a 32×32 spatial size and is centered around a single digit which means that many examples do contain some distractions at the sides.

B. Deep Models

We firstly designed a simple network(SN) of four stacked convolutional layers. The output channels are 20, 50, 50, 50 respectively. The first three convolutional layers are followed by stride-2 pooling layers and the last convolutional layer is followed by ReLU layer and inner-product layer as the classifier. We use the SN network and the MNIST dataset for our first verification on RotateConv.

ResNet is introduced in [10] and makes great contribution in deep learning. Using shortcut connections and deeper networks, it massively improves the performance in various learning tasks while maintaining the efficiency in the model size and data computation. DenseNet is a more recently proposed work in [11]. It replaces the element wise summation layer in ResNet with channel concatenate layer, designs different block convolutions and network depth, and finally gets the state-of-art performance. In order to strictly validate RotateConv, we choose such two efficient models for verification.

We reproduce two ResNet variants, ResNet20 and ResNet44 as illustrated in [10]. ResNet20 has three stages, each stage has 3 residual blocks (include dimension-increase block), and each block has two 3×3 convolutional layers. On each stage, the output channels are 16, 32 and 64 respectively. For ResNet44, settings remain same except that on each stage, there are 7 residual blocks. Besides ResNet, the reproduced DenseNet40 has 3 stages, each stage has 12 blocks, each block has one 3×3 convolutional layer, and each convolutional layer has 23 output channels.

C. Results of Image Classification

Parameter numbers reduce a lot for deep networks. As shown in Table I, $version9$ means the basic 3×3 kernel, $version4$ means 4 parameters RotateConv kernel and $version2$ means 2 parameters RotateConv kernel. These numbers clearly denote the compression degree in the kernel level. Taking ResNet20 for an example, the basic model has $271k$ parameters, $version4$ only has $122k$ parameters and $version2$ has an even smaller number $63k$. More details can be inferred from Table I.

The results show that $version4$ is comparable with $version9$. And in some cases, $version4$ even gets better results. It means that the line segment shape is effective for the image classification task. We can also infer from the table

TABLE I

RESULTS FOR IMAGE CLASSIFICATION ON MNSIT, CIFAR10/100 AND SVHN DATASETS. WE USE FOUR MODELS, SIMPLE NETWORK(SN) AND RESNET20, RESNET44, DENSENET40 FOR THEIR MODEL EFFICIENCY. BASICALLY, FOR ONE MODEL ON ONE DATASET, WE LIST THREE PERFORMANCES CORRESPONDING TO THREE SETTINGS FOR COMPARISON, AS SHOWN IN SECOND ROW ON TABLE, 9 FOR 3×3 BASELINE KERNEL, 4 FOR 4 PARAMETERS ROTATECONV VERSION AND 2 FOR 2 PARAMETERS ROTATECONV VERSION.

Models	SN			ResNet20			ResNet44			DenseNet40		
Versions	9	4	2	9[10]	4	2	9[10]	4	2	9[11]	4	2
#Params	55k	25k	13k	271k	122k	63k	657k	294k	148k	1.9M	0.86M	0.43M
MNIST	**98.95**	98.65	97.41	-	-	-	-	-	-	-	-	-
CIFAR10	-	-	-	**91.70**	90.78	84.83	**92.52**	91.83	87.01	92.63	**92.68**	81.24
CIFAR100	-	-	-	52.84	**53.45**	50.39	56.63	**56.93**	51.22	**70.53**	69.99	50.64
SVHN	-	-	-	95.82	**96.01**	95.17	96.02	**96.40**	95.68	**97.15**	96.84	90.69

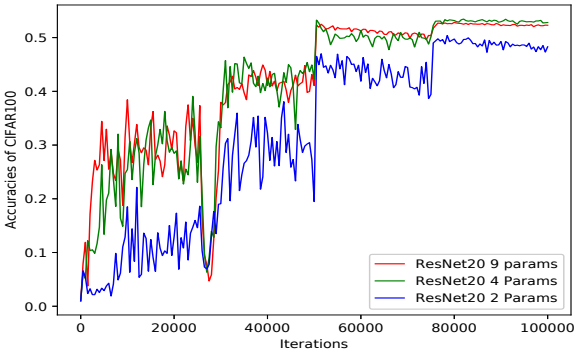
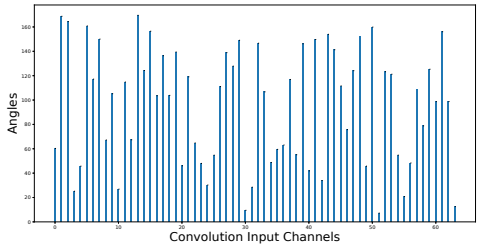
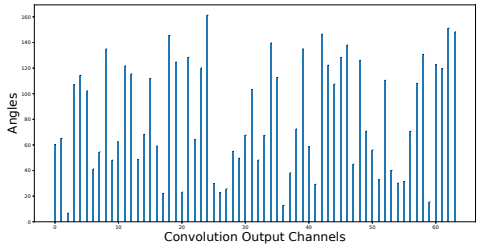


Fig. 4. CIFAR100 test accuracies of ResNet20. Different colours correspond to different versions, red for 9 parameters baseline, green for 4 parameters version and blue for 2 parameters version. Best viewed in color.



(a). Kernel angles along input channels



(b). Kernel angles along output channels

Fig. 5. Kernel angle distributions for layer $Convolution21$ in $ResNet20$ trained on $CIFAR100$. Layer $Convolution21$ has 64 input channels and 64 output channels. (a) For the first output channel, the 64 kernel angles applied on input features. (b) For the first input channel, the 64 kernel angles used by 64 output channels respectively.

that $version2$ still gets acceptable performance though its compression degree is so impressive as $2/9$. The comparison between $version4$ and $version2$ shows that weights diversity is helpful for feature extraction. For example, $version4$ can directly model the difference function with weights $[-1, 0, 1]$, but $version2$ can not. Figure 4 shows the accuracy curves for the model ResNet20 and the dataset CIFAR100. As we can see in the figure, accuracies of $version9$ and $version4$ seesaw by iterations, and finally get almost equal levels. The ascent tendency of $version2$ is a little slower but it still gets 50.39% for best.

D. Analysis of Kernel Angle Distribution

In this section, we firstly choose the model $ResNet20$ without loss of generality, then train it on $CIFAR100$ with 2 parameters RotateConv kernels, and at last we choose the last convolutional layer $Convolution21$ for observation. $Convolution21$ has 64 output channels and 64 input channels. Here we analyze two distributions along input and output channel respectively.

On the one hand, we give the angle distribution for one output channel. For each output channel, it has 64 spatial kernels corresponding to 64 RotateConv angles applied along input channels. Figure 5(a) shows the angle distribution of the first output channel, which is denoted as $\{\theta_{1,j}|j = 1, 2, .., 64\}$ before. We can find that different input features are applied with different RotateConv angles, which is coincident with universal intuition that different features represent different patterns.

On the other hand, we give the angle distribution for one input channel. For each input channel, it has been repeatedly used by 64 different output channels which has 64 different RotateConv angles too. Figure 5(b) shows the angle distribution of the first input channel, which is denoted as $\{\theta_{i,1}|i = 1, 2, .., 64\}$ before. We can find that one single input feature is repeatedly applied with different RotateConv angles, which can be explained that one feature map always contains various patterns and the later operations need respectively select these patterns for further processing.

V. Conclusion and Future Work

The aim of RotateConv is to reduce the number of parameters by using extremely simple kernel shape as line segment. In order to maintain the kernels' modeling capacities, we equip them with the rotatable ability which helps learn diverse features of different angles. The rotatable ability is achieved by using inverse-interpolation which makes angles continuous, differentiable and learnable. We design two variants of RotateConv which only have 2 and 4 parameters respectively. The difference between these two variants is whether using weight sharing. In experiments, we firstly design a simple convolutional network and apply it on the $MNSIT$ dataset for quick verification. We then use three networks, $ResNet20$, $ResNet44$ and $DenseNet40$, and three datasets, $CIFAR10$, $CIFAR100$ and $SVHN$, for further verification. At last, we analyze the kernel angle distributions for better understanding.

RotateConv is a model compression method which needs more contributions. In the future, we will devote to the following three problems. Firstly, RotateConv should be validated on more large scale datasets like ImageNet[29] and COCO[30]. Secondly, it needs more exhaustive analysis on the relationship between the network depth and the kernel complexity. For example, as deeper layer processes more global information, whether it should be guaranteed with more complex kernel shapes deserves further research. Last, RotateConv is a model compression method but not an acceleration one. RotateConv is achieved by inverse-interpolation, but inverse-interpolation brings RotateConv kernels back to 3×3 kernels for computation, which does not reduce computation time. We hope more efforts could be devoted to RotateConv and to make further progress in model compression and acceleration.

Acknowledgment

This work is jointly supported by National Key Research and Development Program of China (2016YFB1001000), National Natural Science Foundation of China (61525306, 61633021, 61721004, 61420106015), Beijing Natural Science Foundation (4162058), and Capital Science and Technology Leading Talent Training Project (Z181100006318030).

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv preprint arXiv:1606.00915*, 2016.

[5] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision*. Springer, 2016, pp. 21–37.

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

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[11] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[13] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating deep convolutional networks using low-precision and sparsity," *arXiv preprint arXiv:1610.00324*, 2016.

[14] D. Gabor, "Theory of communication. part 1: The analysis of information," *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429–441, 1946.

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

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

[17] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[18] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," *arXiv preprint arXiv:1703.06211*, 2017.

[19] J. Ma, W. Wang, and L. Wang, "Irregular convolutional neural networks," *arXiv preprint arXiv:1706.07966*, 2017.

[20] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2554–2564.

[21] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

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

[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*, 2016, pp. 4820–4828.

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

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

[26] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision*. Springer, 2014, pp. 740–755.