Convolution of Convolution: Let Kernels Spatially Collaborate

Rongzhen Zhao  Jian Li  Zhenzhi Wu
Lynxi Technologies
Beijing, China, 100097
{rongzhen.zhao, jian.li, zhenzhi.wu}@lynxi.com

Abstract

In the biological visual pathway especially the retina, neurons are tiled along spatial dimensions with the electrical coupling as their local association, while in a convolution layer, kernels are placed along the channel dimension singly. We propose convolution of convolution, associating kernels in a layer and letting them collaborate spatially. With this method, a layer can provide feature maps with extra transformations and learn its kernels together instead of isolatedly. It is only used during training, bringing in negligible extra costs; then it can be re-parameterized to common convolution before testing, boosting performance gratuitously in tasks like classification, detection and segmentation. Our method works even better when larger receptive fields are demanded. The code is available on site: https://github.com/Genera1Z/ConvolutionOfConvolution.

1. Introduction

In the most recent decades, deep learning methods have been greatly promoting the performance of algorithms on various computation vision (CV) tasks. Particularly, the convolution operation in convolution neural networks (CNNs) is of great importance because of its powerful capability in feature extraction.

For gains in performance or efficiency, various ways have been tried to improve the convolution operation. The very early efforts are light convolution, by lowering connectivities in the channel [1, 10, 30] or space [19, 31, 33], or both [34]. The following trials are increasing the freedom of kernel shape or value [5, 17, 42]. The most recent are dynamic weights generated by the inputs [18, 27, 40]. Some draw attention or multi-scale into convolution [4, 9, 12, 22, 39], which are more like blocks.

In modeling the retina and subsequent visual pathway, as shown in Fig. 1 [11, 36], these methods are no different from the standard convolution: different populations of neurons are modeled as different convolution layers; connections between populations, i.e., chemical synapses, are modeled as weights connecting different layers; different neuron types within a population are modeled by different kernels within a layer, while neurons of the same type are modeled by one same kernel shared over spatial dimensions; the electrical synapses among close neurons of different types have not been realized yet. We address this by employing the spatial association (vertical red arrows on the left) on kernels within a layer.

The electrical synapses provide the electrical coupling effect that neural signals are transduced instantly in local areas. And such an effect also plays an important role in coordinating neighboring neurons to perform visual perception all together [11, 36], which we believe should not be ignored in CNNs’ implementation.

The electrical synapses among close neurons of the same type, in current methods, are implicated in the spatial overlap of those neighboring convolution sliding windows (horizontal red arrows on Fig. 1 left) of one single kernel within a layer; the electrical synapses among close neurons of dif-
different types have not been realized yet.

Inspired by this, we propose the method “convolution of convolution” (CoC), where spatial associations among kernels (vertical red arrows on Fig. 1 left) within a layer are employed to let them collaborate spatially. It can seamlessly replace current kinds of convolution layers. It only brings in negligible extra costs during training, then can be re-parameterized to the original convolution version once finished, which gives various networks re-built of CoC gratuitous performance gains during testing.

Our contributions are:
(1) Proposing method CoC, opening up a new way of thinking for other works to follow up;
(2) Realizing an association to let kernels collaborate spatially for better feature extraction;
(3) Conducting detailed ablation studies on how hyper-parameters affect CoC’s performance;
(4) Evaluating CoC on various backbones and vision tasks to demonstrate its superiority.

2. Related Works

Existing convolution techniques can be roughly categorized to ones lowering connectivity, ones adjusting statistics and ones liberalizing shape or value.

“Lonely” Kernels

Works lowering kernels’ connectivity do not provide constraints that joint kernels. The ingeniously handcrafted topology of light convolutions, like GWC, CWC, 1D-Conv, PSConv and MixConv [1, 10, 19, 33, 34], provides no association among kernels in a layer. It is the same with those sparse convolutions based on L0 regularization or pruning techniques, such as SSL and DeepR [3, 37].

Works liberalizing kernels’ shape or value pay no attention to the association. Neither DeformConv. ActiveConv [5, 17, 42] or alike, which deform shape and modulate weights by extra feature maps, nor WeightNet, DyNet, In-
volution [18, 27, 40] and so on, which dynamically generate weights by inputs, focus on other aspects like the association, except the liberalization of convolution.

Associated Kernels

Some of those works that adjust kernels’ statistics indeed take into account how to learn kernels together. Representative ones, including SO, SN and OCNN [2, 26, 35], supervise kernels in a layer to converge to orthogonal states with extra loss, such that kernels are diverse and weights are made full use. Works standardizing weights by normalizing and/or centering, for instance, WN, CWN and WS [16, 28, 29], are not necessarily doing the association, but worth referring to.

Association vs. Diversity

That we use another convolution to associate kernels in a convolution usually do harm to kernel’s diversity due to convolution’s smoothing effect and linear correlation.

The association and diversity are thus a pair of contradictions. Typical solutions include skip connections and dilation [14, 20, 38]. The aforementioned standardization and orthogonalization are also possible choices.

Re-Parameterization

Works of re-parameterization [6–8] may not have much to do with the topic of association, but their characteristic use-in-train-fuse-in-test is worth learning from – You just pay a price for the performance in training then enjoy the benefits in testing without any loss of efficiency.

3. Proposed Method

Our method Convolution of Convolution (CoC) is firstly presented and then analyzed mathematically, followed by implementation details.

3.1. Convolution of Convolution

Two definitions here – Basic Convolution: corresponding to a standard convolution; Super Convolution: extra convolution imposing spatial association on the basic convolution kernels.
which is shared over the “batch” dimension $c_i$ times. (3) Re-arrange the “output features” back and spatially associated kernels are obtained. (4) Lastly, common convolution can be computed with these kernels.

During training, steps (1˜3) are executed at every iteration to keep the associated kernels up-to-date; everything else is no different from the common case. During testing, steps (1˜3) are calculated in advance, i.e., re-parameterizing CoC back to common convolution, so that the model can enjoy performance gains gratuitously.

Please do not confuse our dividing kernels into groups with the grouped convolution technique. Also DO NOT mix up our re-arranging each group along width and height with increasing the kernel size manyfold.

How CoC exactly works in the forward and backward propagation? The mathematical analyses are followed.

**Forward: Provide Extra Transformations among the Output Channels**

In the forward propagation, CoC’s spatial association is equivalent to providing extra transformations among feature’s output channels.

Suppose a CoC layer with basic kernels $b^0$ and super kernels $s_i$; the “re-arrange-into” operation is denoted as $\xi$ in Eq. 2 and “re-arrange-back” is $\xi^{-1}$; input and output features are $f_i$ and $f_o$, respectively.

As shown in Fig. 3 the upper, a CoC operation following steps (0˜4) mentioned above can be formulated as

$$f_o = F_{coc}(f_i) = \xi^{-1}(s * \xi(b^0)) * f_i \quad (1)$$

where $s * \xi(b^0)$ is the step (2) super convolution, and $\xi^{-1}(s * \xi(b^0))$ is steps (1˜3), i.e. spatial association.

As shown in Fig. 3 the lower, given CoC super convolution is linear, the CoC operation can be equivalent to

$$f_o = F_{coc}(f_i) = \xi^{-1}(s * \xi(b^0 * f_i)) \quad (2)$$

where $b^0 * f_i$ is actually the common convolution

$$f_o = F_{sta}(f_i) = b^0 * f_i \quad (3)$$

and the remaining part in Eq. (2) means the extra transformation among the output channels.

Note: the extra transformation is what common convolutions do not have, and more importantly, it can be re-parameterized as common convolution after training so its computation burden will not present during testing.

Further, the extra transformation in Eq. (2) takes place in every “super sliding” window, where four neighboring common sliding windows share sides, as shown in Fig. 3 the cyan box. So Eq. (2) can be re-formulated as

$$f_o = \theta(\xi^{-1}(s * \xi(b^0 * f_i^{u,v}))) \quad (4)$$

for all possible $u$ and $v$, where $\theta(\cdot)$ is the super sliding operation, and $f_i^{u,v}$ is the input feature patches in the $(u,v)$th super sliding window; $f_o^{u,v}$ is the output super patch:

$$f_o^{u,v} = \xi^{-1}(s * \xi(b^0 * f_i^{u,v}))$$

$$= \xi^{-1}(s * \xi(k_1 * p_{2a}, k_2 * p_{1b}, k_3 * p_{4a}, k_4 * p_{4d}))$$

$$= \xi^{-1}(s * \xi(p_{2a}, p_{1b}, p_{3c}, p_{4d}))$$

$$= \xi^{-1}(s * \xi(f_b^{u,v})) \quad (5)$$

where operator $\circ$ is a convolution performed within each sub-patch instead of the super patch; $k_1$-$k_4$ are the basic kernels $b^0$; $p_{2a}$-$p_{4d}$ are sub-patches composing the super patch $f_i^{u,v}$; $p_{2a}$-$p_{4d}$ consist of $f_b^{u,v}$, as shown in Fig. 4.

Figure 3. The upper is a CoC layer following steps (0˜4), where the super convolution is done the first; the lower is its equivalence, where the super convolution is done the last, i.e., extra transformations are provided among the output channels. Here basic kernels are $b^0$ and super kernels are $s_i$; the “re-arrange-into” and “re-arrange-back” operations are $\xi$ and $\xi^{-1}$; input and output features are $f_i$ and $f_o$.

Figure 4. The meaning of the extra transformation that Eq. (5) describes. For the current super sliding window, (1) Re-arrange sub-patches $p_{2a}$-$p_{4d}$ into a super patch; (2) Convolve the super patch with the super kernel; (3) Re-arrange the returned patch back to the original places. Here $f_i^{u,v}$ is the super patch out of $f_i$, where $p_{2a}$-$p_{4d}$ are their sub-patches. Note: the padding-zero routine is replaced by padding the feature contents surrounding current super sliding window.
patches $p_0^a$, $p_0^b$, $p_0^c$ and $p_0^d$ are re-arranged into a cross-channel super patch by $\xi(\cdot)$; (2) this super patch is then convolved by $s$, producing a new super patch, which is of the same size; (3) the new super patch is re-arranged back by $\xi^{-1}(\cdot)$ to the original shape.

Proof of Eq. (2). For the $(u, v)$th super sliding,

$$
\begin{align*}
    f^u v_0 &= \xi^{-1}(s \ast (b^u + f^u v)) \\
    &= \xi^{-1}(s \ast \xi(\{k_1^a, k_2^a, k_3^a, k_4^a, k_1^b, k_3^b, k_4^b, k_2^b, k_4^c, k_4^d\})) \\
    &= \xi^{-1}(s \ast \left[\begin{array}{c}
    \xi\left[\begin{array}{c}
    k_1^a + k_2^a + k_3^a + k_4^a \\
    k_3^a + k_4^a \\
    k_4^b + k_2^b \\
    k_3^b + k_4^b \\
    k_2^c + k_4^c \\
    k_3^c + k_4^c \\
    k_4^d + k_2^d \\
    k_3^d + k_4^d
\end{array}\right] \\
    \end{array}\right] + \left[\begin{array}{c}
    k_1 + 0 \\
    k_2 + 0 \\
    0 \\
    0
\end{array}\right]) \\
    &= \xi^{-1}(s \ast \left[\begin{array}{c}
    \xi\left[\begin{array}{c}
    p_0^a \\
    p_0^b \\
    p_0^c \\
    p_0^d
\end{array}\right] + \left[\begin{array}{c}
    0 \\
    0 \\
    0 \\
    p_0^d
\end{array}\right])
\end{align*}
$$

(6)

where $\circ$ is a convolution operated in each sub-patch instead of the super patch, which means it is linear to both $s \ast \square$ and $\xi^{-1}(\cdot)$ and thus can be moved out and reduced to $\ast$.

Substitute Eq. (6) into Eq. (4) and the proof is done.

Backward: Learn Kernels within a Layer by Referring to One Another

In the backward propagation, CoC’s association makes kernels learnt by referring to each other.

Suppose two layers conv1 and conv2 and their input, intermediate and output features $x$, $y$ and $z$, and the gradient accumulated to features $y$ is denoted as $G$.

For common convolution in Fig. 5 the upper, according to the chain rule, kernel $k_1$’s gradient in layer conv1 is

$$
\begin{align*}
    g_1 &= G \times \frac{\partial y}{\partial k_1} \\
    &= G \times x
\end{align*}
$$

(7)

where $G$ is the accumulated gradient, dependent on current training examples and the model’s weights, of course including $k_1^T k_4$ in conv1, which empirically contribute minor to $G$; $\partial y / \partial k_1$ is the derivative of conv1’s output $k_1$, which is independent of conv1’s other kernels. Briefly, common convolution kernels are roughly learnt solely.

For CoC drawn in Fig. 5 the lower, denote the aforementioned spatial association as $\alpha(\cdot) = \xi^{-1}(s \ast \xi(\cdot))$. Then $k_1$’s gradient in conv1 is

$$
\begin{align*}
    g_1 &= G \times \frac{\partial y}{\partial k_1} \\
    &= G \times \frac{\partial (k_1^a + k_2^a + k_3^a + k_4^a)}{\partial k_1} + \frac{\partial (k_1^b + k_2^b + k_3^b + k_4^b)}{\partial k_1} + \frac{\partial (k_1^c + k_2^c + k_3^c + k_4^c)}{\partial k_1} + \frac{\partial (k_1^d + k_2^d + k_3^d + k_4^d)}{\partial k_1} \\
    &= G \times \alpha(k_1; k_2; k_3; k_4) \ast x
\end{align*}
$$

(8)

where $k_1' = \alpha(k_1; k_2; k_3; k_4)$ is the $i$th associated kernel, of which both the value and gradient are explicitly dependent on other kernels in this layer. Briefly, CoC kernels are learnt by referring to one another.

3.2. Implementation Details

The naive implementation is shown in Fig. 6. The most tricky part is step (2) “super-convolve”: each of the $c_0/4$ “input features” is convolved by a specific super kernel, whose weights are shared $c_1$ times. Note the lines marked by XXX.

```python
# __init__
c0 = out_channels // 2 ** 2
weight1 = Tensor(out_channels, in_channels, *kernel_size)
co , ci, kh, kw = weight1.shape

# create super conv
c0 // 2 ** 2
weight1 = Tensor(out_channels, in_channels, *kernel_size)
co , ci, kh, kw = weight1.shape

# rearrange basic kernels for spatial association
weight2 = rearrange(weight1, (co, 2, 2) ci kh kw) -> (co, ci (2 kh) (2 kw))
weight3 = weight2.permute(1, 0, 2, 3) XXX

# forward
weight4 = self.conv2d(x, weight6, self.bias, stride, padding, dilation)
weight5 = rearrange(weight4, (co, ci (2 kh) (2 kw)) -> (co, 2, 2) ci kh kw)
```

Figure 6. Pseudo-code of CoC during training. For testing, step #2/3 is moved into __init__ for re-parameterization.

Configurable Hyper-Parameters
For optimization, some hyper-parameters or auxiliary techniques need to consider, as shown in Fig. 7 the left.

(1) Spatial association $sa$: the number (square root) of kernels grouped together for association. E.g., if $sa=2$ then four kernels are associated together;

(2) Super kernel plus $skp$: how much super kernel size expands or dilates upon basic kernel size. Given basic kernel size $3 \times 3$, if $skp=2$ then super kernel size is $5 \times 5$;

(3) Number of super layers $sl$: the number of layers used as the super convolution part;

(4) Association vs. diversity. As discussed in Sec. 2, methods like skip connection $skp$, dilation $dilat$, orthogonalization $orth$ [35], standardization $stnd$ [28] could ensure kernels’ diversity under the association;

(5) Replacing convolution $c1x1$ with convolution layers of kernel size $1 \times 1$ with CoC or not.

Note: Since the association is convolution, points like $skp$, $sl$, $skp$ and $dilat$ should be taken as intrinsic parts of CoC, but $orth$ and $stnd$ are not, because they are actually our competitors yet compatible with CoC.

Please refer to Sec. 4.1 for more information.

4. Experiments

How CoC’s hyper-parameters affect the networks’ performance is expounded here, so that readers could understand our way of thinking better and avoid detours in their further explorations. Then comes the evaluation of our method on typical tasks under the optimal setting of those hyper-parameters.

4.1. Determine Hyper-Parameters

The codebase we use is mmcls\(^1\). The dataset we use is Tiny-ImageNet\(^2\), sub-set of ImageNet with 200 classes and 100k examples, of which the images are resized to $128 \times 128$ from $64 \times 64$. The backbones are ResNet18 and ShuffleNetV2-1x, representing (a) standard and (b) lightweight models respectively. Other training settings are exactly identical. Results are shown in Fig. 7 the right.

Analysis

(1) $sa$: different settings get similar results and $sa=2$ seems to be the best. This is because super convolution can only provide the association within its receptive field, and thus too large $sa$ makes no essential difference. Please refer to Fig. 8 for visual explanation;

(2) $skp$: should be greater than 0, namely, super kernel size had better be larger than basic kernel size, but too large undermines the performance. This can be explained as too large super kernels may correlate the basic weights overly. Please also refer to Fig. 8;

(3) $sl$: not the larger the better. Considering the gap between train/val accuracy in our experiments, the more super layers there are, the easier it gets over-fitting;

\(^1\)https://github.com/open-mmlab/mmclassification
\(^2\)http://cs231n.stanford.edu/tiny-imagenet-200.zip
Figure 8. Analysis of sa, skp & dilat. Suppose basic kernel (green/blue/purple) size is 3×3; super kernel (orange) size is dependent on skp. **Left**: if skp=0, the associated area is 3×3 at most, so it “saturates” when sa>2. **Center**: if skp=2, the associated area is 5×5, and it saturates when sa>3. Besides, when skp>0 and dilat=false, the super kernels linearly correlate almost every element in the basic kernels, which harms kernels’ diversity. **Right**: if dilat is enabled, the over-correlation will be effective alleviated.

(4, 5) **skp, dilat**: always beneficial. They indeed alleviate over-association and retain kernels’ diversity;

(6) **orth** [35]: always beneficial, especially for standard models. This is due to that orthogonalization indeed makes full use of models’ weights;

(7) **stnd** [28]: good for standard models but bad for light models. The reason could be that light models have limited weights thus need some outliers to enrich kernel diversity, which is suppressed by weight standardization;

(8) **c1x1**: replacing conv1x1 with CoC is harmful because conv1x1 is originally designed for channel projection rather than spatial transformation.

**Summary**

First of all, our CoC is even compatible with its competitors like orth and stnd, and joint use of CoC with them would likely create more benefits.

The optimal setting of our CoC is: sa=2, skp=2, sl=1, skp=true, dilat=true and c1x1=false.

For standard models, it is better to use CoC and orthogonalization together; yet for light-weight models, it is quite necessary to discard standardization.

### 4.2. Evaluate on Typical Tasks

Under the optimal setting mentioned above, we evaluate our CoC on multiple typical vision tasks. We carry out these experiments all under their widely recognized settings without special customization.

**Image Classification**

The codebase we use is mmcls. The dataset is ImageNet [3] and input size is 224×224. The backbones are ResNet18/50, HRNet18/18small, MobileNetV3small and ShuffleNetV2-1x [14, 15, 25, 32], representing mainstream architectures, i.e. (1) **simple-feed-forward**, (2) **multi-branch-interact** and (3) **light-weight** respectively.

For **simple feed-forward** and **multi-branch-interact** models, the optimizer is SGD with lr=0.1, nesterov=true, momentum=0.9, and weight-decay=1e-4; The learning-rate decays in “step” mode of ratio=0.1 at epoches #30/60/90, and the maximum number of epoches is 100.

For **light-weight** models, the optimizer is SGD with lr=0.5, momentum=0.9, weight-decay=4e-5; The learning-rate decays in “poly” mode of power=0.9 and min lr=1e-4, and the maximum number of epoches is 300.

All of these models are trained at batch-size=64 on four GPUs of type RTX3090, with identical data augmentation, i.e., resize-crop and random-flip.

According to results Tab. 1 #1/2/3/4 and #9/10/11/12, exclusive use of CoC can surely improve the performance but the advantage is not obvious, just 0.2’0.6%. But according to #2/3/5, #7/8, #10/11/13 and #15/16, CoC with orth or stnd can reach the effect of “one plus one greater than two”, up to 0.8’1.2% of performance gain. Therefore, our method is competitive to some degree if over-association is overcome and kernel diversity is ensured.

**Object Detection & Instance Segmentation**

The codebase we use is mmdet [4]. The dataset is COCO 2017 [5] and input size is 1333×800. The detection model is RetinaNet-ResNet50 [21], and the instance segmentation model is MaskRCNN-HRNet18 [13]. The pretrained weights are loaded from the above classification tasks, i.e. Tab. 1 #7/15.

---


4https://github.com/open-mmlab/mmdetection

5https://cocodataset.org/#detection-2017

![Image of analysis](image.png)
Table 2. Object detection & instance segmentation results on COCO2017/val.

<table>
<thead>
<tr>
<th>#</th>
<th>network</th>
<th>mAP</th>
<th>mAP(_{50})</th>
<th>mAP(_{75})</th>
<th>mAP(_{S})</th>
<th>mAP(_{M})</th>
<th>mAP(_{L})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RetinaNet-r50</td>
<td>36.3</td>
<td>55.1</td>
<td>38.8</td>
<td>20.1</td>
<td>40.1</td>
<td>47.8</td>
</tr>
<tr>
<td>2</td>
<td>RetinaNet-r50-stnd</td>
<td>37.9</td>
<td>56.9</td>
<td>40.6</td>
<td>21.3</td>
<td>42.1</td>
<td>50.0</td>
</tr>
<tr>
<td>3</td>
<td>RetinaNet-r50-orth</td>
<td>37.6</td>
<td>56.4</td>
<td>40.7</td>
<td>21.0</td>
<td>41.9</td>
<td>49.9</td>
</tr>
<tr>
<td>4</td>
<td>RetinaNet-r50-coc</td>
<td>38.1</td>
<td>57.0</td>
<td>40.8</td>
<td>21.5</td>
<td>42.0</td>
<td>50.1</td>
</tr>
<tr>
<td>5</td>
<td>MaskRCNN-hr18</td>
<td>33.9</td>
<td>54.3</td>
<td>36.3</td>
<td>18.9</td>
<td>36.4</td>
<td>45.8</td>
</tr>
<tr>
<td>6</td>
<td>MaskRCNN-hr18-stnd</td>
<td>34.9</td>
<td>56.0</td>
<td>37.4</td>
<td>19.9</td>
<td>37.9</td>
<td>47.2</td>
</tr>
<tr>
<td>7</td>
<td>MaskRCNN-hr18-orth</td>
<td>35.1</td>
<td>55.8</td>
<td>37.9</td>
<td>20.0</td>
<td>37.6</td>
<td>47.5</td>
</tr>
<tr>
<td>8</td>
<td>MaskRCNN-hr18-coc</td>
<td>35.6</td>
<td>56.2</td>
<td>38.2</td>
<td>20.3</td>
<td>38.1</td>
<td>47.8</td>
</tr>
</tbody>
</table>

For both object detection and instance segmentation, the optimizer is SGD with lr=0.01, momentum=0.9 and weight-decay=1e-4; The learning-rate decays in “step” mode of ratio=0.1 at epochs #8/11, and the maximum number of epochs is 12.

All of these models are trained at batch-size=4 on four GPUs of type RTX3090, with identical data augmentation, i.e., random-flip only.

According to Tab. 2, the larger objects or instances are, the better our method performs. Our method improves the detection of small/medium/large objects by 1.4/1.9/2.3 mAP respectively, and improves the segmentation of small/medium/large instances by 1.4/1.7/2.0 mAP respectively.

Semantic Segmentation

The codebase we use is mmseg\(^6\). The dataset is Pascal VOC 0712\(^7\) and input size is 512×512. The models are FCN-R50d8\([23]\) and HRNet-W18\([32]\). The pretrained weights are loaded from the above classification tasks, i.e. Tab. 1 #7/15.

The optimizer is SGD with lr=0.01, momentum=0.9, weight-decay=5e-4; The learning-rate decays in “poly” mode of power=0.9 and min-lr=1e-4, and the maximum number of iters is 20k.

All of these models are trained at batch-size=4 on four GPUs of type RTX3090, with identical data augmentation, i.e., random-crop, random-flip and photo-metric-distortion.

According to Tab. 3, our CoC always improves their performance. The mIoU and mAcc of both FCN and HRNet get nearly 3.0- and 3.5-points’ promotion respectively.

5. Discussions

What Kernels Are Learnt Under Our Spatial Association?

We visualize ResNet’s first convolution kernels, following\([1]\), to intuitively understand what kernels are learnt under our spatial association.

We choose models of ResNet50 and its CoC variant, corresponding to Tab. 1 #6/7. For a consistent comparison, these kernels are normalized by \((w - w.\text{min})/(w.\text{max} - w.\text{min})\), where \(w\) is the kernels. Then kernels learnt in CoC...
are drawn in the unit of the spatial association group, as shown in Fig. 9 the right – every four kernels are tiled along the width and height, just like how they were spatially associated, and the left corner is their super kernel who produced them. The common convolution kernels, as shown in Fig. 9 the left, are drawn in a similar way.

Figure 9. First layers’ kernels of ResNet50 (the left) and its CoC variant (the right). On the right, every four is an association, at which the top left corner is their super kernel.

According to Fig. 9, in the common convolution of ResNet50, the patterns of different styles, e.g., grey vs. color, stripe vs. plane, scatter among these 64 channels irregularly. By contrast, the patterns learnt in CoC are always similar but complementary within each spatial association group. Besides, the spatial distribution of the four sub-patterns within a group clearly echoes their super kernel’s pattern.

Specially, in the red box, the two patterns in the first column are stripes that are shade-light-shade and light-shade-light respectively; the second column has similar looks. From another perspective, this group is vertically symmetric, and their super kernel’s pattern is vertically symmetric too. In the purple box, the sub-patterns are colorful stripes, grey stripes, grey grids and a colorful plane, which somehow “breaks” the aforementioned law that intra-group patterns have similar styles; however, these four sub-patterns possess the symmetry along the main diagonal, and so is their super kernel. These reflect the spatial collaboration that we claim.

Why It’s Relatively Better When Larger Receptive Fields Needed?

This phenomenon suggests that our method offers larger effective receptive fields (RF), which can be visually proven by the class activation map (CAM) [41] technique.

We choose models of ResNet50 and its CoC variant, corresponding to Tab. 1 #6/7, where the former’s theoretical RF (TRF) is 427 and effective RF (ERF) is empirically $1/4 \sim 1/3$ [24], as shown in Fig. 10 the pink squares. Images that contain objects of different scales are selected from VOC12 test set, and are resized and padded into $1024 \times 768$, so that there are object scales both within and beyond the models’ ERF, as shown in Fig. 10 the original images. Tool [torch-cam](https://github.com/frgfm/torch-cam) is used to extract CAMs, where the top 10 class maps are fused together via maximum to cover the quasi-correct classifications, as shown in Fig. 10 the right two columns.

According to Fig. 10, ResNet50 fully perceives small objects that fall into its ERF while partially perceives the large that exceed its ERF; our ResNet50-CoC works much better as shown in Fig. 10 due to its larger ERF and better feature extraction.

Figure 10. CAM of standard ResNet50 (2nd column) and its CoC variant (3rd column).

Specifically, the small boat or cow, which are within standard ResNet50’s effective RF, are fully perceived; but for the large boat or cow, which are beyond ResNet50’s effective RF, only the top and right angles of the large boat or the right horn and eye are perceived. By contrast, our ResNet50- CoC activates the most area of the large boat or almost all the head region of the large cow, reflecting its larger effective RF.

6. Conclusion

A novel method named “convolution of convolution” (CoC) is proposed and explored. It can seamlessly replace current convolution operations and significantly improve models’ capability of extracting spatial patterns with its larger effective receptive field. The ablation study shows how to use such a technique and the evaluation on typical tasks presents how well it may improve current models. It works in training and can be re-parameterized before testing so that considerable performance gains are obtained with no efficiency loss in deployment.

As for future works, we believe similar ways of thinking are also worth exploring: spatial/channel attention for pruning and cross-layer connections among kernels rather than activation features, etc.

Acknowledgement This work was supported by Beijing Science & Technology Plan, China (Z191100007519009) and Science & Technology Innovation 2030 - Key Project of “New Generation of AI” (2020AAA0109100).
References


[27] Jiawei Huang Ningning Ma, Xiangyu Zhang and Jian Sun. Weightnet: Revisiting the design space of weight networks. In Advances in Neural Information Processing Systems (NeurIPS), 2020. 1


