SORT: Second-Order Response Transform for Visual Recognition

Yan Wang¹, Lingxi Xie²(✉), Chenxi Liu², Siyuan Qiao²
Ya Zhang¹(✉), Wenjun Zhang¹, Qi Tian³, Alan Yuille²

¹Cooperative Medianet Innovation Center, Shanghai Jiao Tong University, Shanghai, China
²Department of Computer Science, The Johns Hopkins University, Baltimore, MD, USA
³Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX, USA

tiffany940107@gmail.com 198808xc@gmail.com {cxliu,siyuan.qiao}@jhu.edu
{ya_zhang,zhangwenjun}@sjtu.edu.cn qitian@cs.utsa.edu alan.l.yuille@gmail.com

Abstract

In this paper, we reveal the importance and benefits of introducing second-order operations into deep neural networks. We propose a novel approach named Second-Order Response Transform (SORT), which appends element-wise product transform to the linear sum of a two-branch network module. A direct advantage of SORT is to facilitate cross-branch response propagation, so that each branch can update its weights based on the current status of the other branch. Moreover, SORT augments the family of transform operations and increases the nonlinearity of the network, making it possible to learn flexible functions to fit the complicated distribution of feature space. SORT can be applied to a wide range of network architectures, including a branched variant of a chain-styled network and a residual network, with very lightweight modifications. We observe consistent accuracy gain on both small (CIFAR10, CIFAR100 and SVHN) and big (ILSVRC2012) datasets. In addition, SORT is very efficient, as the extra computation overhead is less than 5%.

1. Introduction

Deep neural networks [27][46][50][16] have become the state-of-the-art systems for visual recognition. Supported by large-scale labeled datasets such as ImageNet [5] and powerful computational resources like modern GPUs, it is possible to train a hierarchical structure to capture different levels of visual patterns. Deep networks are also capable of generating transferrable features for different vision tasks such as image classification [6] and instance retrieval [42], or fine-tuned to deal with a wide range of challenges, including object detection [10][43], semantic segmentation [50][2], boundary detection [45][58], etc.

The past years have witnessed an evolution in designing efficient network architectures, in which the chain-styled modules have been extended to multi-path modules [50] or residual modules [16]. Meanwhile, highway inter-layer connections are verified helpful in training very deep networks [48]. In the previous literatures, these connections are fused in a linear manner, i.e., the neural responses of two branches are element-wise summed up as the output. This limits the ability of a deep network to fit the complicated distribution of feature space, as nonlinearity forms the main contribution to the network capacity [23]. This motivates us to consider higher-order transform operations.

\[ y^R = F_1(x) + F_2(x) \]

\[ y^S = F_1(x) + F_2(x) + F_1(x) \odot F_2(x) \]

\[ y^{R,S} = x + F(x) \]

Figure 1. Two types of modules and the corresponding SORT operations. Left: in a two-branch convolutional block, the two-way outputs, \( F_1(x) \) and \( F_2(x) \), are combined with a second-order transform \( F_1(x) + F_2(x) + F_1(x) \odot F_2(x) \). Right: in a residual-learning building block [16], we can also modify the fusion stage from \( x + F(x) \) to \( x + F(x) + \sqrt{x \odot F(x)} \). Here, \( \odot \) denotes element-wise product, and \( \sqrt{\cdot} \) denotes element-wise square-root.
In this paper, we propose Second-Order Response Transform (SORT), an efficient approach that applies to a wide range of visual recognition tasks. The core idea of SORT is to append a dyadic second-order operation, say element-wise product, to the original linear sum of two-branch vectors. This modification, as shown in Figure 1, brings two-fold benefits. First, SORT facilitates cross-branch information propagation, which rewards consistent responses in forward-propagation, and enables each branch to update its weights based on the current status of the other branch in back-propagation. Second, the nonlinearity of the module becomes stronger, which allows the network to fit more complicated feature distribution. In addition, adding such operations is very cheap, as it requires less than 5% extra time, and no extra memory consumptions. We apply SORT to both deep chain-styled networks and deep residual networks, and verify consistent accuracy gain its weights based on the current status of the other branch in back-propagation. The core idea of SORT also generates more effective deep features to boost the transfer learning performance.

The remainder of this paper is organized as follows. Section 2 briefly reviews related work, and Section 3 illustrates the SORT algorithm and some analyses. Experiments are shown in Section 4 and conclusions are drawn in Section 5.

2. Related Work

2.1. Convolutional Neural Networks

The Convolutional Neural Network (CNN) is a hierarchical model for visual recognition. It is based on the observation that a deep network with enough neurons is able to fit any complicated data distribution. In past years, neural networks were shown effective for simple recognition tasks. More recently, the availability of large-scale training data (e.g., ImageNet) and powerful GPUs make it possible to train deep architectures which significantly outperform the conventional Bag-of-Visual-Words and deformable part models. A CNN is composed of several stacked layers. In each of them, responses from the previous layer are convolved with a filter bank and activated by a differentiable non-linearity. Hence, a CNN can be considered as a composite function, which is trained by back-propagating error signals defined by the difference between supervision and prediction at the top layer. Recently, efficient methods were proposed to help CNNs converge faster and prevent over-fitting, such as ReLU activation Dropout, batch normalization and varying network depth in training. It is believed that deeper networks have stronger ability of visual recognition, but at the same time, deeper networks are often more difficult to be trained efficiently.

An intriguing property of the CNN lies in its transfer ability. The intermediate responses of CNNs can be used as effective image descriptors, and widely applied to various types of vision applications, including image classification and instance retrieval. Also, deep networks pre-trained on a large dataset can be fine-tuned to deal with other tasks, including object detection, semantic segmentation, boundary detection, etc.

2.2. Multi-Branch Network Connections

Beyond the conventional chain-styled networks, it is observed that adding some sideline connections can increase the representation ability of the network. Typical examples include the inception module, in which neural response generated by different kernels are concatenated to convey multi-scale visual information. Meanwhile, the benefit of identity mapping motivates researchers to explore networks with residual connections. These efforts can be explained as the pursuit of building highway connections to prevent gradient vanishing and/or explosion in training very deep networks.

Another family of multi-branch networks follow the bilinear CNN model, which constructs two separate streams to model the co-occurrence of local features. Formulated as the outer-product of two vectors, it requires a larger number of parameters and more computational resources than the conventional models to be trained. An alternative approach is proposed to factorize bilinear models for visual recognition, which largely decreases the number of trainable parameters.

All the multi-branch structures are followed by a module to fuse different sources of features. This can be done by linearly summing them up, concatenating them, or using a bilinear or recurrent transform. In this work, we present an extremely simple and efficient approach to enable effective feature ensemble, which involves introducing a second-order term to apply nonlinear transform in neural responses. Introducing a second-order operation into neural networks has been studied in some old-fashioned models, but we study this idea in modern deep convolutional networks.

3. Second-Order Response Transform

3.1. Formulation

Let $x$ be a set of neural responses at a given layer of a deep neural network. In practice, $x$ often appears as a 3D volume. In a two-branch network structure, $x$ is fed into two individual modules with different parameters, and two intermediate data cubes are obtained. We denote them as $F_1(x; \theta_1)$ and $F_2(x; \theta_2)$, respectively. In the cases without ambiguity, we write $F_1(x)$ and $F_2(x)$ in short. Most often, $F_1(x)$ and $F_2(x)$ are of the same dimensionality, and an element-wise operation is used to summarize them into the...
output set of responses \( y \).

There are some existing examples of two-branch networks, such as the Maxout network \([13]\) and the deep residual network \([16]\). In Maxout, \( F_1(x) \) and \( F_2(x) \) are generated by two individual convolutional layers, i.e.,
\[
 F_m(x) = \sigma[\theta_m^\top x] \quad \text{for} \quad m = 1, 2, \quad \text{where} \quad \theta_m \quad \text{is the} \quad m\text{-th convolutional matrix}, \quad \sigma[\cdot] \quad \text{is the activation function}, \quad \text{and an element-wise max operation is performed to fuse them:} \quad y^M = \max\{F_1(x), F_2(x)\}. \]

In a residual module, \( F_1(x) \) is simply set as an identity mapping (i.e., \( x \) itself), and \( F_2(x) \) is defined as \( x \) followed by two convolutional operations, i.e.,
\[
 F_2(x) = \theta_2^\top \sigma[\theta_2^\top x], \quad \text{and the fusion is performed as linear sum:} \quad y^R = F_1(x) + F_2(x). \]

The core idea of SORT is extremely simple. We append a second-order term, i.e. element-wise product, to the linear term, leading to a new fusion strategy:
\[
y^S = F_1(x) + F_2(x) + g[F_1(x) \odot F_2(x)]. \tag{1} \]

Here, \( \odot \) denotes element-wise product and \( g[\cdot] \) is a differentiable function. The gradient of \( y^S \) over either \( x \) or \( \theta_m \) \((m = 1, 2)\) is straightforward. Note that this modification is very simple yet light-weighted. Based on a specifically implemented layer in popular deep learning tools such as CAFFE \([24]\), SORT requires less than 5\% additional time in training and testing, meanwhile no extra memory is used.

SORT can be applied to a wide range of network architectures, even if the original structure does not have branches. In this case, we need to modify each of the original convolutional layers, i.e., \( y^O = \sigma[\theta^\top x] \). We construct two symmetric branches \( F_1(x) \) and \( F_2(x) \), in which the \( m\)-th branch is defined as \( F_m(x) = \sigma[\theta_m^\top x] \). Then, we perform element-wise fusion \([1]\) beyond \( F_1(x) \) and \( F_2(x) \) by setting \( g[\cdot] \) to be an identity mapping function. Following the idea to reduce the number of parameters \([46]\), we shrink the receptive field size of each convolutional kernel in \( \theta_m \) from \( k \times k \) to \([\frac{k}{2}(k+1)] \times [\frac{k}{2}(k+1)]\). With two cascaded convolutional layers and \( k \) being an odd number, the overall receptive field size of each neuron in the output layer remains unchanged. As we shall see in experiments, the branched structure works much better than the original structure, and SORT consistently boosts the recognition performance beyond the improved baseline.

Another straightforward application of SORT lies in the family of deep residual networks \([16]\). Note that residual networks are already equipped with two-branch structures, i.e., the input signal \( x \) is followed by an identity mapping and the neural response after two convolutions. As a direct variant of \([1]\), SORT modifies the original fusion function from \( y^R = x + F(x) \) to \( y^S = x + F(x) + \sqrt{x \odot F(x)} + \varepsilon \). Here \( \varepsilon = 10^{-4} \) is a small floating point number to avoid numerical instability in gradient computation. Note that in the residual networks, elements in either \( x \) or \( F(x) \) may be negative \([17]\), and we perform a ReLU activation on it before computing the product term. Thus, the exact form of SORT in this case is \( y^S = x + F(x) + \sqrt{\|x\| \odot \|F(x)\|} + \varepsilon \). Similarly, SORT does not change the receptive field size of an output neuron.

### 3.2. Cross-Branch Response Propagation

We first discuss the second-order term. According to our implementation, all the numbers fed into element-wise product are non-negative, i.e., \( \forall i, F_{1,i}(x) \geq 0 \) and \( F_{2,i}(x) \geq 0 \). Therefore, the second-order term is either 0 or a positive value (when both \( F_{1,i}(x) \) and \( F_{2,i}(x) \) are positive). Consider two input pairs, i.e., \( (F_{1,i}(x), F_{1,i}(x)) = (a, 0) \) or \( (F_{1,i}(x), F_{2,i}(x)) = (a_1, a_2) \) where \( a_1 + a_2 = a \).

In the former case we have \( y^S_i = a \), but in the latter case we have \( y^S_i = a_1 + a_2 \). The extra term, i.e., \( a_1 \times a_2 \), is large when \( a_1 \) and \( a_2 \) are close, i.e., \( |a_1 - a_2| \) is small. We explain it as facilitating the consistent responses, i.e., we reward the indices on which two branches have similar response values.

We also note that SORT leads to an improved way of gradient back-propagation. Since there exists a dyadic term \( F_1(x; \theta_1) \odot F_2(x; \theta_2) \), the gradient of \( y^S \) with respect to either one in \( \theta_1 \) and \( \theta_2 \) is related to another. Thus, when the parameter \( \theta_1 \) needs to be updated, the gradient \( \frac{\partial L}{\partial \theta_1} \) is directly related to \( F_2(x) \):
\[
 \frac{\partial L}{\partial \theta_1} = \left( \frac{\partial L}{\partial y^S} \right)^\top \cdot [1 + F_2(x; \theta_2)]^\top \cdot \frac{\partial F_1(x; \theta_1)}{\partial \theta_1}, \tag{2} \]
and similarly, \( \frac{\partial L}{\partial \theta_2} \) is directly related to \( F_1(x) \). This prevents the gradients from being shattered as the network goes deep \([1]\), and reduces the risk of structural over-fitting (i.e., caused by the increasing number of network layers).

As an example, we train deep residual networks \([16]\) with different numbers of layers on the SVHN dataset \([40]\), a relatively simple dataset for street house number recognition. Detailed experimental settings are illustrated in Section \([11]\). The baseline recognition errors are 2.30\% and 2.49\% for the 20-layer and 56-layer networks, respectively, while these numbers become 2.26\% and 2.19\% after SORT is applied. SORT consistently improves the recognition rate, and the gain becomes more significant when a deeper network architecture is used.

In summary, SORT allows the network to consider cross-branch information in both forward-propagation and back-propagation. This strategy improves the reliability of neural responses, as well as the numerical stability in gradient computation.

### 3.3. Global Network Nonlinearity

Nonlinearity makes the major contribution to the representation ability of deep neural networks \([23]\). State-of-
The second-order operation produces nonlinearity in a 2D subset. Here, \( x_\ast = \max \{ x, 0 \} \) and \( y_\ast = \max \{ y, 0 \} \).

<table>
<thead>
<tr>
<th>+</th>
<th>max</th>
<th>( \odot )</th>
<th>LeNet</th>
<th>BigNet</th>
<th>ResNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.10</td>
<td>6.86</td>
<td>7.60</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.07</td>
<td>7.01</td>
<td>7.55</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.03</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.02</td>
<td>6.90</td>
<td>7.63</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>10.34</td>
<td>6.60</td>
<td>7.14</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>10.39</td>
<td>6.57</td>
<td>7.44</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>10.80</td>
<td>6.65</td>
<td>7.90</td>
</tr>
</tbody>
</table>

Table 1. Recognition error rate (%) on the CIFAR10 dataset with different fusion strategies. Here, +, max and \( \odot \) denote three dyadic operators, and multiple checkmarks in one row mean to sum up the results produced by the corresponding operators.

Sometimes, using the second-order terms alone results in non-convergence (denoted by –). All these numbers are averaged over 3 individual runs, with standard deviations of 0.04%–0.08%.

3.4. Relationship to Other Work

We note that some previous work also proposed to use a second-order term in network training. For example, the bilinear CNN [35] computes the outer-product of neural responses from two individual networks to capture feature co-occurrence at the same spatial positions. However, this operation often requires heavy time and memory overheads, as it largely increases the dimensionality of the feature vector, and consequently the number of trainable parameters. Training a bilinear CNN is often slow, even in the improved versions [9, 33]. In comparison, the extra computation brought by SORT is merely ignorable (< 5%). We evaluate [34] and [9] on the CIFAR10 dataset. Using BigNet* [38] as the backbone (see Section 4.1), the error rates of [35], [9] and SORT are 7.17%, 8.01% and 6.81%, and every 20 iterations take 3.7s, 16.5s and 2.1s, respectively. Compared with the baseline, bilinear pooling requires heavier computation and reports even worse results. This was noted in the original paper [35], which shows that good initialization and careful fine-tuning are required, and therefore it was not designed for training-from-scratch.

In a spatial transformer network [22], the product op-
We preprocess the data as in the previous literature [40], selecting 10,000 images as training samples, and the remaining 10,000 testing samples, and 5,000 extra samples per category from the training set for validation, and the remaining 50,000 images as training samples. We also use local contrast normalization (LCN) for data preprocessing [13].

Four baseline network architectures are evaluated:

- **LeNet** [29] is a relatively shallow network with 3 convolutional layers, 3 pooling layers and 2 fully-connected layers. All the convolutional layers have 5 × 5 kernels, and the input cube is zero-padded by a width of 2 so that the spatial resolution of the output remains unchanged. After each convolution including the first fully-connected layer, a nonlinear function known as ReLU [49] is used for activating the neural responses. This common protocol will be used in all the network structures. The pooling layers have 3 × 3 kernels, and a spatial stride of 2. We apply three training sections with learning rates of 10^{-2}, 10^{-3} and 10^{-4}, and 60K, 5K, and 5K iterations, respectively.

- A so-called **BigNet** is trained as a deeper chain-styled network. There are 10 convolutional layers, 3 pooling layers and 3 fully-connected layers in this architecture. The design of BigNet is similar to VGGNet [46], in which small convolutional kernels (3 × 3) are used and the depth is increased. Following [38], we apply four training sections with learning rates of 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4}, and 60K, 30K, 20K and 10K iterations, respectively.

- The deep residual network (ResNet) [16] brings significant performance boost beyond chain-styled networks. We follow the original work [16] to define network architectures with different numbers of layers, which are denoted as ResNet-20, ResNet-32 and ResNet-56, respectively. These architectures differ from each other in the number of residual blocks used in each stage. Batch normalization is applied after each convolution to avoid numerical instability in this very deep network. Following the implementation of [59], we apply three training sections with learning rates of 10^{-1}, 10^{-2}, and 10^{-3}, and 32K, 16K and 16K iterations, respectively.

- The wide residual network (WRN) [60] takes the idea of increasing the number of kernels in each layer and decrease the network depth at the same time. We apply the 28-layer architecture, denoted as WRN-28, which is verified effective in [60]. Following the same implementation of the original ResNets, we apply three training sections with learning rates of 10^{-1}, 10^{-2} and 10^{-3}, and 32K, 16K, and 16K iterations, respectively.

In all the networks, the mini-batch size is fixed as 100. Note that both LeNet and BigNet are chain-styled networks. Using the details illustrated in Section 3.1, we replace each convolutional layer using a two-branch, two-layer module with smaller kernels. This leads to deeper and more powerful networks, and we append an asterisk (*) after the original networks to denote them. SORT is applied to the modified network structure by appending element-wise product to linear sum.

### 4.1.2 Results

Results are summarized in Table 2. One can observe that SORT boosts the performance of all network architectures...
On the accuracy gain when the network goes deeper. For example, fusion function. In addition, we observe more significant convolutional layer as a two-branch module. SORT further consistently. On both LeNet and BigNet, we observe significant accuracy gain brought by replacing of each convolutional layer as a two-branch module. SORT further improves recognition accuracy by using a more effective fusion function. In addition, we observe more significant accuracy gain when the network goes deeper. For example, on the 20-layer ResNet, the relative error rate drops are 4.79, 0.47% and 1.74% for CIFAR10, CIFAR100 and SVHN, and these numbers become much bigger (12.70%, 5.27% and 12.05%, respectively) on the 56-layer ResNet. This verifies our hypothesis in Section 3.2 that SORT alleviates the shattered gradient problem and helps training very deep networks more efficiently. Especially, based on WRN-28, one of the state-of-the-art structures, SORT reduces the recognition error rate of SVHN from 1.93% to 1.48%, giving a relatively 23.32% error drop, meanwhile achieving the new state-of-the-art (the previous record is 1.59% [19]). All these results suggest the usefulness of the second-order term in visual recognition.

### 4.2.2 Results

The recognition results are summarized in Table 2. All the numbers are reported by one single model. Based on the original chain-styled AlexNet, replacing each convolutional layer as a two-branch module produces 36.71% top-1 and 14.77% top-5 error rates, which is significantly lower than the original version, i.e., 43.19% and 19.87%. This is mainly due to the increase in network depth. SORT further reduces the errors by 0.72% and 0.31 (or 1.96% and 2.10% relatively). On the 18-layer ResNet, the baseline top-1 and top-5 error rates are 34.50% and 13.33%, and SORT...
reduces them to 32.37% and 12.61% (6.17% and 5.71% relative drop, respectively).

On a 4-GPU machine, AlexNet* and ResNet-18 need an average of 10.5s and 19.3s to finish 20 iterations. After SORT is applied, these numbers becomes 10.7s and 19.9s, respectively. Given that only less than 5% extra time and no extra memory are used, we can claim the effectiveness and the efficiency of SORT in large-scale visual recognition.

### 4.2.3 Discussions

We also plot the learning curves of both architectures in Figure 4. Very similar phenomena are observed as in small-scale experiments. On AlexNet* which is the branched version of a chain-styled network, SORT helps the network to be trained better. Meanwhile, on ResNet-18, SORT makes the network more difficult to converge. But nevertheless,
with CAFFE, while ResNetT−50 is implemented with T orch [15].

Table 3. Recognition error rate (%) on the ILSVRC2012 dataset using different network architectures. All the results are reported using one single crop in testing. The ResNet-18 is implemented with CAFFE, while ResNetT’s are implemented with Torch [13].

<table>
<thead>
<tr>
<th>Network</th>
<th>Top-1 Error</th>
<th>Top-5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>43.19</td>
<td>19.87</td>
</tr>
<tr>
<td>AlexNet*</td>
<td>36.71</td>
<td>14.77</td>
</tr>
<tr>
<td>AlexNet* -SORT</td>
<td>35.99</td>
<td>14.46</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>34.50</td>
<td>13.33</td>
</tr>
<tr>
<td>ResNet-18 -SORT</td>
<td>32.37</td>
<td>12.61</td>
</tr>
<tr>
<td>ResNetT-18</td>
<td>30.50</td>
<td>11.07</td>
</tr>
<tr>
<td>ResNetT-18 -SORT</td>
<td>29.95</td>
<td>10.80</td>
</tr>
<tr>
<td>ResNetT-34</td>
<td>27.02</td>
<td>8.77</td>
</tr>
<tr>
<td>ResNetT-34 -SORT</td>
<td>26.57</td>
<td>8.55</td>
</tr>
<tr>
<td>ResNetT-50</td>
<td>24.10</td>
<td>7.11</td>
</tr>
<tr>
<td>ResNetT-50 -SORT</td>
<td>23.82</td>
<td>6.72</td>
</tr>
</tbody>
</table>

Table 4. Classification accuracy (%) on the Caltech256 dataset using deep features extracted from different layers of different network structures.

<table>
<thead>
<tr>
<th>Network</th>
<th>pool-5 (std deviation)</th>
<th>fc-6 (std deviation)</th>
<th>fc-7 (std deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>69.19 ± 0.18</td>
<td>71.51 ± 0.25</td>
<td>69.47 ± 0.11</td>
</tr>
<tr>
<td>AlexNet* (std deviation)</td>
<td>74.20 ± 0.17</td>
<td>76.54 ± 0.30</td>
<td>74.42 ± 0.18</td>
</tr>
<tr>
<td>AlexNet* -SORT (std deviation)</td>
<td>74.88 ± 0.19</td>
<td>77.12 ± 0.24</td>
<td>75.06 ± 0.15</td>
</tr>
</tbody>
</table>

in either cases, SORT improves the representation ability and eventually helps the modified structure achieve better recognition performance.

4.3. Transfer Learning Experiments

We evaluate the transfer ability of the trained models by applying them to other image classification tasks. The Caltech256 [14] dataset is used for generic image classification. We use the AlexNet-based models to extract from the pool-5, fc-6 and fc-7 layers, and adopt ReLU activation to filter out negative responses. The neural responses from the pool-5 layer (6 × 6 × 256) are spatially averaged into a 256-dimensional vector, while the other two layers directly produce 4,096-dimensional feature vectors. We perform square-root normalization followed by ℓ2 normalization, and use LIBLINEAR [7] as an SVM implementation and set the slacking variable C = 10. 60 images per category are left out for training the SVM model, and the remaining ones are used for testing. The average accuracy over all categories is reported. We run 10 individual training/testing splits and report the averaged accuracy as well as the standard deviation. Results are summarized in Table 4. One can observe that the improvement on ILSVRC2012 brought by SORT is able to transfer to Caltech256.

5. Conclusions

In this paper, we propose Second-Order Response Transform (SORT), an extremely simple yet effective approach to improve the representation ability of deep neural networks. SORT summarizes two neural responses by considering both sum and product terms, which leads to efficient information propagation throughout the network and more powerful network nonlinearity. SORT can be applied to a wide range of modern convolutional neural networks, and produce consistent recognition accuracy gain on some popular benchmarks. We also verify the increasing effectiveness of SORT on very deep networks.

In the future, we will investigate the extension of SORT. It remains open problems that whether SORT can be applied to multi-branch networks such as Inception [50], DenseNet [19] and ResNetXt [57], or some other applications such as GANs [12] or LSTMs [18].

Acknowledgements. This work was supported by the High Tech Research and Development Program of China 2015AA015801, NSFC 61521062, STCSM 12DZ2272600, the IARPA via DoI/IBC contract number D16PC00007, and ONR N00014-15-1-2356. We thank Xiang Xiang and Zhuotun Zhu for instructive discussions.
References


