Noname manuscript No.
(will be inserted by the editor)

Towards Reversal-Invariant Image Representation

Lingxi Xie · Jingdong Wang · Weiyao Lin · Bo Zhang · Qi Tian

Received: date / Accepted: date

Abstract State-of-the-art image classification approaches are mainly based on robust image representation, such as the Bag-of-Features (BoF) model or the Convolutional Neural Network (CNN) architecture. In real applications, the **orientation** (left/right) of an image or an object might vary from sample to sample, whereas some handcrafted descriptors (*e.g.*, SIFT) and network operations (*e.g.*, convolution) are not reversal-invariant, leading to the unsatisfied stability of image features extracted from these models. To deal with, a popular solution is to augment the dataset by adding a left-right reversed copy for each image. This strategy improves the recognition accuracy to some extent, but also brings the price of almost doubled time and memory consumptions on both the training and testing stages.

In this paper, we present an alternative solution based on designing reversal-invariant representation of local patterns, so that we can obtain the identical representation for an image and its left-right reversed copy. For the BoF model, we design a reversal-invariant version of SIFT descriptor named **Max-SIFT**, and generalize it to the **RIDE** algorithm which can be applied to a large family of local descriptors. For the CNN architecture, we present a simple idea of generating reversal-invariant deep features (**RI-Deep**), and, inspired by which, design reversal-invariant convolution (**RI-Conv**) layers to enlarge the CNN capacity without increasing the model complexity. Experimental results reveal consistent accuracy on various image classification tasks, including scene understanding, fine-grained object recognition, and large-scale visual recognition.

Keywords Image Classification · The BoF Model · CNN · Reversal-Invariant Image Representation

This work was done when Lingxi Xie was an intern at Microsoft Research. This work is supported by the 973 Program of China 2013CB329403 and 2012CB316301, NSFC 61332007, 61273023, 61429201, 61471235, Tsinghua ISRP 20121088071, ARO grants W911NF-15-1-0290 and W911NF-12-1-0057, and Faculty Research Awards, NEC Lab of America.

L. Xie
Geology Building, University of California, Los Angeles, Los Angeles, CA 90024, USA
E-mail: 198808xc@gmail.com

J. Wang
Building 2, No.5 Danling Street, Haidian District, Beijing 100080, China
E-mail: jingdw@microsoft.com

W. Lin
SEIEE Building, Shanghai Jiao Tong University, Minhang District, Shanghai 200240, China
E-mail: wylin@sjtu.edu.cn

B. Zhang
FIT Building, Tsinghua University, Haidian District, Beijing 100084, China
E-mail: dcszb@mail.tsinghua.edu.cn

Q. Tian (✉)
One UTSA Circle, University of Texas at San Antonio, San Antonio, TX 78249-1604, USA
E-mail: qitian@cs.utsa.edu

1 Introduction

Image classification is a fundamental problem in computer vision which implies a large number of applications. One of the most popular approaches for image classification is the Bag-of-Features (BoF) model (Csurka et al, 2004), a statistics-based algorithm in which local features are extracted, encoded and summarized into global image representation. Recently, as the availability of large-scale image databases (Deng et al, 2009) and powerful computational resources, Convolutional Neural Networks (CNN) have been dominant in either large-scale image classification (Krizhevsky et al,

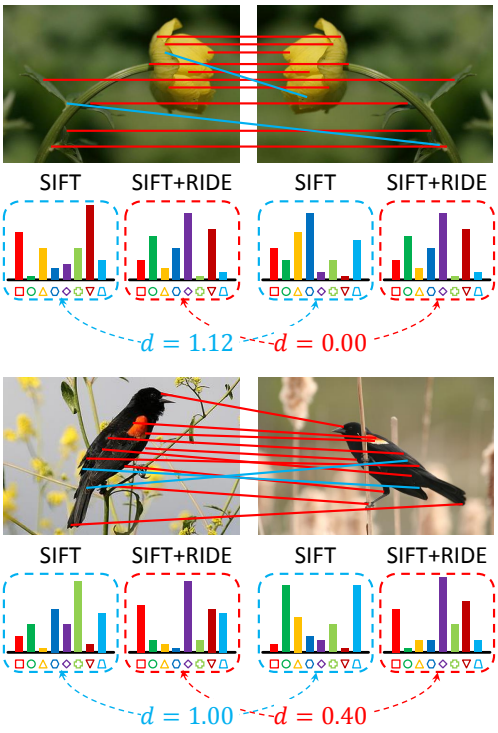

Fig. 1: SIFT (Lowe, 2004) matching with (red) and without (blue) reversal invariance (best viewed in color). In the latter case, it is difficult to find feature matches even between an image and its reversed copy (the above example). RIDE (illustrated in Section 4) brings reversal invariance to local descriptors, and significantly reduces the feature (*e.g.*, BoF) distance between each pair of reversed objects.

2012), or extracting transferrable features (Donahue et al, 2014)(Jia et al, 2014)(Razavian et al, 2014) for various computer vision tasks.

People often capture images or photos without caring about its left/right **orientation**, since an image and its reversed copy often deliver the same visual concept. However, as we shall see in Section 3, statistics-based image representation is not often robust to image reversal. The reason mainly lies in that handcrafted descriptors, such as SIFT (Lowe, 2004) and LCS (Perronnin et al, 2010), might change completely after being reversed (Figure 1), therefore it is difficult to find feature correspondence between an image and its reversed version. Consequently, the BoF representation of an image might be totally different after it is reversed. Meanwhile, most CNN models are somewhat sensitive to image reversal, since convolution is not reversal-invariant. The unsatisfied feature stability limits machine learning algorithms from learning discriminative models. To cope with, researchers propose an effective approach named data augmentation, which works by adding a reversed copy for each image (Chatfield et al, 2011)(Chai et al, 2013), or reversing each training image in the CNN training process with a probability of 50% (Krizhevsky et al, 2012). Although data augmentation consistently improves recognition accuracy, it still suffers the disadvantage of being more computationally expensive, especially on the online testing stage of the BoF model.

This paper presents an alternative idea, *i.e.*, designing reversal-invariant representation of local patterns for both the BoF and CNN models. On the BoF model, we start with observing the difference between the original and reversed descriptors, and then suggest computing the orientation of each descriptor so that we can cancel out the impact of image reversal. For orientation estimation, we adopt an approximated summation on the gradient-based histograms of SIFT. Based on this theory, we propose **Max-SIFT** and **RIDE** (Reversal-Invariant Descriptor Enhancement), two simple, fast yet generalized algorithms which bring reversal invariance to local descriptors. Both Max-SIFT and RIDE guarantee to generate identical representation for an image and its left-right reversed copy. Experiments reveal that Max-SIFT and RIDE produce consistent accuracy improvement to image classification. RIDE even outperforms data augmentation with higher recognition rates and lower time/memory consumptions. Max-SIFT and RIDE appear as preliminary publications (Xie et al, 2015b) and (Xie et al, 2015d), respectively.

In this extended journal version, we generalize the idea to the state-of-the-art CNN architectures. We first propose **RI-Deep**, a simple algorithm which extracts reversal-invariant deep features by post-processing. Then we design a reversal-invariant convolution operation (**RI-Conv**) and plug it into conventional CNNs, so that we can train reversal-invariant deep networks, which generate reversal-invariant deep features directly (without requiring post-processing). RI-Conv enjoys the advantage of enlarging the network capacity without increasing the model complexity. Experiments verify the effectiveness of our algorithms, demonstrating the importance of reversal invariance in training efficient CNN models and transferring deep features.

The remainder of this paper is organized as follows. Section 2 briefly introduces related works. Section 3 elaborates the importance of reversal invariance of image representation. Sections 4 and 5 illustrate our algorithms towards reversal-invariant representation of local patterns, and the application on the BoF and CNN models, respectively. Experiments are shown in each section. Finally, we conclude our work in Section 6.

2 Related Works

2.1 The BoF Model

The BoF model (Csurka et al, 2004) starts with describing local patches. Due to the limited descriptive power of raw image pixels, handcrafted descriptors, such as SIFT (Lowe, 2004), HOG (Dalal and Triggs, 2005) and LCS (Perronnin et al, 2010), are widely adopted. Although these descriptors can be automatically detected using operators such as DoG (Lowe, 2004) and MSER (Matas et al, 2004), the dense sampling strategy (Bosch et al, 2006)(Tuytelaars, 2010) often works better on classification tasks.

Next, a visual vocabulary (codebook) is trained to estimate the feature space distribution. The codebook is often computed with iterative algorithms such as K-Means or GMM. Descriptors are then encoded with the codebook. Popular feature encoding methods include hard quantization, sparse coding (Yang et al, 2009), LLC encoding (Wang et al, 2010), super-vector encoding (Zhou et al, 2010), Fisher vector encoding (Sanchez et al, 2013), *etc.*

On the final stage, quantized feature vectors are aggregated as compact image representation. Sum pooling, max-pooling and ℓ_p-norm pooling (Feng et al, 2011) can be different choices, and visual phrases (Zhang et al, 2009)(Xie et al, 2014a) and/or spatial pyramids (Grauman and Darrell, 2005)(Lazebnik et al, 2006) are constructed for richer spatial context modeling. The representation vectors are then summarized (Xie et al, 2015c) and fed into machine learning algorithms such as the SVM.

It is also important to organize local features according to the property of the image dataset. A popular case is fine-grained object recognition, which is aimed at predicting the object class at a finer level of granularity. Given that each image contains, say, a *bird*, it remains to decide which species is depicted. As observed in (Berg and Belhumeur, 2013)(Chai et al, 2013)(Gavves et al, 2014), the key to fine-grained recognition is the alignment of semantic object parts, such as the *head* or *tail* of a *bird*. Meanwhile, for scene understanding, it is reasonable to capture other types of visual clues to assist recognition, such as orientations (Xie et al, 2014b) and important semantic regions (Lin et al, 2014).

2.2 Convolutional Neural Networks

The Convolutional Neural Network (CNN) serves as a hierarchical model for large-scale visual recognition. It is based on that a network with enough neurons is able to fit any complicated data distribution. In the early years, neural networks were shown effective for simple recognition tasks such as digit recognition (LeCun et al, 1990). More recently, the availability of large-scale training data (*e.g.*, ImageNet (Deng et al, 2009)) and powerful GPUs makes it possible to train deep CNNs (Krizhevsky et al, 2012) which significantly outperform the BoF-based models. A CNN is composed of several stacked layers, in each of which responses from the previous layer are convoluted and activated by a differentiable function. Hence, a CNN can be considered as a composite function, and is trained by back-propagating error signals defined by the difference between supervised and predicted labels at the top level. Recently, efficient methods were proposed to help CNNs converge faster and prevent over-fitting, such as ReLU activation (Krizhevsky et al, 2012), dropout and batch normalization (Ioffe and Szegedy, 2015). It is believed that deeper networks produce better recognition results (Simonyan and Zisserman, 2015)(Szegedy et al, 2015).

The intermediate responses of CNNs, or the so-called deep features, serve as an efficient image description (Donahue et al, 2014), or a set of latent visual attributes. They can be used for various types of vision applications, including image classification (Jia et al, 2014), image retrieval (Razavian et al, 2014)(Xie et al, 2015a) and object detection (Girshick et al, 2014). A discussion of how different CNN configurations impact deep feature performance is available in (Chatfield et al, 2014). Visualization also helps to understanding the behaviour of CNN models (Zeiler and Fergus, 2014).

2.3 Towards Reversal Invariance

One of the major shortcomings of the BoF and CNN models is the unsatisfied stability of image representation. Especially, in fine-grained recognition tasks, objects might have different left/right orientations. Since handcrafted descriptors (such as SIFT) and convolution operations are not reversal-invariant, feature representation of an image and its reversed version might be totally different.

To cope with, researchers propose to augment the image datasets by adding a reversed copy for each original image, and perform classification on the enlarged training and testing sets (Chatfield et al, 2011)(Chai et al, 2013). In (Paulin et al, 2014), it is even suggested to learn a larger image transformation set for data augmentation. Similar strategies are also adopted in the CNN training process, including a popular method which adds reversal on each training sample with a probability of 50%, which, as a part of data augmentation, is often cooperated with other techniques such

Fig. 2: Content-based image retrieval on the right-oriented **Aircraft-100** dataset. We use the same query image with different orientations (best viewed in color).

as image cropping (Krizhevsky et al, 2012). Although data augmentation improves the recognition accuracy consistently, it brings heavier computational overheads, *e.g.*, almost doubled time and memory consumptions on the online testing stage of the BoF model, or the requirement of more training epoches to make the CNN training process converge.

There are also efforts on designing reversal-invariant descriptors for image retrieval. Some of them (Ma et al, 2010)(Xie et al, 2015b) consider geometry-inverted and brightness-inverted variants, and perform a symmetric function, such as dimension-wise summation or maximization, to cancel out the reversal operation. Other examples include defining a set of spatial bins to calculate histograms (Guo and Cao, 2010), or enforcing that the flows of all regions should follow a pre-defined direction (Zhao and Ngo, 2013). These works inspire us that symmetry is the key to reversal invariance (Skelly and Sclaroff, 2007)(Wang et al, 2011).

3 Why Reversal Invariance?

People often take pictures without caring about the left/right orientation, since an image and its left-right reversed copy often have the same semantic meaning. Consequently, there exist both left-oriented and right-oriented objects in almost every popular image datasets, especially in the case of fine-grained object recognition on *animals*, *man-made tools*, *etc.* For example, among 11788 images of the **Bird-200** dataset (Wah et al, 2011), at least 5000 *birds* are oriented to the left and other 5000 oriented to the right. In the **Aircraft-100** dataset (Maji et al, 2013) with 10000 images, we can also find more than 4800 left-oriented and more than 4500 right-oriented *aircrafts*, respectively.

However, we argue that most image representation models are sensitive to image reversal, *i.e.*, the features extracted from an image and its reversed version may be completely different. Let us take a simple case study using the BoF model which encodes SIFT with the Fisher vectors (Perronnin et al, 2010). Detailed settings are shown in Section 4.6. We perform image classification and retrieval tasks on the **Aircraft-100** dataset (Maji et al, 2013). We choose this dataset mainly because that the orientation of an *aircraft* is more easily determined than, say, a *bird*. Based on the original dataset, we manually reverse all the left-oriented images, generating a right-aligned dataset.

With the standard training/testing split (around 2/3 images are used for training and others for testing), the recognition rate is 53.13% on the original dataset and rises up quickly to 63.94% on the right-aligned dataset, with a more-than-10% absolute accuracy gain (a more-than-20% relative gain). This implies that orientation alignment brings a huge benefit to fine-grained

object recognition. As a diagnostic experiment, we use all (10000) images in the right-aligned dataset for training, and evaluate the model on two datasets with exactly the same image contents but different orientations. When testing images are all right-oriented (*i.e.*, performing self-validation), the classification accuracy is 99.73%. However, when testing images are all left-oriented (by reversing right-oriented ones), the accuracy drops dramatically to 46.84%. This experiment indicates that a model learned from right-oriented objects may not recognize left-oriented objects very well.

We also perform image retrieval on the right-aligned dataset to observe the feature quality more directly. Given a query image, we sort the candidates according to the ℓ_2 distance between the representation vectors. Some typical results are shown in Figure 2. When the query is of the same orientation (right) with the database, the search result is satisfying (mAP is 0.4143, the first false-positive is ranked at #18). However, if the query image is reversed, its feature representation changes thoroughly, and the retrieval accuracy drops dramatically (mAP is 0.0025, the first true-positive is ranked at #388). It is worth noting, in the latter case, that the reversed version of the query image is ranked at #514, which means that more than 500 images, most of them coming from different categories, are more similar to the query than its reversed copy!

Although all the above experiments are based on the BoF model with SIFT and Fisher vectors, we emphasize that similar trouble also arises in the case of extracting deep features from a pre-trained neural network. Since convolution is not reversal invariance, the features extracted on an image and its reversed version are often different, even when the network is trained with data augmentation (each training image is reversed with a 50% probability). We shall present detailed analysis on this point in Section 5.

Since an image and its reversed copy might have totally different feature representation, in a fine-grained dataset containing both left-oriented and right-oriented objects, we are implicitly partitioning the images of each class into two (or even more) prototypes. Consequently, the number of training images of each prototype is reduced and the risk of over-fitting increased. With this observation, some algorithms (Chatfield et al, 2011)(Chai et al, 2013) augment the dataset by generating a reversed copy for each image to increase the number of training cases of each prototype, meanwhile the testing stage of deep networks often involves image reversal (Krizhevsky et al, 2012)(Simonyan and Zisserman, 2015). We propose a different idea that generates reversal-invariant image representation in a bottom-up manner.

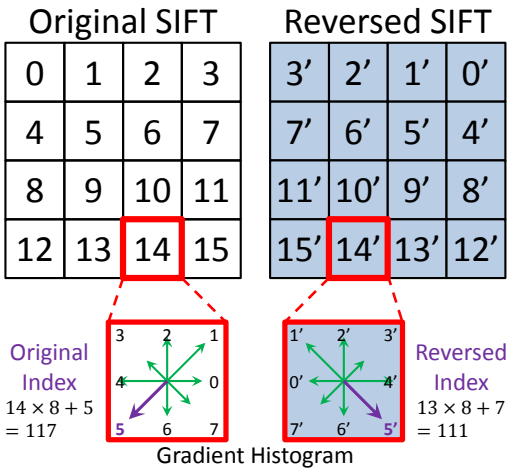

Fig. 3: SIFT and its reversed version. Corresponding grids/gradients are marked with the same number. Numbers in the original SIFT indicate the order of collecting grids/gradients.

4 Reversal Invariance for BoF

This section introduces reversal invariance to the BoF model by designing reversal-invariant local descriptors. We first discuss the basic principle of designing reversal-invariant descriptors, and then provide a simple solution named **Max-SIFT**. After that, we generalize Max-SIFT as **RIDE**, and show that it can be applied to more types of local descriptors. Experiments on the BoF model and Fisher vector encoding verify the effectiveness of our algorithms.

4.1 Reversal-Invariant Local Descriptors

4.1.1 Reversal Invariance as a Symmetric Function

We start from observing how SIFT, a typical handcrafted descriptor, changes with left-right image reversal. The structure of a SIFT descriptor is illustrated in Figure 3. A patch is partitioned into 4×4 spatial grids, and in each grid a 8-dimensional gradient histogram is computed. Here we assume that spatial grids are traversed from top to bottom, then left to right, and gradient intensities in each grid is collected in a counter-clockwise order. When an image is left-right reversed, all the patches on it are reversed as well. In a reversed patch, both the order of traversing spatial grids and collecting gradient values are changed, although the absolute gradient values in the corresponding directions do not change. Taking the lower-right grid in the original SIFT descriptor (#15) as the example. When the image is reversed, this grid appears at the lower-left position

(#12), and the order of collecting gradients in the grid changes from $(0,1,2,3,4,5,6,7)$ to $(4,3,2,1,0,7,6,5)$.

Denote the original SIFT as $\mathbf{d} = (d_0, d_1, \ldots, d_{127})$, in which $d_{i\times 8+j} = a_{i,j}$ for $i = 0,1,\ldots,15$ and $j = 0,1,\ldots,7$. As shown in Figure 3, each index (0 to 127) of the original SIFT is mapped to another index of the reversed SIFT. For example, d_{117} ($a_{14,5}$, the bold arrow in Figure 3) would appear at d_{111} ($a_{13,7}$) when the descriptor is reversed. Denote the **index mapping function** as $f^{\mathrm{R}}(\cdot)$ (*e.g.*, $f^{\mathrm{R}}(117) = 111$), so that the reversed SIFT can be computed as: $\mathbf{d}^{\mathrm{R}} \doteq \mathbf{f}^{\mathrm{R}}(\mathbf{d}) = \left(d_{f^{\mathrm{R}}(0)}, d_{f^{\mathrm{R}}(1)}, \ldots, d_{f^{\mathrm{R}}(127)}\right)$.

Towards reversal invariance, we need to design a **descriptor transformation function** $\mathbf{r}(\mathbf{d})$, so that $\mathbf{r}(\mathbf{d}) = \mathbf{r}\left(\mathbf{d}^{\mathrm{R}}\right)$ for any descriptor $\mathbf{d}$. For this, we define $\mathbf{r}(\mathbf{d}) = \mathbf{s}\left(\mathbf{d}, \mathbf{d}^{\mathrm{R}}\right)$, in which $\mathbf{s}(\cdot,\cdot)$ satisfies *symmetry*, *i.e.*, $\mathbf{s}(\mathbf{d}_1,\mathbf{d}_2) = \mathbf{s}(\mathbf{d}_2,\mathbf{d}_1)$ for any pair $(\mathbf{d}_1,\mathbf{d}_2)$. In this way reversal invariance is achieved: $\mathbf{r}(\mathbf{d}) = \mathbf{s}\left(\mathbf{d},\mathbf{d}^{\mathrm{R}}\right) = \mathbf{s}\left(\mathbf{d}^{\mathrm{R}},\mathbf{d}\right) = \mathbf{s}\left(\mathbf{d}^{\mathrm{R}},\left(\mathbf{d}^{\mathrm{R}}\right)^{\mathrm{R}}\right) = \mathbf{r}\left(\mathbf{d}^{\mathrm{R}}\right)$. We use the fact that $\left(\mathbf{d}^{\mathrm{R}}\right)^{\mathrm{R}} = \mathbf{d}$ holds for any descriptor $\mathbf{d}$.

4.1.2 The Max-SIFT Descriptor

There are a lot of symmetric function $\mathbf{s}(\cdot,\cdot)$, such as dimension-wise summation or maximization. Here we consider an extremely simple case named **Max-SIFT**, in which we choose the one in $\mathbf{d}$ and $\mathbf{d}^{\mathrm{R}}$ with the larger *sequential lexicographic order*. Here, by the sequential lexicographic order we mean to regard each SIFT descriptor as a sequence with length 128, and on each dimension (an element in the sequence), the larger value has the higher priority. Therefore, to compute the Max-SIFT descriptor for $\mathbf{d}$, we only need to compare the dimensions of $\mathbf{d}$ and $\mathbf{d}^{\mathrm{R}}$ one by one and stop at the first difference. Let us denote the Max-SIFT descriptor of $\mathbf{d}$ by $\widehat{\mathbf{d}}$, and use the following notation:

$$\widehat{\mathbf{d}} = \mathbf{r}(\mathbf{d}) = \widehat{\max}\left\{\mathbf{d}, \mathbf{d}^{\mathrm{R}}\right\}. \tag{1}$$

Obviously, $\widehat{\mathbf{d}}$ equals to either $\mathbf{d}$ or $\mathbf{d}^{\mathrm{R}}$, thus we maximally preserve the descriptive power of SIFT.

Algorithm 1 Max-SIFT

1: **Input:** $\mathcal{D} = \{\mathbf{d}_m, \mathbf{l}_m\}_{m=1}^{M}$.
2: **procedure** MAX-SIFT
3: **Reversal:** $\mathcal{D}^{\mathrm{R}} = \left\{\mathbf{d}_m^{\mathrm{R}}, \mathbf{l}_m\right\}_{m=1}^{M}$;
4: **Selection:** $\widehat{\mathbf{d}_m} = \mathbf{r}(\mathbf{d}_m)$, based on (1);
5: **end procedure**
6: **Output:** $\widehat{\mathcal{D}} = \left\{\widehat{\mathbf{d}_m}, \mathbf{l}_m\right\}_{m=1}^{M}$.

The pseudo codes of Max-SIFT are illustrated in Algorithm 1. We point out that there are many other symmetric functions, but their performance is often inferior to **Max-SIFT**. For example, using **Average-SIFT**, *i.e.*, $\mathbf{r}(\mathbf{d}) = \frac{1}{2}\left(\mathbf{d} + \mathbf{d}^{\mathrm{R}}\right)$, leads to 1%–2% accuracy drop on every single image classification case.

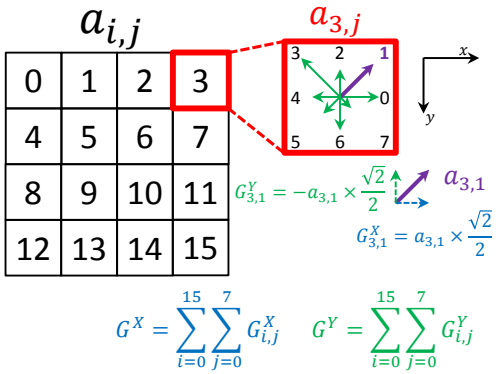

Fig. 4: Estimating the orientation of SIFT.

4.2 RIDE: Generalized Reversal Invariance

4.2.1 The Orientation of SIFT

Let us choose the descriptor from $\mathbf{d}$ and $\mathbf{d}^{\mathrm{R}}$ in a more generalized manner. In general, we can define an **orientation quantization function** $q(\cdot)$, and choose the one in $\left\{\mathbf{d}, \mathbf{d}^{\mathrm{R}}\right\}$ with the larger function value. Ideally, $q(\cdot)$ can capture the orientation property of a descriptor, *e.g.*, $q(\mathbf{d})$ reflects the extent that $\mathbf{d}$ is oriented to the right. Recall that in the original version of SIFT (Lowe, 2004), each descriptor is naturally assigned an orientation angle $\theta \in [0, 2\pi)$, so that we can simply take $q(\mathbf{d}) = \cos\theta$, but orientation is often ignored in the implementation of dense SIFT (Bosch et al, 2006)(Vedaldi and Fulkerson, 2010). We aim at recovering the orientation with fast computations.

The major conclusion is that, the global orientation of a densely-sampled SIFT descriptor can be estimated by its local gradients. For each of the 128 dimensions, we take its gradient value and lookup for its (1 of 8) direction. The gradient value is then decomposed into two components along the x-axis and y-axis, respectively. The left/right orientation of the descriptor is then computed by collecting the x-axis components over all the 128 dimensions. Formally, we define 8 orientation vectors $\mathbf{u}_j$, $j = 0,1,\ldots,7$. According to the definition of SIFT in Figure 3, we have $\mathbf{u}_j = \left(\cos(j\pi/8), \sin(j\pi/8)\right)^{\top}$. The global gradient can be computed as $\mathbf{G}(\mathbf{d}) = (G_x, G_y)^{\top} = \sum_{i=0}^{15}\sum_{j=0}^{7} a_{i,j}\mathbf{u}_j$. The computing process is illustrated in Figure 4. The proof of the estimation is provided in Appendix A.

4.2.2 The RIDE Algorithm

We simply take G_x as the value of quantization function, *i.e.*, $q(\mathbf{d}) = G_x(\mathbf{d})$ for every $\mathbf{d}$. It is worth noting that $q(\mathbf{d}) = -q(\mathbf{d}^{\mathrm{R}})$ holds for any $\mathbf{d}$, therefore we can simply use the sign of $q(\mathbf{d})$ to compute the reversal-invariant descriptor transform $\widetilde{\mathbf{d}}$:

$$\widetilde{\mathbf{d}} = \mathbf{r}(\mathbf{d}) = \begin{cases} \mathbf{d} & q(\mathbf{d}) > 0 \\ \mathbf{d}^{\mathrm{R}} & q(\mathbf{d}) < 0 \\ \widehat{\max}\left\{\mathbf{d}, \mathbf{d}^{\mathrm{R}}\right\} & q(\mathbf{d}) = 0 \end{cases} . \tag{2}$$

We name the algorithm **RIDE** (Reversal-Invariant Descriptor Enhancement). When $q(\mathbf{d}) = 0$, RIDE degenerates to Max-SIFT. Since Max-SIFT first compares d_0 and d_{28} ($f^{\mathrm{R}}(0) = 28$), we can regard it as a special case of RIDE, with $q(\mathbf{d}) = d_0 - d_{28}$.

4.2.3 The Generalization of RIDE

We generalize RIDE to (a) other local descriptors and (b) more types of reversal invariance.

When RIDE is applied on other dense descriptors, we can first extract SIFT descriptors on the same patches, then compute $\mathbf{G}$ to estimate the orientation of those patches, and perform reversal operation if necessary. A generalized flowchart of RIDE is illustrated in Algorithm 2. The extra time overheads in this process mainly come from the computation of SIFT, which can be exempted in the case of using Color-SIFT descriptors. For example, RGB-SIFT is composed of three SIFT vectors $\mathbf{d}_{\mathrm{R}}$, $\mathbf{d}_{\mathrm{G}}$ and $\mathbf{d}_{\mathrm{B}}$, from the individual red, green and blue channels, therefore we can compute $\mathbf{G}_{\mathrm{R}}$, $\mathbf{G}_{\mathrm{G}}$ and $\mathbf{G}_{\mathrm{B}}$ individually, and combine them with $\mathbf{G} = 0.30\mathbf{G}_{\mathrm{R}} + 0.59\mathbf{G}_{\mathrm{G}} + 0.11\mathbf{G}_{\mathrm{B}}$. For other color SIFT descriptors, the only difference lies in the linear combination coefficients. By this trick we can perform RIDE on Color-SIFT descriptors very fast.

Algorithm 2 Generalized **RIDE**

1: **Input:** $\mathcal{D} = \{\mathbf{d}_m, \mathbf{l}_m\}_{m=1}^{M}$.
2: **procedure** RIDE
3: **Reversal:** $\mathcal{D}^{\mathrm{R}} = \{\mathbf{d}_m^{\mathrm{R}}, \mathbf{l}_m\}_{m=1}^{M}$;
4: **SIFT:** $\mathcal{D}^{\mathrm{S}} = \{\mathbf{d}_m^{\mathrm{S}}, \mathbf{l}_m\}_{m=1}^{M}$, if necessary;
5: **Orientation:** $q(\mathbf{d}_m) = G_x(\mathbf{d}_m^{\mathrm{S}})$;
6: **Selection:** $\widetilde{\mathbf{d}_m} = \mathbf{r}(\mathbf{d}_m)$, based on (2);
7: **end procedure**
8: **Output:** $\widetilde{\mathcal{D}} = \left\{\widetilde{\mathbf{d}_m}, \mathbf{l}_m\right\}_{m=1}^{M}$.

In the case that RIDE is applied to fast binary descriptors for image retrieval, we can obtain the orientation vector $\mathbf{G}$ without computing SIFT. Let us take the BRIEF descriptor (Calonder et al, 2010) as an example. For a descriptor $\mathbf{d}$, $G_x(\mathbf{d})$ is obtained by accumulating the *binary tests*. For each tested pixel pair (p_1, p_2) with distinct x-coordinates, if the left pixel has a smaller intensity value, add 1 to $G_x(\mathbf{d})$, otherwise subtract 1 from $G_x(\mathbf{d})$. If the x-coordinates of p_1 and p_2 are the same, this pair is ignored. $G_y(\mathbf{d})$ is similarly computed. We still take $q(\mathbf{d}) = G_x(\mathbf{d})$ to quantize left-right orientation. This idea can also be generalized to other binary descriptors such as ORB (Rublee et al, 2011), which is based on BRIEF.

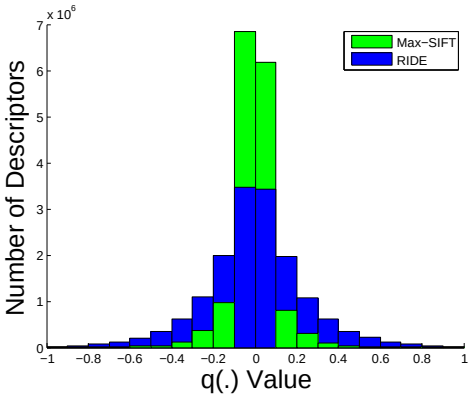

Fig. 5: The distribution of $q(\cdot)$ values on the **Bird-200** dataset. For Max-SIFT, $q(\mathbf{d}) = d_0 - d_{28}$ (see Section 4.2.2). All the SIFT descriptors are ℓ_2-normalized so that $\|\cdot\|_2 = 1$.

RIDE is also capable of cancelling out a larger family of reversal operations, including the upside-down image reversal, and image rotation by 90°, 180° and 270°. For this we need to constrain the descriptor more strictly with global gradient $\mathbf{G} = (G_x, G_y)^{\top}$. Recall that limiting $G_x > 0$ selects 1 descriptor from 2 candidates, resulting in **RIDE-2** (equivalent to **RIDE** mentioned previously) for left-right reversal invariance. Similarly, limiting $G_x > 0$ and $G_y > 0$ selects 1 from 4 descriptors, obtaining **RIDE-4** for both left-right and upside-down reversal invariance, and limiting $G_x > G_y > 0$ obtains **RIDE-8** for both reversal and rotation invariance. We do not use RIDE-4 and RIDE-8 in this paper, since upside-down reversal and heavy rotations are not often observed, whereas the descriptive power of a descriptor is reduced by strong constraints. An experimental analysis of these issues can be found in Appendix B.

4.3 Numerical Stability Issues

Both Max-SIFT and RIDE may suffer from numerical stability issues, especially in areas with low gradient magnitudes. When the quantization function value $q(\mathbf{d})$ is close to 0, small image noises may change the sign of

$q(\mathbf{d})$ and, consequently, the Max-SIFT and/or RIDE descriptors. To quantitatively analyze the impact of image noises, we first estimate the distribution of $q(\mathbf{d})$ on the **Bird-200** dataset (Wah et al, 2011). According to the histogram in Figure 5, one may observe that most SIFT descriptors have relatively small $q(\cdot)$ values using Max-SIFT. With the descriptors normalized ($\|\mathbf{d}\|_2 = 1$ for all $\mathbf{d}$), the median of $q(\mathbf{d})$ values is 0.0556 for Max-SIFT and 0.1203 for RIDE, which implies that RIDE is more robust than Max-SIFT to small image noises. The reason is that RIDE summarizes the information of the whole SIFT descriptor, while Max-SIFT only considers few dimensions.

Back to the image classification experiments on the **Bird-200** dataset, we add a random Gaussian noise with standard deviation 0.1203 (the median of $|q(\cdot)|$ values) to each of the $q(\cdot)$ value of RIDE, and find that random noises only cause the classification accuracy of SIFT case drop by less than 1%, which is relatively smaller compared to the gain of RIDE (6.37%, see Table 2(d)). Experiments on the **Aircraft-100** dataset (Maji et al, 2013) also lead to similar results.

4.4 Applications to Image Classification

We briefly discuss the application of Max-SIFT and RIDE for image classification. Consider an image $\mathbf{I}$, and a set of, say, SIFT descriptors extracted from the image: $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_M\}$. When the image is left-right reversed, the set $\mathcal{D}$ becomes: $\mathcal{D}^{\mathrm{R}} = \{\mathbf{d}_1^{\mathrm{R}}, \mathbf{d}_2^{\mathrm{R}}, \ldots, \mathbf{d}_M^{\mathrm{R}}\}$. If the descriptors are not reversal-invariant, *i.e.*, $\mathcal{D} \neq \mathcal{D}^{\mathrm{R}}$, the feature representation produced by $\mathcal{D}$ and $\mathcal{D}^{\mathrm{R}}$ might be totally different. With Max-SIFT or RIDE, we have $\widehat{\mathbf{d}} = \widehat{\mathbf{d}^{\mathrm{R}}}$ or $\widetilde{\mathbf{d}} = \widetilde{\mathbf{d}^{\mathrm{R}}}$, for any $\mathbf{d}$, therefore $\widehat{\mathcal{D}} = \widehat{\mathcal{D}^{\mathrm{R}}}$ and $\widetilde{\mathcal{D}} = \widetilde{\mathcal{D}^{\mathrm{R}}}$. Consequently, we generate the same representation for an image and its reversed copy.

A simple trick applies when Max-SIFT or RIDE is adopted with Spatial Pyramid Matching (Lazebnik et al, 2006). Note that corresponding descriptors might have different x-coordinates on an image and its reversed copy, *e.g.*, a descriptor appearing at the upper-left corner of the original image can also be found at the upper-right corner of the reversed image, resulting in the difference in spatial pooling bin assignment. To cope with, we count the number of descriptors to be reversed, *i.e.*, those satisfying $\widehat{\mathbf{d}} \neq \mathbf{d}$ or $\widetilde{\mathbf{d}} \neq \mathbf{d}$. If the number is larger than half of the total number of descriptors, we left-right reverse the descriptor set by replacing the x-coordinate of each descriptor with $W - x$, where W is the image width. This is equivalent to predicting the orientation of an image using the orientation of SIFT descriptors (see Section 4.6.3).

Dataset	Images	Trains
Pet-37 (Parkhi et al, 2012)	7390	100
Aircraft-100 (Maji et al, 2013)	10000	67
Flower-102 (Nilsback et al, 2008)	8189	20
Bird-200 (Wah et al, 2011)	11788	30
LandUse-21 (Yang et al, 2010)	2100	80
Indoor-67 (Quattoni et al, 2009)	15620	80
SUN-397 (Xiao et al, 2010)	108754	50
Caltech256 (Griffin, 2007)	30607	60

Table 1: Image classification datasets used in our experiments. We partition them as four fine-grained object recognition, three scene classification and one generic object recognition sets. The final column indicates the number of training images in each category.

4.5 Comparison with Previous Works

Many recently published papers achieve reversal invariance with data augmentation (Wang et al, 2010)(Chatfield et al, 2011)(Chai et al, 2013)(Paulin et al, 2014). In Section 4.6.2, we will show that RIDE works better than data augmentation.

Although some reversal-invariant descriptors have been proposed for image retrieval (Guo and Cao, 2010)(Ma et al, 2010)(Zhao and Ngo, 2013)(Xie et al, 2015b), these descriptors have not been adopted in classification tasks. We implement several of them, and compare it with Max-SIFT and RIDE in Table 3. One can observe that Max-SIFT and RIDE significantly outperform these competitors in every single case. Especially, MI-SIFT (Ma et al, 2010) works even worse than original descriptors, which is probably because it destroys the spatial structure of SIFT and thus harms the descriptive power of SIFT.

4.6 Experiments

4.6.1 Datasets and Settings

We evaluate our algorithm on four publicly available fine-grained object recognition datasets, three scene classification datasets and one generic object classification dataset. The detailed information of the used datasets is listed in Table 1.

Basic experimental settings follow the recent proposed BoF model (Sanchez et al, 2013). An image is scaled, with the aspect ratio preserved, so that there are 300 pixels on the larger axis. We only use the region within the bounding box if it is available. We use VLFeat (Vedaldi and Fulkerson, 2010) to extract dense RootSIFT (Arandjelovic and Zisserman, 2012) descriptors. The spatial stride and window size of dense

	ORIG	MAX	RIDE	AUGM	RIDE×2
S	37.92	41.78	42.28	42.24	**45.61**
L	43.25	–	44.27	45.12	**46.83**
F	52.06	53.92	54.69	54.67	**57.51**
R	44.90	46.73	47.35	46.98	**49.53**
O	46.53	48.39	49.01	48.72	**51.19**

(a) **Pet-37** Results

	ORIG	MAX	RIDE	AUGM	RIDE×2
S	53.13	57.72	57.82	57.16	**60.14**
L	41.82	–	42.86	43.13	**44.81**
F	57.36	60.49	61.27	60.59	**63.62**
R	57.89	61.90	63.09	62.48	**65.11**
O	47.06	52.35	53.12	51.39	**55.79**

(b) **Aircraft-100** Results

	ORIG	MAX	RIDE	AUGM	RIDE×2
S	53.68	58.12	59.12	58.01	**61.09**
L	73.47	–	75.30	75.88	**77.40**
F	76.96	79.59	80.51	79.49	**82.14**
R	71.52	74.00	74.97	74.18	**77.10**
O	76.12	78.40	79.68	78.83	**81.69**

(c) **Flower-102** Results

	ORIG	MAX	RIDE	AUGM	RIDE×2
S	25.77	31.59	32.14	31.60	**34.07**
L	36.18	–	38.50	38.97	**40.16**
F	38.11	43.48	44.73	43.98	**46.38**
R	31.36	38.20	39.16	38.79	**41.73**
O	35.40	41.15	42.18	41.72	**44.30**

(d) **Bird-200** Results

Table 2: Classification accuracy (%) of different models. Evaluated features include SIFT (**S**), LCS (**L**), FUSED (**F**, where SIFT and LCS features are concatenated), RGB-SIFT (**R**) and OPP-SIFT (**O**) features, while models include using the original descriptors (**ORIG**), Max-SIFT (**MAX**), RIDE (**RIDE**) or data augmentation (**AUGM**). Max-SIFT does not work on LCS, thus the LCS part remains unchanged in the FUSED feature. **RIDE×2** denotes using RIDE with doubled codebook size. See the texts in Section 4.6.2 for details.

sampling are 6 and 12, respectively. On the same set of patches, LCS, RGB-SIFT and Opponent-SIFT (van de Sande et al, 2010) descriptors are also extracted. Max-SIFT or RIDE is thereafter computed for each type of descriptors. We can only apply Max-SIFT on SIFT-based descriptors, thus the LCS descriptors remain unchanged. The dimensions of SIFT, LCS and color SIFT descriptors are reduced by PCA to 64, 64 and 128, respectively. We cluster the descriptors with a GMM of 32 components, and use the improved Fisher vectors (IFV) for feature encoding. A spatial pyramid with 4 regions (the entire image and three horizontal stripes) is adopted. Features generated by SIFT and LCS descriptors are concatenated as the FUSED feature. The final vectors are square-root normalized followed by ℓ_2 normalized (Lapin et al, 2014), and then fed into LibLINEAR (Fan et al, 2008), a scalable SVM implementation, with the slacking parameter $C = 10$. Averaged accuracy by category is reported on the fixed training/testing split provided by the authors.

To compare our results with the state-of-the-art classification results, **strong** Fisher vectors are extracted by resizing the images to 600 pixels in the larger axis, using spatial stride 8, window size 16, and clustering 256 GMM components.

4.6.2 Image Classification Results

We first report fine-grained object recognition accuracy with different descriptors in Table 2. Beyond original descriptors, we adopt both RIDE and data augmentation. By augmentation we mean to generate a reversed copy for each training/testing image, use the enlarged set to train the model, test with both original and reversed samples, and predict the label with a soft-max function (Paulin et al, 2014).

In Table 2, one can see that both Max-SIFT and RIDE produces consistent accuracy gain beyond original descriptors (**ORIG**). Moreover, when we use SIFT or Color-SIFT descriptors, RIDE also produces higher accuracy than that using data augmentation (**AUGM**). When the LCS descriptors are used, **RIDE** works a little worse than **AUGM**, which is probably because the orientation of LCS (not a gradient-based descriptor) is not very well estimated with SIFT gradients.

We shall emphasize that data augmentation requires almost doubled computational costs than those of RIDE (see Section 4.6.5 for details), since the time/memory complexity of many classification models is proportional to the number of training/testing images. To make fair comparison, we double the codebook size used in RIDE to obtain longer features, since it is a common knowledge that larger codebooks often lead to better classification results. Such a model, denoted by **RIDE×2**, works better than **AUGM** in every single case.

We also use strong features and compare Max-SIFT and RIDE with other reversal-invariant descriptors, namely MI-SIFT (Ma et al, 2010), FIND (Guo and Cao, 2010) and F-SIFT (Zhao and Ngo, 2013). We compute these competitors for each SIFT component in RGB-SIFT, and leave LCS unchanged in the FUSED feature. Results are shown in Table 3. The consisten-

	P-37	A-100	F-102	B-200	L-21	I-67	S-397	C256
ORIG	60.24	74.61	83.53	47.61	93.64	63.17	48.35	58.77
MAX	62.80	77.54	85.82	49.93	94.13	64.12	49.39	59.21
RIDE	**63.49**	**78.92**	**86.45**	**50.81**	**94.71**	**64.93**	**50.12**	**60.25**
MI(Guo and Cao, 2010)	58.91	72.26	81.06	45.59	92.86	61.49	46.51	55.39
FIND(Guo and Cao, 2010)	59.63	74.06	82.91	47.49	93.14	62.91	47.87	56.72
F(Zhao and Ngo, 2013)	61.06	75.95	84.72	48.21	93.64	63.36	48.61	58.57
(Angelova and Zhu, 2013)	54.30	–	80.66	–	–	–	–	–
(Maji et al, 2013)	–	48.69	–	–	–	–	–	–
(Murray and Perronnin, 2014)	56.8	–	84.6	33.3	–	–	–	–
(Paulin et al, 2014)	–	–	–	45.2	–	–	–	–
(Pu et al, 2014)	–	–	–	44.2	–	–	–	–
(Wang et al, 2014)	59.29	–	75.26	–	–	–	–	–
(Juneja et al, 2013)	–	–	–	–	–	63.10	–	–
(Kobayashi, 2014)	–	–	–	–	92.8	63.4	46.1	57.4
(Xie et al, 2014b)	–	–	–	–	–	63.48	45.91	–
(Lapin et al, 2014)	–	–	–	–	–	–	49.5	–

Table 3: Classification accuracy (%) comparison with recent works. We use RGB-SIFT on the **Aircraft-100** dataset, and the FUSED (SIFT with LCS) features on other datasets. We implement MI-SIFT (Ma et al, 2010), FIND (Guo and Cao, 2010) and F-SIFT (Zhao and Ngo, 2013) by ourselves.

t 3%-4% gain verifies that RIDE makes stable contribution to visual recognition. Moreover, researchers design complex part-based recognition algorithms on the **Bird-200** dataset (Chai et al, 2013)(Gavves et al, 2014)(Xie et al, 2013) (Zhang et al, 2013)(Zhang et al, 2014b)(Zhang et al, 2014a)(Li et al, 2015). We also evaluate RIDE on the detected parts provided by symbiotic segmentation and localization (Chai et al, 2013) and gravitational alignment (Gavves et al, 2014). RIDE boosts the recognition accuracy of (Chai et al, 2013) and (Gavves et al, 2014) from 56.6% to 60.7% and from 65.3% to 67.4%, respectively. In comparison, (Gavves et al, 2014) applies data augmentation to boost the accuracy from 65.3% to 67.0%. RIDE produces better results with only half time/memory consumption. With the parts learned by deep CNNs (Zhang et al, 2014a), we get 73.1% with the FUSED features.

To reveal that Max-SIFT and RIDE can be applied to generalized classification, we perform experiments on the scene classification and generic object recognition tasks. The FUSED (SIFT with LCS) features are used, and the results are summarized in Table 3. It is interesting to see that Max-SIFT and RIDE also work well to outperform the recent competitors. Thus, although Max-SIFT and RIDE are motivated by the observation on fine-grained cases, it enjoys good recognition performance on a wide range of image datasets.

4.6.3 Object Orientation Prediction

As an diagnostic experiment, we predict the left/right orientation of an image based on the orientation quantization function $q(\cdot)$. We use the **Aircraft-100** dataset, in which the orientation (left or right) of each *aircraft* is manually labeled. We adopt the ground-truth bounding box to crop the image, so that the objects are better aligned. After cropping, all the images are resized so that the longer axis has 600 pixels, and dense SIFT descriptors are extracted using the VLFeat library (Vedaldi and Fulkerson, 2010).

We use 2/3 images (approximately 67 per category) for training. Without the loss of generality, we assume that all the training images are oriented to right. For each testing image, we compute its **orientation score** by accumulating clues from each descriptor. Suppose the width and height of the testing image are W and H, then a descriptor on the position (x, y) has the "relative position" $(x/W, y/H)$. On each training image, we seek for the nearest descriptor measured by the ℓ_2 distance of relative positions, and compare their orientation quantization function values: if the values are of the same sign, add 1 to the score, otherwise subtract 1 from the score. After all the descriptors are processed, if the score is positive, then this testing image is oriented to right; if it is negative, to left; if it is 0, we perform a random guess (this situation merely happens).

One can note that different results are produced with different orientation quantization functions. Using Max-SIFT ($q(\mathbf{d}) = d_0 - d_{28}$), the prediction accuracy is 65.45%, whereas using RIDE ($q(\mathbf{d}) = G_x(\mathbf{d})$), it is 54.69%, barely above random guess (50%). Considering information from all 128 dimensions, RIDE produces more accurate prediction than Max-SIFT, thus better image alignment. As we have seen in Section 3, this significantly helps image classification.

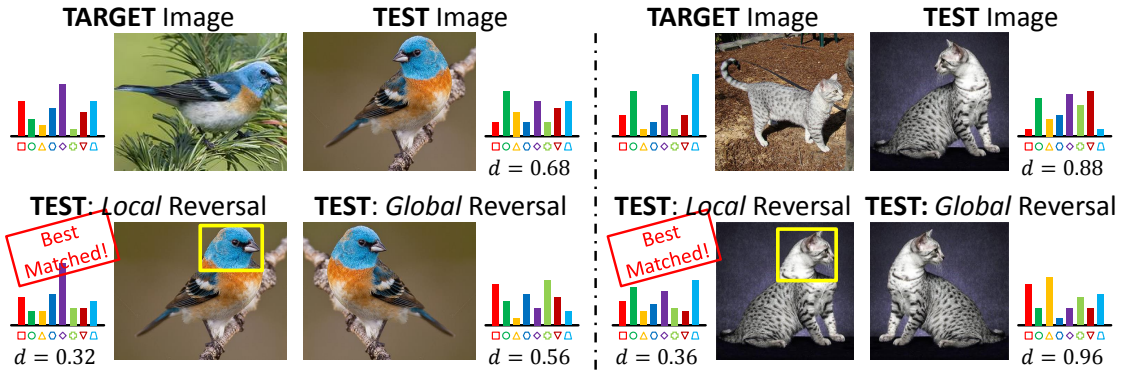

Fig. 6: *Global* vs. *local* image reversal. Local reversal (with *manually* labeled regions in the yellow boxes) allows more flexible image representation, and produces smaller feature distances between the **test** and **target** images.

4.6.4 Global Reversal vs. Local Reversal

Based on the above experiments, one can conclude that RIDE produces powerful image features by predicting the orientation of objects in an implicit manner.

An essential difference between RIDE and data augmentation comes from the comparison of local and global image reversal. By local reversal we mean that RIDE can decide whether to reverse every single descriptor individually, while data augmentation only allows to choose one image from two candidates, *i.e.*, either original or globally reversed. Figure 6 compares both strategies in an intuitive manner. In these cases, we aim at matching a **target** image with a possibly reversed **test** image. With global reversal, we have only two choices and the flexibility of our model is limited. With local reversal, however, it is possible to reverse smaller regions such as the turned *head* of the *bird* or *cat*. By this we can find larger numbers of true feature matches and obtain more similar image representation, *i.e.*, smaller feature distance. Therefore, it is not difficult to understand the reason why RIDE works even better than data augmentation.

4.6.5 Computational Costs

We report the time/memory cost of RIDE with SIFT in Table 4. The time cost of Max-SIFT is consistently lower than RIDE, and the memory cost is the same.

Since the only extra computation of RIDE comes from gradient accumulation and descriptor permutation, the additional time cost of RIDE is merely about 1% of SIFT computation. RIDE does not require any extra memory storage. However, if the dataset is augmented with left-right image reversal, one needs to compute and store two instances for each image, descriptor and feature vector, resulting in almost doubled time and memory overheads, which is comparable with using a double-sized codebook, whereas the latter produces better classification results.

	ORIG	**RIDE**	**AUGM**	**RIDE×2**
Descriptor	2.27 hrs	2.29 hrs	2.30 hrs	2.29 hrs
Codebook	0.13 hrs	0.13 hrs	0.13 hrs	0.27 hrs
Encoding	0.78 hrs	0.78 hrs	1.56 hrs	1.28 hrs
Recognition	1.21 hrs	1.21 hrs	2.46 hrs	2.42 hrs
(RAM cost)	3.71 GB	3.71 GB	7.52 GB	7.51 GB

Table 4: Time/memory cost in each step of the BoF model. All the data are recorded with SIFT descriptors with 32 GMM components on the **Bird-200** dataset (Wah et al, 2011).

4.7 Summary

In this section, we explore reversal invariance in the context of the BoF model. We propose the **Max-SIFT** descriptor and the **RIDE** (Reversal-Invariant Descriptor Enhancement) algorithm which bring reversal invariance to local descriptors. Our idea is inspired by the observation that most handcrafted descriptors are not reversal-invariant, whereas many fine-grained datasets contain objects with different left/right orientations. Max-SIFT and RIDE cancels out the impact of image/object reversal by estimating the orientation of each descriptor, and then forcing all the descriptors to have the same orientation. Experiments reveal that both of them significantly improve the accuracy of fine-grained object recognition and scene classification with very few computational costs. Both Max-SIFT and RIDE are robust to small image noises. Compared with data augmentation, RIDE produces better results with lower time/memory consumptions.

5 Reversal Invariance for CNN

In this section, we generalize the above ideas from BoF to CNN. We first present a simple strategy to improve deep features, which demonstrates the importance of reversal invariance in CNN. Motivated by which, we propose a new convolution operation so that we can train reversal-invariant deep networks directly.

5.1 Reversal-Invariant Deep Features (RI-Deep)

5.1.1 Average-Deep and Max-Deep

We start with observing the behavior of deep features, which are the neuron responses of a testing image extracted from a pre-trained CNN model. In general, when an image is reversed, the neuron responses on each layer change accordingly, because the convolution operation is not reversal-invariant. In most deep CNN models (Krizhevsky et al, 2012)(Szegedy et al, 2015)(Simonyan and Zisserman, 2015), data augmentation with image reversal is widely adopted on both the training and testing stages. In training, each sample is reversed with a probability of 50%, so that the network can see objects with different orientations. In testing, neuron responses on both the original and reversed versions are computed and averaged. We shall verify in the later experiments that data augmentation modules in training and testing are complementary to each other.

Let us denote an image as $\mathbf{I}$ and its left-right reversed version as $\mathbf{I}^{\mathrm{R}}$. Given a deep CNN model $\mathcal{M}$ and a specified layer number l, the feature vector extracted on the l-th layer is $\mathbf{f}_l(\mathbf{I};\mathcal{M}) \in \mathbb{R}^{K_l}$, where K_l is the number of channels (convolution kernels) on that layer. With the reversed image, we can also compute the reversed deep feature: $\mathbf{f}_l^{\mathrm{R}}(\mathbf{I};\mathcal{M}) \doteq \mathbf{f}_l\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$. Most often, $\mathbf{f}_l(\mathbf{I};\mathcal{M}) \neq \mathbf{f}_l^{\mathrm{R}}(\mathbf{I};\mathcal{M})$.

Inspired by Section 4.1.1, we seek for a **deep feature transformation function** $\mathbf{r}(\cdot)$, which satisfies $\mathbf{r}(\mathbf{I}) = \mathbf{r}\big(\mathbf{I}^{\mathrm{R}}\big)$ for any image $\mathbf{I}$. Here, we choose two simple symmetric operations, named **Avg-Deep** and **Max-Deep**, respectively:

$$\mathbf{r}_l^{\mathrm{AVG}}(\mathbf{I}) = \frac{1}{2}\left[\mathbf{f}_l(\mathbf{I};\mathcal{M}) + \mathbf{f}_l^{\mathrm{R}}(\mathbf{I};\mathcal{M})\right], \tag{3}$$

$$\mathbf{r}_l^{\mathrm{MAX}}(\mathbf{I}) = \max\left\{\mathbf{f}_l(\mathbf{I};\mathcal{M}), \mathbf{f}_l^{\mathrm{R}}(\mathbf{I};\mathcal{M})\right\}, \tag{4}$$

where $\max\{\cdot,\cdot\}$ denotes the element-wise maximization of two vectors. This strategy is different with that used in Section 4.1.2 which chooses the one with the larger sequential lexicographic order. Let us take a little space to illustrate the difference between SIFT descriptors and deep features. SIFT is a type of handcrafted descriptor in which each dimension corresponds to the intensity of gradient. If we simply take the dimension-wise average or maximum of a SIFT and its reversed version, the inner structure as well as the relationship between corresponding dimensions may be damaged, leading to significant accuracy drop. In a deep feature vector, however, each dimension corresponds to the extent that a visual concept or attribute arises, therefore it is reasonable to take dimension-wise operation to consider the visual attributes contained in both the original and reversed images.

We point out that Avg-Deep is similar to the testing strategy used in the state-of-the-art CNNs (Krizhevsky et al, 2012)(Simonyan and Zisserman, 2015)(Szegedy et al, 2015). In which, using both the original and reversed testing images produces around 0.2%–0.5% accuracy gain. We shall verify that this strategy is also useful in transferring features for image classification.

Regarding computational costs, both Avg-Deep and Max-Deep require doubled time complexity on the feature extraction stage, but they do not need extra time or memory on the online testing stage. Considering that the feature extraction is performed only once, the extra cost is thus reasonable.

5.1.2 Image Classification Experiments

We evaluate the models on all the eight datasets introduced in Section 4.6. For pre-trained deep networks, we use the **AlexNet** and the **VGGNet** (both the 16-layer and 19-layer models), provided by the **MatConvNet** library (Vedaldi and Lenc, 2014). To demonstrate the importance of reversal invariance, we also train another version of the **AlexNet**, in which we do not add reversal data augmentation in the training process. The top-5 recognition error rate on the **ILSVRC2012** validation set increases by about 2% (19.9% vs. 21.9%).

Most often, it is reasonable to pre-process the testing image according to the way of network training. For the **AlexNet**, we simply resize each image to 227×227 pixels and feed it into the network. In the original testing process (Krizhevsky et al, 2012), the image is resized to 256×256 and five sub-images are cropped at different positions and the average response is computed. While this strategy improves the accuracy consistently, we do not use it so that the feature extraction stage is accelerated. For the **VGGNet**-16 and **VGGNet**-19, we maximally preserve the aspect ratio of the input image, constrain the width and height divisible by 32 (the down-sampling rate), and the number of pixels is approximately 512^2. Such a strategy improves the performance of deep features significantly, compared to resizing all images to 224×224 pixels. After the neuron responses are computed, we extract the features from

Model	**C256**	**L-21**	**I-67**	**S-397**	**P-37**	**A-100**	**F-102**	**B-200**
AlexNet (w/o AUGM), ORIG	67.69	93.81	53.91	41.01	76.95	44.68	84.56	43.43
AlexNet (w/o AUGM), AVG	70.39	95.74	58.10	44.47	79.60	52.74	87.17	47.98
AlexNet (w/o AUGM), MAX	70.17	95.64	57.77	44.19	79.40	53.06	86.88	47.82
AlexNet (w/ AUGM), ORIG	70.48	95.07	57.78	44.77	80.85	49.34	87.27	47.17
AlexNet (w/ AUGM), AVG	71.75	95.55	59.76	46.42	81.79	53.89	88.34	49.28
AlexNet (w/ AUGM), MAX	71.57	95.50	59.45	46.24	81.55	53.86	88.26	49.15
VGGNet-16 (w/ AUGM), ORIG	82.69	95.71	75.78	60.43	93.09	67.18	93.69	71.62
VGGNet-16 (w/ AUGM), AVG	83.09	96.02	76.06	61.50	93.31	68.20	94.01	72.66
VGGNet-16 (w/ AUGM), MAX	83.12	95.83	75.93	61.39	93.25	68.37	93.97	72.73
VGGNet-19 (w/ AUGM), ORIG	83.51	95.10	75.49	61.30	93.10	68.20	93.57	71.70
VGGNet-19 (w/ AUGM), AVG	83.90	95.07	75.93	62.40	93.17	69.31	93.83	72.55
VGGNet-19 (w/ AUGM), MAX	83.90	94.98	75.83	62.25	93.12	69.44	93.84	72.59
SIFT+LCS, original	58.77	93.64	63.17	48.35	60.24	74.61	83.53	47.61
Max-SIFT+LCS	59.21	94.13	64.12	49.39	62.80	77.54	85.82	49.93
SIFT+LCS, w/ RIDE	60.25	94.71	64.93	50.12	63.49	78.92	86.45	50.81
Chatfield *et.al* (Chatfield et al, 2014)	77.61	–	–	–	–	–	–	–
Donahue *et.al* (Donahue et al, 2014)	–	–	–	40.94	–	–	–	64.96
Razavian *et.al.* (Razavian et al, 2014)	–	–	69.0	–	–	–	86.8	61.8
Zeiler *et.al* (Zeiler and Fergus, 2014)	74.2	–	–	–	–	–	–	–
Zhou *et.al.* (Zhou et al, 2014)	–	–	69.0	54.3	–	–	–	–
Krause *et.al.* (Krause et al, 2015)	–	–	–	–	–	–	–	82.8
Lin *et.al.* (Lin et al, 2015)	–	–	–	–	–	–	–	80.26
Qian *et.al.* (Qian et al, 2015)	–	–	–	–	81.18	–	89.45	67.86
Xie *et.al.* (Xie et al, 2015a)	–	94.71	70.13	54.87	90.03	–	86.82	62.02

Table 5: Classification accuracy (%) without or with reversal-invariant deep features. We also list the results of the BoF model (please refer to Table 3) and several recent works using deep features for comparison. All our results are obtained with the *fc-6* features (4096D) after ReLU activation.

each layer by average-pooling over all spatial positions. Throughout the rest part, we use the features extracted from the *fc-6* layer, activated by ReLU (Krizhevsky et al, 2012). These feature vectors are ℓ_2-normalized and sent to LIBLINEAR (Fan et al, 2008), a scalable SVM implementation, with the slacking parameter $C = 10$. Averaged accuracy by category is reported. Results are summarized in Table 5.

5.1.3 Discussions

First, it is obvious that feature quality, reflected by classification accuracy, is improved with data augmentation techniques, either in the training stage (reversing each training sample with the probability of 50%) or in the testing stage (computing the average or maximal neuron responses on an image and its reversed copy), which reveals the importance or reversal invariance in training CNN models and transferring CNN features. In most cases, Avg-Deep works slightly better than Max-Deep, whereas Max-Deep is often faster, since the feature vector contains more 0-entries in this situation.

Let us take the results produced by the **AlexNet** as an example. On the one hand, when the network is trained with both original and reversed samples, the validation accuracy on **ILSVRC2012** is improved by about 2%, and, consequently, consistent accuracy gain on each dataset is obtained. On the other hand, both Avg-Deep and Max-Deep boost the classification accuracy, sometimes even by a large margin, *e.g.*, more than 8% on the **Aircraft-100** dataset. Even when the network is trained with data augmentation, Avg-Deep and Max-Deep still improve the classification rate consistently, although the gain becomes relatively smaller (approximately 2% on the **Aircraft-100** dataset) due to the marginal effect. Considering that in most cases the baseline is already high, meanwhile both Avg-Deep and Max-Deep are extremely easy to implement, the accuracy gain is significant yet effortless to get.

To compare with the BoF model with handcrafted descriptors, we also copy a part of Table 3 here. We can see that, in most cases, deep features outperform BoF significantly, except in the **Aircraft-100** dataset: this set contains 100 *aircraft* models which are rigid (suitable for handcrafted descriptors) and do not appear in the pre-training set (the **ILSVRC2012** dataset) of the deep networks. The BoF model obtains higher accuracy only in this dataset. In contrast, in the **Pet-37** dataset, all the objects (*cats* or *dogs*) are deformable and the pre-training set contains a lot of these concepts, therefore the performance of deep features is dominant to that of the BoF model. Finally, we observe that the reported

accuracy on the **Bird-200** dataset is inferior to some recent publications, mainly because we do not use part-based models, which are crucial for classifying birds.

It is instructive to note that the accuracy gain brought by reversal invariance differs from case to case. For example, on the **Aircraft-100** and **Bird-200** datasets, the accuracy gain is impressive ($> 1\%$ using **VGGNet-19**), however in the **LandUse-21** and **Pet-37** datasets, it is less significant ($< 0.2\%$). The reason lies in the intrinsic property of the datasets and their relationship with the pre-training data. The *orientation* of an *aircraft* or a *bird* is more significant, and also more meaningful in visual recognition, than that of a *scene* captured from the sky. Moreover, all the above networks are pre-trained with the **ILSVRC2012** dataset, which contains a large number of *cat* and *dog* images (but no *aircraft* images), therefore it is easier to achieve reversal invariance when the testing image contains a similar visual concept.

The above experiments suggest that designing reversal invariance also helps to improve the quality of deep features. In what follows, we will design intrinsic reversal-invariant convolution modules, *i.e.*, Avg-Conv and Max-Conv, which lead to a more direct way of generating reversal-invariant deep features. These two strategies will be compared in Section 5.3.

5.2 Reversal-Invariant Convolution (RI-Conv)

5.2.1 Average-Conv and Max-Conv

As an alternative solution to post-processing deep features towards reversal invariance, we show that directly training a reversal-invariant deep CNN is possible and more efficient. Here, we call a CNN model *reversal-invariant* if it produces symmetric neural responses on each pair of symmetric images, *i.e.*, for an arbitrary image $\mathbf{I}$, taking $\mathbf{I}$ and $\mathbf{I}^{\mathrm{R}}$ as the input, the neuron responses on each layer of the pre-trained network $\mathcal{M}$, *i.e.*, $\mathbf{f}_l(\mathbf{I};\mathcal{M})$ and $\mathbf{f}_l\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$, are symmetric to each other: $\mathbf{f}_l^{\mathrm{R}}(\mathbf{I};\mathcal{M}) = \mathbf{f}_l\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$. In such a network, when we extract features on a fully-connected layer (*e.g.*, *fc-6* in the **AlexNet**), the original and reversed outputs are exactly the same since the spatial resolution is 1×1. If the features are extracted on an earlier layer (*e.g.*, *conv-5* in the **AlexNet**), we can also achieve reversal invariance by performing average-pooling or max-pooling the responses over all the spatial locations, similar to the strategy used in Section 5.1 and some previous publications (He et al, 2015).

The key to constructing a reversal-invariant CNN model is to guarantee that all the network layers are performing *symmetric* operations. Among the frequently used network operations (*e.g.*, convolution, pooling, normalization, non-linear activation, *etc.*), only convolution is non-symmetric, *i.e.*, a local patch and its reversed copy may produce different convolution outputs. We aim at designing a new *reversal-invariant* convolution operation to replace the original one.

Mathematically, let l be the index of a convolution layer with K_l convolution kernels, and $\mathbf{f}_{l-1} \doteq \mathbf{f}_{l-1}(\mathbf{I};\mathcal{M})$ is the input of the l-th layer. $\boldsymbol{\theta}_l \in \mathbb{R}^{W_l \times H_l \times K_l}$ and $\mathbf{b}_l \in \mathbb{R}^{K_l}$ are the weighting and bias parameters, respectively. The convolution operation takes each small patch $\mathbf{f}_{l-1}^{(a,b)}$ with the same spatial scale as the kernels, computes its inner-product with each kernel, and adds the bias to the result. For the k-th kernel, $k = 1, 2, \ldots, K_l$, we have:

$$f_l^{(a,b,k)}(\mathbf{I};\mathcal{M}) = \left\langle \mathbf{f}_{l-1}^{(a,b)}, \boldsymbol{\theta}_l^{(k)} \right\rangle + b_l^{(k)}, \tag{5}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner-product operation.

Inspired by the reversal-invariant deep features, reversal invariance is achieved if we perform a symmetric operation on the neuron responses from a patch and its reversed copy. Again, we take the element-wise average and maximal responses, leading to the **Avg-Conv** and the **Max-Conv** formualtion:

$$\begin{aligned} r_{l,\mathrm{AVG}}^{(a,b,k)}(\mathbf{I};\mathcal{M}) &= \frac{1}{2}\left[f_l^{(a,b,k)}(\mathbf{I};\mathcal{M}) + f_l^{(a,b,k)}\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)\right] \\ &= \frac{1}{2}\left[\left\langle \mathbf{f}_{l-1}^{(a,b)}, \boldsymbol{\theta}_l^{(k)} \right\rangle + \left\langle \mathbf{f}_{l-1}^{(a,b),\mathrm{R}}, \boldsymbol{\theta}_l^{(k)} \right\rangle\right] + b_l^{(k)} \\ &= \left\langle \frac{1}{2}\left[\mathbf{f}_{l-1}^{(a,b)} + \mathbf{f}_{l-1}^{(a,b),\mathrm{R}}\right], \boldsymbol{\theta}_l^{(k)} \right\rangle + b_l^{(k)}, \end{aligned} \tag{6}$$

$$\begin{aligned} r_{l,\mathrm{MAX}}^{(a,b,k)}(\mathbf{I};\mathcal{M}) &= \max\left\{f_l^{(a,b,k)}(\mathbf{I};\mathcal{M}), f_l^{(a,b,k)}\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)\right\} \\ &= \max\left\{\left\langle \mathbf{f}_{l-1}^{(a,b)}, \boldsymbol{\theta}_l^{(k)} \right\rangle, \left\langle \mathbf{f}_{l-1}^{(a,b),\mathrm{R}}, \boldsymbol{\theta}_l^{(k)} \right\rangle\right\} + b_l^{(k)}. \end{aligned} \tag{7}$$

Since Avg-Conv and Max-Conv simply perform the corresponding pooling operation on two convoluted data blobs, it is straightforward to derive the formula of back-propagation. In the case of Avg-Conv, we can accelerate both forward-propagation and back-propagation by modifying the input data (the original input $\mathbf{f}_{l-1}^{(a,b)}$ is replaced by the average of $\mathbf{f}_{l-1}^{(a,b)}$ and $\mathbf{f}_{l-1}^{(a,b),\mathrm{R}}$), thus the time-consuming convolution process is performed only once. In the case of Max-Conv, we need to create a mask blob to store the index of forward-propagated units, as in max-pooling layers.

In what follows, we will plug the reversal-invariant convolution modules into the conventional CNN models. We name a CNN model **RI-CNN** if all the convolution layers in it, including the fully-connected layers, are reversal-invariant. We start with discussing its property of reversal invariance, and the cooperation with data augmentation strategies.

5.2.2 Reversal Invariance and Data Augmentation

It is obvious that both Avg-Conv and Max-Conv are symmetric operations. We prove that an RI-CNN is reversal-invariant, *i.e.*, the feature vectors extracted from an image and its reversed copy are identical.

We use mathematical induction, with the starting point that an image and its reversed copy are symmetric to each other, *i.e.*, $\mathbf{f}_0^{\mathrm{R}}(\mathbf{I};\mathcal{M}) = \mathbf{f}_0\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$. Now, given that $\mathbf{f}_{l-1}^{\mathrm{R}}(\mathbf{I};\mathcal{M}) = \mathbf{f}_{l-1}\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$, we derive that $\mathbf{f}_l^{\mathrm{R}}(\mathbf{I};\mathcal{M}) = \mathbf{f}_l\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$, if both of them are computed with a reversal-invariant convolution operation on the l-th layer. For this, we assume that when padding (increasing data width/height with 0-valued stripes) is used, the left and right padding width must be the same, so that the **geometric symmetry** is guaranteed.

Consider a patch $\mathbf{f}_{l-1}^{(a,b)}(\mathbf{I};\mathcal{M})$. According to symmetry, we have $\mathbf{f}_{l-1}^{(a,b),\mathrm{R}}(\mathbf{I};\mathcal{M}) = \mathbf{f}_{l-1}^{(W_{l-1}-a-1,b)}\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$, where $(W_{l-1}-a-1,b)$ is the left-right symmetric position to (a,b). These two patches are fed into the k-th convolution kernel $\boldsymbol{\theta}_k$, and the outputs are $f_l^{(a,b,k)}(\mathbf{I};\mathcal{M})$ and $f_l^{(W_{l-1}-a-1,b,k)}\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$. These two scalars equal to each other since both Avg-Conv and Max-Conv are symmetric, thus the neuron responses on the l-th layer are also symmetric: $\mathbf{f}_l^{\mathrm{R}}(\mathbf{I};\mathcal{M}) = \mathbf{f}_l\big(\mathbf{I}^{\mathrm{R}};\mathcal{M}\big)$. This finishes the induction, *i.e.*, the neuron responses on each layer are symmetric.

We point out that such a good property in feature extraction can be a significant shortcoming in the network training process, since an RI-CNN model suffers from the difficulty to cooperate with "reversal data augmentation". Here by reversal data augmentation we mean to reverse each training sample with the probability of 50%. As an RI-CNN model generates exactly the same (symmetric) neuron responses for an image and its reversed copy, these two training samples actually produce the same gradients with respect to the network parameters on each layer. Consequently, reversing a training image cannot provide any "new" information to the network training process. As we shall see in Section 5.2.5, using reversal-invariant convolution operations increases the capacity of the CNN model, the decrease of training data may cause over-fitting, which harms the generalization ability of the model.

To deal with, we intentionally damage the reversal-invariant property of the network in the training process. For this, we crop the training image into a smaller size, so that the *geometric symmetry* does not hold any more. Taking the **AlexNet** as an example. The original input image size is 227×227, in which geometric symmetry holds on each convolutional/pooling layer. If the size becomes $S'\times S'$ where S' is a little smaller than 227, then in some layers, the padding margin on the left side is not the same as that on the right side. By the way, S' shall be at least 199, so that the input of the *fc-6* layer still has a spatial resolution of 6×6. In practise, we simply use $S'=199$, so that we can generate as many training images as possible. As we shall see in Section 5.2.4, this strategy slightly improves the baseline accuracy.

CIFAR10	w/o AUGM	w/ AUGM
LeNet	18.11 ± 0.20	16.99 ± 0.22
LeNet-AVG	21.01 ± 0.35	20.99 ± 0.26
LeNet-MAX	16.93 ± 0.18	$\mathbf{16.64}\pm0.17$
CIFAR100	w/o AUGM	w/ AUGM
LeNet	46.08 ± 0.26	44.55 ± 0.10
LeNet-AVG	47.79 ± 0.41	47.55 ± 0.31
LeNet-MAX	43.90 ± 0.19	$\mathbf{43.65}\pm0.16$

Table 6: **CIFAR** classification error rate (%) with respect to different training strategies.

5.2.3 CIFAR Experiments

The **CIFAR10** and **CIFAR100** datasets (Krizhevsky and Hinton, 2009) are subsets of the 80 million tiny images database (Torralba et al, 2008). Both of them have 50000 training samples and 10000 testing samples, each of which is a 32×32 color image, uniformly distributed among the categories (they have 10 and 100 categories, respectively). It is a popular dataset for training relatively small-scale neural networks for simple recognition tasks.

We use a modified version of the **LeNet** (LeCun et al, 1990). A $32\times32\times3$ image is passed through three units consisting of convolution, ReLU and max-pooling operations. Using abbreviation, the network configuration for **CIFAR10** can be written as:

`[C5(S1P2)@32-MP3(S2)]-[C5(S1P2)@32-MP3(S2)]-[C5(S1P2)@64-MP3(S2)]-FC10.`

On **CIFAR100**, we replace the final layer as `FC100` in order to categorize 100 classes. A 2-pixel wide padding is added to each convolution operation so that the width and height of the data remain unchanged. We do not produce multiple sizes of input images, since the **LeNet** is not symmetric itself: on each pooling layer, the left padding margin is 0 while the right margin is 1. We apply 120 training epoches with the learning rate 10^{-3}, followed by 20 epoches with the learning rate 10^{-4}, and another 10 epoches with the learning rate 10^{-5}.

We train six different models individually, *i.e.*, training a network with the original version of convolution, Avg-Conv or Max-Conv (three choices), and using data augmentation (probabilistic training image reversal) or not (two choices). We name these models as **LeNet**,

LeNet-AUGM ("AUGM" for augmentation), **LeNet**-AVG, **LeNet**-AVG-AUGM, **LeNet**-MAX and **LeNet**-MAX-AUGM, respectively. For instance, **LeNet**-MAX indicates the network with Max-Conv but without data augmentation. To reveal the statistics significance, we train 5 independent models in each case, and report the average accuracy.

Results are summarized in Table 6. One can observe similar phenomena on both datasets. First, Avg-Conv causes dramatic accuracy drop, and we will analyze the reason of bad performance in Section 5.2.5. On the other side, data augmentation and Max-Conv improve the recognition accuracy consistently, In the **CIFAR10** dataset, both data augmentation and Max-Conv boost the accuracy by about 1%, and these two strategies cooperate with each other to beat the baseline by 1.5%. In the **CIFAR100** dataset, Max-Conv alone contributes a more-than-2% accuracy gain, which is higher than the 1.5% gain by data augmentation, and the cooperation obtains a nearly 2.5% gain. As a last note, the improvement on **CIFAR100** is much larger than that on **CIFAR10**, which indicates that **CIFAR100** is a more difficult dataset (with more categories), and that Max-Conv enlarges the capacity of **LeNet** to fit this challenging recognition task.

5.2.4 ILSVRC2012 Classification Experiments

We also evaluate our model on the **ILSVRC2012** classification dataset (Russakovsky et al, 2015), a subset of the **ImageNet** database (Deng et al, 2009) which contains 1000 object categories. The training set, validation set and testing set contain 1.3M, 50K and 150K images, respectively. We use the **AlexNet** (provided by the **CAFFE** library (Jia et al, 2014), sometimes referred to as the **CaffeNet**). The input image is of size 199×199, randomly cropped from the original 256×256 image (see Section 5.2.2). The **AlexNet** structure is abbreviated as:

```
[C11(S4)@96-MP3(S2)-LRN]-[C5(S1P2)@256-
MP3(S2)-LRN]-[C3(S1P1)@384]-[C3(S1P1)@384]-
[C3(S1P1)@256-MP3(S2)]-FC4096-D0.5-FC4096-
D0.5-FC1000.
```

Following the setting of **CAFFE**, a total of 450000 mini-batches (approximately 90 epoches) are used for training, each of which has 256 image samples, with the initial learning rate 0.01, momentum 0.9 and weight decay 0.0005. The learning rate is decreased to 1/10 after every 100000 mini-batches.

We individually train four models, *i.e.*, using original convolution or Max-Conv, using data augmentation or not. Similarly, we name these variations as **AlexNet**, **AlexNet**-AUGM, **AlexNet**-MAX and **AlexNet**-MAX-AUGM, respectively. Considering the large computational costs, we only train two individual networks for each setting. We do not train models based on Avg-Conv according to the dramatic accuracy drop in **CIFAR** experiments.

ILSVRC2012, top-1	w/o AUGM	w/ AUGM
AlexNet	43.05 ± 0.19	42.52 ± 0.17
AlexNet-MAX	42.16 ± 0.05	$\mathbf{42.10} \pm 0.07$
ILSVRC2012, top-5	w/o AUGM	w/ AUGM
AlexNet	20.62 ± 0.08	19.52 ± 0.05
AlexNet-MAX	19.42 ± 0.03	$\mathbf{19.12} \pm 0.07$

Table 7: **ILSVRC2012** classification error rate (%) with respect to different training strategies.

Result are summarized in Table 7. As we have slightly modified the data augmentation strategy, the baseline performance (80.48% top-5 accuracy) is slightly better than that reported using the standard setting (approximately 80.1% top-5 accuracy [1]). With Max-Conv, the top-5 accuracy is boosted to 80.88%, which show that Max-Conv and data augmentation cooperate to improve the recognition performance. We emphasize that the 0.40% accuracy gain is not small, given that the network structure is unchanged. Meanwhile, the conclusions drawn in **CIFAR** experiments also hold in this large-scale image recognition task.

5.2.5 Discussions

The success of data augmentation and Max-Conv implies that it is instructive to force the network to learn reversal invariance by constructing corresponding specific structures. This part provides several discussions based on the experimental results.

We first provide another perspective on the behavior of reversal-invariant convolution. Let us consider a convolution layer (the l-th layer), in which we compute the inner product of a patch $\mathbf{f}_{l-1}^{(a,b)}$ (probably together its reversed copy) and each of the K_l convolution kernels $\boldsymbol{\theta}_k$, $k = 1, 2, \ldots, K_l$. Since inner production measures the similarity between $\mathbf{f}_{l-1}^{(a,b)}$ and $\boldsymbol{\theta}_k$. the patches with similar appearance to $\boldsymbol{\theta}_k$ will get a significant neuron response. In this situation, $\boldsymbol{\theta}_k$ behaves like a *codeword* and K_l is the *codebook* size. Meanwhile, we note that image *patterns* are often left-right asymmetric, *e.g.*, a *slash* may have either a positive or a negative angle. Without reversal-invariant convolution, we need two different codewords to encode a visual pattern and its

[1] `https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val`

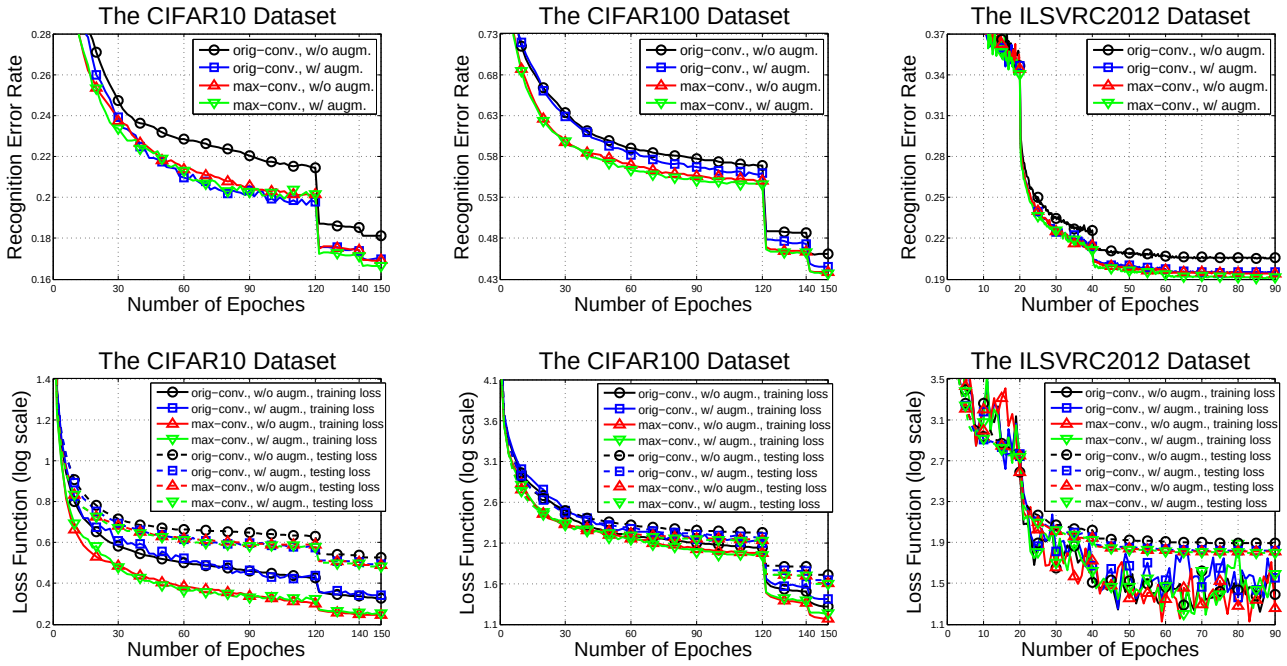

Fig. 7: Error rate curves and training/testing loss curves on the **CIFAR** datasets and the **ILSVRC2012** dataset. We report top-1 and top-5 error rates in **CIFAR** and **ILSVRC2012**, respectively.

reversed version, which significantly decreases the capacity of the limited codebook size (K_l), and, consequently, the capacity of the network. Reversal-invariant convolution brings the opportunity for each local patch to be compared with a codeword and its reversed copy, so that equivalently, we need only one codeword to store a visual pattern and its reversed version.

Now, it is easy to see the difference between Avg-Conv and Max-Conv. Both of them compute the similarity between each codeword and each original/reversed local patch, after that, Avg-Conv considers the average response and Max-Conv gets the larger response. Which means that, in the context of average-convolution, a local patch can get a high response if it is similar to both the codeword itself and its reversed copy, which is not reasonable since image *patterns* are often left-right asymmetric. In opposite, Max-Conv animates those local patches which are similar to either the original or reversed codeword. Therefore in **CIFAR** experiments, Avg-Conv causes dramatic accuracy drop, while Max-Conv boosts the performance significantly. This is the reason why we do not train Avg-Conv models in **ILSVRC2012** experiments.

To take a closer observation on the network training with data augmentation and/or reversal-invariant convolution, we plot the testing error rate as well as the training/testing loss with respect to the number of training epoches. Note that both strategies augment the training data, *i.e.*, data augmentation implicitly increases the number of training samples, meanwhile reversal-invariance convolution makes it possible to "see" more variations of local patches. From the results shown in Figure 7, we can see that using data augmentation slows down the network training since it introduce regularization to the training process. However, with Max-Conv, network training converges faster since the network capacity is significantly enlarged. These two strategies cooperate with each other to make full use of the enlarged model capacity, meanwhile prevent overfitting.

5.3 Model Comparison

We compare the two strategies discussed in this section, *i.e.*, training a non-reversal-invariant deep network and post-processing deep features for reversal invariance, and training a reversal-invariant deep network which generates reversal-invariant deep features directly.

We use the networks trained in the previous experiments, namely **AlexNet**-AUGM and **AlexNet**-MAX-AUGM, to extract deep features on the image classification dataset used in Section 5.1. Results are summarized in Table 8. We observe that, features extracted from **AlexNet**-MAX produce consistently higher accuracy than the original features extracted from **AlexNet**, comparable to the Avg-Deep and Max-Deep features

Model	**C256**	**L-21**	**I-67**	**S-397**	**P-37**	**A-100**	**F-102**	**B-200**
AlexNet, ORIG feature	70.75	95.14	58.04	45.12	81.02	49.89	87.39	47.53
AlexNet, AVG feature	71.97	95.60	60.01	46.64	81.98	54.10	88.40	49.53
AlexNet, MAX feature	71.81	95.55	59.77	46.47	81.73	54.03	88.29	49.42
AlexNet-MAX, ORIG feature	71.78	95.67	59.91	46.47	81.92	54.11	88.17	49.55

Table 8: Classification accuracy (%) comparison with deep features extracted using different strategies. Note that the first part of this table is not the same as in Table 5, since we have used a different way of training **AlexNet** (see Section 5.2.2). With Max-Conv, we do not need to post-process the feature vector since it is naturally reversal-invariant.

from **AlexNet**. Therefore, we can observe the benefit of designing intrinsically reversal-invariant modules.

5.4 Summary

In this part, we generalize the idea of reversal-invariant representation from the BoF model to deep CNNs, and verify that reversal invariance is also important in both deep feature extraction and deep network training. We propose two effective algorithms (**RI-Deep** and **RI-Conv**), both of which are easy to implement. First, computing neuron responses on a testing image as well as its reversed version makes it possible to extract reversal-invariant deep features from a pre-trained network which is not reversal-invariant. Second, a small modification in convolution leads to a deep network which is intrinsically reversal-invariant, which has larger capacity yet unchanged complexity, meanwhile makes the feature extraction more effective. Reversal-invariant convolution also cooperates well with data augmentation, creating the possibility of applying deep neural networks to even larger databases.

6 Conclusions

It is important to consider reversal invariance in order to achieve more robust image representation, but conventional BoF and CNN models often lack of an explicit implementation of reversal invariance. This paper presents a basic idea that designs reversal-invariant local patterns, such as **Max-SIFT** and **RIDE** (local descriptors), **RI-Deep** (deep features) and **RI-Conv** (convolution), so that reversal invariance is guaranteed in the representation based on the BoF and CNN models. The proposed algorithms are very easy to implement yet efficient to carry out, meanwhile producing consistent accuracy improvement. The success of our algorithms also reveals that designing invariance directly is often more effective than using data augmentation, and that these two strategies can often cooperate with each other towards better image representation.

A Orientation Estimation of Dense SIFT

In this section, we aim at proving an approximated estimation of SIFT orientation based on its local gradient values. The approximation is used in Section 4.2.1 of the main article.

A.1 The Implementation of SIFT

The implementation of SIFT is based on the original paper (Lowe, 2004). In this subsection, we briefly review the process of orientation assignment and descriptor representation. Part of the statements refer to (Lowe, 2004).

First let us assume that the assignment of descriptor scale is finished, which fits the case of dense sampling (Bosch et al, 2006) where all the descriptors have the same, fixed window size. Denote an image as $\mathbf{I} = [a(x,y)]_{W\times H}$. The gradient magnitude, $m(x,y)$, and orientation, $\theta(x,y)$, is pre-computed for each pixel:

$$\begin{cases} m(x,y) = \left[\Delta_x(x,y)^2 + \Delta_y(x,y)^2\right]^{1/2} \\ \theta(x,y) = \arctan\left[\Delta_y(x,y)\,/\Delta_x(x,y)\right] \end{cases}, \tag{8}$$

in which $\Delta_x(x,y)$ and $\Delta_y(x,y)$ are defined as:

$$\begin{cases} \Delta_x(x,y) = a(x+1,y) - a(x-1,y) \\ \Delta_y(x,y) = a(x,y+1) - a(x,y-1) \end{cases}. \tag{9}$$

The magnitude and orientation on each pixel are then used to estimate the dominant orientation of that descriptor. An orientation histogram is constructed using the gradient orientation of the pixels within a region around the keypoint. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a smoothing parameter σ that is 1.5 times that of the scale of the keypoint. Peaks in the orientation histogram correspond to *dominant* orientations of local gradients. The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation. Therefore, for locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale but different orientations.

The above method works well on image matching and retrieval (Lowe, 2004), but we do not need to assign multiple orientations for a descriptor in the classification tasks. As an alternation, we can also estimate a unique *accumulated* orientation using the following method. Every gradient magnitude is decomposed along both x and y axes, *i.e.*,

$$\begin{cases} m_x(x,y) = m(x,y) \times \cos\theta(x,y) \\ m_y(x,y) = m(x,y) \times \sin\theta(x,y) \end{cases}, \tag{10}$$

and all the decomposed components are accumulated on x and y axes, respectively:

$$\begin{cases} G_x = \sum_{x,y} m_x(x,y) \\ G_y = \sum_{x,y} m_y(x,y) \end{cases} . \tag{11}$$

Finally we get a 2-D vector $\mathbf{G} = (G_x, G_y)^\top$ indicating the orientation of that descriptor.

Of course, we can also follow the orientation assignment of original SIFT implementation (Lowe, 2004). In practise, we have implemented RIDE with both *dominant* and *accumulated* orientations, and found that the latter one is slightly better. Another reason why we prefer the *accumulated* orientation is that it is a continuous value in $[0, 2\pi)$, which makes it easier for us to design the RIDE-8 algorithm.

In descriptor representation, we inherit $m(x,y)$ and $\theta(x,y)$ values of each pixel. The implementation of dense SIFT (Vedaldi and Fulkerson, 2010) does not rotate the descriptor region. The region of a descriptor is partitioned into 4×4 grids, and an 8-bin orientation histogram is constructed in each grid. The central orientation value of the j-th bin is $\theta_j = j\pi/4$, $j = 0, 1, \ldots, 7$. Then the gradient magnitude of each pixel is then trilinearly quantized onto at most two bins. By trilinear we mean that if the orientation of a pixel, $\theta(x,y)$, is closest to two standard orientation, say, $\theta_a < \theta(x,y) < \theta_b$, then the coefficients assigned to the bins are:

$$\begin{cases} m_a = m(x,y) \times \dfrac{\theta_b - \theta(x,y)}{\theta_b - \theta_a} \\ m_b = m(x,y) \times \dfrac{\theta(x,y) - \theta_a}{\theta_b - \theta_a} \end{cases} . \tag{12}$$

An 8-dimensional orientation histogram is thereafter obtained in each of the 4×4 grids. Finally, the 128-dimensional descriptor is constructed by concatenating the histogram vectors from all 4×4 grids.

A.2 Orientation Estimation

The main goal of this part is to prove the next theorem for orientation approximation:

Theorem: Given a densely sampled SIFT descriptor $\mathbf{d} = (d_k, \theta_k)_{k=1,2,\ldots,128}$, where d_k and θ_k are the gradient value and the histogram orientation for the k-th dimension, respectively. Its *accumulated* orientation θ approximately satisfies:

$$\tan\theta = \frac{G_y(x,y)}{G_x(x,y)} = \frac{\sum_{x,y} m_y(x,y)}{\sum_{x,y} m_x(x,y)} \approx \frac{\sum_k d_k \sin\theta_k}{\sum_k d_k \cos\theta_k}. \tag{13}$$

For this, we only need to prove the following lemma:

Lemma: When a gradient value (m, θ) with an arbitrary orientation is quantized as (m_a, θ_a) and (m_b, θ_b) $(\theta_a < \theta < \theta_b)$ with the trilinear interpolation, *i.e.*, using (12):

$$\begin{cases} m_a = m \times \dfrac{\theta_b - \theta}{\theta_b - \theta_a} \\ m_b = m \times \dfrac{\theta - \theta_a}{\theta_b - \theta_a} \end{cases} , \tag{14}$$

the impacts on SIFT descriptor representation, before and after quantization, are approximately the same, *i.e.*,

$$\begin{cases} m\cos\theta \approx m_a \cos\theta_a + m_b \cos\theta_b \\ m\sin\theta \approx m_a \sin\theta_a + m_b \sin\theta_b \end{cases} . \tag{15}$$

Proof: we only prove the first formula, since the proof of the other one is very similar.

Using (14) to substitute m_a and m_b in (15) yields:

$$\begin{aligned} & m_a \cos\theta_a + m_b \cos\theta_b \\ &= m \times \frac{\theta_b - \theta}{\theta_b - \theta_a} \times \cos\theta_a + m \times \frac{\theta - \theta_a}{\theta_b - \theta_a} \times \cos\theta_b \\ &= m \times \left(\frac{\theta_b - \theta}{\theta_b - \theta_a} \times \cos\theta_a + \frac{\theta - \theta_a}{\theta_b - \theta_a} \times \cos\theta_b \right). \end{aligned} \tag{16}$$

Let us make the approximation that:

$$\begin{cases} \dfrac{\theta_b - \theta}{\theta_b - \theta_a} \approx \dfrac{\sin(\theta_b - \theta)}{\sin(\theta_b - \theta_a)} \\ \dfrac{\theta - \theta_a}{\theta_b - \theta_a} \approx \dfrac{\sin(\theta - \theta_a)}{\sin(\theta_b - \theta_a)} \end{cases} , \tag{17}$$

thus (16) becomes:

$$\begin{aligned} & m_a \cos\theta_a + m_b \cos\theta_b \\ &= m \times \frac{\theta_b - \theta}{\theta_b - \theta_a} \times \cos\theta_a + m \times \frac{\theta - \theta_a}{\theta_b - \theta_a} \times \cos\theta_b \\ &\approx m \times \left[\frac{\sin(\theta_b - \theta)}{\sin(\theta_b - \theta_a)} \times \cos\theta_a + \frac{\sin(\theta - \theta_a)}{\sin(\theta_b - \theta_a)} \times \cos\theta_b \right] \\ &= \frac{m \times [\sin(\theta_b - \theta)\cos\theta_a + \sin(\theta - \theta_a)\cos\theta_b]}{\sin(\theta_b - \theta_a)} \\ &= \frac{m \times (\sin\theta_b \cos\theta \cos\theta_a - \cos\theta \sin\theta_a \cos\theta_b)}{\sin(\theta_b - \theta_a)} \\ &= \frac{m \times \cos\theta \times (\sin\theta_b \cos\theta_a - \cos\theta_b \sin\theta_a)}{\sin(\theta_b - \theta_a)} \\ &= m\cos\theta, \end{aligned}$$

which finishes the proof. □

We provide a discussion on the approximation (17). Given that $\theta_b - \theta_a = \pi/4$, the maximum relative error of the approximation is less than 11%. Let us define $f(x) = \frac{\sin x}{x}$. Since $\lim_{x\to 0} f(x) = 1$ and $f(x)$ is a monotonically decreasing function, large errors of (17) appear when $\theta_b - \theta$ or $\theta - \theta_a$ is quite small, in which case the m_a or m_b is also quite small thus the absolute estimation error is ignorable. Therefore, we can conclude that the approximation (17) is reasonable.

B Generalized RIDE: RIDE-4 and RIDE-8

In this section, we provide a detailed discussion of generalizing RIDE to dealing with more types of reversal and rotation invariance. It is a supplementary explanation to Section 4.2.3 of the main article.

B.1 RIDE-2, RIDE-4 and RIDE-8

We start from an alternative description of the RIDE-2, RIDE-4 and RIDE-8 algorithms.

Recall that we have computed a 2-D global gradient vector $\mathbf{G} = (G_x, G_y)^\top$, in which G_x and G_y estimate the horizontal and vertical orientation of a descriptor, respectively. If it is constrained that $G_x \geqslant 0$ holds for a descriptor $\mathbf{d}$, we need to generate a left-right reversed version of that descriptor, $\mathbf{d}^{\mathrm{R}}$, and select the one in $\mathbf{d}$ and $\mathbf{d}^{\mathrm{R}}$ that satisfies $G_x \geqslant 0$. Such a descriptor, denoted as $r_2(\mathbf{d})$, is left-right reversal-invariant. If $G_x = 0$ for $\mathbf{d}$, both $\mathbf{d}$ and $\mathbf{d}^{\mathrm{R}}$ satisfy the condition. In such cases, we choose the one with the larger sequential lexicographic order.

If we need to achieve upside-down reversal invariance, the value G_y should also be constrained, *i.e.*, $G_y \geqslant 0$. We then

Algorithm	**Aircraft-100**-1	**Aircraft-100**-2	**Aircraft-100**-4	**Aircraft-100**-8
ORIG	**58.75**	48.52	39.33	25.11
RIDE-2	55.22	**55.22**	43.20	29.71
RIDE-4	47.44	47.44	**47.44**	35.41
RIDE-8	43.47	43.47	43.47	**43.47**

Table 9: Classification accuracy (%) of different versions of RIDE on different versions of the **Aircraft-100** dataset.

generate 3 other versions of a descriptor $\mathbf{d}$, namely $\mathbf{d}_0$, $\mathbf{d}_1$, $\mathbf{d}_2$ and $\mathbf{d}_3$, in which $\mathbf{d}_0$ is just $\mathbf{d}$, $\mathbf{d}_1$ is the left-right reversed version of $\mathbf{d}$, $\mathbf{d}_2$ is the upside-down reversed version of $\mathbf{d}$, and $\mathbf{d}_3$ is the left-right and upside-down reversed version of $\mathbf{d}$. Obviously, there exists at least one of them that satisfies both $G_x \geqslant 0$ and $G_y \geqslant 0$. If more than one candidates satisfy the conditions, we choose the one with the largest sequential lexicographic order. Such a descriptor, denoted as $r_4(\mathbf{d})$, is both left-right and upside-down invariant.

The last type of variant comes from rotating the descriptor by 90°. Adding the 90°-rotation option into left-right and upside-down reversals obtains up to 8 descriptor versions. We generate all these variants and select one from them by constraining $G_x \geqslant G_y \geqslant 0$, *i.e.*, $G_x \geqslant 0$, $G_y \geqslant 0$ and $G_x \geqslant G_y$. If more than one candidates satisfy the conditions, we choose the one with the largest sequential lexicographic order. Such a descriptor, denoted as $r_8(\mathbf{d})$, is invariant through all the reversal and rotation operations.

We provide an intuitive explanation of RIDE-2, RIDE-4 and RIDE-8 algorithms. All the reversal and rotation operations change the orientation of a descriptor correspondingly. RIDE-2, in which $G_x \geqslant 0$, limits the orientation to falling into an interval of a 180° range. This range is further shrunk into 90° in RIDE-4, and 45° in RIDE-8. A descriptor with **any** orientation can be aligned into the range with one or a few reversal or rotation variations, and in this way we cancel out the reversal and rotation operations and achieve the desired reversal invariance.

B.2 Experiments

We evaluate the original descriptors with RIDE-2, RIDE-4 and RIDE-8 on the **Aircraft-100** dataset (Maji et al, 2013). We use four different versions of the dataset. The **aligned** version, denoted as **Aircraft-100**-1, is the one in which all the objects are manually aligned to the right. Other three versions, denoted as **Aircraft-100**-2, **Aircraft-100**-4 and **Aircraft-100**-8, are generated by randomly assigning one of 2, 4 and 8 image transformations to each image in the aligned dataset. Here, 2 transformations include unchanged and the left-right reversal, 4 transformations are constructed by adding the option of upside-down reversal to 2 transformations, and 8 transformations are constructed by adding the option of 90° rotation to 4 transformations. The property of **Aircraft-100**-2 is very similar to the original (unaligned) version of the **Aircraft-100** dataset.

The basic setting follows what is used in the main article (Section 4.6.1). We only use the SIFT descriptor, and do not use spatial pyramids in the following experiments. The classification results are summarized in Table 9. One can observe that on the **Aircraft-100**-1 dataset, the system with original descriptors (**ORIG**) works best. After original descriptors are processed by RIDE, classification accuracy drops dramatically. The underlying reason is that RIDE harms the descriptive power of original descriptors by performing a one-of-many selection. The more candidates generated for selection, the heavier accuracy drop is observed.

However, in the case of **Aircraft-100**-2, **RIDE-2** works better than **ORIG**. This implies that **RIDE-2** captures the left-right reversal invariance. Although the descriptive power of SIFT is reduced, the benefit of reversal invariance is larger than the loss in descriptive power. However, when we use **RIDE-4** and **RIDE-8**, the descriptive power continues to drop but we do not obtain any new invariance, resulting in the accuracy drop from **RIDE-2** to both **RIDE-4** and **RIDE-8**. Similar results are also observed in the **Aircraft-100**-4 dataset, *i.e.*, **RIDE-4** is just enough to capture left-right and upside-down reversal. In **Aircraft-100**-8 dataset, all the reversal and rotation variance might be encountered, therefore **RIDE-8** produces the highest accuracy.

The above experiments verify that RIDE increases the robustness of descriptors but harms the descriptive power. According to Table 9, one type of reversal/rotation variance, if **not** captured, causes about 10% accuracy drop, meanwhile performing RIDE to capture an unnecessary invariance causes about 5% accuracy drop. Therefore it is not wise to cover those unnecessary types of invariance: **the best strategy is to take what we need.**

Consequently, we do not use **RIDE-4** and **RIDE-8** in all the experiments presented in the main article, since all the evaluated datasets, either on fine-grained object recognition or scene understanding, often do not contain upside-down reversed or 90°-rotated objects. **RIDE-2** produces the best classification accuracy in such cases.

References

Angelova A, Zhu S (2013) Efficient Object Detection and Segmentation for Fine-Grained Recognition. Computer Vision and Pattern Recognition

Arandjelovic R, Zisserman A (2012) Three Things Everyone Should Know to Improve Object Retrieval. Computer Vision and Pattern Recognition

Berg T, Belhumeur P (2013) POOF: Part-based One-vs-One Features for Fine-Grained Categorization, Face Verification, and Attribute Estimation. Computer Vision and Pattern Recognition

Bosch A, Zisserman A, Munoz X (2006) Scene Classification via pLSA. International Conference on Computer Vision

Calonder M, Lepetit V, Strecha C, Fua P (2010) BRIEF: Binary Robust Independent Elementary Features. European Conference on Computer Vision

Chai Y, Lempitsky V, Zisserman A (2013) Symbiotic Segmentation and Part Localization for Fine-Grained Categorization. International Conference on Computer Vision

Chatfield K, Lempitsky V, Vedaldi A, Zisserman A (2011) The Devil is in the Details: An Evaluation of Recent

Feature Encoding Methods. British Machine Vision Conference
Chatfield K, Simonyan K, Vedaldi A, Zisserman A (2014) Return of the Devil in the Details: Delving Deep into Convolutional Nets. British Machine Vision Conference
Csurka G, Dance C, Fan L, Willamowski J, Bray C (2004) Visual Categorization with Bags of Keypoints. Workshop on Statistical Learning in Computer Vision, European Conference on Computer Vision
Dalal N, Triggs B (2005) Histograms of Oriented Gradients for Human Detection. Computer Vision and Pattern Recognition
Deng J, Dong W, Socher R, Li L, Li K, Fei-Fei L (2009) ImageNet: A Large-Scale Hierarchical Image Database. Computer Vision and Pattern Recognition
Donahue J, Jia Y, Vinyals O, Hoffman J, Zhang N, Tzeng E, Darrell T (2014) DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. International Conference on Machine Learning
Fan R, Chang K, Hsieh C, Wang X, Lin C (2008) LIBLINEAR: A Library for Large Linear Classification. Journal of Machine Learning Research 9:1871–1874
Feng J, Ni B, Tian Q, Yan S (2011) Geometric Lp-norm Feature Pooling for Image Classification. Computer Vision and Pattern Recognition
Gavves E, Fernando B, Snoek C, Smeulders A, Tuytelaars T (2014) Local Alignments for Fine-Grained Categorization. International Journal on Computer Vision
Girshick R, Donahue J, Darrell T, Malik J (2014) Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. Computer Vision and Pattern Recognition
Grauman K, Darrell T (2005) The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. International Conference on Computer Vision
Griffin G (2007) Caltech-256 Object Category Dataset. Technical Report: CNS-TR-2007-001
Guo X, Cao X (2010) FIND: A Neat Flip Invariant Descriptor. International Conference on Patter Recognition
He K, Zhang X, Ren S, Sun J (2015) Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 37(9):1904–1916
Ioffe S, Szegedy C (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. International Conference on Machine Learning
Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) CAFFE: Convolutional Architecture for Fast Feature Embedding. ACM International Conference on Multimedia
Juneja M, Vedaldi A, Jawahar C, Zisserman A (2013) Blocks that Shout: Distinctive Parts for Scene Classification. Computer Vision and Pattern Recognition
Kobayashi T (2014) Dirichlet-based Histogram Feature Transform for Image Classification. Computer Vision and Pattern Recognition
Krause J, Jin H, Yang J, Fei-Fei L (2015) Fine-Grained Recognition without Part Annotations. Computer Vision and Pattern Recognition
Krizhevsky A, Hinton G (2009) Learning Multiple Layers of Features from Tiny Images. Technical Report, University of Toronto 1(4):7
Krizhevsky A, Sutskever I, Hinton G (2012) ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems
Lapin M, Schiele B, Hein M (2014) Scalable Multitask Representation Learning for Scene Classification. Computer Vision and Pattern Recognition
Lazebnik S, Schmid C, Ponce J (2006) Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. Computer Vision and Pattern Recognition
LeCun Y, Denker J, Henderson D, Howard R, Hubbard W, Jackel L (1990) Handwritten Digit Recognition with a Back-Propagation Network. Advances in Neural Information Processing Systems
Li L, Guo Y, Xie L, Kong X, Tian Q (2015) Fine-Grained Visual Categorization with Fine-Tuned Segmentation. International Conference on Image Processing
Lin D, Lu C, Liao R, Jia J (2014) Learning Important Spatial Pooling Regions for Scene Classification. Computer Vision and Pattern Recognition
Lin D, Shen X, Lu C, Jia J (2015) Deep LAC: Deep Localization, Alignment and Classification for Fine-grained Recognition. Computer Vision and Pattern Recognition
Lowe D (2004) Distinctive Image Features from Scale-Invariant Keypoints. International Journal on Computer Vision 60(2):91–110
Ma R, Chen J, Su Z (2010) MI-SIFT: Mirror and Inversion Invariant Generalization for SIFT Descriptor. Conference on Image and Video Retrieval
Maji S, Rahtu E, Kannala J, Blaschko M, Vedaldi A (2013) Fine-Grained Visual Classification of Aircraft. Technical Report
Matas J, Chum O, Urban M, Pajdla T (2004) Robust Wide-Baseline Stereo from Maximally Stable Extremal Regions. Image and Vision Computing 22(10):761–767
Murray N, Perronnin F (2014) Generalized Max Pooling. Computer Vision and Pattern Recognition
Nilsback M, Zisserman A (2008) Automated Flower Classification over a Large Number of Classes. International Conference on Computer Vision, Graphics & Image Processing
Parkhi O, Vedaldi A, Zisserman A, Jawahar C (2012) Cats and Dogs. Computer Vision and Pattern Recognition
Paulin M, Revaud J, Harchaoui Z, Perronnin F, Schmid C (2014) Transformation Pursuit for Image Classification. Computer Vision and Pattern Recognition
Perronnin F, Sánchez J, Mensink T (2010) Improving the Fisher Kernel for Large-scale Image Classification. European Conference on Computer Vision
Pu J, Jiang Y, Wang J, Xue X (2014) Which Looks Like Which: Exploring Inter-class Relationships in Fine-Grained Visual Categorization. European Conference on Computer Vision
Qian Q, Jin R, Zhu S, Lin Y (2015) Fine-Grained Visual Categorization via Multi-stage Metric Learning. Computer Vision and Pattern Recognition
Quattoni A, Torralba A (2009) Recognizing Indoor Scenes. Computer Vision and Pattern Recognition
Razavian A, Azizpour H, Sullivan J, Carlsson S (2014) CNN Features off-the-shelf: an Astounding Baseline for Recognition. Computer Vision and Pattern Recognition
Rublee E, Rabaud V, Konolige K, Bradski G (2011) ORB: an Efficient Alternative to SIFT or SURF. International Conference on Computer Vision
Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg A, Fei-Fei L (2015) ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (1):1–42

Sanchez J, Perronnin F, Mensink T, Verbeek J (2013) Image Classification with the Fisher Vector: Theory and Practice. International Journal on Computer Vision 105(3):222–245

van de Sande K, Gevers T, Snoek C (2010) Evaluating Color Descriptors for Object and Scene Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 32(9):1582–1596

Simonyan K, Zisserman A (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition. International Conference on Learning Representations

Skelly L, Sclaroff S (2007) Improved Feature Descriptors for 3D Surface Matching

Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going Deeper with Convolutions. Computer Vision and Pattern Recognition

Torralba A, Fergus R, Freeman W (2008) 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(11):1958–1970

Tuytelaars T (2010) Dense Interest Points. Computer Vision and Pattern Recognition 32(9):1582–1596

Vedaldi A, Fulkerson B (2010) VLFeat: An Open and Portable Library of Computer Vision Algorithms. ACM International Conference on Multimedia

Vedaldi A, Lenc K (2014) MatConvNet-Convolutional Neural Networks for MATLAB. arXiv preprint, arXiv: 14124564

Wah C, Branson S, Welinder P, Perona P, Belongie S (2011) The Caltech-UCSD Birds-200-2011 Dataset. Technical Report: CNS-TR-2011-001

Wang J, Yang J, Yu K, Lv F, Huang T, Gong Y (2010) Locality-Constrained Linear Coding for Image Classification. Computer Vision and Pattern Recognition

Wang Z, Fan B, Wu F (2011) Local Intensity Order Pattern for Feature Description. International Conference on Computer Vision

Wang Z, Feng J, Yan S (2014) Collaborative Linear Coding for Robust Image Classification. International Journal on Computer Vision (1):1–12

Xiao J, Hays J, Ehinger K, Oliva A, Torralba A (2010) SUN Database: Large-scale Scene Recognition from Abbey to Zoo. Computer Vision and Pattern Recognition

Xie L, Tian Q, Hong R, Yan S, Zhang B (2013) Hierarchical Part Matching for Fine-Grained Visual Categorization. International Conference on Computer Vision

Xie L, Tian Q, Wang M, Zhang B (2014a) Spatial Pooling of Heterogeneous Features for Image Classification. IEEE Transactions on Image Processing 23(5):1994–2008

Xie L, Wang J, Guo B, Zhang B, Tian Q (2014b) Orientational Pyramid Matching for Recognizing Indoor Scenes. Computer Vision and Pattern Recognition

Xie L, Hong R, Zhang B, Tian Q (2015a) Image Classification and Retrieval are ONE. International Conference on Multimedia Retrieval

Xie L, Tian Q, Zhang B (2015b) Image Classification with Max-SIFT Descriptors. International Conference on Acoustics, Speech and Signal Processing

Xie L, Tian Q, Zhang B (2015c) Simple Techniques Make Sense: Feature Pooling and Normalization for Image Classification. IEEE Transactions on Circuits and Systems for Video Technology

Xie L, Wang J, Lin W, Zhang B, Tian Q (2015d) RIDE: Reversal Invariant Descriptor Enhancement. International Conference on Computer Vision

Yang J, Yu K, Gong Y, Huang T (2009) Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification. Computer Vision and Pattern Recognition

Yang Y, Newsam S (2010) Bag-of-Visual-Words and Spatial Extensions for Land-Use Classification. International Conference on Advances in Geographic Information Systems

Zeiler M, Fergus R (2014) Visualizing and Understanding Convolutional Networks. European Conference on Computer Vision

Zhang N, Farrell R, Iandola F, Darrell T (2013) Deformable Part Descriptors for Fine-Grained Recognition and Attribute Prediction. International Conference on Computer Vision

Zhang N, Donahue J, Girshick R, Darrell T (2014a) Part-based R-CNNs for Fine-Grained Category Detection. European Conference on Computer Vision

Zhang S, Tian Q, Hua G, Huang Q, Li S (2009) Descriptive Visual Words and Visual Phrases for Image Applications. ACM International Conference on Multimedia

Zhang X, Xiong H, Zhou W, Tian Q (2014b) Fused One-vs-All Mid-Level Features for Fine-Grained Visual Categorization. ACM International Conference on Multimedia

Zhao W, Ngo C (2013) Flip-Invariant SIFT for Copy and Object Detection. IEEE Transactions on Image Processing 22(3):980–991

Zhou B, Lapedriza A, Xiao J, Torralba A, Oliva A (2014) Learning Deep Features for Scene Recognition Using Places Database. Advances in Neural Information Processing Systems

Zhou X, Yu K, Zhang T, Huang T (2010) Image Classification using Super-Vector Coding of Local Image Descriptors. European Conference on Computer Vision