

# Towards Reversal-Invariant Image Representation

Lingxi Xie<sup>1</sup> · Jingdong Wang<sup>2</sup> · Weiyao Lin<sup>3</sup> · Bo Zhang<sup>4</sup> · Qi Tian<sup>5</sup>

Received: 10 February 2016 / Accepted: 1 November 2016 / Published online: 28 November 2016  
© Springer Science+Business Media New York 2016

**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**, a generalized **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 increase the CNN capacity without increasing the model complexity. Experiments reveal consistent accuracy gain on various image classification tasks, including scene understanding, fine-grained object recognition, and large-scale visual recognition.

Communicated by Takayuki Okatani.

✉ Jingdong Wang  
jingdw@microsoft.com

✉ Qi Tian  
qitian@cs.utsa.edu

Lingxi Xie  
198808xc@gmail.com

Weiyao Lin  
wylin@sjtu.edu.cn

Bo Zhang  
dcszb@mail.tsinghua.edu.cn

<sup>1</sup> The Johns Hopkins University, Malone Hall, Baltimore, MD 21218, USA

<sup>2</sup> Microsoft Research, Building 2, No. 5 Danling Street, Haidian District, Beijing 100080, China

<sup>3</sup> Shanghai Jiao Tong University, SEIEE Building, Minhang District, Shanghai 200240, China

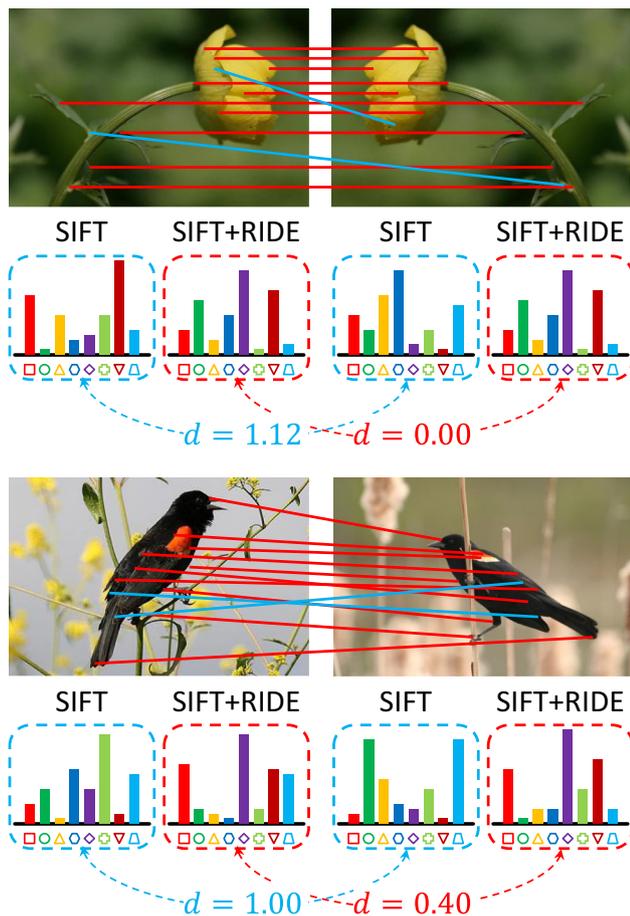
<sup>4</sup> Tsinghua University, FIT Building, Haidian District, Beijing 100084, China

<sup>5</sup> University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249-1604, USA

**Keywords** Image classification · BoF · CNN · Reversal-invariant image representation

## 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. 2012), or extracting transferrable



**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 Sect. 4) brings reversal invariance to local descriptors, and significantly reduces the feature (e.g., BoF) distance between each pair of reversed objects (Color figure online)

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 Sect. 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) might change completely after being reversed (Fig. 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. Although this idea has been previously used to deal with descriptor matching issues (Guo and Cao 2010; Ma et al. 2010; Zhao and Ngo 2013), we argue that these existing approaches are not mainly designed for image classification, and their performance is below satisfaction due to the lack of consideration on some key properties. We will detail this point in Sect. 4.5.

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 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 work. 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 Sect. 6.

## 2 Related Work

### 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 (Clintchant et al. 2007), 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  $\ell_p$ -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. 2014b).

### 2.2 Convolutional Neural Networks

The Convolutional Neural Network (CNN) serves as a hierarchical model for large-scale visual recognition. It is based on 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 them, 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 overfitting, 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 or a set of latent visual attributes (Donahue et al. 2014). 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 understanding the behaviour of CNN models (Zeiler and Fergus 2014).

### 2.3 The Invariance of Descriptors

One of the major shortcomings of the BoF and CNN models is the unsatisfied stability of image representation. An important way of improvement is to study the *invariance* of local descriptors or patch operators. SIFT (Lowe 2004) achieves scale and rotation invariance by selecting the maxima in the scale space, and picking up a dominant orientation, via gradient computation, and rotating the local patch accordingly. Other examples include Shape Context (Belongie et al. 2002), SURF (Bay et al. 2008), BRIEF (Calonder et al. 2010), BRISK (Leutenegger et al. 2011), ORB (Rublee et al. 2011), FREAK (Alahi et al. 2012), etc. Radial transform (Takacs et al. 2013) and polar analysis (Liu et al. 2014) play important roles in generating rotation-invariant local features.

In some vision tasks such as fine-grained recognition, 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 this point, 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 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 epochs 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 setting extra flag bits to represent the reversal information (Guo and Cao 2010), or enforcing that the flows of all regions should follow a pre-defined direction (Zhao and Ngo 2013). These pieces of work inspire us that symmetry is the key to reversal invariance (Skelly and Sclaroff 2007; Wang et al. 2011).

Despite their success, all of these methods are mainly designed for descriptor matching or object retrieval, and their performance on image classification is below satisfaction (see Table 3). In this paper, we propose efficient algorithms towards reversal-invariant representation, which benefits the recognition task.

### 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 11,788 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 10,000 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 Sect. 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.

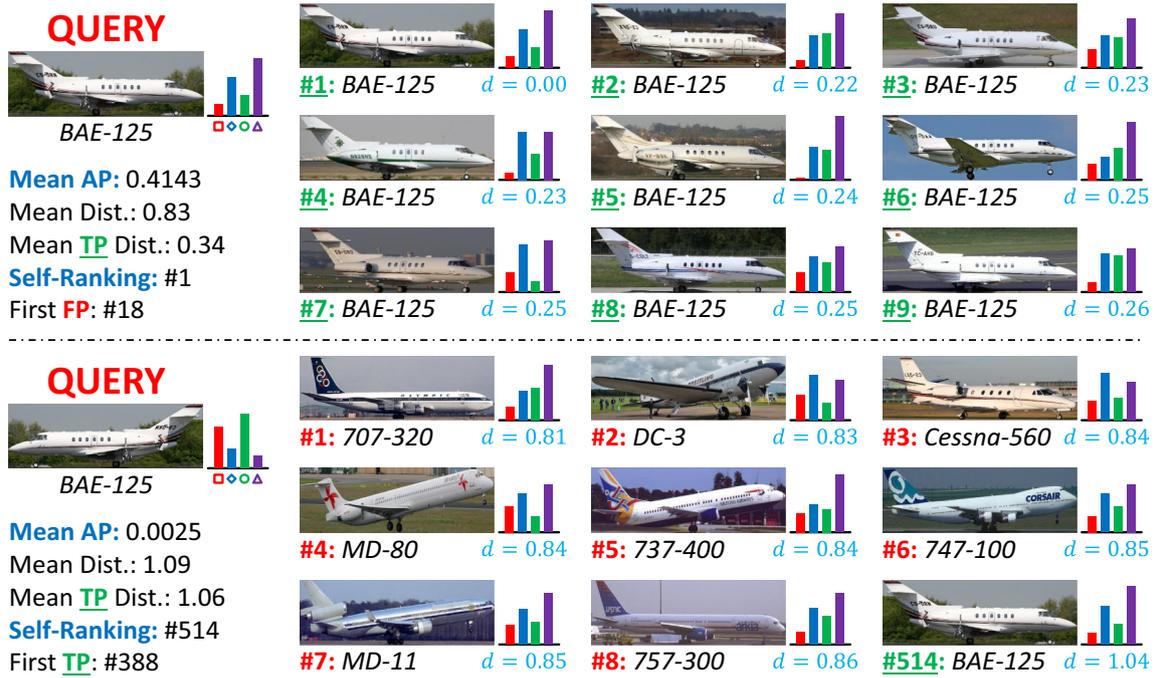
With the standard training/testing split (around 2/3 images are used for training and others for testing), the recognition rate is 53.13% on the *original* dataset and rises up to 63.94% on the *right-aligned* dataset, with a more-than-10% absolute accuracy gain (a more-than-20% relative gain). This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

Still on the **Aircraft-100** dataset. To diagnose, we use all (10,000) images in the right-aligned dataset for training, and evaluate the model with the entire right-aligned and left-aligned datasets, respectively. When we test on the right-aligned dataset, i.e., the training images are identical to the testing image, the classification accuracy is 99.73% (not surprising since we are just performing self-validation). However, when we test on the left-aligned dataset, i.e., each testing image is the reversed version of a training image, the accuracy drops dramatically to 46.84%. This experiment reveals that a model learned from right-oriented objects may not recognize left-oriented objects very well.

Lastly, we perform image retrieval on the right-aligned dataset to directly evaluate the feature quality. Given a query image, we sort the candidates according to the  $\ell_2$ -distance between the representation vectors. Some typical results are shown in Fig. 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. This means that more than 500 images, most of them coming from different categories, are more similar to the query than its reversed copy, because the image feature is not reversal-invariant.

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 will present detailed analysis on this point in Sect. 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



**Fig. 2** Content-based image retrieval on the right-oriented **Aircraft-100** dataset (best viewed in color). We use the same query image with different orientations. *AP* for average precision, *TP* for true-positive, *FP* for false-positive (Color figure online)

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 followed by score average (Krizhevsky et al. 2012; Simonyan and Zisserman 2015). We propose a different idea that generates reversal-invariant image representation in a bottom-up manner.

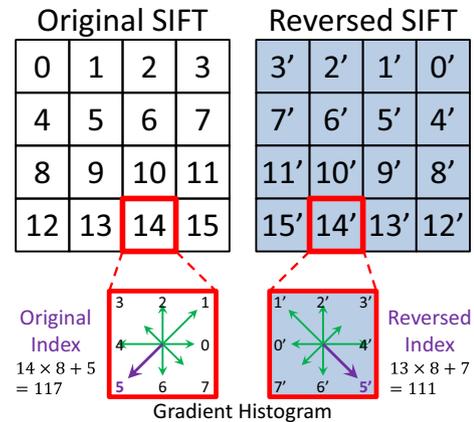
### 4 Reversal Invariance for BoF

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 struc-



**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

ture of a SIFT descriptor is illustrated in Fig. 3. A patch is partitioned into  $4 \times 4$  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  $\mathbf{d} = (d_0, d_1, \dots, d_{127})$ , in which  $d_{i \times 8 + j} = a_{i,j}$  for  $i = 0, 1, \dots, 15$  and  $j = 0, 1, \dots, 7$ . As shown in Fig. 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 Fig. 3) would appear at  $d_{111}$  ( $a_{13,7}$ ) when the descriptor is reversed. Denote the **index mapping function** as  $f(\cdot)$  (e.g.,  $f(0) = 28, f(117) = 111$ ), so that the reversed SIFT can be computed as:  $\mathbf{d}^R \doteq \mathbf{f}(\mathbf{d}) = (d_{f(0)}, d_{f(1)}, \dots, d_{f(127)})$ .

Towards reversal invariance, we need to design a **descriptor transformation function**  $\mathbf{r}(\mathbf{d})$ , so that  $\mathbf{r}(\mathbf{d}) = \mathbf{r}(\mathbf{d}^R)$  for any descriptor  $\mathbf{d}$ . For this, we define  $\mathbf{r}(\mathbf{d}) = \mathbf{s}(\mathbf{d}, \mathbf{d}^R)$ , 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:  $\mathbf{r}(\mathbf{d}) = \mathbf{s}(\mathbf{d}, \mathbf{d}^R) = \mathbf{s}(\mathbf{d}^R, \mathbf{d}) = \mathbf{s}(\mathbf{d}^R, (\mathbf{d}^R)^R) = \mathbf{r}(\mathbf{d}^R)$ . We use the fact that  $(\mathbf{d}^R)^R = \mathbf{d}$  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  $\mathbf{d}^R$  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 a higher priority. The generalized algorithm for selecting the vector with the maximal sequential lexicographic order is provided in Algorithm 1.

---

#### Algorithm 1 Selecting the Vector with the Maximal Sequential Lexicographic Order

---

```

1: Input:  $T$  descriptors  $\{v_1, \dots, v_T\}$  of equal length  $L$ .
2: procedure MAXSEQLEXORDER
3:   Initialization:  $\mathcal{S} = \{1, 2, \dots, T\}$ .
4:   for  $l = 1, 2, \dots, L$  do
5:      $\mathcal{S} \leftarrow \{t \mid t \in \mathcal{S} \wedge v_{t,l} = \max_{t \in \mathcal{S}} \{v_{t,l}\}\}$ ;
6:     if  $|\mathcal{S}| = 1$  then
7:       break
8:     end if
9:   end for
10:  Finalization:  $t^* = \min_{t \in \mathcal{S}} \{t\}$ ;
11: end procedure
12: Output:  $\widehat{\mathbf{v}} = \max \{v_1, \dots, v_T\} \doteq v_{t^*}$ .

```

---

Therefore, to compute the Max-SIFT descriptor for  $\mathbf{d}$ , we only need to compare the dimensions of  $\mathbf{d}$  and  $\mathbf{d}^R$  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:

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

where  $\widehat{\max} \{ \cdot, \cdot, \dots, \cdot \}$  denotes the element with the maximal sequential lexicographic order (see Algorithm 1).

---

#### Algorithm 2 Max-SIFT

---

```

1: Input:  $\mathcal{D} = \{\mathbf{d}_m\}_{m=1}^M$ .
2: procedure MAX-SIFT
3:   for  $m = 1, 2, \dots, M$  do
4:     Reversal: computing  $\mathbf{d}_m^R$ ;
5:     Selection:  $\widehat{\mathbf{d}}_m = \widehat{\max} \{ \mathbf{d}_m, \mathbf{d}_m^R \}$ ;
6:   end for
7: end procedure
8: Output:  $\widehat{\mathcal{D}} = \{ \widehat{\mathbf{d}}_m \}_{m=1}^M$ .

```

---

The pseudo codes of Max-SIFT are illustrated in Algorithm 2. 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.,  $\mathbf{r}(\mathbf{d}) = \frac{1}{2}(\mathbf{d} + \mathbf{d}^R)$ , leads to 1–3% accuracy drop on every single image classification case.

## 4.2 RIDE: Generalized Reversal Invariance

### 4.2.1 The Orientation of SIFT

Let us choose the descriptor from  $\mathbf{d}$  and  $\mathbf{d}^R$  in a more generalized manner. In general, we define an **orientation quantization function**  $q(\cdot)$ , and choose the one in  $\{\mathbf{d}, \mathbf{d}^R\}$  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 accumulating clues from the 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  $\mathbf{u}_j, j = 0, 1, \dots, 7$ . According to the definition of SIFT in Fig. 3, we have  $\mathbf{u}_j = (\cos(j\pi/4), \sin(j\pi/4))^T$ . The global gradient can

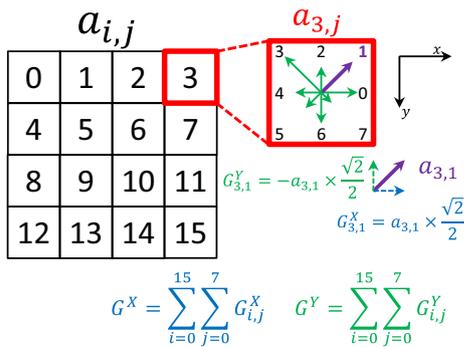


Fig. 4 Estimating the orientation of SIFT

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 computation of  $\mathbf{G}(\mathbf{d})$  is illustrated in Fig. 4. The proof is provided in Appendix 1.

#### 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  $q(\mathbf{d}) = -q(\mathbf{d}^R)$  holds for any  $\mathbf{d}$ , therefore we can simply use the sign of  $q(\mathbf{d})$  to compute the reversal-invariant descriptor transform  $\tilde{\mathbf{d}}$ :

$$\tilde{\mathbf{d}} = \mathbf{r}(\mathbf{d}) = \begin{cases} \mathbf{d} & q(\mathbf{d}) > 0 \\ \mathbf{d}^R & q(\mathbf{d}) < 0 \\ \widehat{\max\{\mathbf{d}, \mathbf{d}^R\}} & q(\mathbf{d}) = 0 \end{cases} \quad (2)$$

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  $d_{28}$  ( $f(0) = 28$ , see Sect. 4.1.1), we can approximate it as a special case of RIDE, with  $q(\mathbf{d}) = d_0 - d_{28}$ .

#### 4.2.3 Generalized 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 3. 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  $\mathbf{d}_R$ ,  $\mathbf{d}_G$  and  $\mathbf{d}_B$ , from the individual red, green and blue channels, therefore we can compute  $\mathbf{G}_R$ ,  $\mathbf{G}_G$  and  $\mathbf{G}_B$  individually, and combine them with  $\mathbf{G} = 0.30\mathbf{G}_R + 0.59\mathbf{G}_G + 0.11\mathbf{G}_B$ . For other color SIFT descriptors, the only difference lies in the linear com-

ination coefficients. By this trick we can perform RIDE on Color-SIFT descriptors very fast.

#### Algorithm 3 Generalized RIDE

---

```

1: Input:  $\mathcal{D} = \{\mathbf{d}_m\}_{m=1}^M$ .
2: procedure RIDE
3:   for  $m = 1, 2, \dots, M$  do
4:     Reversal: computing  $\mathbf{d}_m^R$ ;
5:     SIFT: computing  $\mathbf{d}_m^S$  if necessary;
6:     Orientation:  $q(\mathbf{d}_m) = G_x(\mathbf{d}_m^S)$ ;
7:     Selection:  $\tilde{\mathbf{d}}_m = \mathbf{r}(\mathbf{d}_m)$ , using (2);
8:   end for
9: end procedure
10: Output:  $\tilde{\mathcal{D}} = \{\tilde{\mathbf{d}}_m\}_{m=1}^M$ .

```

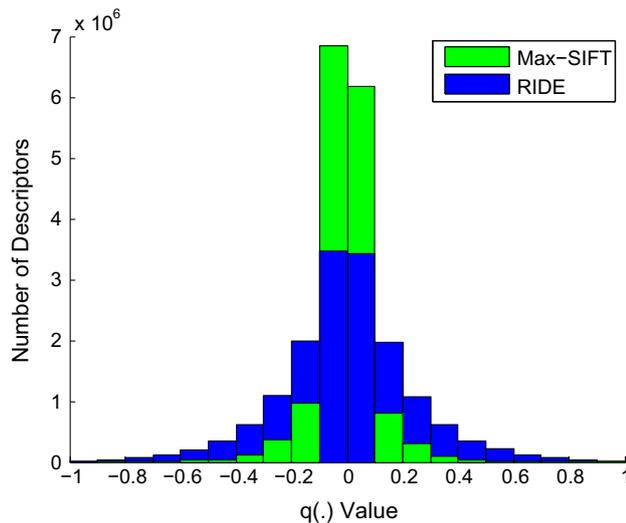
---

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 (Ruble et al. 2011), which is based on BRIEF.

RIDE is also capable of cancelling out a larger family of reversal operations, including upside-down image reversal, and image rotation by  $90^\circ$ ,  $180^\circ$  and  $270^\circ$ . For this we need to take more information from the global gradient vector  $\mathbf{G} = (G_x, G_y)^\top$ . 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 this issue can be found in Appendix 2.

#### 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 noise may change the sign of  $q(\mathbf{d})$  and, consequently, the Max-SIFT and/or RIDE descriptors. To quantitatively analyze the impact of image noise, we first estimate the dis-



**Fig. 5** The distribution of  $q(\cdot)$  values on the **Bird-200** dataset. For Max-SIFT,  $q(\mathbf{d}) = d_0 - d_{28}$  (see the texts in Sect. 4.2.2). All the SIFT descriptors are  $\ell_2$ -normalized so that  $\|\cdot\|_2 = 1$  (Color figure online)

tribution of  $q(\mathbf{d})$  on the **Bird-200** dataset (Wah et al. 2011). According to the histogram in Fig. 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 noise. The reason is that RIDE summarizes the information of the whole SIFT descriptor, while Max-SIFT only considers few dimensions.

Consider image classification on the **Bird-200** dataset. We add random Gaussian noise with standard deviation 0.1203 (the median of all  $|q(\cdot)|$  values) to each of the  $q(\cdot)$  value of RIDE, and find that random noise only causes the classification accuracy drop by less than 1%, which is relatively small compared to the gain of RIDE (6.37%, see Table 2(d)).

Experimental results are very similar on the **Aircraft-100** dataset (Maji et al. 2013).

#### 4.4 Application to Image Classification

The benefit brought by the Max-SIFT and RIDE to image classification is straightforward. Consider an image  $\mathbf{I}$ , and a set of, say, SIFT descriptors extracted from the image:  $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_M\}$ . When the image is left-right reversed, the set  $\mathcal{D}$  becomes:  $\mathcal{D}^R = \{\mathbf{d}_1^R, \mathbf{d}_2^R, \dots, \mathbf{d}_M^R\}$ . If the descriptors are not reversal-invariant, i.e.,  $\mathcal{D} \neq \mathcal{D}^R$ , the feature representation produced by  $\mathcal{D}$  and  $\mathcal{D}^R$  might be totally different. With Max-SIFT or RIDE, we have  $\widehat{\mathbf{d}} = \widehat{\mathbf{d}}^R$  or  $\widetilde{\mathbf{d}} = \widetilde{\mathbf{d}}^R$ , for any  $\mathbf{d}$ , therefore  $\widehat{\mathcal{D}} = \widehat{\mathcal{D}}^R$  or  $\widetilde{\mathcal{D}} = \widetilde{\mathcal{D}}^R$ . 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 Sect. 4.7.1). Despite this, a safer way is to use symmetric pooling bins. In our experiments (see Sect. 4.6.1), we use a spatial pyramid with 4 regions (the entire image and three horizontal stripes).

#### 4.5 Comparison with Previous Work

We first discuss the relationship between RIDE and the original SIFT descriptor. SIFT aligns the descriptor by rotating the detected region, so that the dominant orientation is pointed to the upright direction. After this, we have  $G_x = 0$  (see “The Implementation of SIFT”) thus RIDE degenerates to Max-SIFT. However, in the ordinary implementation of dense sampling where descriptors are not aligned beforehand (Vedaldi and Fulkerson 2010), RIDE serves as a quick and efficient manner of achieving reversal invariance, which does not require considering rotation invariance explicitly.

Some recent work achieves reversal invariance with data augmentation (Wang et al. 2010; Chatfield et al. 2011; Chai et al. 2013; Paulin et al. 2014). This strategy aims at increasing the number of training samples for each prototype (e.g., left/right orientation, etc.). In Sect. 4.6.2, we will show that RIDE works better and faster than data augmentation, arguably because RIDE allows local reversal on each part of the object, which is more flexible than global reversal (see Sect. 4.7.3).

Although some reversal-invariant descriptors are proposed for descriptor matching (Ma et al. 2010; Guo and Cao 2010; Zhao and Ngo 2013) or object retrieval (Guo and Cao 2010; Zhao and Ngo 2013; Xie et al. 2015b), these descriptors have not been adopted in fine-grained recognition. We implement several of them, and compare it with our algorithm in Table 3. One can observe that Max-SIFT and RIDE significantly outperform these competitors in every single case. We will provide detailed analysis on this issue in Sect. 4.7.2.

RIDE shares the idea of accumulating gradients with FIND (Guo and Cao 2010), but uses a different way of computation. FIND partitions the descriptor bins into two parts (left and right) according to their spatial positions, and accumulates each gradient intensity without considering its horizontal (left/right) components. That is to say, all gradi-

**Table 1** Image classification datasets used in our experiments

Dataset	Abbreviation	# Classes	# Images	# Training samples / Class
<b>Pet-37</b> (Parkhi et al. 2012)	P-37	37	7390	100
<b>Aircraft-100</b> (Maji et al. 2013)	A-100	100	10000	67
<b>Flower-102</b> (Nilsback and Zisserman 2008)	F-102	102	8189	20
<b>Bird-200</b> (Wah et al. 2011)	B-200	200	11788	30
<b>LandUse-21</b> (Yang and Newsam 2010)	L-37	21	2100	80
<b>Indoor-67</b> (Quattoni and Torralba 2009)	I-67	67	15620	80
<b>SUN-397</b> (Xiao et al. 2010)	S-397	397	108754	50
<b>Caltech256</b> (Griffin 2007)	C-256	257	30607	60

Among them, four datasets are designed for fine-grained object recognition, three for scene classification and one for generic object recognition

ent bins at the same cell contribute equally, although they are pointed to different directions. RIDE improves this strategy by decomposing each bin into a horizontal component and a vertical component, and accumulates each component with their (plus/minus) sign separately into a global gradient vector  $\mathbf{G}$ . According to our proof in Appendix 1, our strategy better utilizes the gradient information. In experiments, RIDE works better than FIND in orientation prediction (see Sect. 4.7.1), which, we believe, is the main reason of the accuracy gap in classification (see Table 3).

Finally, we discuss the difference between RIDE and F-SIFT (Zhao and Ngo 2013). We predict the orientation of each descriptor based on the description vector, while F-SIFT normalizes each local patch before extracting the descriptor on it. In other words, RIDE computes orientation based on the descriptor, while F-SIFT based on the patch. We point out that prediction based on the descriptor is more reliable: we are dealing with descriptors, not patches, in the following feature encoding stage. In experiments, RIDE works better than F-SIFT in orientation prediction (see Sect. 4.7.1). We believe this is the main reason of the accuracy gap in classification (see Table 3). This principle generalizes to the CNN case (see Sect. 5.2.5). In addition, RIDE works much faster than F-SIFT (see Sect. 4.6.3).

In summary, although our algorithm shares some ideas with the previous approach, we achieve better performance in image classification by generating more discriminative image-level features. Moreover, our approach enjoys the advantage of easy implementation and low computational costs.

## 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 summarized 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 sampling are 6 and 12, respectively. On the same set of patches, LCS, RGB-SIFT and Opponent-SIFT (Sande et al. 2010) descriptors are also extracted. Max-SIFT or RIDE is thereafter computed for each type of descriptors. In the former case, we can only apply Max-SIFT on SIFT-based descriptors, thus the LCS descriptors remain unchanged. When we apply RIDE to RootSIFT, we compute RIDE on the original SIFT, obtained by dimension-wise squaring RootSIFT.

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  $\ell_2$  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.

**Table 2** Classification accuracy (%) of different models

	ORIG	MAX	RIDE	AUGM	RIDE×2
<b>(a) Pet-37 Results</b>					
S	37.92	41.78	42.28	42.24	<b>45.61</b>
L	43.25	–	44.27	45.12	<b>46.83</b>
F	52.06	53.92	54.69	54.67	<b>57.51</b>
R	44.90	46.73	47.35	46.98	<b>49.53</b>
O	46.53	48.39	49.01	48.72	<b>51.19</b>
<b>(b) Aircraft-100 Results</b>					
S	53.13	57.72	57.82	57.16	<b>60.14</b>
L	41.82	–	42.86	43.13	<b>44.81</b>
F	57.36	60.49	61.27	60.59	<b>63.62</b>
R	57.89	61.90	63.09	62.48	<b>65.11</b>
O	47.06	52.35	53.12	51.39	<b>55.79</b>
<b>(c) Flower-102 Results</b>					
S	53.68	58.12	59.12	58.01	<b>61.09</b>
L	73.47	–	75.30	75.88	<b>77.40</b>
F	76.96	79.59	80.51	79.49	<b>82.14</b>
R	71.52	74.00	74.97	74.18	<b>77.10</b>
O	76.12	78.40	79.68	78.83	<b>81.69</b>
<b>(d) Bird-200 Results</b>					
S	25.77	31.59	32.14	31.60	<b>34.07</b>
L	36.18	–	38.50	38.97	<b>40.16</b>
F	38.11	43.48	44.73	43.98	<b>46.38</b>
R	31.36	38.20	39.16	38.79	<b>41.73</b>
O	35.40	41.15	42.18	41.72	<b>44.30</b>

Bold values indicate the highest classification accuracy on each dataset. 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 Sect. 4.6.2 for details

#### 4.6.2 Image Classification Results

We first report fine-grained object recognition accuracy with different descriptors in Table 2. Beyond original descriptors, we implement Max-SIFT, RIDE and data augmentation. By data augmentation we mean to generate a reversed copy for each training/testing image, use the augmented 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 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 compared to RIDE (see Sect. 4.6.3 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. Besides descriptor computation, all the other stages are exactly the same. Results are shown in Table 3. The consistent 3–4% gain verifies that RIDE makes stable contribution to visual recognition.

It is worth noting that Bird-200 is a little bit different from other datasets, since part detection and description often play important roles in this fine-grained recognition task. Recently, researchers design complicated part-based recognition algorithms, including 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), and Zhang et al. (2016)). We also evaluate RIDE with SIFT on the detected parts provided by some of these approaches. 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), the baseline performance is 71.5% with SIFT and LCS descriptors. With RIDE, we obtain 73.1%, which is close to the accuracy using deep features (73.9% reported in the paper).

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 the fine-grained case, it enjoys good recognition performance on a wide range of image datasets.

#### 4.6.3 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.

**Table 3** Classification accuracy (%) comparison with some recent work

	P-37	A-100	F-102	B-200	L-21	I-67	S-397	C256
<b>ORIG</b>	60.24	74.61	83.53	47.61	93.64	63.17	48.35	58.77
<b>Max-SIFT</b>	62.80	77.54	85.82	49.93	94.13	64.12	49.39	59.21
<b>RIDE</b>	<b>63.49</b>	<b>78.92</b>	<b>86.45</b>	<b>50.81</b>	<b>94.71</b>	<b>64.93</b>	<b>50.12</b>	<b>60.25</b>
<b>MI-SIFT</b> (Ma et al. 2010)	58.91	72.26	81.06	45.59	92.86	61.49	46.51	55.39
<b>FIND</b> (Guo and Cao 2010)	59.63	74.06	82.91	47.49	93.14	62.91	47.87	56.72
<b>F-SIFT</b> (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	–

Bold values indicate the highest classification accuracy on each dataset

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

**Table 4** Time/memory cost in each step of the BoF model

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

All the data are recorded with SIFT descriptors with 32 GMM components on the **Bird-200** dataset (Wah et al. 2011)

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. This is significantly lower than some previous methods such as F-SIFT (about 30% extra time Zhao and Ngo 2013). 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 much better classification results.

## 4.7 Discussion

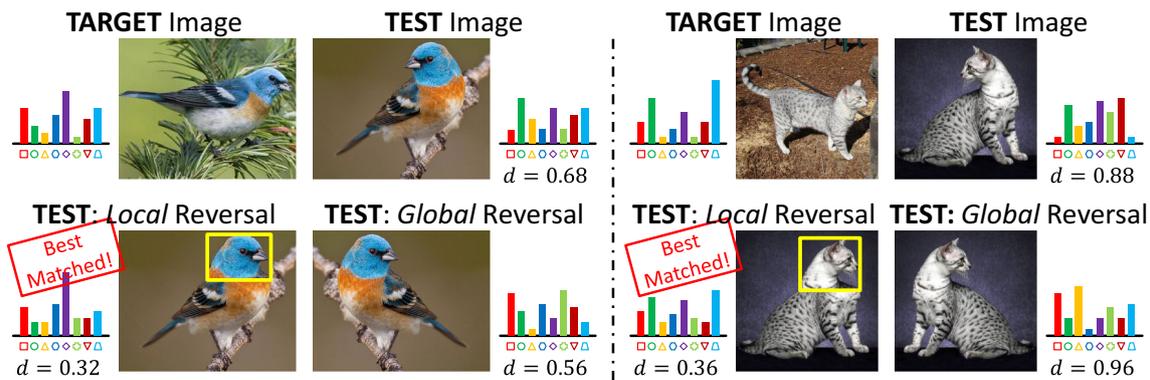
### 4.7.1 Object Orientation Prediction

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 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”  $(x/W, y/H)$ . The descriptor starts from an evidence 0. On each training image, we seek for the nearest descriptor measured by the  $\ell_2$  distance of relative positions, and compare their orientation quantization function values: if the values are of the same sign, add 1 to the evidence, otherwise subtract 1 from the evidence. Each descriptor contributes the sign of the evidence (in  $\{+1, -1, 0\}$ ) to the orientation score of the entire image. After all the descriptors are processed, if the total score is positive, then this testing image is oriented to right; if it is negative, to left; if it is 0, a random guess is taken (this rarely happens).

One can note that different results are produced with different orientation quantization functions. Using RIDE ( $q(\mathbf{d}) = G_x(\mathbf{d})$ ), the prediction accuracy is 65.45%, whereas using Max-SIFT ( $q(\mathbf{d}) = d_0 - d_{28}$ , see Sect. 4.2.2), the accuracy drops to 54.69%, barely above the chance level



**Fig. 6** Global versus 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 (Color figure online)

(50%). We also implement FIND (Guo and Cao 2010) (ignoring gradient components) and F-SIFT (Zhao and Ngo 2013) (computing on raw pixels) for orientation prediction, and the accuracy is 56.19 and 57.41%, respectively. In summary, RIDE predicts the orientation of a local patch more accurately. As we have analyzed in Sect. 3, this significantly helps image classification.

#### 4.7.2 Classification Versus Retrieval

We discuss the difference between image classification and object retrieval, followed by the reason that some descriptors designed for image retrieval (Ma et al. 2010) fail to achieve good performance in image classification (see Table 3).

To retrieve a partial-duplicate or near-duplicate object from a set of candidate images, the most important thing is to find a sufficient number of local feature matches. As hand-crafted descriptors such as SIFT are often sensitive to image transformation (e.g., reversal, rotation, etc.), a good strategy is to cancel out these possible variations by pooling over the corresponding dimensions of each descriptor. For instance, MI-SIFT (Ma et al. 2010) averages the values of each four-bin group to cancel out image reversal. Although this may result in precision drop, i.e., introducing some false matches, it is still possible to filter them out with post-processing such as spatial verification.

As to image classification, especially fine-grained object recognition, things are different: the most important task is to generate some *discriminative* image features to distinguish one class from others. Therefore, it is often unreasonable to destroy the structure of a local descriptor for the purpose of invariance. To this point, we propose to maximally preserve the descriptive power of local features and prevent dimension-wise operations.

#### 4.7.3 Global Reversal Versus Local Reversal

We point out that RIDE benefits image representation in the perspective of regional feature matching.

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.

## 4.8 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 cancel 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 noise. 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, if an image is left-right reversed, the neuron responses on each layer will 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 learn from 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  $\mathbf{I}^R$ . 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 on the same layer, i.e.,  $\mathbf{f}_l(\mathbf{I}^R; \mathcal{M})$ . Most often,  $\mathbf{f}_l(\mathbf{I}; \mathcal{M}) \neq \mathbf{f}_l(\mathbf{I}^R; \mathcal{M})$ .

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

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

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

where  $\max\{\cdot, \cdot\}$  denotes the element-wise maximization of two vectors. This strategy is different from that used in Sect. 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, each dimension in it corresponds to the gradient intensity on a specific direction. If we simply take the dimension-wise average or maximum

of a SIFT and its reversed version (like MI-SIFT (Ma et al. 2010)), 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 both Average-Deep and Max-Deep are similar to the testing strategy used in some state-of-the-art CNNs (Krizhevsky et al. 2012; Simonyan and Zisserman 2015; Szegedy et al. 2015). In which, using visual clues in 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 Average-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 Sect. 4.6. We use the **AlexNet** and the **VGGNet** (both the 16- and 19-layer models, provided by the **MatConvNet** library (Vedaldi and Lenc 2014)) as the pre-trained deep networks. To demonstrate the importance of reversal invariance, we also train another version of the **AlexNet**, in which we do not use image reversal as 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  $227 \times 227$  pixels and feed it into the network. In the original testing process (Krizhevsky et al. 2012), the image is resized to  $256 \times 256$  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  $224 \times 224$  pixels. After the neuron responses are computed, we extract the features from each layer by average-pooling over all spatial positions. Throughout the rest part, we use the features extracted from the *fc-6*

**Table 5** Classification accuracy (%) without or with reversal-invariant deep features

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

We also list the results of the BoF model (please refer to Table 3) and several recent work using deep features for comparison. All the results are obtained with the  $fc-6$  features (4096D) after ReLU activation

layer, activated by ReLU (Krizhevsky et al. 2012). These feature vectors are  $\ell_2$ -normalized and sent to LIBLINEAR (Fan et al. 2008), a scalable SVM implementation, with the slackening parameter  $C = 10$ . Averaged accuracy by category is reported. Results are summarized in Table 5. Each number is the mean of 10 random training/testing splits.

### 5.1.3 Discussion

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

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, we obtain consistent accuracy gain using the transferred features for recognition. On the other hand, both Average-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, Average-Deep and Max-Deep still improve the classification rate consistently, although the gain becomes relatively small (approximately 2% on the Aircraft-100 dataset) due to the marginal effect. Considering that the baseline is already high in most cases and that both Average-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, there-

fore 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 (see the related contents in Sect. 4.6.2).

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 related 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., Average-Conv and Max-Conv, which lead to a more direct way of generating reversal-invariant deep features. These two strategies will be compared in Sect. 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 say a CNN model is *reversal-invariant* if it produces symmetric neural responses on each pair of symmetric images, i.e., for an arbitrary image  $\mathbf{I}$ , if we take  $\mathbf{I}$  and  $\mathbf{I}^R$  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  $\mathbf{f}_l(\mathbf{I}^R; \mathcal{M})$ , are symmetric to each other:  $\mathbf{f}_l^R(\mathbf{I}; \mathcal{M}) = \mathbf{f}_l(\mathbf{I}^R; \mathcal{M})$ . In a reversal-invariant 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  $1 \times 1$ . 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 over the responses at all spatial locations, similar to the strategy used in Sect. 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 a patch 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,  $k = 1, 2, \dots, K_l$ , we have:

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

Here,  $f_l^{(a,b,k)}$  denotes the unit at the spatial position  $(a, b)$ , and convoluted by the  $k$ -th kernel, on the  $l$ -th layer, and  $\mathbf{p}_{l-1}^{(a,b)}$  denotes the related data patch on the previous layer. Note that  $\mathbf{p}_{l-1}^{(a,b)}$  and  $\boldsymbol{\theta}_l^{(k)}$  are of the same dimension.  $\langle \cdot, \cdot \rangle$  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 **Average-Conv** and the **Max-Conv** formulations:

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

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

Since Average-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 Average-Conv, we can accelerate both forward-propagation and back-propagation by modifying the input data (the original input  $\mathbf{p}_{l-1}^{(a,b)}$  is replaced by the average of  $\mathbf{p}_{l-1}^{(a,b)}$  and  $\mathbf{p}_{l-1}^{(a,b),R}$ ), 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 made 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 Average-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.,  $\mathbf{f}_0^R(\mathbf{I}; \mathcal{M}) = \mathbf{f}_0(\mathbf{I}^R; \mathcal{M})$ . Now, given that  $\mathbf{f}_{l-1}^R(\mathbf{I}; \mathcal{M}) = \mathbf{f}_{l-1}(\mathbf{I}^R; \mathcal{M})$ , we derive that  $\mathbf{f}_l^R(\mathbf{I}; \mathcal{M}) = \mathbf{f}_l(\mathbf{I}^R; \mathcal{M})$ , 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, i.e., the **geometric symmetry** is guaranteed.

Consider a patch  $\mathbf{p}_{l-1}^{(a,b)}(\mathbf{I}; \mathcal{M})$ . According to symmetry, we have  $\mathbf{p}_{l-1}^{(a,b),R}(\mathbf{I}; \mathcal{M}) = \mathbf{p}_{l-1}^{(W_{l-1}-a-1,b)}(\mathbf{I}^R; \mathcal{M})$ , 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  $\theta_k$ , and the outputs are  $f_l^{(a,b,k)}(\mathbf{I}; \mathcal{M})$  and  $f_l^{(W_{l-1}-a-1,b,k)}(\mathbf{I}^R; \mathcal{M})$ . These two scalars equal to each other since both Average-Conv and Max-Conv are symmetric, thus the neuron responses on the  $l$ -th layer are also symmetric:  $\mathbf{f}_l^R(\mathbf{I}; \mathcal{M}) = \mathbf{f}_l(\mathbf{I}^R; \mathcal{M})$ . This finishes the induction, i.e., the neuron responses of a pair of symmetric inputs are symmetric.

We point out that such a good property in feature extraction can be a significant drawback 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. Since using reversal-invariant convolution operations increases the capacity of the CNN model (see Sect. 5.2.5), the decrease of training data may cause overfitting, 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  $227 \times 227$ , 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  $6 \times 6$ . In practice,

we simply use  $S' = 199$ , so that we can generate as many training images as possible. As we shall see in Sect. 5.2.4, this strategy also improves the baseline accuracy slightly.

### 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 50,000 training samples and 10,000 testing samples, each of which is a  $32 \times 32$  color image, uniformly distributed among all 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  $32 \times 32 \times 3$  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.

Here, C5 (S1P2) means a convolutional layer with a kernel size 5, a spatial stride 1 and a padding width 2; MP3 (S2) refers to a max-pooling layer with a kernel size 5 and a spatial stride 2, and FC10 indicates a fully-connected layer with 10 outputs. In **CIFAR100**, we replace FC10 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 epochs with the learning rate  $10^{-3}$ , followed by 20 epochs with the learning rate  $10^{-4}$ , and another 10 epochs with the learning rate  $10^{-5}$ .

We train six different models individually, i.e., training a network with the original version of convolution, Average-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. Note that reversal-invariant convolution also applies to the fully-connected layer. 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, Average-Conv causes dramatic accuracy drop (we will analyze the reason in Sect. 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

**Table 6** CIFAR classification error rate (%) with respect to different versions of **LeNet**

	w/o AUGM	w/ AUGM
<b>CIFAR10</b>		
<b>LeNet</b>	18.11 ± 0.20	16.99 ± 0.22
<b>LeNet-AVG</b>	21.01 ± 0.35	20.99 ± 0.26
<b>LeNet-MAX</b>	16.93 ± 0.18	<b>16.64 ± 0.17</b>
<b>CIFAR100</b>		
<b>LeNet</b>	46.08 ± 0.26	44.55 ± 0.10
<b>LeNet-AVG</b>	47.79 ± 0.41	47.55 ± 0.31
<b>LeNet-MAX</b>	43.90 ± 0.19	<b>43.65 ± 0.16</b>

Bold values indicate the lowest recognition error

**Table 7** CIFAR classification error rate (%) with respect to different versions of **BigNet**

	w/o AUGM	w/ AUGM
<b>CIFAR10</b>		
<b>BigNet</b>	10.29 ± 0.29	7.80 ± 0.18
<b>BigNet-AVG</b>	12.13 ± 0.19	9.53 ± 0.19
<b>BigNet-MAX</b>	7.63 ± 0.12	<b>7.49 ± 0.23</b>
<b>CIFAR100</b>		
<b>BigNet</b>	34.51 ± 0.17	31.03 ± 0.24
<b>BigNet-AVG</b>	36.74 ± 0.19	32.61 ± 0.30
<b>BigNet-MAX</b>	30.92 ± 0.32	<b>30.76 ± 0.24</b>

Bold values indicate the lowest recognition error

Max-Conv boost the accuracy by about 1%, and these two strategies cooperate with each other to outperform 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 combination gives 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 challenging dataset (with more categories), and that Max-Conv increases the network capacity to benefit the recognition task.

To compare with the state-of-the-art, we also evaluate our algorithm on a larger network named **BigNet**. It is a 10-layer network (Nagadomi 2014) which resembles the **VGGNet** in using small convolutional kernels. We inherit all the settings from the original author, and report the accuracy produced by six different versions as above. Results summarized in Table 7 deliver the same conclusion as using the small network (**LeNet**).

We also put our result in the **CIFAR** datasets in the context of some recent publications in this dataset. Results are summarized in Table 8.

**Table 8** Comparison with some recent work in **CIFAR** classification error rates (%)

	CIFAR10	CIFAR100
Goodfellow et al. (2013)	9.38	38.57
Lin et al. (2014a)	8.81	35.68
Lee et al. (2015)	7.97	34.57
Liang and Hu (2015)	7.09	31.75
Lee et al. (2016)	<b>6.05</b>	32.37
<b>LeNet-MAX</b>	16.64	43.65
<b>BigNet-MAX</b>	<b>7.49</b>	<b>30.76</b>

Bold values indicate the lowest recognition error

**Table 9** ILSVRC2012 classification error rate (%) with respect to different versions of **AlexNet**

	w/o AUGM	w/ AUGM
<b>ILSVRC2012, top-1</b>		
<b>AlexNet</b>	43.05 ± 0.19	42.52 ± 0.17
<b>AlexNet-MAX</b>	42.16 ± 0.05	<b>42.10 ± 0.07</b>
<b>ILSVRC2012, top-5</b>		
<b>AlexNet</b>	20.62 ± 0.08	19.52 ± 0.05
<b>AlexNet-MAX</b>	19.42 ± 0.03	<b>19.12 ± 0.07</b>

Bold values indicate the lowest recognition error

#### 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 1.3 M, 50 K and 150 K 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 Sect. 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-FC1000.

Here, LRN means local response normalization (Krizhevsky et al. 2012) and D0.5 means Dropout with a drop ratio 0.5. Following the setting of **CAFFE**, a total of 450,000 mini-batches (approximately 90 epochs) 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 100,000 mini-batches.

We individually train four models, i.e., using original convolution or Max-Conv, using data augmentation or not. Simi-

larly, 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 Average-Conv according to the dramatic accuracy drop in **CIFAR** experiments. Note that Max-Conv also applies to the fully-connected layers.

Result are summarized in Table 9. 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<sup>1</sup>). 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 Discussion

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 some discussion 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  $\mathbf{p}_{l-1}^{(a,b)}$  (probably together its reversed copy) and each of the  $K_l$  convolution kernels  $\theta_k$ ,  $k = 1, 2, \dots, K_l$ . Since inner production measures the similarity between  $\mathbf{p}_{l-1}^{(a,b)}$  and  $\theta_k$ , the patches with similar appearance to  $\theta_k$  will get a significant neuron response. In this situation,  $\theta_k$  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 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 code-word 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 Average-Conv and Max-Conv. Both of them compute the similarity between each codeword and each original/reversed local patch. After that, Average-Conv considers the average response and Max-

Conv gets the larger response. Thus, 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. Another way of understanding is that Eqn (6) is equivalent to averaging each kernel and its reversed copy and then convoluting with the patch, thus Average-Conv constrains all the input patches to be left-right symmetric. In opposite, Max-Conv animates those local patches which are similar to either the original or reversed codeword, thus contributes to improve the discriminative power and invariance of the deep feature. Consequently, in the experiments, Average-Conv causes dramatic accuracy drop, while Max-Conv boosts the performance significantly.

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 epochs. Note that both strategies augment the training data: 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 Fig. 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 increased. These two strategies cooperate with each other to make full use of the increased model capacity, meanwhile prevent over-fitting.

Another strategy for reversal invariance in CNN is to pre-compute the orientation of each patch before convolution, e.g., using the method presented in Zhao and Ngo (2013) for patch normalization. We point out that RI-Conv is more effective than Zhao and Ngo (2013). RI-Conv always chooses the patch orientation which generates the higher score in template matching, while Zhao and Ngo (2013) determines the patch orientation without considering the learned templates in the network. As we have shown in Sect. 4.5, the former strategy often works better. In addition, RI-Conv is very easy to implement, while Zhao and Ngo (2013) may produce complicated gradients on the input patch.

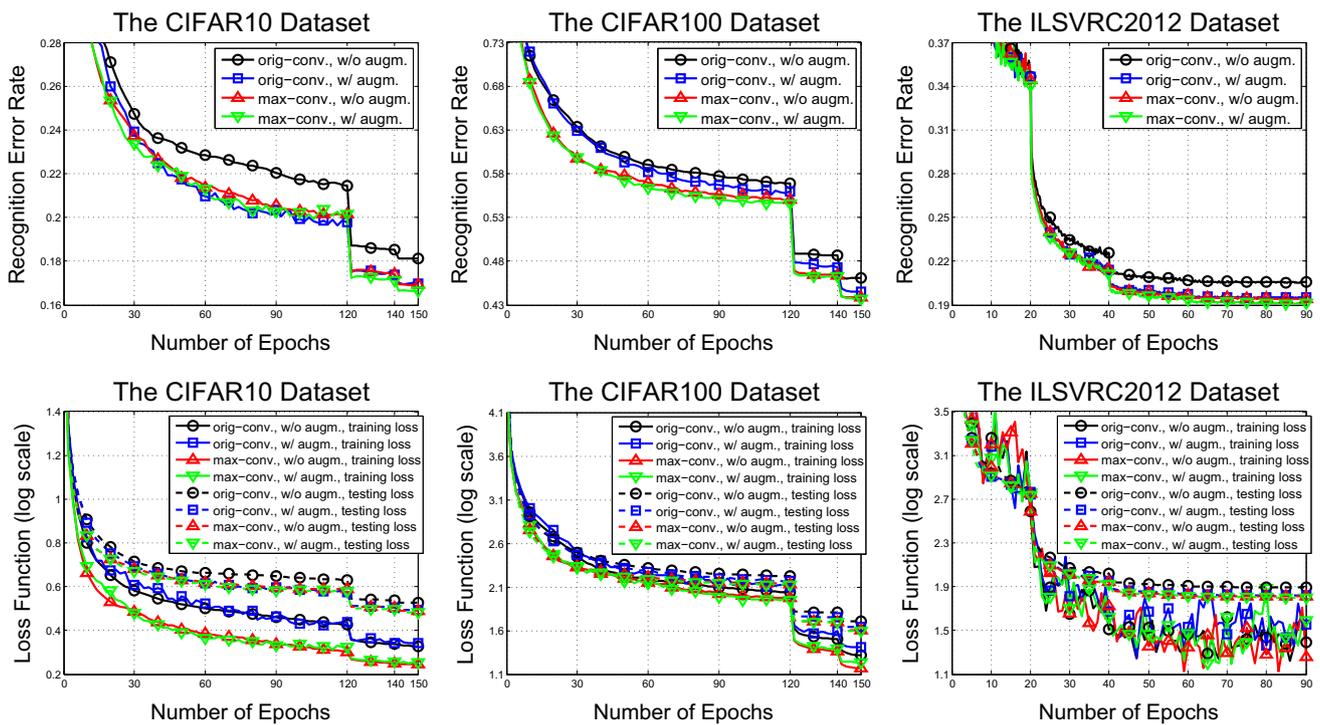
### 5.3 Model Comparison

We compare the strategies discussed in this section, i.e., training a non-reversal-invariant deep network followed by post-processing deep features for reversal invariance, and training a reversal-invariant deep network to generate 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 Sect. 5.1. Results are summarized in Table 10.

<sup>1</sup> <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

**Table 10** 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 Sect. 5.2.2). With Max-Conv, we do not need to post-process the feature vector since it is naturally reversal-invariant

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

We observe that the features extracted from **AlexNet-MAX** produce consistently higher accuracy than the original non-reversal-invariant features. The performance is comparable to the Average-Deep and Max-Deep features, while the computation is cheaper. In one word, designing intrinsically reversal-invariant modules is helpful to visual recognition.

**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**). 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.

Last but not least, the Max-Conv operator is easy to implement yet fast to carry out (less than 20% extra time is required).

**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 implemen-

tation 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 visual recognition.

**Acknowledgements** 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.

### Appendix 1: 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 Sect. 4.2.1 of the main article.

#### 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:

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

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

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

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  $\sigma$  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.,

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

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

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

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

Of course, we can also follow the orientation assignment of original SIFT implementation (Lowe 2004). In practice, 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  $4 \times 4$  grids, and an 8-bin orientation histogram is constructed in each grid. The central orientation value of the  $j$ -th bin is  $\theta_j = j\pi/4, j = 0, 1, \dots, 7$ . 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:

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

An 8-dimensional orientation histogram is thereafter obtained in each of the  $4 \times 4$  grids. Finally, the 128-dimensional

descriptor is constructed by concatenating the histogram vectors from all  $4 \times 4$  grids.

**Orientation Estimation**

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

**Theorem** *Given a densely sampled SIFT descriptor  $\mathbf{d} = (d_k, \theta_k)_{k=1,2,\dots,128}$ , where  $d_k$  and  $\theta_k$  are the gradient value and the histogram orientation for the  $k$ -th dimension, respectively. Its accumulated orientation  $\theta$  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  $(m, \theta)$  with an arbitrary orientation is quantized as  $(m_a, \theta_a)$  and  $(m_b, \theta_b)$  ( $\theta_a < \theta < \theta_b$ ) with the trilinear interpolation, i.e., using (12):*

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

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:

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

thus (16) becomes:

$$\begin{aligned} & m_a \cos \theta_a + m_b \cos \theta_b \\ &= m \times \frac{\theta_b - \theta}{\theta_b - \theta_a} \times \cos \theta_a + m \times \frac{\theta - \theta_a}{\theta_b - \theta_a} \times \cos \theta_b \end{aligned}$$

$$\begin{aligned} & \approx m \times \left[ \frac{\sin(\theta_b - \theta)}{\sin(\theta_b - \theta_a)} \times \cos \theta_a + \frac{\sin(\theta - \theta_a)}{\sin(\theta_b - \theta_a)} \times \cos \theta_b \right] \\ &= \frac{m \times [\sin(\theta_b - \theta) \cos \theta_a + \sin(\theta - \theta_a) \cos \theta_b]}{\sin(\theta_b - \theta_a)} \\ &= \frac{m \times (\sin \theta_b \cos \theta \cos \theta_a - \cos \theta \sin \theta_a \cos \theta_b)}{\sin(\theta_b - \theta_a)} \\ &= \frac{m \times \cos \theta \times (\sin \theta_b \cos \theta_a - \cos \theta_b \sin \theta_a)}{\sin(\theta_b - \theta_a)} \\ &= m \cos \theta, \end{aligned} \tag{18}$$

which finishes the proof.  $\square$

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  $\lim_{x \rightarrow 0} f(x) = 1$  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.

**Appendix 2: Generalized RIDE Descriptors**

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 Sect. 4.2.3 of the main article.

**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  $G_x \geq 0$  holds for a descriptor  $\mathbf{d}$ , we need to generate a left-right reversed version of that descriptor,  $\mathbf{d}^R$ , and select the one in  $\mathbf{d}$  and  $\mathbf{d}^R$  that satisfies  $G_x \geq 0$ . 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  $\mathbf{d}^R$  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.,  $G_y \geq 0$ . We then 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  $G_x \geq 0$  and  $G_y \geq 0$ . If more than one candidates satisfy the conditions, we choose the one with the largest sequential

**Table 11** Classification accuracy (%) of different versions of RIDE on different versions of the **Aircraft-100** dataset. The training/testing split is fixed in all cases

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

Bold values indicate the highest classification accuracy on each dataset

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  $90^\circ$ . Adding the  $90^\circ$ -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  $G_x \geq G_y \geq 0$ , i.e.,  $G_x \geq 0$ ,  $G_y \geq 0$  and  $G_x \geq G_y$ . 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  $G_x \geq 0$ , limits the orientation to falling into an interval of a  $180^\circ$  range. This range is further shrunk into  $90^\circ$  in RIDE-4, and  $45^\circ$  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.

## 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  $90^\circ$  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 (Sect. 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 11. 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 11, 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  $90^\circ$ -rotated objects. **RIDE-2** produces the best classification accuracy in such cases.

## References

- Alahi, A., Ortiz, R., & Vandergheynst, P. (2012). FREAK: Fast Retina Keypoint. In *IEEE conference on computer vision and pattern recognition*.
- Angelova, A., & Zhu, S. (2013). Efficient object detection and segmentation for fine-grained recognition. In *IEEE conference on computer vision and pattern recognition*.
- Arandjelovic, R., & Zisserman, A. (2012). Three things everyone should know to improve object retrieval. In *IEEE conference on computer vision and pattern recognition*.

- Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3), 346–359.
- Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4), 509–522.
- Berg, T., & Belhumeur, P. (2013). POOF: Part-based one-vs-one features for fine-grained categorization, face verification, and attribute estimation. In *IEEE conference on computer vision and pattern recognition*.
- Bosch, A., Zisserman, A., & Munoz, X. (2006). Scene classification via pLSA. In *International conference on computer vision*.
- Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010). BRIEF: Binary robust independent elementary features. In *European conference on computer vision*.
- Chai, Y., Lempitsky, V., & Zisserman, A. (2013). Symbiotic segmentation and part localization for fine-grained categorization. In *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. In *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. In *British machine vision conference*.
- Clinchant, S., Csurka, G., Perronnin, F., & Renders, J. (2007). *XRCEs participation to ImageEval*. ImageEval workshop at CVIR.
- 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. In *IEEE conference on 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. In *IEEE conference on 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. In *IEEE conference 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. In *IEEE conference on 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*, 111, 191–212.
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE conference on computer vision and pattern recognition*.
- Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. In *International conference on machine learning*.
- Grauman, K., & Darrell, T. (2005). The pyramid match kernel: Discriminative classification with sets of image features. In *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. In *International conference on pattern 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. In *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. In *ACM international conference on multimedia*.
- Juneja, M., Vedaldi, A., Jawahar, C., & Zisserman, A. (2013). Blocks that shout: Distinctive parts for scene classification. In *IEEE conference on computer vision and pattern recognition*.
- Kobayashi, T. (2014). Dirichlet-based histogram feature transform for image classification. In *IEEE conference on computer vision and pattern recognition*.
- Krause, J., Jin, H., Yang, J., & Fei-Fei, L. (2015). Fine-grained recognition without part annotations. In *IEEE conference on computer vision and pattern recognition*.
- Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*. Technical report, University of Toronto, (Vol. 1(4), p. 7).
- Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*.
- Lapin, M., Schiele, B., & Hein, M. (2014). Scalable multitask representation learning for scene classification. In *IEEE conference on computer vision and pattern recognition*.
- Lazebnik, S., Schmid, C., & Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE conference on 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. In *Advances in neural information processing systems*.
- Lee, C., Gallagher, P., & Tu, Z. (2016). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *International conference on artificial intelligence and statistics*.
- Lee, C., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-supervised nets. In *International conference on artificial intelligence and statistics*.
- Leutenegger, S., Chli, M., & Siegwart, R. (2011). BRISK: Binary robust invariant scalable keypoints. In *International conference on computer vision*.
- Li, L., Guo, Y., Xie, L., Kong, X., & Tian, Q. (2015). Fine-grained visual categorization with fine-tuned segmentation. In *International conference on image processing*.
- Liang, M., & Hu, X. (2015). Recurrent convolutional neural network for object recognition. In *IEEE conference on computer vision and pattern recognition*.
- Lin, M., Chen, Q., & Yan, S. (2014a). Network in network. In *International conference on learning representations*.
- Lin, D., Lu, C., Liao, R., & Jia, J. (2014b). Learning important spatial pooling regions for scene classification. In *IEEE conference on 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. In *IEEE conference on computer vision and pattern recognition*.
- Liu, K., Skibbe, H., Schmidt, T., Blein, T., Palme, K., Brox, T., et al. (2014). Rotation-invariant HOG descriptors using fourier analysis in polar and spherical coordinates. *International Journal of Computer Vision*, 106(3), 342–364.
- 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. In *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. In *IEEE conference on computer vision and pattern recognition*.
- Nagadomi. (2014). The Kaggle CIFAR10 network. <https://github.com/nagadomi/kaggle-cifar10-torch7/>
- Nilsback, M., & Zisserman, A. (2008). Automated flower classification over a large number of classes. In *International conference on computer vision, graphics & image processing*.
- Parkhi, O., Vedaldi, A., Zisserman, A., & Jawahar, C. (2012). Cats and dogs. In *IEEE conference on computer vision and pattern recognition*.
- Paulin, M., Revaud, J., Harchaoui, Z., Perronnin, F., & Schmid, C. (2014). Transformation pursuit for image classification. In *IEEE conference on computer vision and pattern recognition*.
- Perronnin, F., Sánchez, J., & Mensink, T. (2010). Improving the Fisher Kernel for large-scale image classification. In *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. In *European conference on computer vision*.
- Qian, Q., Jin, R., Zhu, S., & Lin, Y. (2015). Fine-grained visual categorization via multi-stage metric learning. In *IEEE conference on computer vision and pattern recognition*.
- Quattoni, A., & Torralba, A. (2009). Recognizing indoor scenes. In *IEEE conference on computer vision and pattern recognition*.
- Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE conference on computer vision and pattern recognition*.
- Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In *IEEE conference on international conference on computer vision*.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (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.
- Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International conference on learning representations*.
- Skelly, L., & Sclaroff, S. (2007). Improved feature descriptors for 3D surface matching. In *Proceedings of SPIE: 6762, two- and three-dimensional methods for inspection and metrology V*
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *IEEE conference on computer vision and pattern recognition*.
- Takacs, G., Chandrasekhar, V., Tsai, S., Chen, D., Grzeszczuk, R., & Girod, B. (2013). Fast computation of rotation-invariant image features by an approximate radial gradient transform. *IEEE Transactions on Image Processing*, 22(8), 2970–2982.
- 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.
- 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.
- Vedaldi, A., & Fulkerson, B. (2010). VLFeat: An open and portable library of computer vision algorithms. In *ACM international conference on multimedia*.
- Vedaldi, A., & Lenc, K. (2014). MatConvNet-convolutional neural networks for MATLAB. [arXiv:1412.4564](https://arxiv.org/abs/1412.4564).
- 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, Z., Fan, B., & Wu, F. (2011). Local intensity order pattern for feature description. In *IEEE conference on 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.
- Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., & Gong, Y. (2010). Locality-constrained linear coding for image classification. In *IEEE conference on computer vision and pattern recognition*.
- Xiao, J., Hays, J., Ehinger, K., Oliva, A., & Torralba, A. (2010). SUN satabase: Large-scale scene recognition from Abbey to Zoo. In *IEEE conference on computer vision and pattern recognition*.
- Xie, L., Hong, R., Zhang, B., & Tian, Q. (2015a). Image classification and retrieval are ONE. In *International conference on multimedia retrieval*.
- Xie, L., Tian, Q., Hong, R., Yan, S., & Zhang, B. (2013). Hierarchical part matching for fine-grained visual categorization. In *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., Tian, Q., & Zhang, B. (2015b). Image classification with Max-SIFT descriptors. In *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. In *IEEE transactions on circuits and systems for video technology*.
- Xie, L., Wang, J., Guo, B., Zhang, B., & Tian, Q. (2014b). Orientational pyramid matching for recognizing indoor scenes. In *IEEE conference on computer vision and pattern recognition*.
- Xie, L., Wang, J., Lin, W., Zhang, B., & Tian, Q. (2015d). RIDE: Reversal invariant descriptor enhancement. In *International conference on computer vision*.
- Yang, J., Yu, K., Gong, Y., & Huang, T. (2009). Linear spatial pyramid matching using sparse coding for image classification. In *IEEE conference on computer vision and pattern recognition*.
- Yang, Y., & Newsam, S. (2010). Bag-of-visual-words and spatial extensions for land-use classification. In *International conference on advances in geographic information systems*.
- Zeiler, M., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*.
- Zhang, N., Donahue, J., Girshick, R., & Darrell, T. (2014a). Part-based R-CNNs for fine-grained category detection. In *European conference on computer vision*.
- Zhang, N., Farrell, R., Iandola, F., & Darrell, T. (2013). Deformable part descriptors for fine-grained recognition and attribute prediction. In *International conference on computer vision*.
- Zhang, S., Tian, Q., Hua, G., Huang, Q., & Li, S. (2009). Descriptive visual words and visual phrases for image applications. In *ACM international conference on multimedia*.
- Zhang, X., Xiong, H., Zhou, W., Lin, W., & Tian, Q. (2016). Picking deep filter responses for fine-grained image recognition. In *IEEE conference on computer vision and pattern recognition*.
- Zhang, X., Xiong, H., Zhou, W., & Tian, Q. (2014b). Fused one-vs-all mid-level features for fine-grained visual categorization. In *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. In *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. In *European conference on computer vision*.