High-Speed and Local-Changes Invariant Image Matching

Chao ZHANG\textsuperscript{(a), Student Member} and Takuya AKASHI\textsuperscript{(b), Member}

SUMMARY In recent years, many variants of key point based image descriptors have been designed for the image matching, and they have achieved remarkable performances. However, to some images, local features appear to be inapplicable. Since these images usually have many local changes around key points compared with a normal image, we define this special image category as the image with local changes (IL). An IL pair (ILP) refers to an image pair which contains a normal image and its IL. ILP usually loses local visual similarities between two images while still holding global visual similarity. When an IL is given as a query image, the purpose of this work is to match the corresponding ILP in a large scale image set. As a solution, we use a compressed HOG feature descriptor to extract global visual similarity. For the nearest neighbor search problem, we propose random projection indexed KD-tree forests (rKDFS) to match ILP efficiently instead of exhaustive linear search. rKDFS is built with large scale low-dimensional KD-trees. Each KD-tree is built in a random projection indexed subspace and contributes to the final result equally through a voting mechanism. We evaluated our method by a benchmark which contains 35,000 candidate images and 5,000 query images. The results show that our method is efficient for solving local-changes invariant image matching problems.

key words: local-changes invariant, image matching, feature compression, random projection indexed KD-tree forests

1. Introduction

During the last decade, image matching and retrieval technology have been widely studied. At the same time, with the development of internet and image editing technology, images are showing more and more diversity in our daily life. The explosion of image data requires image descriptors to be not only lighter but also more discriminative for matching and retrieval tasks. SIFT\textsuperscript{[1]} feature and other key point based image features well solved this problem and are now becoming one of the most popular research branches. However, key point based frameworks have become less effective against images with local changes (IL). IL can not be defined by a single image since “changes” exist with respect to a normal image. IL is an image in an IL pair (ILP). An ILP includes two images: a normal image and its IL.

Main reasons for why local features are less effective in matching ILP can be concluded as: 1) In some IL, detection of corner points is difficult. 2) An ILP may contains many local changes (e.g. an ILP contains two photos which are taken at same place but in different seasons). 3) Multiple similar regions may exist in an IL. An ILP can be image with image, image with sketch, image with noise image, image with painting, image with synthetic image, image with blur image, image with edge image, etc. Figure 1 shows some examples of ILP. As we can see, various local changes can be considered such as changes of illumination, color, edge, shape, texture. The SURF\textsuperscript{[2]} descriptor failed in matching with each key point, which will lead to a failure of matching the whole image.

Image matching is a basic research topic for various applications of computer vision, such as near-duplicate image detection (NDID), content-based image retrieval (CBIR), texture classification. The task is to search a large database of candidate images with a query image and then finds the result which matches to the query image. In our problem, there are one query image as the input and one result image as the output. The input and output are exactly the same without the local changes. Most of the image matching problems are studied from two points of view: 1) image feature presentation 2) nearest neighbor (NN) search technique.

Low-level features, such as histogram based gradient features and key point based features, usually have trade-off problem between the number of dimensions and the discriminative ability. In other words, features composed of more dimensions usually have stronger ability to present...
an image’s visual similarity. Although higher-level features learned by sparse coding/deep learning can surely present image’s visual similarity well with less dimensions, the coding process is time consuming. In our research, the original high-dimensional HOG feature is projected onto a low-dimensional subspace while trying to keep discriminative ability based on [3]. According to the compressive sensing theory, a small number of randomly generated linear measurements can preserve most of the salient information. The projection processing does not cost much time when the projection matrix is very sparse.

On the other hand, most of the current data structures for effective NN search can only index data points in low-dimensional feature space. These data structures become less efficient with the growing of dimension number due to the curse of dimensionality. The difficult point is that it is hard to solve the exact NN problem efficiently in high-dimensional feature space while the accuracy is low when solving the exact NN problem in low-dimension feature space. In this paper, our solution is to find numerous approximate nearest neighbors (ANN) in thousands of low-dimension feature spaces and then vote for the best ANN as the final output. By doing this, we can accelerate the matching procedure while achieve satisfactory matching accuracy. To the best of our knowledge, there is little study on image matching problem with various types of local changes. Our method builds a large number of random projected subspaces to reduce noise’s effect brought by local changes, and finally grasps the global similarity between query and candidate images.

2. Related Work

2.1 Feature Descriptors for Visual Similarity Evaluation

Among recent state-of-art works, local feature descriptors for quantifying images’ visual similarity have been proved to be very effective. SIFT [1] and its variants are representative. Furthermore, D.C. Hauagge et al. [4] proposed a local feature descriptor which based on detecting and representing local symmetries for matching pairs of photos taken at urban scenes. K. Grauman et al. [5] proposed a technique that compares images by matching their distributions of local invariant features.

On the other hand, global feature descriptors are also used for evaluating the visual similarity. A. Oliva et al. [6] noted the global image features play an important role on scene perception. S. Lazebnik et al. [7] noted that a global feature representation can be surprisingly effective for identifying the overall scene. P. Li et al. [8] proposed a method to enrich the discriminative ability of local feature with global information. They noted that the current local descriptors will fail to match when an image has multiple similar regions. C. Zhang et al. [9] proposed a compressed HOG descriptor for IL image matching. They used random projection to compress the high-dimensional HOG feature into low-dimensional feature. However, in matching procedure, only a simple brute-force method with L1 distance measure is applied.

2.2 Image Matching and NN Search

NN search problem for image matching has been widely studied. For exact image matching, brute-force is an efficient method especially the number of feature dimension is large. A. Torralba et al. [10] applied brute-force search to match images which are converted into binary code from GIST descriptor. When the number of feature dimension is small, many data structures can be applied for image matching such as Kd-tree, R-tree, Ball tree, SR-tree. C. Silpa et al. [11] introduced an optimized Kd-tree [12] algorithm which is used to match SIFT descriptors. On the other hand, instead of finding the nearest image to the query image, approximate image matching aims to find images which are within a certain distance threshold to query image, such as image retrieval. ANN search can deal with the high-dimensional feature effectively by reducing the dependency on dimensionality. Y. Ke et al. [13] employed local sensitive hashing (LSH) to index the local descriptors for near-duplicate detection. LSH also applied the random projection for searching ANN over high-dimensional data. P. Wu et al. [14] used multiple randomly projected kd-trees to search ANN. Each kd-tree search the ANN in a random projected low-dimensional space and rank the results by distance at last.

2.3 Image Matching with IL

To the best of our knowledge, there are few papers for studying all types of IL. Sketches and paintings are most studied problems belong to ILP matching. A. Shrivastava et al. [15] defined IL as cross-domain images, the authors mainly considered the matching task for sketches, paintings and photos taken in different seasons which are all included in the definition of IL. They learned the weights for each HOG feature’s dimension with single positive query image and a very large set of negative images by SVM. The training process is very time consuming and hard to be finished within query time. Other similar papers include [16] for matching sketches with photographs, [17] for matching paintings with photographs, [18] for matching images under different illumination conditions. Furthermore, [9] used a compressed HOG descriptor and brute-force NN search to match the ILP. In this paper, the dimension number of original HOG descriptor is reduced from 6384 to 500 in order to reduce the burden of matching time. However, after projection with a single random sparse matrix, the original feature lost original information naturally. The balance between matching accuracy and matching time is still not be solved well in this paper.

Our work is mainly based on work [9], [14]. We use the feature descriptor proposed in [9] and enhance the ANN method proposed in [14] for high-speed exact ILP matching.
3. Methodology

3.1 Problem Setting

We have a set \( P_c \) of \( n \) pre-processed candidate image feature vectors \( \{p_1, p_2, \ldots, p_n\} \), where \( p_i \in \mathbb{R}^m \), \( m \) is the number of compressed feature vector’s dimension. Given an arbitrary query feature vector \( q_n \in \mathbb{R}^m \) from query set \( P_q \), return \( p_i \) which is closest to \( q_n \) under the distance measurement function. In the following sections, we will first introduce how to generate set \( P_c \) and \( P_q \) from candidate image set \( I_c \) and query set \( I_q \), and then we will introduce how to find \( p_i \) by using random projection indexed KD-tree forests (rKDFs).

3.2 Feature Compression

HOG feature \([19]\) counts occurrences of gradient orientations in cells/blocks/windows and merge them into one feature vector \( p' \in \mathbb{R}^n \), \( n \) is the dimension number of original HOG feature. In this paper, we treat the whole image as a single window, and construct grid-like structure for extracting feature with units called block and cell. We define \( R \) as a \( m \times n \) random measurement matrix, and \( r_{ij} = R(i, j) \) denotes the entry in row \( i \), column \( j \) of matrix \( R \). Each \( r_{ij} \) is independent with others and decided by the following probability distribution,

\[
 r_{ij} = \sqrt{s} \times \begin{cases} 
 +1 & \text{with probability} \quad \frac{1}{2s} \\
 0 & \text{with probability} \quad 1 - \frac{1}{s} \\
 -1 & \text{with probability} \quad \frac{1}{2s} 
\end{cases}, \quad (1)
\]

Achlioptas et al. \([3]\) state that when the \( s = 1 \) or \( 3 \), \( R \) satisfies the Johnson-Lindenstrauss lemma. Such kind of matrices can achieve favorable compression performance. The method of \([14]\) also uses \( s = 3 \) to generate the random measurement matrix. When \( s = 3 \), only \( 1/3 \) data need to be processed. However, when the size of candidate image set \( I_c \) is very large, the procedure of pre-processing becomes time consuming. For each query image, although compression operation only needs to be performed once, we hope to avoid large amount of numerical calculation in order to reduce query time as much as possible. Fortunately, this random sparse matrix has been proved to be effective even \( s \gg 3 \) \([20]\). In this paper, we set \( s > n/2 \). Therefore, only \( 2/n \) data need to be processed at most. Parameter \( s \) is determined by rule of thumb. For example, in both \([20]\) and \([9]\), \( s \) is set as \( n/4 \) for efficient compression procedure. In addition, since no floating-point arithmetic is needed expect a square root operation, the compression process needs little computational cost. Also, this random measurement matrix only needs to be generated once during the pre-processing procedure.

The process of compression can be seen as a projection from the high-dimensional space to low-dimensional space.

We define \( p' \) as high-dimensional HOG feature (\( p' \in \mathbb{R}^n \)), \( p \) as low-dimensional compressed feature (\( p \in \mathbb{R}^m \)). For the sparse random projection, \( n > m \). The compression procedure can be presented as,

\[
 p'^{(m \times 1)} = R^{(m \times n)} p^{(n \times 1)}. \quad (2)
\]

This quick and simple matrix multiplication complies our requirements for computing speed. However, with larger \( s \), the loss of feature’s information will be unavoidable. We apply multiple random matrices to remedy this problem. We use \( p_{ij} \) to denote the feature vector of image \( I_i \), which is compressed by random matrix \( R_{ij} \). As a result, each image will be presented by \( \gamma \) compressed feature vectors in total instead of a single vector. Although dimension number of each \( p_{ij} \) is much smaller than the descriptor proposed in \([9]\), the combination of all the \( p_{ij} \) can hold more information. Furthermore, lower-dimensional \( p_{ij} \) is much easier to be processed by KD-tree.

The theoretical foundation of why such a simple matrix can do data compression well is proved in \([21]\). R. Baraniuk \textit{et al.} give a simple proof that \( R \) satisfies the restricted isometry property. At the same time \( R \) satisfies the Johnson-Lindenstrauss lemma, thus it has high probability to reconstruct \( p' \) from \( p \) with minimum error. Figure 2 shows the feature compression procedure. \( R \) is created with Eq. (1), black squares represent positive entries, and the white squares represent negative entries. In order to calculate \( i \)th dimension’s value \( p_{ij} \), dimensions of \( p' \) are randomly selected and combined according to \( R \).

The whole compression procedure can also be considered as a procedure to improve the original HOG’s feature level. The problem of HOG feature is that it is not clear which dimension of \( p' \) performs a more important role in further application, which dimension of \( p' \) is useless. After compression, each dimension of \( p \) is calculated from multiple dimensions of \( p' \), thus more information is included in \( p \)’s single dimension than \( p' \). As we all know that with higher level features, less dimensions are needed to hold the same discriminative ability. From this point of view, we can also understand why the random projection works for feature dimension reduction with less loss of discriminative...
Algorithm 1 Feature extraction and compression.

Require: Candidate image set : I_c
Require: Query image set : I_q
Require: Compressed feature set of candidate images: P_c
Require: Compressed feature set of query images : P_q
Require: Number of random matrices : γ
Require: Parameter : s
1: for i from 1 to γ do
2: Generate projection matrix R_i with Equation 1 and s
3: end for
4: for each image I_i in I_c ∪ I_q do
5: I_i = gaussianBlur(I_i)
6: p_i’ = extractHOG(I_i)
7: for j from 1 to γ do
8: p_ij = compress(p_i’, R_j)
9: I_j = normalize(p_ij)
10: if I_j ∈ I_c then
11: Push p_ij to P_c
12: else
13: Push p_ij to P_q
14: end if
15: end for
16: end for
17: Return P_c and P_q

ability.

The preprocessing algorithm can be concluded with Algorithm 1.

3.3 Random Projection Indexed KD-tree Forests

KD-tree [22] is a widely used tree structure for searching ANN in multi-dimensional data space. It is a binary tree with each node has a hyper-plane (typically one dimension) to divide the data space into two subspaces. The feature vectors which are left to the hyper-plane will be assigned to left child node, and the feature vectors which are right to the hyper-plane will be assigned to right child node. As one of the ANN algorithms, KD-tree works effectively when dealing with low-dimensional data. However, KD-tree works poorly especially the number of feature vector’s dimension is large since it will degrade to linear search [23]. In our matching problem, KD-tree seems to be inapplicable because the dimension number of HOG feature is large. By using the compression method mentioned above, dimension number of HOG feature can be reduced. In our condition, we set compressed feature vector’s dimension extremely small to build KD-tree in an effective way (e.g., n = 6384, m = 10). Such KD-tree is very light both in memory and search time. However, much information on the original feature vectors will be lost naturally and ANN of query ICL becomes hard to search by single KD-tree. To solve this problem, our idea is to build a large scale KD-tree forests (e.g., α = 8,000) with each tree indexed by a random matrix. Each tree in rKDFs is built with different input compressed data sets which are generated by different random matrices. “indexed” in rKDFs means that we use one random matrix to discriminate a certain tree from others. ANN results returned by a single KD-tree in rKDFs are very inaccurate but better than random guesses, because the compressed feature space’s dimension is too small to reflect the original feature space’s data distribution. We vote with ANNs provided by each tree by a histogram and at last select the ANN which is most voted as the final output. Figure 3 illustrates the whole processing. P_cj denotes a compressed feature set which is compressed via random matrix R_j.

We now introduce how to build rKDFs. P_c includes γ feature subspaces which are returned by Algorithm 1. We build δ trees in one subspace in parallel. δ trees in one subspace form a tree group. For each tree, data in the according subspace is partitioned recursively from the root node to leaf nodes. In initialization process, dimensions with large variance are selected as candidate dimensions to partition the data (e.g., five dimensions). At each node, we first randomly select a dimension for splitting and then calculate the median value of this dimension. After that, all the feature vectors in the node will be split into two child nodes according to the median value. The split operation will stop until the depth of the tree reaches to the depth threshold λ. In order to store all the trees, we need space complexity about O(αm × |P_c|). The building algorithm is concluded in Algorithm 2.

We now introduce how to search with rKDFs. To find the best ANN of a given feature vector p ∈ P_q, we need to search with γ × δ trees. After preprocessing, p has already been projected into γ subspaces. In each subspace, we search ANNs of p with a tree group which is returned by Algorithm 2. However, these ANNs are very inaccurate since each is outputted by a single tree. In order to boost the accuracy, ANNs searched by a tree group are ranked by distance and output best β ANNs for voting the final NN. The voting mechanism is established under this assumption: exact NN of a query image has higher probability to appear.
in the ANNs of each sub feature space. We need time complexity about \(O(an \times \log \{P_c|\})\) to search with one query. The searching algorithm is concluded in Algorithm 3.

The differences between our matching method and [14] can be concluded as following. 1) We introduced randomized kd-tree forests [24] to divide tress into groups according to different subspaces. 2) Method of [14] uses only about 20 trees to search ANNs, in our condition, number of trees is 8,000 and more. 3) Method in [14] ranked all the ANNs by distance and treat the top rank which is closest to the query in subspace as the final NN. This method will become less effective when the dimension number of original feature vector is very large like the HOG feature. Because the distance measurement in subspace can not well reflect the distance in the original feature space. Our method vote with all the ANNs to determine the final NN which appeared most frequently as an ANN. 4) Our method compresses feature dimension from thousands to 10 while method of [14] compresses feature dimension from hundreds to 10. Random projection is a random method which does not depend on any training data, thus building large number of KD-trees in a same subspace is risky and unwarranted. Our method disperses the risk to each subspace and achieve better performance overall.

**Algorithm 2** Build rKDFs.

**Require:** Compressed feature set of candidate images: \(P_c\)

**Require:** Number of random matrices : \(\gamma\)

**Require:** Number of trees in one group : \(\delta\)

**Require:** Maximum depth of one tree: \(\lambda\)

Ensure: \(|P_c| > 0\)

1: for \(j\) from 1 to \(\gamma\) do
2: for \(i\) from 1 to \(\delta\) do
3: Initialize KD-tree \(KT\) with root node \(n\) and data \(P_{cj}\)
4: while depth\((n) < \lambda\) do
5: splitNode\((n)\)
6: \(n = \text{findLeaf}(KT)\)
7: end while
8: Push \(KT\) into tree group \(KG_j\)
9: end for
10: end for
11: Return KD-tree groups \(KG\)

**Algorithm 3** Search with rKDFs.

**Require:** Compressed feature set of query images : \(P_q\)

**Require:** Compressed feature set of candidate images : \(P_c\)

**Require:** Number of random matrices : \(\gamma\)

**Require:** KD-tree groups \(KG\)

**Require:** Number of ANNs outputted by a single KD-tree group: \(\beta\)

Ensure: \(|P_q| > 0\) and \(|P_c| > 0\)

1: for \(i\) from 1 to \(|P_q|\) do
2: initialize histogram with \(|P_c|\) bins
3: for \(j\) from 1 to \(\gamma\) do
4: \(ANN = \text{search}(KG_j, P_{cj}, \beta)\)
5: \(\text{vote(histgram, ANN)}\)
6: end for
7: Return arg max \(\sum_{i=1}^{\beta} \text{bin}_i\)
8: end for

| Sub figure No. | \(\alpha\) | \(\beta\) | \(s\) | \(d\) | \(k\) | \(n\) | \(m\) |
|---------------|----------|----------|-------|-------|-----|-----|------|
| (a)           | –        | 10       | 6000  | 35000 | 31  | 6384| 10   |
| (b)           | 400      | –        | 6000  | 35000 | 31  | 6384| 10   |
| (c)           | 400      | 10       | –     | 35000 | 31  | 6384| 10   |
| (d)           | 8000     | 10       | 6000  | –     | 31  | 6384| 10   |
| (e)           | 8000     | 10       | 2000  | 35000 | –   | 2700| 10   |
| (f)           | 400      | 10       | 6000  | 35000 | 31  | 6384| –    |

4. Experiment

4.1 Experiment Environment

We use the benchmark\(^*\) used in [9] to evaluate our method. It is a challenging benchmark which contains 5,000 query images and 35,000 candidate images. 5,000 query images are modified with local changes based on normal images which are randomly selected from 35,000 candidate images. Width of images is between 454 pixels to 1272 pixels, the of images is between 482 pixels to 1024 pixels. Many types of IL are included in the query set, and the local changes can be mainly concluded into three categories: changes of color-texture information, changes of edge-gradient information, and changes with special filters. Color-texture information can be changed by the adding of text and scribbling, illumination changes, image binarization, etc. Edge-gradient information can be changed by image rotation, local deformation, text adding, scribbling, etc. Special image filters will largely change image’s local feature and keep global similarity like crayon drawing, oil paint, pencil drawing, pixel explosion, stained glass, etc. This benchmark is an one-to-one matching task benchmark, a query image and its ground truth are exactly the same without the local changes. To the best of our knowledge, there are few similar benchmarks for one-to-one image matching task involving various types of local changes.

We did all the experiments with a PC equipped with Intel Core-i5 2.5GHz CPU and 6 GB RAM.

4.2 Effect of Parameters

In this section, we systematically report the experimental results for studying how each parameter affects the performance of our matching method. In this paper, some parameters are fixed to reduce complexity of experiment. We set the number of trees \(\delta = 4\) in each tree group, image size as \(320 \times 240\), gradient angle’s range as \([0^\circ, 180^\circ]\), \(\sigma_x\) of Gaussian blur as 8 and \(\sigma_y\) of Gaussian blur as 6. Figure 5 (shown in last page) summarizes the effects of various parameters. Experimental conditions for each sub figure are given out.

\(^*\)This benchmark is publicly available for repeatability and comparison. It can be downloaded from the URL below: http://cvhost.scv.cis.iwate-u.ac.jp/research/projects/Local_Changes_Invariant_Image_Matching_Database.html
Table 2 Example of parameter settings for extracting HOG feature. \( n \) is the dimension number of HOG feature. To calculate the error rate, we set \( \alpha = 8000 \), \( \beta = 10 \), \( d = 35000 \), \( k = 31 \), \( m = 10 \).

| \( n \) | error rate | block | block stride | cell | bin | s |
|-------|------------|-------|--------------|------|-----|---|
| 1836  | 0.149      | (64,64)|(16,16)      | (64,64)| 9  | 1000|
| 2396  | 0.099      | (32,32)|(16,16)      | (32,32)| 9  | 2000|
| 2700  | 0.066      | (16,16)|(16,16)      | (16,16)| 9  | 2000|
| 5508  | 0.110      | (64,64)|(16,22)      | (32,32)| 9  | 5000|
| 6384  | 0.076      | (32,32)|(16,16)      | (16,16)| 6  | 6000|
| 9576  | 0.071      | (32,32)|(16,16)      | (16,16)| 9  | 6000|
| 12768 | 0.080      | (32,32)|(16,16)      | (16,16)| 12 | 6000|
| 29576 | 0.087      | (64,64)|(16,16)      | (16,16)| 9  | 6000|
| 35964 | 0.092      | (32,32)|(8,8)        | (16,16)| 9  | 6000|

Fig. 4 Examples of comparison with method [9] in Table 1. Two evaluation criteria are observed during experiments: error rate and matching time per query image in milliseconds. Error rate is defined as follows,

\[
\text{error rate} = 1.0 - \frac{\sum_{p \in P_q} \text{match}(p, P_c)}{|P_q|},
\]

match function returns 1 if a query image can be approximately matched according to ground truth, returns 0 if not.

As Fig. 5 (a) shows, increasing the number of trees \( \alpha \) of rKDFs improves the performance significantly. Error rate stops decreasing from a certain value of \( \alpha \). The matching time per query image increases linearly as \( \alpha \) increases. As Fig. 5 (b) shows, when the number of ANNs outputted by each tree group is increased, the voting process appears to be more accurate. As Fig. 5 (c) shows, with the increase of \( s \), error rate declines in a stepwise fashion. Larger \( s \) leads the algorithm to generate a more sparse random matrix to compress the original HOG feature vector. In our method, larger \( s \) shows to be a more appropriate choice. As one of the possible reasons, excessive compression may cause bad influence on calculating the visual similarity of IL conversely. As Fig. 5 (d) shows, with the increase of candidate images, error rate maintains. This can explain that our method is robust in change of search space. We can also find that the processing time increase linearly with the increase of \( d \), this is very important for practical applications. From Fig. 5 (e) and (f), we can find that there exists minimum error rate while increasing \( k \) and \( m \) step by step. We can tune both the parameters by a validation set. Furthermore, the dimension number of original HOG feature vector \( n \) also plays an important role. Some parameter tuning examples are shown in Table 2. We found out that parameter setting with 9 bins performs better than 6 bins, the best accuracy is achieved when block size, cell size, block stride are all set to (16,16). After tuning on a validation data set, we report the lowest error rate in the next section.

4.3 Comparison

We compare our method with others from two aspects for the different needs of practical applications, 1) Considering accuracy as priority, 2) Considering matching time as priority. The compared methods are listed below:

- **N-BoF-SIFT**: In experiment, we combine BoF with SIFT [1]. The visual vocabulary is built by randomly sampled images from the candidate dataset and each packaged feature is finally normalized. We extract 128 dimension SIFT descriptors for all the detected key points and then use K-means clustering method, which is usually used in many BoF implementations, to cluster visual words. Initial centroid positions of K-means are chosen according to [25]. For the assignment task, we use fast approximate nearest neighbor (FLANN) [26] to assign the novel features to the closest terms in the vocabulary. After normalization, we use the packaged feature to match the data set by L1 distance.

- **RP-HOG**: The method in [9] compresses the HOG feature with random projection and then match the NN by brute-force with L1 distance measurement.

- **N-HOG**: Original HOG features [19] are extracted from each image and then normalized by L2 norm. We use brute-force method to match the NN with L1 distance measurement.

- **GIST**: Gist feature [6] is a global image feature which convolutes a gradient filter to encode the amount and strength of edges. After Gist is extracted, we use brute-force method to match the NN with L1 distance measurement.

- **Fisher Vector**: GMM is used to construct a visual word dictionary at first. We extract SIFT feature as the local feature of each image. Fisher Vector is encoded by the SIFT feature and the prior obtained GMM, and finally normalized by L2 norm.

- **VLAD**: VLAD can be seen as the simplification of Fisher Vector. In experiment, K-Means is used instead of GMM for visual word generating, and KD-tree is used for vector quantization. It is also normalized by L2 norm at last.

Table 3 and 4 show the comparison results. From Table 3 we can see that SIFT appears to be very ineffective
Fig. 5 Effect of parameters on error rate and processing time per query image. (a) Increasing the number of trees: $\alpha$. (b) Increasing the number of ANNs of each tree group: $\beta$. (c) Increasing the number of compression factor: $s$. (d) Increasing the size of candidate dataset: $d$. (e) Increasing the size of blur kernel: $k$. (f) Increasing the dimension number after compression: $m$.

Fig. 6 Examples of successful ILP matching.

even the visual words are set to 10,000. By this point we can prove the effectiveness of our dataset for evaluating the KF images. Furthermore, our method is about 54% faster than N-HOG at a same accuracy level. We successfully converted a high-dimensional feature matching problem into a low-dimensional matching problem and improved the accu-
racy. From Table 4 we can see that our method is about 74% faster than RP-HOG at a same accuracy level and outperforms other methods both in accuracy and time. Although Fisher Vector and VLAD can perform very well in standard image retrieval tasks, it can not perform well in our problem. The main reason can be concluded as: both of the methods are developed based on the local features such as SIFT, the matching error brought by local features can be further expanded during the transformation of features. As a conclusion, our method can match ILP in high-speed with large scale candidate database, at the same time, accuracy is satisfactory. Dense sampling methods such as SIFT Flow [27] is recently showing the effectiveness. However, to compute a 128-dimensional SIFT feature for each pixel is very time consuming and impractical. With the increase of data base’s size, both the time and space complexity grow dramatically.

Figure 4 intuitively shows some matching examples comparing to [9]. A synthesized image example and a distorted image example are shown to be mismatched by [9] while our method can still match correctly.

5. Limitations and Future Work

This paper presented a problem which aims to match a special category of images called IL. The proposed method applied a compressed HOG descriptor for extraction and introduced rKDFs for high-speed NN search. There still exist some limitations in practical applications. The main limitation is that compressed HOG descriptor is not rotation invariant, thus IL presented in different rotation angles will fail in matching. Furthermore, our method will fail in matching when multiple candidate images in a candidate data set appear to be visually similar (e.g., successive frames in video). Our method will also fail in matching when both images showing a same basic geometric shape (e.g., an image of sun and an image of ball).

In the future, we plan to develop rotation invariant descriptor which can also grasp global visual similarity in order to solve the problems mentioned above. Furthermore, since each tree can perform matching independently, the matching process can be further accelerated by GPU.

References

[1] D.G. Lowe, “Distinctive image features from scale-invariant keypoints,” Int. J. Comput. Vision., vol.60, no.2, pp.91–110, 2004.

[2] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” European Conference on Computer Vision (ECCV), vol.3951, pp.404–417, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[3] D. Achlioptas, “Database-friendly random projections: Johnson-lindenstrauss with binary coins,” J. Comput. Syst. Sci., vol.66, no.4, pp.671–687, 2003.

[4] D.C. Hauagge and N. Snavely, “Image matching using local symmetry features,” 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp.206–213, IEEE, 2012.

[5] K. Grauman and T. Darrell, “Efficient image matching with distributions of local invariant features,” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), pp.627–634, 2005.

[6] A. Oliva and A. Torralba, “Building the gist of a scene: The role of global image features in recognition,” Progress in Brain Research, vol.155, pp.23–36, 2006.

[7] S. Lazebnik, C. Schmid, and J. Ponce, “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories,” 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR’06), pp.2169–2178, 2006.

[8] P. Li, H. Yan, G. Cui, and Y. Du, “Image local invariant features matching using global information,” 2012 IEEE International Conference on Information Science and Technology, pp.627–633, 2012.

[9] C. Zhang and T. Akashi, “Compressive image retrieval with modified images,” Asian Control Conference (ASCC), pp.814–819, IEEE, 2015.

[10] A. Torralba, R. Fergus, and Y. Weiss, “Small codes and large image databases for recognition,” 2008 IEEE Conference on Computer Vision and Pattern Recognition, pp.1–8, 2008.

[11] C. Silpa-Anan and R. Hartley, “Optimised kd-trees for fast image descriptor matching,” 2008 IEEE Conference on Computer Vision and Pattern Recognition, pp.1–8, 2008.

[12] J.H. Friedman, J.L. Bentley, and R.A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” ACM Trans. Math. Softw., vol.3, no.3, pp.209–226, 1977.

[13] Y. Ke, R. Sukthankar, and L. Huston, “Efficient near-duplicate detection and sub-image retrieval,” ACM Multimedia (ACMMM), p.5, 2004.

[14] P. Wu, S.C.H. Hoi, D.D. Nguyen, and Y. He, “Randomly projected kdtrees with distance metric learning for image retrieval,” Advances in Multimedia Modeling, Lecture Notes in Computer Science, vol.6524, pp.371–382, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[15] A. Shrivastava, T. Malisiewicz, A. Gupta, and A.A. Efros, “Data-driven visual similarity for cross-domain image matching,” ACM Trans. Graph., vol.30, no.6, pp.154:1–154:10, 2011.

[16] M. Eitz, K. Hildebrand, T. Boubekeur, and M. Alexa, “Sketch-based image retrieval: Benchmark and bag-of-features descriptors,” IEEE Trans. Vis. Comput. Graphics, vol.17, no.11, pp.1624–1636, 2011.

[17] B.C. Russell, J. Sivic, J. Ponce, and H. Dessaules, “Automatic alignment of paintings and photographs depicting a 3d scene,” 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp.545–552, 2011.

[18] H.Y. Chong, S.J. Gortler, and T. Zickler, “A perception-based color space for illumination-invariant image processing,” ACM Trans. Graph., vol.27, no.3, p.61, 2008.

[19] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), ed. C. Schmid, S. Soatto, and C. Tomasi, pp.886–893, 2005.

[20] C. Zhang, Y. Yamagata, and T. Akashi, “Robust visual tracking of moving objects,” Int. J. Comput. Vision., vol.60, no.2, pp.91–110, 2004.
[22] J.T. Robinson, “The K-D-B-tree: a search structure for large multidimensional dynamic indexes,” ACM SIGMOD Conference on Management of Data (SIGMOD), pp.10–18, ACM, 1981.

[23] A. Gionis, P. Indyk, R. Motwani, et al., “Similarity search in high dimensions via hashing,” International Conference on Very Large Databases (VLDB), pp.518–529, 1999.

[24] A. Vedaldi and B. Fulkerson, “Vlfeat: An open and portable library of computer vision algorithms,” ACM Multimedia (ACMMM), pp.1469–1472, ACM, 2010.

[25] D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding,” ACM-SIAM Symposium on Discrete Algorithms (SODA), pp.1027–1035, Society for Industrial and Applied Mathematics, 2007.

[26] M. Muja and D.G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP), pp.331–340, 2009.

[27] C. Liu, J. Yuen, and A. Torralba, “Sift flow: Dense correspondence across scenes and its applications,” IEEE Trans. Pattern Anal. Mach. Intell., vol.33, no.5, pp.978–994, 2011.

[28] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, “Visual categorization with bags of keypoints,” European Conference on Computer Vision (ECCV), pp.22, 2004.

[29] F. Perronnin and C. Dance, “Fisher kernels on visual vocabularies for image categorization,” 2007 IEEE Conference on Computer Vision and Pattern Recognition, pp.1–8, IEEE, 2007.

[30] J. Delhumeau, P.-H. Gosselin, H. Jégou, and P. Pérez, “Revisiting the vlad image representation,” ACM international conference on Multimedia (ACMMM), pp.653–656, 2013.

Chao Zhang received a B.E. in the the College of Information Engineering from Northwest A & F University in 2011. He is now a Ph.D. candidate at Design and Media Technology, Graduate School of Engineering, Iwate University. His research interests include image matching, online tracking, and object detection. He is a student member of the IEICE and IEEE.

Takuya Akashi received a Ph.D. in the System Design Engineering from the University of Tokushima in 2006. Since April 2009, he has been with the Department of Electrical Engineering, Electronics and Computer Science, Iwate University. He is currently an associate professor. His research interests include evolutionary algorithms, image processing and human sensing. He is a member of the IEEE, RISP and IEICE.