Evaluation of Product Quantization for Image Search

Wei-Ta Chu[1], Chun-Chang Huang[1], Jen-Yu Yu[2]

[1]Department of Computer Science and Information Engineering
National Chung Cheng University, Taiwan
wtchu@cs.ccu.edu.tw, icecandyandscheme@gmail.com

[2]Information and Communication Research Lab,
Industrial Technology Research Institute, Hsinchu, Taiwan
KevinYu@itri.org.tw

Abstract. Product quantization is an effective quantization scheme, with that a high-dimensional space is decomposed into a Cartesian product of low-dimensional subspaces, and quantization in different subspaces is conducted separately. We briefly discuss the factors for designing a product quantizer, and then design experiments to comprehensively investigate how these factors influence performance of image search. By this evaluation we reveal design principles that have not been well investigated before.

Keywords: Product quantization, image search.

1 Introduction

Measuring distances, e.g., Euclidean distances, between data points is a fundamental step for data clustering, classification, and retrieval. In many applications, such as video annotation [10] and image retrieval [11], with such distances the nearest neighbors to a query data point are thus determined. However, in many multimedia applications, we have to process tremendous amount of high-dimensional data points. Exact and exhaustive distance measurement thus suffers from efficiency issues and curse of dimensionality.

Rich methods have been proposed for efficient multidimensional indexing. For example, the KD-tree [1] describes data points by dividing the data space, and search time can be effectively reduced. However, for high-dimensional data, the method is not much more efficient than the brute-force search. Approximate nearest neighbor (ANN) search is thus proposed. Instead of finding the exact nearest neighbors, ANN approaches find N data points that are nearest to a query point with high probability. The Euclidean Locality-Sensitive Hashing (E2LSH) [2], for example, is a widely adopted ANN approach that hashes similar high-dimensional data points into the same bucket with high probability. However, memory needed to store the indexing structure of E2LSH is critical. For applications such as large-scale scene recognition or content-based image retrieval, the trade-off of memory usage of indexing structure and search efficiency becomes severe.

Jegou et al. [3] propose a scheme called product quantization (PQ) to construct a large codebook with little cost. The concept is to “decompose a high-dimensional space into a Cartesian product of low-dimensional spaces and to quantize each subspace separately.” A high-dimensional vector $\boldsymbol{x} = (x_1, x_2, ..., x_D)$ is first divided into lower-dimensional subvectors $\{\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_p\}$, where $\boldsymbol{u}_i = (x_{(i-1)\times d+1}, x_{(i-1)\times d+2}, ..., x_{i\times d})$ is the *i*th d-dimensional subvector, $d \ll D$. For the training vectors $\{\boldsymbol{x}_k\}_{k=1}^{N}$, their *i*th corresponding subvectors $\{\boldsymbol{u}_i\}$ are collected and clustered by the K-means algorithm to form a small codebook. The codebooks constructed based on each set of subvectors are then combined by the Cartesian product to form a large codebook. This scheme effectively constructs a large codebook for multidimensional indexing. PQ has been demonstrated to be scalable to large amount of data [3] and suitable to mobile visual search [9].

Overall, PQ significantly reduces the time required to construct a large codebook, reduces the influence of the curse of dimensionality, and reduces the memory space for storing the codebook. As stated in [3], performance of PQ can be analyzed from the following three aspects, but the authors did not conduct deep investigation in their paper.

- The way to splitting vectors
- The dimension of subvectors
- The number of quantization levels for each subquantizer

Details of these factors will be described in the next section. In this paper, we describe images by bag of visual words based on various codebooks, which are constructed by product quantizers based on different settings. We investigate how different codebooks affect performance of content-based image retrieval. From experimental results, we make a few suggestions for codebook construction.

In the following text, we briefly describe product quantization and the issues to be addressed in Section 2. Section 3 describes evaluation settings, including feature extraction and construction of product quantizers. Various experimental results and discussion are provided in Section 4, followed by the conclusion in Section 5.

2 Product Quantization

Let $\boldsymbol{x} \in R^d$ be a query vector and $\mathcal{Y} = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_n\}$ be the set of vectors in a database. We wish to find the nearest neighbor of $\boldsymbol{x}$ from the database. By the definition of asymmetric distance [3], the approximate distance between $\boldsymbol{x}$ and a database vector $\boldsymbol{y}_i$ is computed as

$$d_c(\boldsymbol{x}, \boldsymbol{y}_i)^2 = \|\boldsymbol{x} - q_c(\boldsymbol{y}_i)\|^2, \tag{1}$$

where $q_c(\cdot)$ denotes a quantizer with K centroids. The vector $c_i = q_c(\boldsymbol{y}_i) \in R^d$ is encoded by $\log_2 K$ bits if K is a power of 2.

The approximate nearest neighbor of $\boldsymbol{x}$ is obtained by minimizing the distance

$$NN_a(\boldsymbol{x}) = \arg\min_i \|\boldsymbol{x} - q_c(\boldsymbol{y}_i)\|^2. \tag{2}$$

Note that the "asymmetric" distance means the query vector $\boldsymbol{x}$ is not quantized, but the database vector $\boldsymbol{y}_i$ is. The results reported in [3] show that the asymmetric version significantly outperforms the symmetric version (quantizing both the query vector and database vectors) in terms of memory usage and search accuracy.

To obtain good approximation, the number of quantization levels K should be large. For example, $K = 2^{64}$ if a 64-bit code is desired. However, learning such a big codebook needs tremendous computation, and storing 2^{64} centroids in floating points is not feasible. The work in [3] addresses this issue by decomposing a high-dimensional space into a Cartesian product of low-dimensional subspaces, and quantizes each subspace separately. A database vector $\boldsymbol{y} \in R^d$ is first equally split into m subvectors $\boldsymbol{y}^1, ..., \boldsymbol{y}^m \in R^{d/m}$. A product quantizer is defined as a function

$$q_c(\boldsymbol{y}) = (q^1(\boldsymbol{y}^1), ..., q^m(\boldsymbol{y}^m)), \tag{3}$$

which maps the vector $\boldsymbol{y}$ to a tuple of indices. Each quantizer $q^j(\cdot)$ has K_s quantization levels, and is learned by the K-means algorithm based on the set of the jth subvectors $\{\boldsymbol{y}^j\}$ in the database.

With the product quantization scheme, the combination of q^j, $j = 1, ..., m$, forms a large codebook consisting of $K = (K_s)^m$ centroids. The square distance in eq. (1) is calculated using the decomposition

$$d_c(\boldsymbol{x}, \boldsymbol{y}_i)^2 = \|\boldsymbol{x} - q_c(\boldsymbol{y}_i)\|^2 = \sum_{j=1}^{m} \|\boldsymbol{x}^j - q^j(\boldsymbol{y}^j)\|^2, \tag{4}$$

where $\boldsymbol{x}^j$ is the jth subvector of $\boldsymbol{x}$. Before calculating the distance described in eq. (4), m tables are constructed for a given query. For the ith entry of the jth table, the distance between the subvector $\boldsymbol{x}^j$ and the ith centroid of q^j, $i = 1, ..., K_s$, is stored. The complexity of table generation is $O(\frac{d}{m} \times m \times K_s) = O(d \times K_s)$. When $K_s \ll n$, this complexity can be ignored compared to the summation cost of $O(d \times n)$ in eq. (1).

To briefly describe the idea of product quantization, both [3] and the aforementioned example assume that a high-dimensional vector is equally split into m subvectors. However, different splitting schemes may have significantly different performance. We can investigate a product quantizer from three aspects:

- The way to splitting vectors: It would be better that different subvectors are uncorrelated. On the contrary, consecutive components are usually correlated, and they are better quantized using the same subquantizer.
- The dimension of subvectors: Dimensions of subvectors should be appropriately chosen to avoid the curse of dimensionality.
- The number of quantization levels for each subquantizer: Numbers of quantization levels for different subvectors may be different, depending on the "importance" of corresponding subvectors for a given task.

It is obvious that these three issues are not independent. In [3], by equally splitting SIFT descriptors [4] and GIST descriptors [5] into four or eight subvectors, the authors verify that different component grouping schemes give significantly different performance. They just briefly discuss the first issue. In this paper, we would deeply investigate search accuracy and memory usage from all these three aspects.

3 Evaluation Settings

3.1 Dataset and Features

Similar to [3], we extract feature vectors from three datasets: learning, database, and query. The learning set comes from the first 100k images of the MIRFLICKR image collection [6]. Based on the learning set, various product quantizers are constructed. Both the database set and the query set are from the Holidays image collection [7]. Given a query image, we would like to find its nearest images, based on the representation with product quantization. Figure 1 shows some sample images from the Holidays image collection. They were all captured by amateur photographers with low-end cameras. In retrieving nearest images, some of them are relatively simpler (the left part of Figure 1), and some of them are extremely challenging (the right part).

In contrast to [3], we extract pyramid of histogram of oriented gradients (PHOG) [8] to describe images. For each image, oriented gradients are first computed using a 3×3 Sobel mask with Gaussian smoothing. Orientations of pixels are quantized into B bins. For a given image region, a B-dimensional histogram of oriented gradients (HOG) can be constructed. To form the level ℓ pyramid, the image is divided into 2^{ℓ} cells horizontally and vertically. The final level ℓ PHOG descriptor is a concatenation of all HOG descriptors extracted from cells in levels $0, 1, ..., \ell$. Therefore, for example, a level 1 PHOG descriptor has dimension of $(2^0 \times 2^0 + 2^1 \times 2^1) \times B = 5B$, and a level 2 PHOG descriptor has dimension of $(1 + 4 + 16) \times B = 21B$.

Figure 2 illustrates examples of level 1 PHOG and level 2 PHOG if $B = 16$. The first 16 components (denoted by $\boldsymbol{h}_0$) in level 1 PHOG correspond to the HOG of the cell in level 0, and the following 64 components (denoted by $\boldsymbol{h}_1$) correspond to the HOG of the 4 cells in level 1. Similarly, the last 256 components (denoted by $\boldsymbol{h}_2$) in level 2 PHOG correspond to HOG of the 16 cells in level 2.

The reasons to use PHOG for evaluation are that different scales of information are concatenated as a high-dimensional vector, and components in different dimensions are often not independent. Figure 3 shows two sample correlation matrices between components in different dimensions of level 1 PHOG and level 2 PHOG, respectively. We clearly see that some dimensions are correlated, and from the viewpoint of subvector splitting, they are better categorized into the same subvector. In contrast to SIFT that just encodes gradient information of a local patch and GIST that encodes the global shape distribution, we can design much more variations of splitting schemes for PHOG, and investigate how quantization within levels and across levels affect the performance.

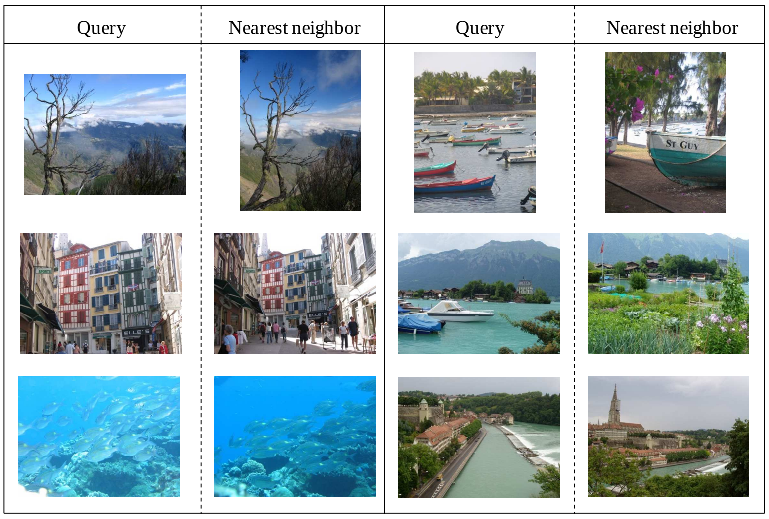

Fig. 1. Sample images from the Holidays image collection.

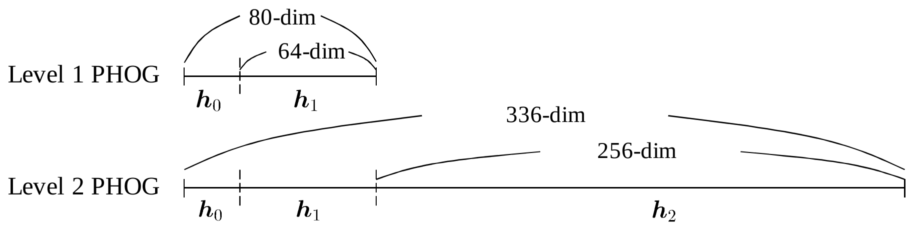

Fig. 2. Illustration of level 1 PHOG and level 2 PHOG.

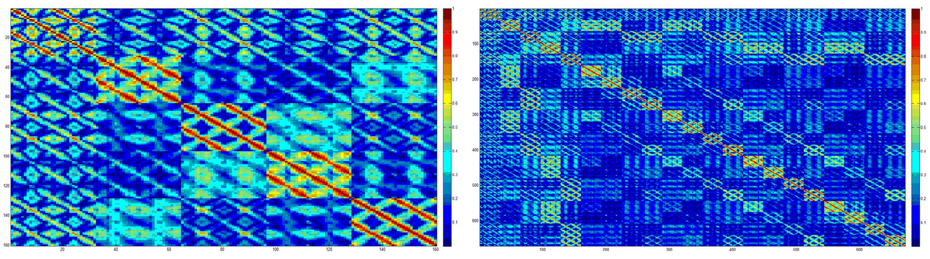

Fig. 3. Two sample correlation matrices for the level 1 PHOG (left) and level 2 PHOG (right). Gradients in both PHOGs are quantized into 32 bins.

3.2 Experiment Settings

We design experiments to investigate how different product quantization schemes influence the performance of nearest image search. The search quality is measured by recall@100, i.e., the proportion of query vectors for which the nearest neighbor is ranked in the first 100 positions [3]. To construct different PHOGs, the orientation of gradients is quantized into 16 or 32 bins ($B = 16$ or 32). Two levels of PHOGs, i.e., level 1 and level 2, are extracted for evaluation. Table 1 shows the overview of our experiment settings.

Two main factors determine how we implement product quantization: how to split a high-dimensional vector into subvectors, and the length of code using to represent a subvector (the number of quantization level of each subquantizer determines the code length). By varying the two factors, we divide our experiments into three categories:

- Category 1: A vector is equally divided into m subvectors, and for different subquantizers the same number of quantization levels is used. As shown in Table 1, a level 1 PHOG may be equally divided into 5, 10, or 20 subvectors. Note that the first two of the 10 subvectors, for example, correspond to the HOG of the cell at level 0. That is, the $\boldsymbol{h}_0$ in Figure 2 is split into two subvectors $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$. Components in the same subvector $\boldsymbol{v}_i,\ i = 0, 1$, belong to the same level of HOG. However, the components $\boldsymbol{v}_0$ convey information of orientations different from that conveyed by the components in $\boldsymbol{v}_1$. A level 2 PHOG may be equally divided into 21, 42, or 84 subvectors.
- Category 2: A vector is equally divided into m subvectors, but for different subquantizers different numbers of quantization levels are used. In the first subcategory Category 2-1, taking the same instance as Category 1, we may respectively construct a subquantizer with 16 quantization levels for $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$, and respectively construct a subquantizer with 32 quantization levels for $\boldsymbol{v}_2, ..., \boldsymbol{v}_9$. In the second subcategory Category 2-2, oppositely, the numbers of quantization levels from $\boldsymbol{v}_0$ to $\boldsymbol{v}_9$ change from large to small.
- Category 3: A vector is unequally divided into m subvectors, and for different subquantizers the same number of quantization levels is used. The level ℓ HOG $\boldsymbol{h}_\ell$ is equally split into two subvectors. That is, for a level 2 PHOG, $\boldsymbol{h}_0$, $\boldsymbol{h}_1$, and $\boldsymbol{h}_2$ are split into six subvectors, $\{\boldsymbol{v}_0, \boldsymbol{v}_1\}$, $\{\boldsymbol{v}_2, \boldsymbol{v}_3\}$, and $\{\boldsymbol{v}_4, \boldsymbol{v}_5\}$, respectively. If $B = 16$, the dimensions of the three sets of subvectors are 8, 32, and 128, respectively. Note that $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ respectively represents half of orientation information of level 0 HOG. The subvectors $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$ respectively represents the level 1 HOG in the left half and right half of an image (please refer to [8] for details of PHOG). We can further equally split each subvectors into shorter subvectors. In this category, dimensions of subvectors corresponding to different levels are different.

Table 1. Overview of experiment settings. The bold-faced numbers correspond to the examples we give in the text.

	#bins of HOG	PHOG levels	#subvectors (m)	#quantization levels
Cat 1	$\{16, 32\}$	$\{1, 2\}$	$\{5, \mathbf{10}, 20\}$ $\{21, 42, 84\}$	$\{8, 16, 32, 64, 128\}$
Cat 2-1	$\{16, 32\}$	$\{1, 2\}$	$\{5, \mathbf{10}, 20\}$ $\{21, 42, 84\}$	$\{(8, 16), \mathbf{(16, 32)}, (32, 64)\}$ $\{(8, 16, 32)$ $(16, 32, 64),$ $(32, 64, 128)\}$
Cat 2-2	$\{16, 32\}$	$\{1, 2\}$	$\{5, \mathbf{10}, 20\}$ $\{21, 42, 84\}$	$\{(64, 32), \mathbf{(32, 16)}, (16, 8)\}$ $\{(128, 64, 32)$ $(64, 32, 16),$ $(32, 16, 8)\}$

4 Experiments

4.1 Experiment Settings

We first describe performance variations of different splitting schemes, i.e., Category 1 vs. Category 3. In Category 1, subvectors corresponding to different levels have the same dimension, while in Category 3, subvectors corresponding to level 0 have lower dimension than that corresponding to level 1, and so on.

Figure 4 shows performance variations of different splitting schemes, based on level 1 and level 2 PHOGs. From both subplots, we clear see that with non-uniform splitting (Category 3), we achieve the same recall@100 with shorter code lengths than that based on uniform splitting (Category 1). This trend conforms to studies in many multimedia applications. Generally, slightly better performance is obtained based on PHOGs with gradients quantizing into 32 bins. We also observe that level 2 PHOGs provide slightly better performance. Therefore, in the following experiments, we mainly do performance comparison based on level 2 PHOGs with gradients quantizing into 32 bins.

4.2 Number of Quantization Intervals

Experiments in Category 2 are designed to investigate how numbers of quantization intervals for different levels of HOG influence the nearest image search. Figure 5 shows performance variations of experiments in Category 2-1 and Category 2-2. As we can see, Category 2-2 obviously provides better performance because the same search accuracy can be achieved based on codes of shorter lengths. In Category 2-2, coarse representations of HOGs (levels 0/1) are more finely quantized. This figure reveals a fact about product quantization that has never been verified before: we are able to derive a more effective descriptor if more important information (in this case, lower level HOGs) is quantized more finely.

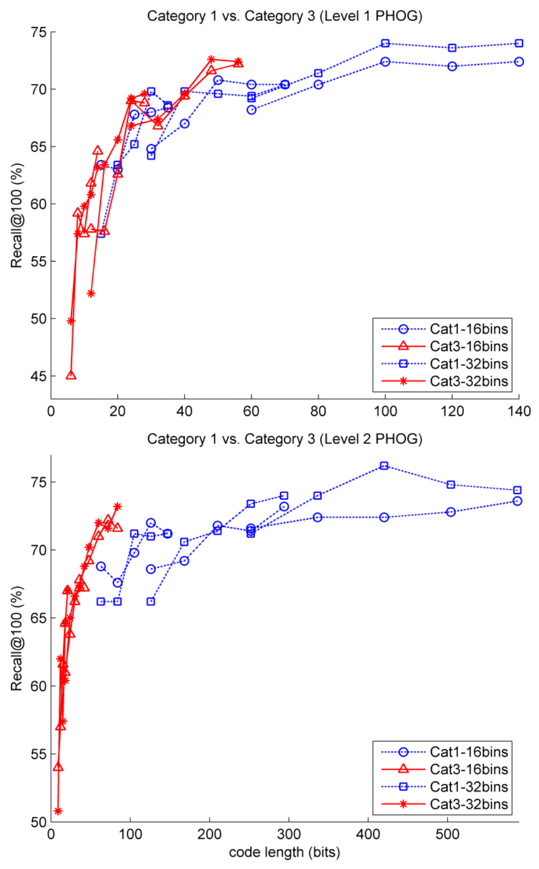

Fig. 4. Performance comparison of Category 1 vs. Category 3 experiments, based on level 1 (top) and level 2 (bottom) PHOGs.

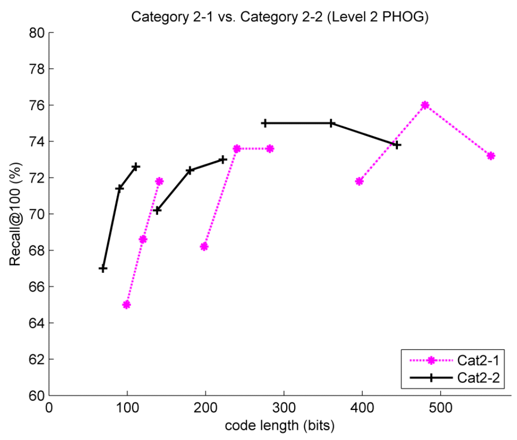

Fig. 5. Performance comparison of two experiments in Category 2, based on level 2 PHOGs.

4.3 The Influence on Retrieval Rank

All experiments described above and the ones reported in [3] show retrieval performance in terms of recall@100, but not reflect how different product quantization schemes influence the rank of retrieval results. Based on the Category 2 settings, we evaluate the rank of the first correctly retrieved image for each query. Figure 6 shows the average rank versus code lengths. The curves again show that Category 2-2 provides better retrieval results, because lower rank values mean this representation more accurately captures the characteristics of images. Combining the results in Figures 5 and 6, we know that if more important information is quantized more finely, both recall (represented by recall@100 in Figure 5) and precision (represented by rank values in Figure 6) can be improved.

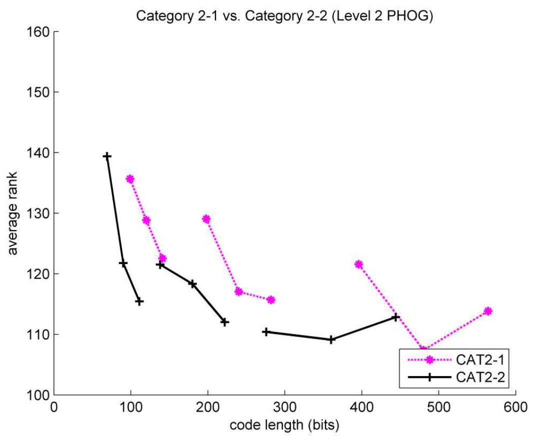

Fig. 6. Average retrieval rank of two experiments in Category 2, based on level 2 PHOGs.

5 Conclusion

In this paper we design experiments to deeply investigate how different product quantization schemes influence the performance of nearest image retrieval. Based on the PHOG descriptors, three categories of experiments are conducted for evaluating the factors of how subvectors are split and how subvectors are quantized. The experimental results reveal some information that has not been verified before: 1) appropriate nonuniform splitting yields better performance; 2) if more important information is quantized more finely, both recall and precision can be improved. In the future, product quantization can be applied to applications with large-scale nearest neighbor search, with appropriate designs of splitting and subvector quantization.

Acknowledgement. The work was partially supported by the National Science Council of Taiwan, Republic of China under research contract NSC 101-2221-E-194-055-MY2.

References

1. Friedman, J.H., Bentley, J.L., and Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. on Mathematical Software (1977), vol. 3, no. 3, pp. 209-226.
2. Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V.: Locality-sensitive hashing scheme based on p-stable distributions. Proc. of Annual Symposium on Computational Geometry (2004), pp. 253-262.
3. Jegou, H., Douze, M., and Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Analysis and Machine Intelligence (2011), vol. 33, no. 1, pp. 117-128.
4. Lowe, D.: Distinctive image features from scale invariant keypoints. International Journal of Computer Vision (2004), vol. 60, no. 2, pp. 91-110.
5. Oliva, A., and Torralba, A.: Modeling the shape of the scene: a holistic representation of the spatial envelope. International Journal of Computer Vision (2001), vol. 42, no. 3, pp. 145-175.
6. Huiskes, M.J., and Lew, M.S.: The MIR Flickr retrieval evaluation. Proc. of ACM International Conference on Multimedia Information Retrieval (2008), pp. 39-43.
7. Jegou, H., Douze, M., and Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. Proc. of European Conference on Computer Vision (2008).
8. Bosch, A., Zisserman, A., and Munoz, X.: Representing shape with a spatial pyramid kernel. Proc. of ACM International Conference on Image and Video Retrieval (2007), pp. 401-408.
9. Wang, C., Duan, L.-Y., Wang, Y., and Gao, W.: PQ-WGLOH: A bit-rate scalable local feature descriptor. Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (2012), pp. 941-944.
10. Wang, M., Hua, X.-S., Tang, J., and Hong, R.: Beyond distance measurement: constructing neighborhood similarity for video annotation. IEEE Transactions on Multimedia, vol. 11, no. 3, pp. 465-476, 2009.

11. Wang, M., Yang, K., Huan, X.-S., and Zhang, H.-J.: Towards a relevant and diverse search of social images. IEEE Transactions on Multimedia, vol. 12, no. 8, pp. 829-842, 2010.