Learning Binary Codes for High-Dimensional Data Using Bilinear Projections

Yunchao Gong
UNC Chapel Hill
yunchao@cs.unc.edu

Sanjiv Kumar
Google Research
sanjivk@google.com

Henry A. Rowley
Google Research
har@google.com

Svetlana Lazebnik
University of Illinois
slazebni@illinois.edu

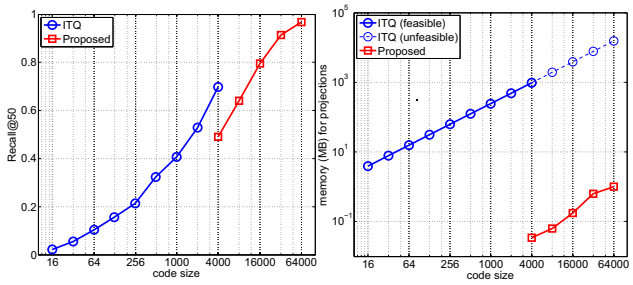
Abstract

Recent advances in visual recognition indicate that to achieve good retrieval and classification accuracy on large-scale datasets like ImageNet, extremely high-dimensional visual descriptors, e.g., Fisher Vectors, are needed. We present a novel method for converting such descriptors to compact similarity-preserving binary codes that exploits their natural matrix structure to reduce their dimensionality using compact bilinear projections instead of a single large projection matrix. This method achieves comparable retrieval and classification accuracy to the original descriptors and to the state-of-the-art Product Quantization approach while having orders of magnitude faster code generation time and smaller memory footprint.

1. Introduction

Today, image search and recognition are being performed on ever larger databases. For example, the ImageNet database [2] currently contains around 15M images and 20K categories. To achieve high retrieval and classification accuracy on such datasets, it is necessary to use extremely high-dimensional descriptors such as Fisher Vectors (FV) [23, 24, 28], Vectors of Locally Aggregated Descriptors (VLAD) [14], or Locality Constrained Linear Codes (LLC) [32]. Our goal is to convert such descriptors to binary codes in order to improve the efficiency of retrieval and classification without sacrificing accuracy.

There is a lot of recent work on learning similarity-preserving binary codes [6, 9, 10, 11, 15, 18, 19, 20, 21, 27, 30, 31, 33], but most of it is focused on relatively low-dimensional descriptors such as GIST [22], which are not sufficient for state-of-the-art applications. By contrast, we are interested in descriptors with tens or hundreds of thousands of dimensions. Perronnin et al. [24, 28] have found that to preserve discriminative power, binary codes for such descriptors must have a very large number of bits. Indeed, Figure 1(a) shows that compressing a 64K-dimensional descriptor to 100-1,000 bits (typical code sizes in most similarity-preserving coding papers) results in an unacceptable loss of retrieval accuracy compared to code sizes of 16K-64K bits. Thus, our goal is to *convert very high-dimensional real vectors to long binary strings*. To do so, we must overcome significant computational challenges.



(a) NN search recall. (b) Storage requirements.

Figure 1. Comparison of our proposed binary coding method with state-of-the-art ITQ method [6] for retrieval on 50K Flickr images (1K queries) with 64,000-dimensional VLAD descriptors. (a) Recall of 10NN from the original feature space at top 50 retrieved points. (b) Storage requirements for projection matrices needed for coding (note the logarithmic scale of the vertical axis). The dashed curve for ITQ is an estimate, since running ITQ for more than 4K bits is too expensive on our system. While ITQ has slightly higher accuracy than our method for shorter code sizes, larger code sizes are needed to get the highest absolute accuracy, and ITQ cannot scale to them due to memory limitations. By contrast, our method can be used to efficiently compute very high-dimensional codes that achieve comparable accuracy to the original descriptor.

A common step of many binary coding methods is linear projection of the data (e.g., PCA or a Gaussian random projection). When the dimensionality of both the input feature vector and the resulting binary string are sufficiently high, the storage and computation requirements for even a random projection matrix become extreme (Figure 1(b)). For example, a 64K×64K random projection matrix takes roughly 16GB of memory and the projection step takes more than one second. Methods that require the projection matrix to be learned, such as Iterative Quantization (ITQ) [6] become infeasible even more quickly since their training time scales cubically in the number of bits.

There are a few works on compressing high-dimensional descriptors such as FV and VLAD. Perronnin et al. [24]

1063-6919/13 $26.00 © 2013 IEEE
DOI 10.1109/CVPR.2013.69

have investigated a few basic methods including thresholding, Locality Sensitive Hashing (LSH) [1], and Spectral Hashing (SH) [33]. A more powerful method, Product Quantization (PQ) [13], has produced state-of-the-art results for compressing FV for large-scale image classification [28]. However, descriptors like FV and VLAD require a random rotation to balance their variance prior to PQ [14], and as explained above, rotation becomes quite expensive for high dimensions.

In this paper, we propose a bilinear method that takes advantage of the natural two-dimensional structure of descriptors such as FV, VLAD, and LLC and uses two smaller matrices to implicitly represent one big rotation or projection matrix. This method is inspired by bilinear models used for other applications [26, 29, 34]. We begin with a method based on random bilinear projections and then show how to efficiently learn the projections from data. We demonstrate the promise of our method, dubbed *bilinear projection-based binary codes (BPBC)*, through experiments on two large-scale datasets and two descriptors, VLAD and LLC. For most scenarios we consider, BPBC produces little or no degradation in performance compared to the original continuous descriptors; furthermore, it matches the accuracy of PQ codes while having much lower running time and storage requirements for code generation.

2. Bilinear Binary Codes

Most high-dimensional descriptors have a natural matrix or tensor structure. For example, a HOG descriptor is a two-dimensional grid of histograms, and this structure has been exploited for object detection [26]. A Fisher Vector can be represented as a $k \times 2l$ matrix, where k is the visual vocabulary size and l is the dimensionality of the local image features (the most common choice is SIFT with l=128). VLAD, which can be seen as a simplified version of FV, can be represented as a $k \times l$ matrix. Finally, an LLC descriptor with s spatial bins can be represented as a $k \times s$ matrix.

Let $\boldsymbol{x} \in \mathbb{R}^d$ denote our descriptor vector. Based on the structure and interpretation of the descriptor,[1] we reorganize it into a $d_1 \times d_2$ matrix with $d = d_1 d_2$:

$$\boldsymbol{x} \in \mathbb{R}^{d_1 d_2 \times 1} \quad \mapsto \quad X \in \mathbb{R}^{d_1 \times d_2}. \tag{1}$$

We assume that each vector $\boldsymbol{x} \in \mathbb{R}^d$ is zero-centered and has unit norm, as L_2 normalization is a widely used preprocessing step that usually improves performance [25].

We will first introduce a randomized method to obtain d-bit bilinear codes in Section 2.1 and then explain how to learn data-dependent codes in Section 2.2. Learning of reduced-dimension codes will be discussed in Section 2.3.

[1] We have also tried to randomly reorganize descriptors into matrices but found it to produce inferior performance.

2.1. Random Bilinear Binary Codes

To convert a descriptor $\boldsymbol{x} \in \mathbb{R}^d$ to a d-dimensional binary string, we first consider the framework of [1, 6] that applies a random rotation $R \in \mathbb{R}^{d \times d}$ to $\boldsymbol{x}$:

$$H(\boldsymbol{x}) = \operatorname{sgn}(R^T \boldsymbol{x}). \tag{2}$$

Since $\boldsymbol{x}$ can be represented as a matrix $X \in \mathbb{R}^{d_1 \times d_2}$, to make rotation more efficient, we propose a bilinear formulation using two random orthogonal matrices $R_1 \in \mathbb{R}^{d_1 \times d_1}$ and $R_2 \in \mathbb{R}^{d_2 \times d_2}$:

$$H(X) = \operatorname{vec}\Big(\operatorname{sgn}(R_1^T X R_2)\Big), \tag{3}$$

where $\operatorname{vec}(\cdot)$ denotes column-wise concatenation.

It is easy to show that applying a bilinear rotation to $X \in \mathbb{R}^{d_1 \times d_2}$ is equivalent to applying a $d_1 d_2 \times d_1 d_2$ rotation to $\operatorname{vec}(X)$. This rotation is given by $\hat{R} = R_2 \otimes R_1$, where $\otimes$ denotes the Kronecker product:

$$\operatorname{vec}(R_1^T X R_2) = (R_2^T \otimes R_1^T) \operatorname{vec}(X) = \hat{R}^T \operatorname{vec}(X)$$

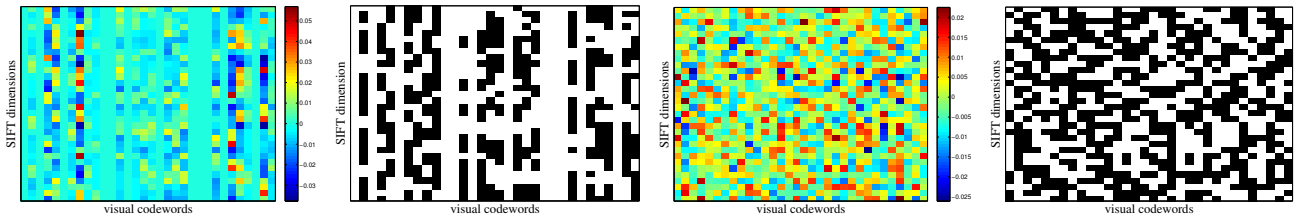
follows from the properties of the Kronecker product [16]. Another basic property of the Kronecker product is that if R_1 and R_2 are orthogonal, then $R_2 \otimes R_1$ is orthogonal as well [16]. Thus, a bilinear rotation is simply a special case of a full rotation, such that the full rotation matrix $\hat{R}$ can be reconstructed from two smaller matrices R_1 and R_2.

While the degree of freedom of our bilinear rotation is more restricted than a full rotation, the projection matrices are much smaller, and the projection speed is much faster. In terms of time complexity, performing a full rotation on $\boldsymbol{x}$ takes $O((d_1 d_2)^2)$ time, while our approach is $O(d_1^2 d_2 + d_1 d_2^2)$. In terms of space for projections, full rotation takes $O((d_1 d_2)^2)$, and our approach only takes $O(d_1^2 + d_2^2)$. For example, as will be shown in Section 3.4, for a 64K-dimensional vector, a full rotation will take roughly 16GB of RAM, while the bilinear rotations only take 1MB of RAM. The projection time for a full rotation is more than a second, vs. only 3 ms for bilinear rotations.

2.2. Learning Bilinear Binary Codes

In this section, we present a method for learning the rotations R_1 and R_2 that is inspired by two-sided Procrustes analysis [29] and builds on our earlier work [5, 6, 7].

Following [6], we want to find a rotation $\hat{R}$ such that the angle θ_i between a rotated feature vector $\hat{R}^T \boldsymbol{x}_i = \operatorname{vec}(R_1^T X_i R_2)$ and its binary encoding (geometrically, the nearest vertex of the binary hypercube), $\operatorname{sgn}(\hat{R}^T \boldsymbol{x}) = \operatorname{vec}(\operatorname{sgn}(R_1^T X_i R_2))$, is minimized. Given N training



(a) Original VLAD descriptor (b) Original binary code (c) Bilinearly rotated VLAD (d) Bilinearly rotated binary code

Figure 2. Visualization of the VLAD descriptor and resulting binary code (given by the sign function) before and after learned bilinear rotation. We only show the first 32 SIFT dimensions and visual codewords. Before the rotation, we can clearly see a block pattern, with many zero values. After the rotation, the descriptor and the binary code look more whitened.

points, we want to maximize

$$\begin{aligned}
&\sum_{i=1}^{N} \cos(\theta_i) \\
&= \sum_{i=1}^{N} \left(\frac{\operatorname{sgn}(\hat{R}^T \boldsymbol{x}_i)^T}{\sqrt{d}} (\hat{R}^T \boldsymbol{x}_i) \right) \qquad (4) \\
&= \sum_{i=1}^{N} \left(\frac{\operatorname{vec}(\operatorname{sgn}(R_1^T X_i R_2))^T}{\sqrt{d}} \operatorname{vec}(R_1^T X_i R_2) \right) \\
&= \frac{1}{\sqrt{d}} \sum_{i=1}^{N} \left(\operatorname{vec}(B_i)^T \operatorname{vec}(R_1^T X_i R_2) \right) \\
&= \frac{1}{\sqrt{d}} \sum_{i=1}^{N} \operatorname{tr}(B_i R_2^T X_i^T R_1), \qquad (5)
\end{aligned}$$

where $B_i = \operatorname{sgn}(R_1^T X_i R_2)$. Notice that (4) involves the large projection matrix $\hat{R} \in \mathbb{R}^{d \times d}$, direct optimization of which is challenging. However, after reformulation into bilinear form (5), the expression only involves the two small matrices R_1 and R_2. Letting $\mathcal{B} = \{B_1, \ldots, B_N\}$, our objective function is as follows:

$$\mathcal{Q}(\mathcal{B}, R_1, R_2) = \max_{\mathcal{B}, R_1, R_2} \sum_{i=1}^{N} \operatorname{tr}(B_i R_2^T X_i^T R_1) \qquad (6)$$

$$\text{s.t.} \;\; B_i \in \{-1, +1\}^{d_1 \times d_2}, \;\; R_1^T R_1 = I, \;\; R_2^T R_2 = I.$$

This optimization problem can be solved by block coordinate ascent by alternating between the different variables $\{B_1, \ldots, B_N\}$, R_1, and R_2. We describe the update steps for each variable below, assuming the others are fixed.

(S1) Update B_i. When R_1 and R_2 are fixed, we independently solve for each B_i by maximizing

$$\mathcal{Q}(B_i) = \operatorname{tr}(B_i R_2^T X_i^T R_1) = \textstyle\sum_{k=1}^{d_1} \sum_{l=1}^{d_2} B_i^{kl} \widetilde{V}_i^{lk},$$

where $\widetilde{V}_i^{lk}$ denote the elements of $\widetilde{V}_i = R_2^T X_i^T R_1$. $\mathcal{Q}(B_i)$ is maximized by $B_i = \operatorname{sgn}(\widetilde{V}_i^T)$.

(S2) Update R_1. Expanding (6) with R_2 and B_i fixed, we have the following:

$$\begin{aligned}
\mathcal{Q}(R_1) &= \sum_{i=1}^{N} \operatorname{tr}(B_i R_2^T X_i^T R_1) \\
&= \operatorname{tr}\Big(\sum_{i=1}^{N} (B_i R_2^T X_i^T) R_1 \Big) = \operatorname{tr}(D_1 R_1)\,,
\end{aligned}$$

where $D_1 = \sum_{i=1}^{N} (B_i R_2^T X_i^T)$. The above expression is maximized with the help of polar decomposition: $R_1 = V_1 U_1^T$, where $D_1 = U_1 S_1 V_1^T$ is the SVD of D_1.

(S3) Update R_2:

$$\begin{aligned}
\mathcal{Q}(R_2) &= \sum_{i=1}^{N} \operatorname{tr}(B_i R_2^T X_i^T R_1) \\
&= \sum_{i=1}^{N} \operatorname{tr}(R_2^T X_i^T R_1 B_i) \\
&= \operatorname{tr}\Big(R_2^T \sum_{i=1}^{N} (X_i^T R_1 B_i) \Big) = \operatorname{tr}(R_2^T D_2)\,,
\end{aligned}$$

where $D_2 = \sum_{i=1}^{N} (X_i^T R_1 B_i)$. Analogously to the update rule for R_1, the update rule for R_2 is $R_2 = U_2 V_2^T$, where $D_2 = U_2 S_2 V_2^T$ is the SVD of D_2.

We cycle between these updates for several iterations to obtain a local maximum. The convergence of the above program is guaranteed in finite number of iterations as the optimal solution of each step is exactly obtained, each step is guaranteed not to decrease the objective function value, and the objective function is bounded from above. In our implementation, we initialize R_1 and R_2 by random rotations and use three iterations. We have not found significant improvement of performance by using more iterations. The time complexity of this program is $O(N(d_1^3 + d_2^3))$ where d_1 and d_2 are typically fairly small (e.g., $d_1 = 128,\ d_2 = 500$).

Figure 2 visualizes the structure of a VLAD descriptor and the corresponding binary code before and after a learned bilinear rotation.

2.3. Learning with Dimensionality Reduction

The formulation of Section 2.2 is used to learn d-dimensional binary codes starting from d-dimensional descriptors. Now, to produce a code of size $c = c_1 \times c_2$, where $c_1 < d_1$ and $c_2 < d_2$, we need projection matrices $R_1 \in \mathbb{R}^{d_1 \times c_1}$, $R_2 \in \mathbb{R}^{d_2 \times c_2}$ such that $R_1^T R_1 = I$ and $R_2^T R_2 = I$. Each B_i is now a $c_1 \times c_2$ binary variable. Consider the cosine of the angle between a lower-dimensional projected vector $\hat{R}^T \boldsymbol{x}_i$ and its binary encoding $\operatorname{sgn}(\hat{R}^T \boldsymbol{x})$:

$$\cos(\theta_i) = \frac{\operatorname{sgn}(\hat{R}^T \boldsymbol{x}_i)^T}{\sqrt{c}} \frac{\hat{R}^T \boldsymbol{x}_i}{\|\hat{R}^T \boldsymbol{x}_i\|_2}\,,$$

where $\hat{R} \in \mathbb{R}^{d_1 d_2 \times c_1 c_2}$ and $\hat{R}^T \hat{R} = I$. This formulation differs from that of (4) in that it contains $\|\hat{R}^T \boldsymbol{x}_i\|_2$ in the denominator, which makes the optimization difficult [5]. Instead, we follow [5] to define a relaxed objective function based on the sum of linear correlations

$$\mathcal{Q}(\mathcal{B}, R_1, R_2) = \sum\nolimits_{i=1}^{N} \left(\frac{\operatorname{sgn}(\hat{R}^T \boldsymbol{x}_i)^T}{\sqrt{c}} (\hat{R}^T \boldsymbol{x}_i) \right).$$

The optimization framework for this objective is similar to that of Section 2.2. For the three alternating optimization steps, (S1) remains the same. For (S2) and (S3), we compute the SVD of D_1 and D_2 as $U_1 S_1 V_1^T$ and $U_2 S_2 V_2^T$ respectively, and set the two rotations to $R_1 = \hat{V}_1 U_1^T$ and $R_2 = \hat{U}_2 V_2^T$, where $\hat{V}_1$ is the top c_1 singular vectors of V_1 and $\hat{U}_2$ is the top c_2 singular vectors of U_2. To initialize the optimization, we generate random orthogonal directions.

2.4. Distance Computation for Binary Codes

At retrieval time, given a query descriptor, we need to compute its distance to every binary code in the database. The most popular measure of distance for binary codes is the Hamming distance. We compute it efficiently by putting bits in groups of 64 and performing an XOR and bit count (popcount). For improved accuracy, we use *asymmetric distance*, in which the database points are quantized but the query is not [3, 8, 13]. In this work, we adopt the lower-bounded asymmetric distance proposed in [3] to measure the distance between the query and binary codes. For a query $\boldsymbol{x} \in \mathbb{R}^c$ and database points $\boldsymbol{b} \in \{-1, +1\}^c$, the lower-bounded asymmetric distance is simply the L_2 distance between $\boldsymbol{x}$ and $\boldsymbol{b}$: $d_a(\boldsymbol{x}, \boldsymbol{b}) = \|\boldsymbol{x}\|_2^2 + c - 2\boldsymbol{x}^T \boldsymbol{b}$. Since $\|\boldsymbol{x}\|_2^2$ is on the query side and c is fixed, in practice, we only need to compute $\boldsymbol{x}^T \boldsymbol{b}$. We do this by putting bits in groups of 8 and constructing a 1×256 lookup table to make the dot-product computation more efficient.

3. Experiments

3.1. Datasets and Features

We test our proposed approach, bilinear projection-based binary codes (**BPBC**), on two large-scale image datasets and two feature types. The first one is the INRIA Holiday dataset with 1M Flickr distractors (**Holiday+Flickr1M**) [12]. There are 1,419 images in the Holiday dataset corresponding to 500 different scene instances, and each instance has three images on average. There is a set of 500 query images, and the remaining 919 images together with the 1M Flickr images are used as the database. We use the SIFT features of interest points provided by [14] and cluster them to 500 k-means centers. Then we represent each image by a $128 \times 500 = 64,000$ dimensional VLAD. The vectors are power-normalized (element-wise square-rooted) and L_2-normalized as in [25].

The second dataset is the **ILSVRC2010** subset of ImageNet [2], which contains 1.2M images and 1,000 classes. On this dataset, we use the publicly available SIFT features, which are densely extracted from the images at three different scales. We cluster the features into 200 centers and then aggregate them into VLAD vectors of $128 \times 200 = 25,600$ dimensions. These vectors are also power- and L_2-normalized. In addition, we compute LLC features [32] on this dataset using a 5,000-dimensional visual codebook and a three-level spatial pyramid (21 spatial bins). The resulting features have $5,000 \times 21 = 105,000$ dimensions. Unlike VLAD descriptors, which are dense and have both positive and negative values, the LLC descriptors are nonnegative and sparse. For improved results, we zero-center and L_2-normalize them.

3.2. Experimental Protocols

To learn binary codes using the methods of Sections 2.2 and 2.3, we randomly sample 20,000 training images from each dataset. We then set aside a number of query images that were not used for training and run nearest neighbor searches against all the other images in the dataset. For Holiday+Flickr1M, we sample the training images from the Flickr subset only and use the 500 predefined queries. For ILSVRC2010, we use 1,000 random queries. For each dataset, we evaluate two types of retrieval tasks:

1. Retrieval of ground-truth nearest neighbors from the original feature space. These are defined as the top ten Euclidean nearest neighbors for each query based on original descriptors. Our performance measure for this task is the recall of 10NN for different numbers of retrieved points [9, 13].
2. Retrieval of "relevant" or "semantically correct" neighbors. For Holiday+Flickr1M, these are defined as the images showing the same object instance (recall from Section 3.1 that each query has around three matches). The standard performance measure for this task on this dataset is mean average precision (mAP) [8, 12, 14]. For ILSVRC2010, we define the ground-truth "relevant" neighbors as the images sharing the same category label. In this case, there are very many matches for each query. Following [6, 18, 20], we report performance using precision at top k retrieved images (precision@k).

In addition, in Section 3.8 we perform image classification experiments on the ILSVRC2010 dataset.

3.3. Baseline Methods

Our main baseline is the state-of-the-art Product Quantization (PQ) approach [13]. PQ groups the data dimensions in batches of size s and quantizes each group with k codebook centers. In our experiments, we use $s = 8$ and

Feature dim.	LSH	PQ	RR+PQ	BPBC
128×10	0.12	2.8	2.92	0.08
128×100	9.35	26.5	35.85	0.54
128×200	29.14	47.3	76.44	0.86
128×500	186.22	122.3	308.52	3.06
128×1000	–	269.5	–	9.53

Table 1. Average time (ms) to encode a single descriptor for LSH, PQ, and BPBC. The VLAD feature dimension is $l \times k$.

Feature dim.	LSH	PQ	RR+PQ	BPBC
128×10	6.25	1.25	7.50	0.06
128×100	625	12.5	637	0.10
128×200	2500	25.0	2525	0.22
128×500	15625	62.5	15687	1.02
128×1000	62500	125	62625	3.88

Table 2. Memory (MB) needed to store the projections (or codebooks), assuming each element is a float (32 bits).

$k = 256$ following [14]. At query time, PQ uses asymmetric distance to compare an uncompressed query point to quantized database points. Namely, the distances between the code centers and corresponding dimensions of the query are first computed and stored in a lookup table. Then the distances between the query and database points are computed by table lookup and summation.

For high-dimensional features with unbalanced variance, Jégou et al. [14] recommend randomly rotating the data prior to PQ.[2] This baseline will be referred to as RR+PQ. Whenever the descriptor dimensionality in our experiments is too high for us to perform the full random rotation, we perform a bilinear rotation instead (BR+PQ).[3]

As simpler baselines, we also consider LSH based on random projection [1] and the $\alpha = 0$ binarization scheme proposed in [24], which simply takes the sign of each dimension. There exist many other methods aimed at lower-dimensional data and shorter codes, e.g., [6, 20, 30, 33], but on our data, they produce poor results for small code sizes and do not scale to larger code sizes (recall Figure 1).

3.4. Computation and Storage Requirements

First, we evaluate the scalability of our method compared to LSH, PQ, and RR+PQ for converting d-dimensional vectors to d-bit binary strings. For this test, we use the VLAD features. All running times are evaluated on a machine with 24GB RAM and a 6-core 2.6GHz CPU. Table 1 reports code generation time for different VLAD sizes and Table 2 reports the memory requirements for storing the projection matrix. It is clear that our bilinear formulation is orders of magnitude more efficient than LSH and both versions of PQ both in terms of projection time and storage.

[2]The original PQ paper [13] states that a random rotation is not needed prior to PQ. However, the conclusions of [13] are mainly drawn from low-dimensional data like SIFT and GIST, whose variance already tends to be roughly balanced.

[3]As another efficient alternative to random rotation prior to PQ, we have also considered a random permutation, but found that on our data it has no effect.

Code size	SD (binary)	ASD (binary)	ASD (PQ)	Eucl. (est.)
122×100	0.33	4.48	4.59	∼120
128×200	0.60	11.29	11.28	∼241

Table 3. Retrieval time per query (seconds) on the ILSVRC2010 dataset with 1.2M images and two different code sizes. This is the time to perform exhaustive computation of distances from a single query to all the 1.2M images in the database. "SD" denotes symmetric (Hamming) and "ASD" asymmetric distance. For Euclidean distance, all the original descriptors do not fit in RAM, so the timing is extrapolated from a smaller subset. The actual timing is likely to be higher due to file I/O.

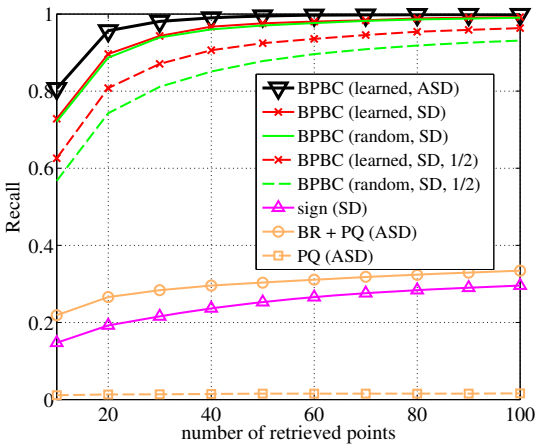
Table 3 compares the speed of Hamming distance (SD) vs. asymmetric distance (ASD) computation for two code sizes on ILSVRC2010. As one would expect, computing Hamming distance using XOR and popcount is extremely fast. The speed of ASD for PQ vs. our method is comparable, and much slower than SD. However, note that for binary codes with ASD, one can first use SD to find a short list and then do re-ranking with ASD, which will be much faster than exhaustive ASD computation.

3.5. Retrieval on Holiday+Flickr1M

Next, we evaluate retrieval performance on the Holiday+Flickr1M dataset using VLAD features with $500 \times 128 = 64,000$ dimensions. As explained in Section 3.1, we use the predefined 500 Holiday queries. For 64,000-dimensional features, evaluating RR+PQ is prohibitively expensive, so instead we try to combine the bilinear rotation with PQ (denoted as BR+PQ). For BPBC with dimensionality reduction (Section 2.3), we use bilinear projections $R_1 \in \mathbb{R}^{500\times400}, R_2 \in \mathbb{R}^{128\times80}$. This reduces the dimensionality in half.

Figure 3(a) shows the recall of 10NN from the original feature space for different numbers of retrieved images. PQ without rotation fails on this dataset; BR+PQ is slightly better, but is still disappointing. This is due to many Flickr images (e.g., sky and sea images) having few interest points, resulting in VLAD with entries that are mostly zero. Bilinear rotation appears to be insufficient to fully balance the variance in this case, and performing the full random rotation is too expensive.[4] On the other hand, all versions of BPBC show good performance. For a code size of 32,000 (dimensionality reduction by a factor of two), learned ro-

[4]Jégou et al. [14] report relatively strong performance for RR+PQ on Holiday+Flickr1M, but they use lower-dimensional VLAD ($d = 2,048$ and $d = 8,192$) followed by PCA compression to 32-128 dimensions. These parameter settings are motivated by the goal of [14] to produce extremely compact image codes. By contrast, our goal is to produce higher-dimensional codes that do not lose discriminative power. Indeed, by raising the dimensionality of the code, we are able to improve the retrieval accuracy in absolute terms: the mAP for our BPBC setup (Figure 3(b)) is about 0.4 vs. about 0.2 for the PQ setup of [14].



(a) Recall of 10NN.

Method	Rate	mAP
VLAD (float)	1	39.0
Sign (SD)	32	25.6
PQ (with bilinear rotation, ASD)	32	24.0
PQ (w/o rotation, ASD)	32	2.3
BPBC (learned, ASD)	32	40.1
BPBC (learned, SD)	32	40.1
BPBC (random, SD)	32	40.3
BPBC (learned, SD, 1/2)	64	38.8
BPBC (random, SD, 1/2)	64	38.6

(b) Instance-level retrieval results.

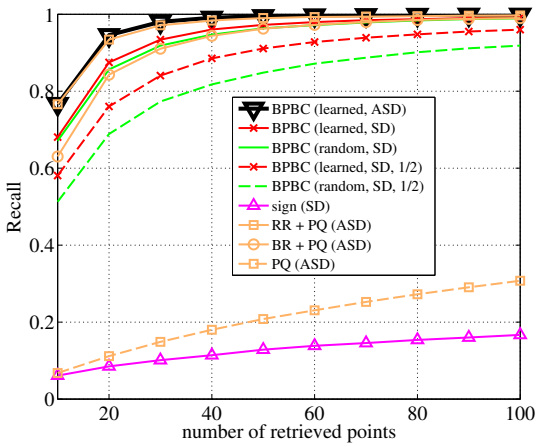
Figure 3. Results on the Holiday+Flickr1M dataset with 64,000-dimensional VLAD. (a) Recall of ground truth 10NN from the original feature space. (b) Instance-level retrieval results (mAP). "SD" (resp. "ASD") denote symmetric (resp. asymmetric) distance. "Sign" refers to binarization by thresholding each dimension at zero. "Random" refers to the method of Section 2.1 and "learned" refers to the methods of Sections 2.2 and 2.3. "1/2" refers to reducing the code dimensionality in half with the method of Section 2.3. "Rate" is the factor by which storage is reduced compared to the original descriptors.

tation works much better than random, while for the full-dimensional BPBC, learned and random rotations perform similarly. Asymmetric distance (ASD) further improves the recall over symmetric distance (SD).

Next, Figure 3(b) reports instance-level image retrieval accuracy measured by mean average precision (mAP), or the area under the recall-precision curve. Both learned and random BPBC produce comparable results to the original descriptor. PQ without rotation works poorly, and BR+PQ is more reasonable, but still worse than our method. Note that for this task, unlike for the retrieval of 10NN, ASD does not give any further improvement over SD. This is good news, since SD computation is much faster (recall Table 3).

3.6. Retrieval on ILSVRC2010 with VLAD

As discussed in Section 3.1, our VLAD descriptors for the ILSVRC2010 dataset have dimensionality 25,600. Random rotation for this descriptor size is still feasible, so we



(a) Recall of 10NN.

Method	Rate	P@10	P@50
VLAD (float)	1	17.73	7.29
Sign (SD)	32	13.15	3.87
PQ (with full rotation, ASD)	32	18.06	7.41
PQ (with bilinear rotation, ASD)	32	16.98	6.96
PQ (w/o rotation, ASD)	32	11.32	3.14
BPBC (learned, ASD)	32	18.01	7.49
BPBC (learned, SD)	32	18.07	7.42
BPBC (random, SD)	32	18.17	7.60
BPBC (learned, SD, 1/2)	64	17.80	7.25
BPBC (random, SD, 1/2)	64	16.85	6.78

(b) Categorical image retrieval results.

Figure 4. Results on ILSVRC2010 with 25,600-dimensional VLAD. (a) Recall of ground-truth 10NN from the original feature space. (b) Semantic precision at 10 and 50 retrieved images. See caption of Figure 3 for notation.

are able to evaluate RR+PQ. We randomly sample 1,000 query images and use the rest as the database. For BPBC with dimensionality reduction, we construct bilinear projections $R_1 \in \mathbb{R}^{200\times160}$, $R_2 \in \mathbb{R}^{128\times80}$, which reduces the dimensionality in half. Figure 4(a) compares the recall of 10NN from the original feature space with increasing number of retrieved points. The basic PQ method on this dataset works much better than on Holiday+Flickr1M (in particular, unlike in Figure 3, it is now better than simple thresholding). This is because the images in ILSVRC2010 are textured and contain prominent objects, which leads to VLAD with reasonably balanced variance. Furthermore, RR+PQ is feasible for the VLAD dimensionality we use on ILSVRC2010. We can see from Figure 4(a) that the improvement from PQ to RR+PQ is remarkable, while BR+PQ is somewhat weaker than RR+PQ. Overall, RR+PQ is the strongest of all the baseline methods, and full-dimensional BPBC with asymmetric distance is able to match its performance while having a lower memory footprint and faster code generation time (recall Tables 1 and 2). The relative performance of the other BPBC variants is the same as in Figure 3(a).

Next, we evaluate the semantic retrieval accuracy on this dataset. Figure 4(b) shows the average precision for top k

images retrieved. We can observe that RR+PQ and most versions of BPBC have comparable precision to the original uncompressed features. As in Section 3.5, using ASD as opposed to SD does not give any gains in semantic precision for our method. Thus, our method has an important advantage over PQ at retrieval time, since unlike PQ, it can be used with the faster Hamming distance.

3.7. Retrieval on ILSVRC2010 with LLC

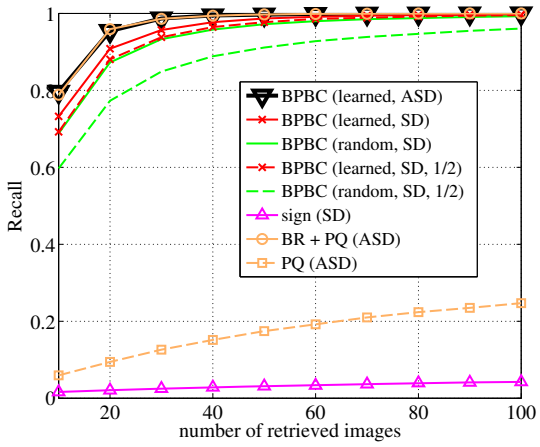
To demonstrate that the BPBC method is applicable to other high-dimensional descriptors besides VLAD, we also report retrieval results on the ILSVRC2010 dataset with LLC features. As discussed in Section 3.1, these features have the highest dimensionality yet in all our experiments: $5,000 \times 21 = 105,000$. To reduce dimensionality by a factor of two, we use bilinear projections $R_1 \in \mathbb{R}^{5000\times2500}, R_2 \in \mathbb{R}^{21\times21}$ (note that the dimensionality of the second side, representing the number of spatial bins, is already low at 21, and we have found that trying to reduce it further can lead to unstable behavior for our learning algorithm).

Figure 5(a) reports the recall for 10NN. Most of the trends are similar to those of Section 3.6. Full-dimensional BPBC with ASD once again has the best performance, together with BR+PQ (evaluating RR+PQ is once again infeasible). Learned rotation works significantly better than the random one. Next, Table 5(b) reports the semantic precision analogously to Table 4(b). As in Table 4(b), PQ with rotation and different versions of BPBC work similarly. By comparing Table 4(b) and 5(b), we can see that LLC features have higher absolute precision than VLAD, which confirms that extremely high-dimensional features and codes, far from overfitting, are necessary to obtain better performance on very large-scale many-category datasets.

3.8. Image Classification

Finally, we demonstrate the effectiveness of our proposed codes for SVM classification on ILSVRC2010. We adopt the setting of Sánchez and Perronnin [28] where the classifier is trained on compressed or encoded descriptors but tested on the original ones. For PQ-compressed data, we perform decoding in order to train the SVM (decoding consists of looking up and concatenating the codewords corresponding to each subset of dimensions) and test on original descriptors (rotated if necessary). Note that unlike [28], we use batch training, and decoding all the training descriptors ahead of time does not actually save us on storage. Instead, our goal is simply to establish a baseline classification accuracy. For BPBC-compressed data, we train directly on the binarized vectors $\mathrm{sgn}(\hat{R}^T \mathrm{vec}(X))$ but test on un-binarized vectors $\hat{R}^T \mathrm{vec}(X)$.

Our classifier is LIBLINEAR SVM [4] and the feature is 25,600-dimensional VLAD. To limit the running time for



(a) Recall of 10NN.

Method	Rate	P@10	P@50
VLAD (float)	1	21.50	10.06
Sign (SD)	32	10.99	2.58
PQ (with bilinear rotation, ASD)	32	21.41	10.11
PQ (w/o rotation, ASD)	32	12.67	4.20
BPBC (learned, ASD)	32	21.78	10.11
BPBC (learned, SD)	32	21.54	10.11
BPBC (random, SD)	32	21.73	10.35
BPBC (learned, SD, 1/2)	64	21.22	9.96
BPBC (random, SD, 1/2)	64	21.27	9.88

(b) Categorical image retrieval results.

Figure 5. Retrieval results on ILSVRC dataset with 105,000-dimensional LLC features. (a) Recall for 10NN from the original feature space. (b) Categorical retrieval precision at 10 and 50 retrieved images. See caption of Figure 3 for notation.

SVM training and parameter tuning, and to make sure we can hold all the training data in memory, we randomly sample 100 classes from ILSVRC2010. We use five random splits into 50% for training, 25% for validation, and 25% for testing. To generate negative data, we sample 200 points per class, which does not sacrifice accuracy too much. For each method, we validate the SVM hyperparameter C on a grid of $[2 \times 10^{-5}, 2 \times 10^{2}]$ with order-of-magnitude increments.

Classification results are reported in Table 4. The full-dimensional BPBC incurs very little loss of accuracy over the original features, and the learned and random rotations work comparably. When dimension is reduced in half, the classification accuracy drops, with the random method degrading more than the learned one. RR+PQ outperforms the best version of BPBC by about 0.3%, but it is not clear whether the difference is statistically significant. Interestingly, PQ without RR still produces reasonable classification results, while its performance on retrieval tasks is severely degraded. Evidently, the SVM training can compensate for the quantization error, whereas nearest-neighbor search without a supervised learning stage cannot. Similar reasoning applies to binarization by taking the sign, whose performance is also fairly good for this task.

Method	Rate	Classification accuracy
VLAD (float)	1	44.87 ± 0.30
Sign	32	41.10 ± 0.34
PQ	32	44.05 ± 0.33
RR+PQ	32	44.64 ± 0.13
BPBC (learned)	32	44.34 ± 0.21
BPBC (random)	32	44.27 ± 0.19
BPBC (learned, 1/2)	64	43.06 ± 0.20
BPBC (random, 1/2)	64	41.28 ± 0.20

Table 4. Image classification results on 100 classes randomly sampled classes from the ILSVRC2010 dataset. The set of classes is fixed but results are averaged over five different training/validation/test splits. The visual feature is 25,600-dimensional VLAD. "Rate" is the factor by which storage is reduced.

An interesting question is as follows: instead of starting with d-dimensional data and reducing the dimensionality to $d/2$ through binary coding, can we obtain the same accuracy if we start with $d/2$-dimensional features and maintain this dimensionality in the coding step? To answer this question, we have performed classification on 12,800-dimensional VLAD for the same 100 classes and obtained an average accuracy of 43.08%. This is comparable to the accuracy of our 12,800-dimensional binary descriptor learned from the 25,600-dimensional VLAD. On the other hand, classifying 12,800-dimensional codes computed from 12,800-dimensional VLAD gives an accuracy of 41.63%. Thus, starting with the highest possible dimensionality of the original features appears to be important for learning binary codes with the most discriminative power.

4. Discussion and Future Work

This paper has presented a novel bilinear rotation formulation for learning binary codes for high dimensional feature vectors that exploits the natural two-dimensional structure of many existing descriptors. Our approach matches the accuracy of Product Quantization for retrieval and classification tasks while being much more efficient in terms of memory and computation. As recent progress on recognition shows that using descriptors with *millions* of dimensions can lead to even better performance [17, 28], it will be interesting to apply our approach on such data.

Acknowledgments. We thank Ruiqi Guo for helpful discussions. Gong and Lazebnik were supported by NSF grant IIS 1228082, DARPA Computer Science Study Group (D12AP00305), Microsoft Research Faculty Fellowship, and Xerox.

References

[1] M. S. Charikar. Similarity estimation techniques from rounding algorithms. *STOC*, 2002.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. *CVPR*, 2009.

[3] W. Dong, M. Charikar, and K. Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. *SIGIR*, 2008.

[4] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *JMLR*, 2008.

[5] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik. Angular quantization based binary codes for fast similarity search. *NIPS*, 2012.

[6] Y. Gong and S. Lazebnik. Iterative quantization: A Procrustean approach to learning binary codes. *In: CVPR*, 2011.

[7] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A Procrustean approach to learning binary codes for large-scale image retrieval. *PAMI*, 2012.

[8] A. Gordo and F. Perronnin. Asymmetric distances for binary embeddings. *CVPR*, 2011.

[9] J. He, R. Radhakrishnan, S.-F. Chang, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. *CVPR*, 2011.

[10] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. *CVPR*, 2012.

[11] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. *CVPR*, 2008.

[12] H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large-scale image search. *ECCV*, 2008.

[13] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 2011.

[14] H. Jégou, M. Douze, C. Schmid, and P. Perez. Aggregating local descriptors into a compact image representation. *CVPR*, 2010.

[15] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[16] A. J. Laub. *Matrix Analysis for Scientists and Engineers*. SIAM.

[17] Y. Lin, L. Cao, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, and T. Huang. Large-scale image classification: Fast feature extraction and SVM training. In *CVPR*, 2011.

[18] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. *CVPR*, 2012.

[19] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang. Compact hyperplane hashing with bilinear functions. *ICML*.

[20] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. *ICML*, 2011.

[21] M. Norouzi, R. Salakhutdinov, and D. Fleet. Hamming distance metric learning. In *NIPS*, 2012.

[22] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 2001.

[23] F. Perronnin and C. R. Dance. Fisher kernels on visual vocabularies for image categorization. *CVPR*, 2007.

[24] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed Fisher vectors. In *CVPR*, 2010.

[25] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. *ECCV*, 2010.

[26] H. Pirsiavash, D. Ramanan, and C. Fowlkes. Bilinear classifiers for visual recognition. *NIPS*, 2009.

[27] M. Raginsky and S. Lazebnik. Locality sensitive binary codes from sift-invariant kernels. *NIPS*, 2009.

[28] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR*, 2011.

[29] P. Schönemann. On two-sided orthogonal Procrustes problems. *Psychometrika*, 1968.

[30] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. *CVPR*, 2008.

[31] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. *CVPR*, 2010.

[32] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. *CVPR*, 2010.

[33] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *NIPS*, 2008.

[34] J. Ye, R. Janardan, and Q. Li. Two-dimensional linear discriminant analysis. *NIPS*, 2004.