Factorized Binary Codes for Large-Scale Nearest Neighbor Search

Frederick Tung
ftung@cs.ubc.ca
James J. Little
little@cs.ubc.ca

Department of Computer Science
University of British Columbia
Vancouver, Canada

Abstract

Hashing algorithms for fast large-scale nearest neighbor search transform data points into compact binary codes by applying a set of learned or randomly generated hash functions. Retrieval accuracy generally increases with the number of hash functions, but increasing the number of hash functions also increases the storage requirements of the resulting binary codes. We present a novel factorized binary codes approach that uses an approximate matrix factorization of the binary codes to increase the number of hash functions while maintaining the original storage requirements. The proposed approach does not assume a particular algorithm for generating the hash functions, and requires only that we can discover and take advantage of correlations among the hash functions. Experiments on publicly available datasets suggest that factorized binary codes work particularly well for locality-sensitive hashing algorithms.

1 Introduction

Nearest neighbor search is a ubiquitous problem in computer vision, and forms an essential building block in algorithms ranging from semantic segmentation [15, 26, 29, 32], to 3D reconstruction [16, 25], to object recognition [27, 28], image inpainting [8], image captioning [21], and others. Given a previously unseen query point $\mathbf{q} \in \mathbf{R}^d$, we seek its closest matches in a database $\mathbf{X} \in \mathbf{R}^{n \times d}$, where the number of database points n may be large. While the problem formulation is simple, a wide variety of techniques have been developed to efficiently find nearest neighbors, with different tradeoffs in speed, memory consumption, accuracy, and training requirements.

One class of techniques for nearest neighbor search is *hashing algorithms* for constructing compact binary codes. Hashing algorithms transform the original data points into compact bit string signatures that require significantly less storage space and can be compared quickly using bit operations. In particular, the Hamming distance between two bit strings can be computed quickly by taking an XOR and counting the ones. This operation is supported directly in modern hardware, and can be accelerated even further using algorithms for efficient Hamming distance computation [20].

We can think of the bits in a binary code as the decisions of a set of hash functions or hyperplanes, possibly in some kernelized space. These hyperplanes are learned or generated

Pages 34.1-34.12
DOI: https://dx.doi.org/10.5244/C.30.34

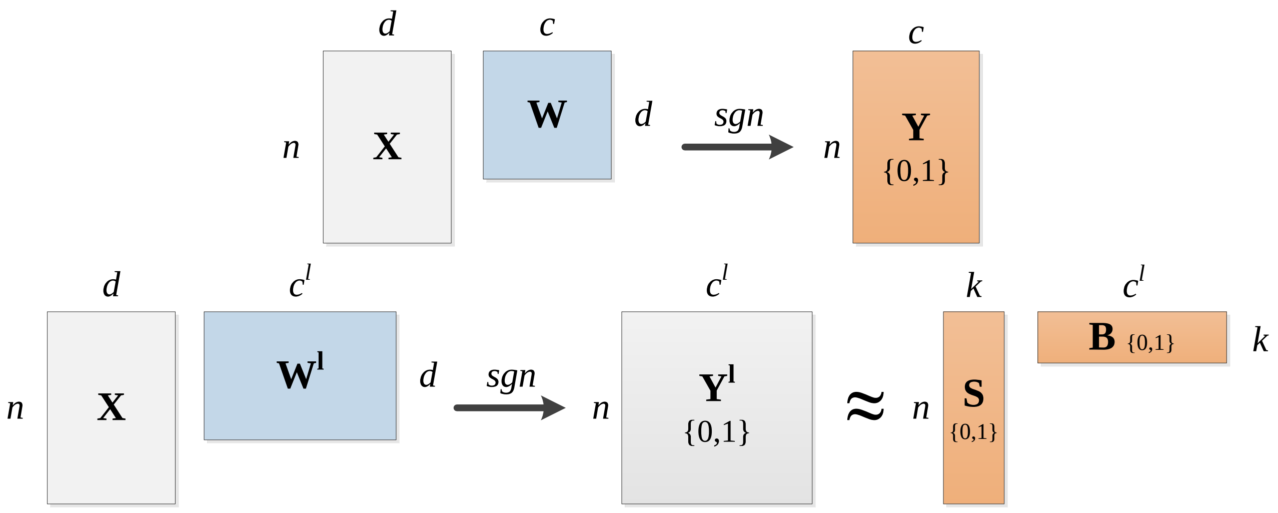

Figure 1: An overview of the factorized binary codes approach. *Top:* Hashing algorithms for learning compact binary codes transform the original data points $\mathbf{X} \in \mathbf{R}^{n \times d}$ into binary codes $\mathbf{Y} \in \{0,1\}^{n \times c}$ by sign-thresholding $\mathbf{XW}$, where $\mathbf{W} \in \mathbf{R}^{d \times c}$ is a set of hash functions or hyperplanes. In general, performance improves with the number of hash functions c. However, increasing c also increases the length of the binary codes, leading to higher storage requirements. *Bottom:* The proposed approach increases the number of hash functions to c^l without increasing storage requirements by approximating the resulting long binary codes $\mathbf{Y}^l$ as the Boolean product of two binary matrices $\mathbf{S}$ and $\mathbf{B}$. The total number of bits in $\mathbf{S}$ and $\mathbf{B}$ is restricted to the bit budget of the original $\mathbf{Y}$ (i.e. the areas highlighted in orange are the same).

by the hashing algorithm. When computing the binary code $\mathbf{y}$ for a data point $\mathbf{x} \in \mathbf{R}^d$, the ith bit of $\mathbf{y}$ is determined by the side of the ith hyperplane $\mathbf{w}_i$ on which $\mathbf{x}$ lies:

$$\mathbf{y}_i = \mathrm{sgn}(\mathbf{w}_i \cdot \mathbf{x}) = \mathrm{sgn}(\mathbf{w}_i^T \mathbf{x}) \tag{1}$$

where $\mathrm{sgn}(\cdot)$ returns 0 if its argument is negative and 1 otherwise, and $\mathbf{w}_i \in \mathbf{R}^d$. Stacking the hash functions and data points in matrix form, we have

$$\mathbf{Y} = \mathrm{sgn}(\mathbf{XW}) \tag{2}$$

where $\mathbf{X} \in \mathbf{R}^{n \times d}$, $\mathbf{W} \in \mathbf{R}^{d \times c}$, $\mathbf{Y} \in \{0,1\}^{n \times c}$, and c is the number of hash functions, or the number of bits in the generated binary code.

Typically, nearest neighbor search performance improves as the number of hash functions increases, i.e. as c increases. However, as the number of hash functions increases, the matrix $\mathbf{Y}$ of binary codes also increases in size, leading to higher storage requirements. If we wish to improve retrieval performance by doubling the number of hash functions, we have to store binary codes that are twice the length.

What if we could double (or triple, quadruple, etc.) the number of hash functions *without* increasing the storage requirements of the binary codes? Then we would obtain improved retrieval performance without incurring additional storage costs. Of course, we cannot expect to increase the number of hash functions for free. Some form of approximation is required.

Figure 1 illustrates the factorized binary codes approach. Given $\mathbf{X}$, $\mathbf{W}$, and $\mathbf{Y}$ as defined in Eq. (2), define a 'long' code length $c^l > c$, and form the matrix $\mathbf{W^l} \in \mathbf{R}^{d \times c^l}$, which appends $(c^l - c)$ new hash functions to the c existing hash functions in $\mathbf{W}$. The new hash functions are generated using the same procedure as the existing hash functions, according to the underlying hashing algorithm. The resulting binary code matrix $\mathbf{Y}^l$ is $n \times c^l$ in size. We approximate $\mathbf{Y}^l$ using the Boolean product of two binary matrices $\mathbf{S}$ and $\mathbf{B}$, such that $\mathbf{S}$ and $\mathbf{B}$ together contain the same number of bits as the original $\mathbf{Y}$.

The proposed approach is general and does not assume a particular hashing algorithm for generating the hash functions. Factorization assumes we can discover and take advantage of correlations among the generated hash functions. Consequently, our method cannot be applied on top of orthogonal hashing algorithms such as PCA hashing and iterative quantization [7], since these generate uncorrelated hash functions. On the other hand, we observe a significant boost in retrieval performance for several locality-sensitive hashing algorithms, which generate random hash functions. Locality-sensitive hashing algorithms require no training, can be kernelized, and have been successfully applied in many large-scale computer vision problems, including image retrieval, feature matching, and object classification [4, 11, 14].

After describing related work in nearest neighbor search (Section 2), we will explain the factorized binary codes approach in detail (Section 3), and present experiments on two publicly available datasets (Section 4).

2 Related work

Hashing algorithms for nearest neighbor search represent different ways to generate the hash functions or hyperplanes $\mathbf{W}$ in Eq. (2). These algorithms differ in their training requirements, speed, and retrieval accuracy.

Locality-sensitive hashing (LSH) [3, 6] is a general technique for fast nearest neighbor search that is data independent. In LSH, the probability of two data points being hashed to the same value is proportional to some measure of similarity between the data points. For example, suppose the hash functions (hyperplanes) in $\mathbf{W}$ are randomly drawn from a d-dimensional, zero-mean Gaussian distribution [3]. Then for any such hyperplane $\mathbf{w}$,

$$Pr[\text{sgn}(\mathbf{w}^T\mathbf{x}_i) = \text{sgn}(\mathbf{w}^T\mathbf{x}_j)] = 1 - \frac{\theta(\mathbf{x}_i, \mathbf{x}_j)}{\pi} \tag{3}$$

where $\theta(\mathbf{x}_i, \mathbf{x}_j)$ is the angle between $\mathbf{x}_i$ and $\mathbf{x}_j$.

Raginsky and Lazebnik [22] proposed a locality-sensitive hashing method that preserves the similarity induced by a shift-invariant kernel (SKLSH). SKLSH does not make any assumptions about the distribution of the data points and has theoretical convergence guarantees as the number of hash functions increases. In particular, the normalized Hamming distance between two binary codes converges to a monotonic function of the kernel value. SKLSH is realized by mapping data points through random Fourier features (RFF) [23] followed by random threshold quantization.

Kulis and Grauman [14] formulated a locality-sensitive hashing method that operates on arbitrary kernels (KLSH). Each hash function is a random hyperplane in the implicit high-dimensional kernel space that can be computed by a weighted sum of kernel evaluations over a sampled subset of database points. The underlying feature space does not need to be known.

Recently, Jiang et al. [11] presented a theoretical analysis of KLSH, re-interpreting the algorithm as LSH after PCA projection in a high-dimensional (possibly infinite-dimensional) kernel space, and giving the first retrieval performance bounds for KLSH. Interestingly, the performance bounds are shown to be determined by the dimensionality of the kernel PCA subspace and the number of sampled database points, and not the number of hash functions. In standard KLSH, the dimensionality of the kernel PCA subspace equals the number of sampled database points minus one. Jiang et al.'s analysis leads naturally to a low-rank extension of KLSH: improved results may be attainable by projecting into a smaller-dimensional subspace than the number of sampled database points minus one.

Though locality-sensitive hashing algorithms have asymptotic convergence properties, a large number of hash functions is typically required to obtain good performance in practice, leading to high storage costs. As a result, many data-driven hashing methods have been developed, which rely on training or optimization to find a small set of hyperplanes that obtains good performance with more compact binary codes [7, 13, 18, 30]. Besides hashing, another important class of techniques for nearest neighbor search is vector quantization algorithms, which encode data points using learned codebooks that can be combined in product [5, 9, 10, 19] or additive [1, 2] forms. Vector quantization algorithms tend to achieve high retrieval accuracy but are slower than hashing algorithms.

3 Factorized binary codes

Hashing algorithms compute compact binary codes $\mathbf{Y} \in \{0,1\}^{n\times c}$ by transforming database points $\mathbf{X} \in \mathbf{R}^{n\times d}$ using a learned or generated set of hash functions (hyperplanes) $\mathbf{W} \in \mathbf{R}^{d\times c}$ via Eq. (2), possibly in a kernelized space. In general, increasing the number of hash functions c improves the retrieval accuracy. However, increasing c also increases the length of the binary codes, leading to higher storage requirements. For example, doubling the number of hash functions doubles the size of the binary code matrix $\mathbf{Y}$. We would like to use more hash functions to improve retrieval accuracy but must balance this with the increased storage requirements of $\mathbf{Y}$.

The factorized binary codes approach increases the number of hash functions while maintaining the storage requirements of the original binary codes $\mathbf{Y}$. An overview is shown in Fig. 1. Define $c^l > c$ and augment the set of hash functions $\mathbf{W}$ with additional hash functions to obtain $\mathbf{W^l} \in \mathbf{R}^{d\times c^l}$. The additional hash functions are generated using the same hashing algorithm as the original hash functions. We do not assume a particular algorithm for generating hash functions, and will demonstrate several possible algorithms in the experiments. The augmented matrix $\mathbf{W^l}$ produces 'long' binary codes $\mathbf{Y}^l \in \{0,1\}^{n\times c^l}$:

$$\mathbf{Y}^l = \operatorname{sgn}(\mathbf{X}\mathbf{W}^l) \tag{4}$$

Next, we approximate $\mathbf{Y}^l$ as the Boolean product of two factor matrices $\mathbf{S}$ and $\mathbf{B}$, both of which are also binary:

$$\mathbf{Y}^l \approx \mathbf{S} \circ \mathbf{B} \tag{5}$$

where $\mathbf{S} \in \{0,1\}^{n\times k}$, $\mathbf{B} \in \{0,1\}^{k\times c^l}$, and $\circ$ denotes the Boolean product. The Boolean product of two binary matrices is the same as their regular matrix multiplication but with $1+1=1$. The storage requirement of the original binary codes $\mathbf{Y}$ is nc bits. We restrict the total number of bits in $\mathbf{S}$ and $\mathbf{B}$ to also be nc bits. This is achieved by setting $k = \lfloor \frac{nc}{n+c^l} \rfloor$, as

$$nk + kc^l \leq nc \Rightarrow k \leq \frac{nc}{n+c^l}, \tag{6}$$

provided c^l is not so extremely large as to make $k = 0$. Hence, **S** and **B** require the same number of bits as the original **Y** and there is no additional storage overhead. To achieve the factorization in Eq. (5), we seek the factors **S** and **B** that minimize the reconstruction error

$$|\mathbf{Y}^l - \mathbf{S} \circ \mathbf{B}| = \sum_{i=1}^{n} \sum_{j=1}^{c^l} |\mathbf{Y}^l_{ij} - (\mathbf{S} \circ \mathbf{B})_{ij}| \tag{7}$$

This problem is NP-hard and cannot be approximated within any factor in polynomial time unless P=NP [17]. However, we can obtain an approximate solution using a greedy association technique [17] from the data mining community (other binary matrix factorization techniques are also possible, e.g. [33]). Intuitively, we can think of the matrix $\mathbf{B} \in \{0,1\}^{k \times c^l}$ as a basis vector matrix encoding a set of k binary basis vectors, each of dimensionality c^l. A basis vector in **B** represents a set of correlated attributes, or hash functions in our case. For example, in document retrieval, a basis vector may encode a set of words that define a topic. Finding the basis vector matrix $\mathbf{B} \in \{0,1\}^{k \times c^l}$ and usage matrix $\mathbf{S} \in \{0,1\}^{n \times k}$ to minimize Eq. 7 then becomes a problem of discovering or mining the correlations present in the 'long' codes $\mathbf{Y}^l$. The association technique in [17] forms an association matrix using standard data mining and then draws basis vectors from the association matrix in a greedy manner to optimize a cover measure. The overall time complexity of the association technique is $O(kn(c^l)^2)$. The technique requires a single parameter τ, which we set by validation on a held-out set.

Processing novel queries. Now we have binary factor matrices **S** and **B**, with which we can approximately reconstruct the 'long' binary codes $\mathbf{Y}^l$ computed using the augmented set of hash functions. Given a previously unseen query $\mathbf{q} \in \mathbf{R}^d$, the 'long' binary code $\mathbf{y_q} \in \{0,1\}^{c^l}$ is computed using the augmented set of hash functions

$$\mathbf{y}_q = \mathrm{sgn}(\mathbf{W}^{lT}\mathbf{q}) \tag{8}$$

and matched with the approximate binary codes $\tilde{\mathbf{Y}}^l$ as reconstructed using the factors **S** and **B**:

$$\tilde{\mathbf{Y}}^l = \mathbf{S} \circ \mathbf{B} \approx \mathbf{Y}^l \tag{9}$$

In summary, the factorized binary codes approach increases the number of hash functions from c to c^l and approximates the resulting 'long' binary codes $\mathbf{Y}^l \in \{0,1\}^{n \times c^l}$ as the Boolean product of two binary factor matrices $\mathbf{S} \in \{0,1\}^{n \times k}$ and $\mathbf{B} \in \{0,1\}^{k \times c^l}$, such that the factor matrices require no more storage than the original binary codes $\mathbf{Y} \in \{0,1\}^{n \times c}$. The factor matrices that approximately minimize Eq. 7 are obtained by discovering correlations among the hash functions using a greedy association technique [17]. We observe the proposed approach to work particularly well with locality-sensitive hashing algorithms [3, 11, 14, 22], which generate random hash functions.

4 Experimental results

We performed experiments on two publicly available datasets: CIFAR-10 [12] and LM+SUN [26]. The CIFAR-10 dataset [12] is a 60,000-image, 10-class subset of the Tiny Images dataset [28] of 32×32 resolution images. The LM+SUN dataset [26] is a standard semantic segmentation dataset consisting of 45,676 images from the LabelMe [24] and SUN [31]

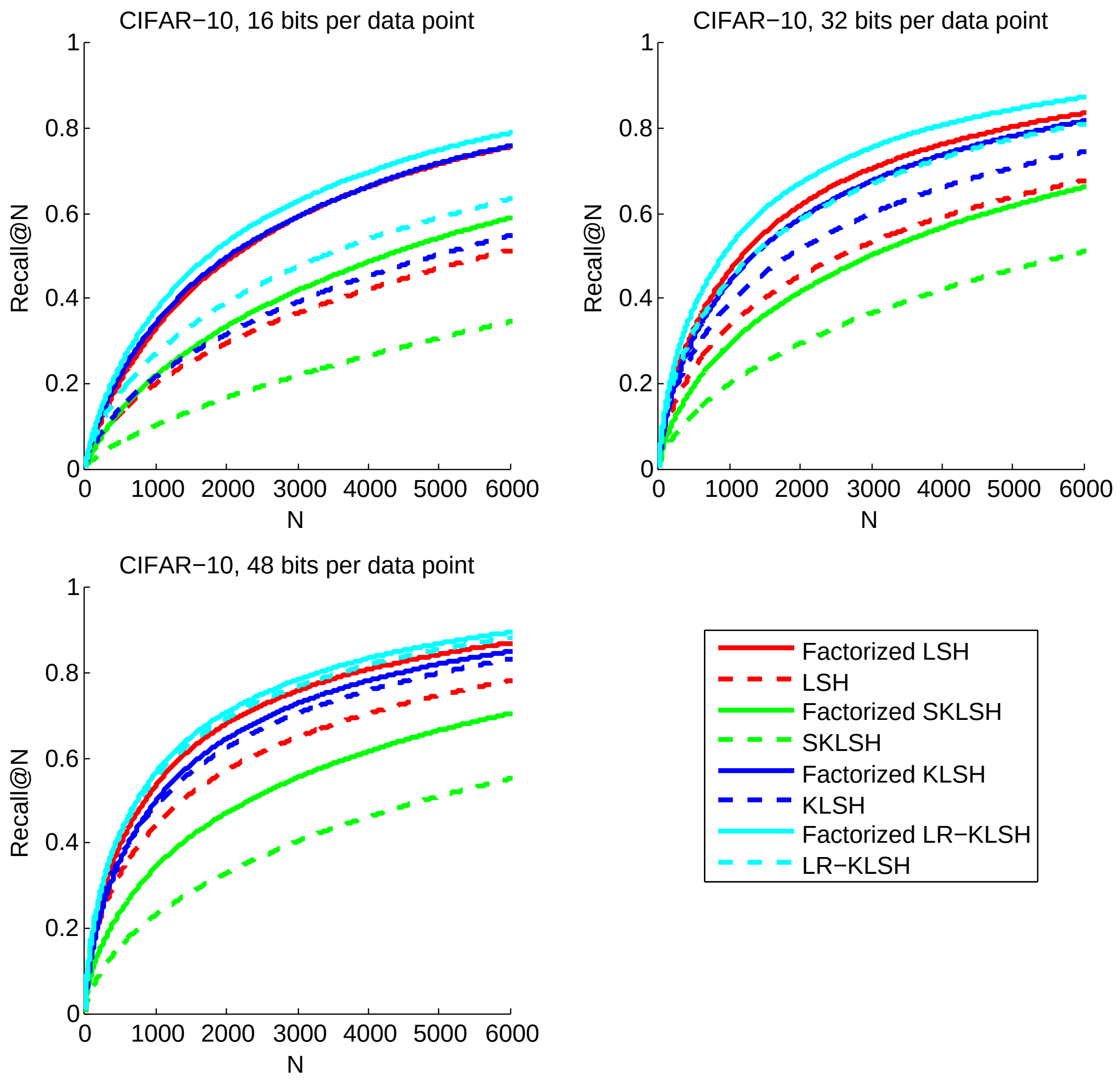

Figure 2: Experimental results on the CIFAR-10 [12] dataset. Dashed curves show the retrieval accuracy of conventional binary codes generated using LSH [3], SKLSH [22], KLSH [14], and LR-KLSH [11]. Solid curves show the retrieval accuracy of the corresponding factorized binary codes.

datasets. We computed 384-dimensional Gist descriptors for all images. For CIFAR-10, we randomly generated five training-testing splits, each containing 1,000 test queries. Results are averaged over the five splits. For LM+SUN, we used the standard training-testing split. Results are averaged over five trials.

Retrieval performance is measured using recall@N curves [5, 10, 19], which plot the proportion of true neighbors retrieved in the first N Hamming neighbors. The ground truth is considered to be the query's 10 nearest Euclidean neighbors.

The proposed approach does not assume any particular algorithm for generating hash functions, only that we can discover and take advantage of correlations among the generated hash functions. We applied factorized binary codes to several locality-sensitive hashing algorithms, including traditional LSH [3], shift-invariant kernel preserving LSH (SKLSH) [22], kernelized LSH (KLSH) [14], and the recently presented low-rank extension of kernelized LSH (LR-KLSH) [11]. In the following we will denote the factorized versions by Factorized LSH, Factorized SKLSH, Factorized KLSH, and Factorized LR-KLSH. We set $c^l = 1024$ in all experiments, and varied the storage budget c from 16 bits, to 32 bits, to 48 bits per database point.

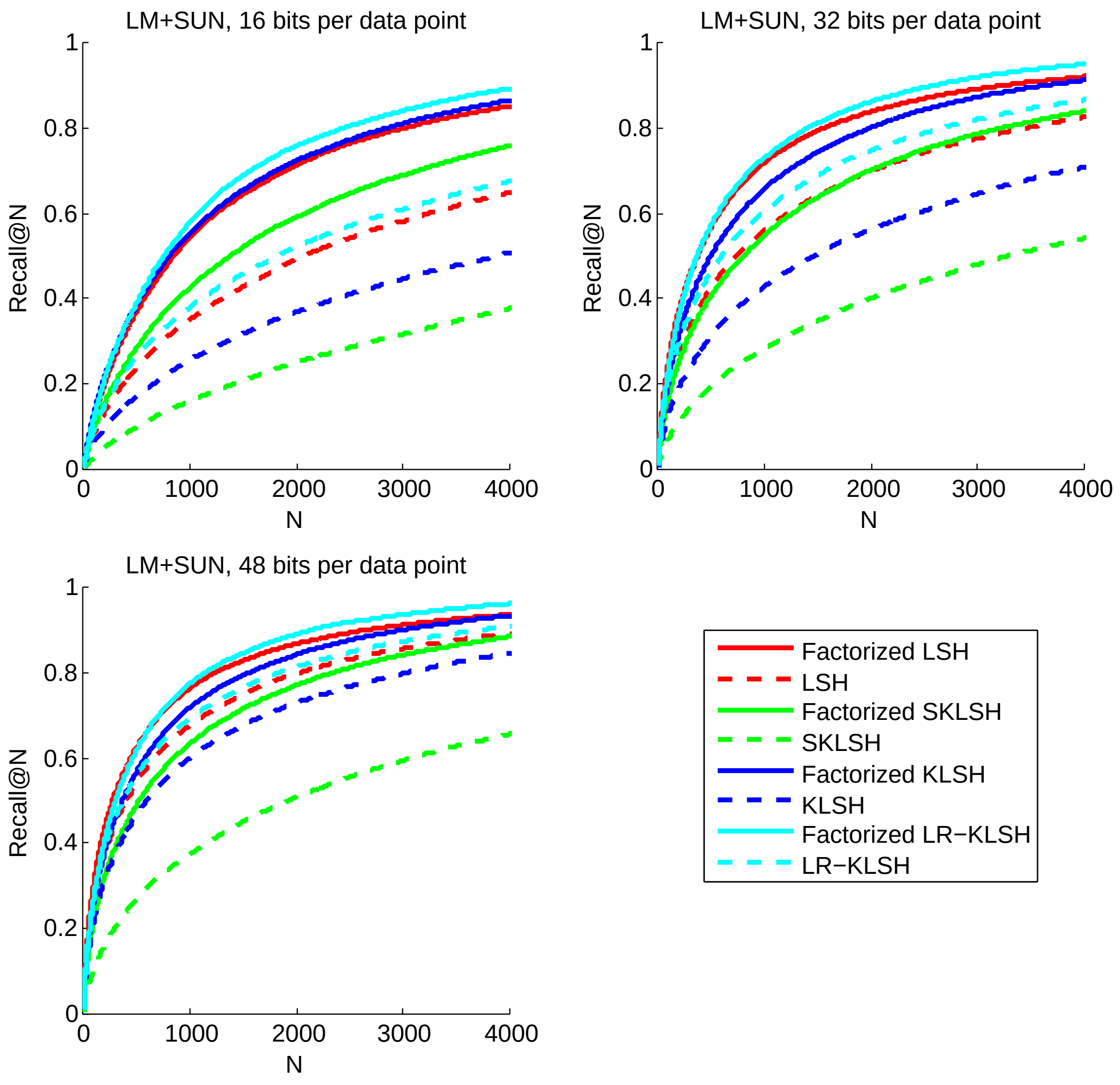

Figure 3: Experimental results on the LM+SUN [26] dataset. Dashed curves show the retrieval accuracy of conventional binary codes generated using LSH [3], SKLSH [22], KLSH [14], and LR-KLSH [11]. Solid curves show the retrieval accuracy of the corresponding factorized binary codes.

Fig. 2 shows the retrieval results on the CIFAR-10 dataset. The dashed curves show the recall performance of the conventional c-bit binary codes generated by LSH, SKLSH, KLSH, and LR-KLSH. The solid curves of the corresponding colors show the recall performance using factorized binary codes. We observed a consistent improvement in retrieval accuracy across the four hashing algorithms. For example, at 32 bits per database point, factorized binary codes lift the recall@1000 of LSH by 13% (or 39% relative), SKLSH by 9.2% (47% relative), KLSH by 5.4% (14% relative), and LR-KLSH by 7.7% (17% relative).

Fig. 3 shows similar retrieval results on the LM+SUN dataset. Factorized binary codes lift the recall across all four hashing algorithms. For example, at 32 bits per database point, an improvement of 16% (29% relative), 27% (94% relative), 23% (54% relative), and 13% (21% relative) in recall@1000 is obtained for LSH, SKLSH, KLSH, and LR-KLSH, respectively.

Fig. 4 shows typical qualitative results on LM+SUN at 32 bits per database point. Retrieval is performed using Factorized LSH. We observe that semantic neighbors are generally well preserved despite the high compression level. Replacing the 384-dimensional Gist descriptor with a 32-bit binary code reduces storage requirements by two orders of magnitude (1,536 or 3,072 bytes to 4 bytes).

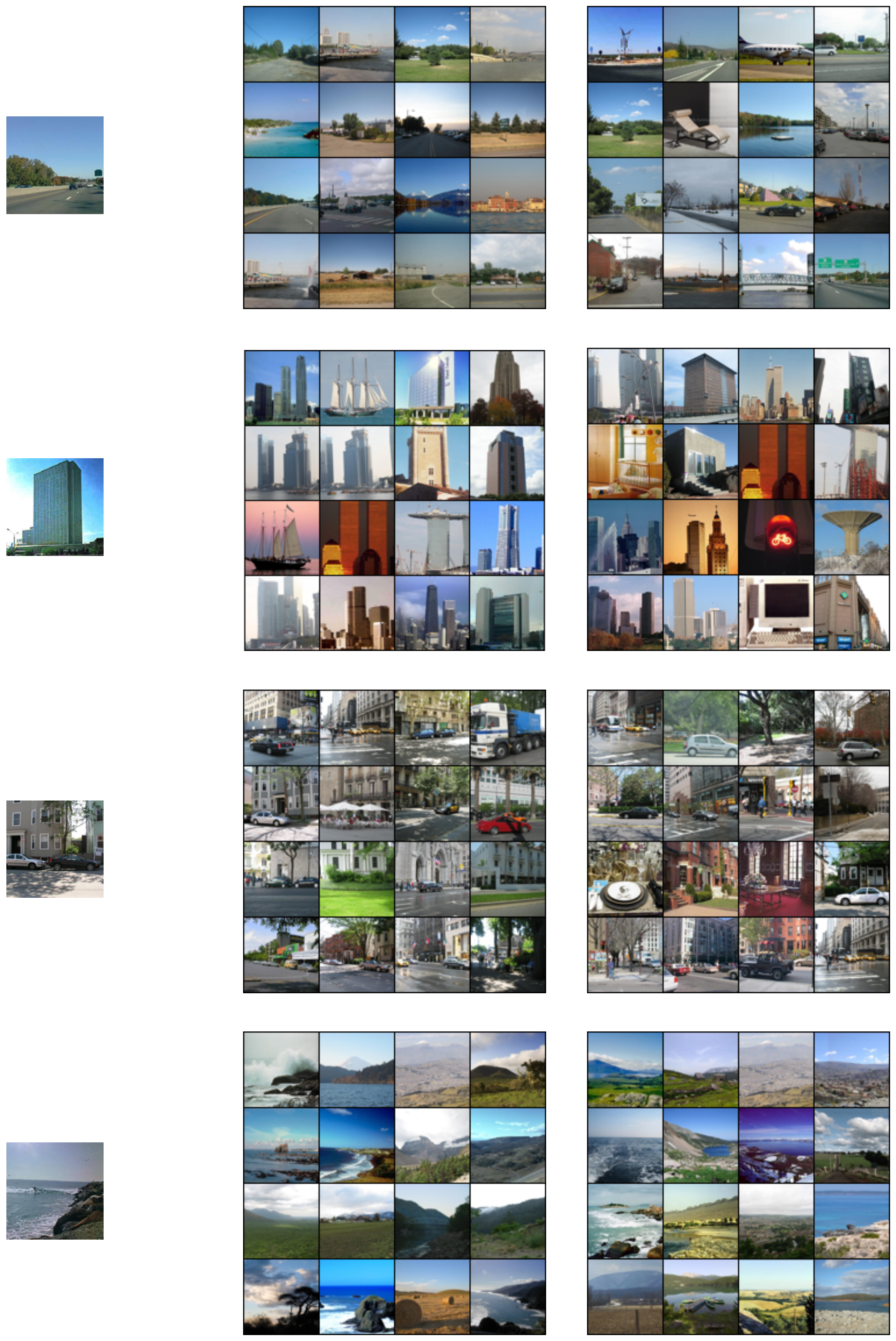

Figure 4: Qualitative results on the LM+SUN [26] dataset (collected from LabelMe [24] and SUN [31]). From left to right: query image, ground truth (Gist) neighbors, retrieved neighbors using Factorized LSH at a storage budget of 32 bits per database point. Neighbors are ordered from left to right, then top to bottom within each block.

Table 1: Search times on LM+SUN (all 500 queries)

	32 bits	48 bits
LSH	131 ± 19 ms	210 ± 4 ms
Factorized LSH	331 ± 26 ms	339 ± 28 ms
SKLSH	139 ± 21 ms	212 ± 5 ms
Factorized SKLSH	406 ± 43 ms	394 ± 70 ms
KLSH	137 ± 13 ms	240 ± 37 ms
Factorized KLSH	347 ± 31 ms	336 ± 23 ms
LR-KLSH	137 ± 16 ms	235 ± 19 ms
Factorized LR-KLSH	355 ± 20 ms	345 ± 19 ms

Table 2: Factorization learning times on LM+SUN

	16 bits	32 bits	48 bits
Factorized LSH	12.4 ± 0.2 min	24.8 ± 0.2 min	37.3 ± 0.4 min
Factorized SKLSH	14.0 ± 0.2 min	27.9 ± 0.8 min	41.8 ± 0.9 min
Factorized KLSH	12.6 ± 0.4 min	25.3 ± 0.4 min	36.4 ± 1.2 min
Factorized LR-LSH	14.1 ± 0.6 min	28.5 ± 0.6 min	41.6 ± 0.9 min

Computation time. Table 1 shows search times on LM+SUN (over all 500 queries). Time overhead is incurred from comparing the longer binary codes $\mathbf{y_q}$. Search times for factorized codes do not vary significantly between 32 and 48 bits because $c^l = 1024$ in all experiments. In addition, there is a one-time cost to compute $\tilde{\mathbf{Y}}^l$ using Eq. 9. Table 2 shows factorization learning times on LM+SUN. All timings are obtained using a desktop with a 3.60GHz CPU, single threaded execution with hardware XOR acceleration (popcount).

Applicability and future work. Factorization assumes we can discover correlations among the hash functions used to generate the 'long' binary codes $\mathbf{Y}^l$. Consequently, our approach cannot be applied on top of orthogonal hashing algorithms such as PCA hashing or iterative quantization [7], which generate uncorrelated hash functions by construction.

The performance of factorized binary codes depends on how accurately $\mathbf{Y}^l$ can be approximated by $\mathbf{S}$ and $\mathbf{B}$, and on how much improvement in retrieval is derived by increasing the number of hash functions. We believe a promising direction for future work will be to develop a more accurate method for finding $\mathbf{S}$ and $\mathbf{B}$.

5 Conclusion

Hashing algorithms for large-scale nearest neighbor search typically improve as the number of hash functions increases. This paper has presented factorized binary codes, which use an approximate binary matrix factorization to increase the number of hash functions while maintaining the original storage costs of the binary codes. The presented approach is general and does not assume a particular hashing algorithm. We believe that factorized binary codes will be a useful technique for boosting the performance of hashing algorithms that can derive a large benefit from additional hash functions. We plan to improve the technique further by pursuing more effective binary matrix factorization.

Acknowledgements. This work was funded in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2014.

[2] A. Babenko and V. Lempitsky. Tree quantization for large-scale similarity search and classification. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2015.

[3] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. ACM Symposium on Theory of Computing*, 2002.

[4] T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[5] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2013.

[6] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. International Conference on Very Large Data Bases*, 1999.

[7] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: a Procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[8] J. Hays and A. A. Efros. Scene completion using millions of photographs. In *Proc. ACM SIGGRAPH*, 2007.

[9] J.-P. Heo, Z. Lin, and S.-E. Yoon. Distance encoded product quantization. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2014.

[10] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[11] K. Jiang, Q. Que, and B. Kulis. Revisiting kernelized locality-sensitive hashing for improved large-scale image retrieval. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[13] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems*, 2009.

[14] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1092–1104, 2012.

[15] S. Liu, X. Liang, L. Liu, X. Shen, J. Yang, C. Xu, L. Lin, X. Cao, and S. Yan. Matching-CNN meets KNN: quasi-parametric human parsing. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[16] K. Matzen and N. Snavely. Scene chronology. In *Proc. European Conference on Computer Vision*, 2014.

[17] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering*, 20(10):1348–1362, 2008.

[18] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proc. International Conference in Machine Learning*, 2011.

[19] M. Norouzi and D. J. Fleet. Cartesian k-means. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2013.

[20] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in Hamming space with multi-index hashing. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2012.

[21] V. Ordonez, G. Kulkarni, and T. L. Berg. Im2Text: describing images using 1 million captioned photographs. In *Advances in Neural Information Processing Systems*, 2011.

[22] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems*, 2009.

[23] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

[24] B. C. Russell, A. Torralba, K. Murphy, and W. T. Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77 (1-3):157–173, 2008.

[25] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. In *Proc. ACM SIGGRAPH*, 2006.

[26] J. Tighe and S. Lazebnik. Superparsing: scalable nonparametric image parsing with superpixels. *International Journal of Computer Vision*, 101(2):329–349, 2013.

[27] T. Tommasi and B. Caputo. Frustratingly easy NBNN domain adaptation. In *Proc. IEEE International Conference on Computer Vision*, 2013.

[28] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[29] F. Tung and J. J. Little. CollageParsing: Nonparametric scene parsing by adaptive overlapping windows. In *Proc. European Conference on Computer Vision*, 2014.

[30] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2008.

[31] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. SUN database: large-scale scene recognition from abbey to zoo. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 3485–3492, 2010.

[32] J. Yang, B. Price, S. Cohen, and M. Yang. Context driven scene parsing with attention to rare classes. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[33] Z. Zhang, T. Li, C. Ding, and X. Zhang. Binary matrix factorization with applications. In *Proc. International Conference on Data Mining*, 2007.