MULTI-BIN SEARCH: IMPROVED LARGE-SCALE CONTENT-BASED IMAGE RETRIEVAL

Abdelrahman Kamel[⋆] *Youssef B. Mahdi*[⋆] *Khaled F. Hussain*[⋆]

[⋆] Computer Science Department, Faculty of Computers and Information, Assiut University, Assiut, Egypt

ABSTRACT

The challenge of large-scale image retrieval has been recently addressed by many promising approaches. In this work, we propose a new approach that jointly optimizes the search accuracy and time by using binary local image descriptors, such as BRIEF and BRISK, and binary hashing methods, such as Locality Sensitive Hashing (LSH) and Spherical Hashing. We propose a Multi-bin search method that highly improves the retrieval precision of binary hashing methods. Also, we introduce a reranking scheme that increases the retrieval precision, but with a slight increase in search time. Evaluations on the University of Kentucky Benchmark (UKB) dataset show that the proposed approach greatly improves the retrieval precision of recent binary hashing approaches.

Index Terms— Image retrieval, Binary Hashing, Multi-bin search

1. INTRODUCTION

Image similarity search has been intensively addressed recently. This is due to its importance to many machine learning and computer vision applications. Such applications include object recognition [1], finding partial image duplicates on the web [2] and searching individual video frames [3]. Searching large-scale image datasets containing millions of images using brute-force matching is not a practical task. Most state-of-the-art large-scale image retrieval approaches use SIFT [4] feature descriptors to represent images. The SIFT descriptor has a very high discriminative power. Its drawback is the high dimensionality. To decrease descriptors' dimensionality, many binary descriptors have been introduced [5, 6, 7, 8, 9]. Binary descriptors are small and very fast to generate. In this work we use binary descriptors to achieve better speedup.

To face the curse of dimensionality, many methods have been introduced based on approximating the search process which is known as approximate nearest neighbor (ANN) search. Some approaches use the bag-of-visual-words (BoVW) model [3] to represent single images with vectors of unified dimensionality. Others [1] use clustering along with indexing data structures to quantize descriptors. Recent approaches [10, 11, 12] use hashing algorithms to encode local image descriptors into binary codes. In this work we use some hashing algorithms to solve the problem of high dimensionality.

Quantization errors are the major problem facing hashing methods. A query descriptor may generate a hash code that is different from its neighbors' hash codes resulting in retrieving wrong results, i.e. false positives. Trying to correct these quantization errors, we propose a simple and intuitive method. In this method, we do not only examine the single bin that contains the query descriptor, we also examine the surrounding bins in order to minimize quantization errors. Of course, examining more data will increase the search time, but the search accuracy would be highly increased.

Hashing methods approximates the problem of searching for nearest neighbors of a query descriptor to only searching for its nearest bin. In this work, we do not just search for the nearest bin or bins, we also search inside bins to find which descriptors are nearer to the query than others. This extra step requires some very fast algorithms due to the large number of descriptors that may reside in a single bin.

2. RELATED WORK

2.1. Binary Descriptors

Recent approaches to image representation favor binary descriptors such as BRISK [7], BRIEF [5], BFROST [6], ORB [8] and FREAK [9] over other high-dimensional ones like SIFT [4] and SURF [13]. This is due to their small size and fast generation, although they are less discriminative. Also, the Hamming distance can be used directly to measure the distance between pairs of binary descriptors, this is another advantage of binary descriptors. The Hamming distance d_{HD} is simply the number of different bits in a pair of binary strings. The number of ones in a binary string is known as its population count (*popcnt*), so we may write the Hamming distance as: $d_{HD} = popcnt(v_1 \oplus v_2)$.

In this work, we chose to use the BRISK binary descriptor, which was proved to provide high discriminative power compared to other binary descriptors and significantly lower computational cost. The low computational cost of BRISK is due to its use of a scale-space FAST-based detector [14].

2.2. Hashing Methods

Hashing descriptors into binary codes has been proved efficient for solving ANN problems. One of the most famous hashing methods is the Locality Sensitive Hashing (LSH) [15]. In LSH, descriptors are projected on a set of random vectors which is randomly generated from a specific distribution such as Gaussian distribution. Each random vector represents a hyperplane that works as a hash function. Each hash function generates a binary bit depending on whether a vector resides above or below the hyperplane. The total bits resulting from the hash functions constitutes the binary code.

Another efficient hashing approach is Spherical Hashing (SH) [12], in which all data vectors are projected over hypersphere-based, instead of hyperplanes, hashing functions. Each hash function is represented by a hypersphere with some center vector and a radius. Unlike LSH, each spherical hash function generates a binary bit depending on whether a vector resides inside or outside a hypersphere.

In this work, we apply our improvement algorithms on LSH and SH. Evaluations will show that our improvements adds a high increase in search precision to each method.

2.3. Fast Exhaustive-Search

One of the proposed improvements is to find the descriptors, inside a target bin, that are nearer to the query descriptor than others. To do this, a fast algorithm is required to examine all descriptors inside the target bin.

Many exhaustive-search equivalent algorithms were introduced recently. These algorithms yields the same or too close results to exahustive-search results with a significant speedup. Most of these algorithms depends on examining data points before matching them with the query point. Examples are Partial Distortion Elimination (PDE) [16], Projection Kernels (PK) [17], Low Resolution Pruning (LRP) [18] and Norm Ordered Matching (NOM) [19].

We preferred to use NOM which has been proved to exceed other algorithms, in addition, it is easy to implement and adapt. Also, NOM and PDE can be used with any metric distance measure, including the Hamming distance which we use, where PK and LRP are designed for L_2 distance. In NOM, not all descriptors are examined to find the nearest ones, but only a partition of them based on their norm values.

3. MULTI-BIN SEARCH

In this section, we clarify the approximated search process based on hashing methods. Then we present the Multi-bin search method which can be applied on any hashing or quantization method[1].

[1]Source code of Multi-bin search can be downloaded from `https://sites.google.com/site/multibinsearch/`

3.1. Approximate Search with Binary Hashing

In large-scale image retrieval, the set of images S usually consists of thousands or millions of images and the total number of descriptors grows to the order of billions. At this scale, hashing or quantization methods are used to approximate the search process.

Any binary hashing method can be defined as follows: given a set of data vectors, we need to represent each data vector v_i with a binary code b_i as $b_i = \{0,1\}^l$ where l is the length of the binary code. In order to make this mapping from data vectors to binary codes, we need a set of l hashing functions $H(v) = (h_1(v), h_2(v), .., h_l(v))$. Each hash function generates a single bit in the binary code, 0 or 1. The goal of the hashing functions is to generate the same binary code for the approximate nearest neighbor vectors.

After applying the hashing algorithm, the data vectors can be divided into a set G of bins $G = \{W_1, W_2, .., W_{n_w}\}$ each bin contains the vectors with the same binary code. The maximum number of bins would be 2^l, i.e., $n_w \leq 2^l$. The approximation here is to consider all the data vectors in a single bin as approximate nearest neighbors, i.e., matches to each other. So, the set of nearest neighbor vectors to a query vector x found inside a bin W is $NN(x) = \{y | y \in W, x \in W\}$. Of course, the precision here depends on the algorithm used for hashing and the length of the generated binary code. It is clear that increasing the binary code length increases the precision.

3.2. Nearest Neighbor Bins

To minimize quantization errors, for each target bin W_i, we *pre-compute* and store its nearest n_{w_i} bins. These nearest bins are selected based on some distance threshold t_w, which is empirically chosen, where the distance between two bins $d(W_1, W_2)$ is computed as the distance between their centers, i.e., their average vector. This distance threshold is, of course, dependable on the binary code length. So, the set containing the indices of the nearest bins to some other bin W_i is $NN(W_i) = \{j | d(W_i, W_j) \leq t_w\}$.

Of course, the computation of nearest neighbor bins is done offline as a pre-processing step. After computing these nearest bins we search inside them, in addition to the target bin, for the nearest neighbors of the query vector. Only vectors within a distance threshold t_v, which is empirically chosen, are considered near neighbors to x.

3.3. Single Bin Search

Exhaustively searching a target bin would be very time consuming due to the large number of vectors inside a single bin, even if the distance measure used to match vectors is very fast like Hamming distance. So, up to now, the problem is the time cost of searching for near neighbor vectors inside a targeted bin. To solve this problem we used NOM, mentioned earlier. A version of NOM adapted to match our case

Algorithm 1 Searching a target bin using NOM

Input: a query vector x, the target bin W_i where $x \in W_i$, a set of population counts of vectors inside the bin $P_i = \{p_y | y \in W_i\}$, and a distance threshold t_v
Output: a set $NN_i(x)$ containing the nearest neighbor vectors of the query vector x within the distance threshold t_v
begin
$p_x \leftarrow popcnt(x)$
$lowerBound \leftarrow p_x - t_v$
$upperBound \leftarrow p_x + t_v$
for each $y | y \in W_i$ **do**
  if $lowerBound \leq p_y \leq upperBound$ **then**
    if $d_{HD}(x, y) < t_v$ **then**
      Add y to the set $NN(x)$
    end if
  end if
end for each
end

Algorithm 2 Approximated image matching

Input: two sets of image descriptor vectors $I_1 = \{v_{11}, v_{12}, .., v_{1n}\}, I_2 = \{v_{21}, v_{22}, .., v_{2m}\}$ where $n > m$, their corresponding population counts $P_1 = \{p_{11}, p_{12}, .., p_{1n}\}, P_2 = \{p_{21}, p_{22}, .., p_{2m}\}$, and a distance threshold t_v
Output: a score s representing the matching percentage between I_1 and I_2
begin
$s \leftarrow 0$
for $i = 1$ to n **do**
  $lowerBound \leftarrow p_{1i} - t_v$
  $upperBound \leftarrow p_{1i} + t_v$
  for $j = 1$ to m **do**
    if $lowerBound \leq p_{2j} \leq upperBound$ **then**
      if $d_{HD}(v_{1i}, v_{2j}) < t_v$ **then**
        $s \leftarrow s + 1$
      end if
    end if
  end for
end for
$s \leftarrow s/(n + m)$
end

is shown in Algorithm 1. In this version, the population count is *pre-computed* offline for all vectors. The algorithm computes lower and upper bounds of population count that ensure a vector may result in a distance less than the threshold. If a vector's population count is outside these bounds, it is assured that it will result in a distance greater than the threshold, so it is skipped. As a result, the distance computation are skipped for a large number of the vectors inside the bin.

A remaining issue with the proposed approach is how to tell that an image in the dataset is a match for a query image depending on matching individual descriptors of the query image. Here, we use a simplified scoring approach to tell a percentage of matching between a query image and the candidate images in the dataset. For each query image descriptor, we search for its approximate nearest neighbors and add *one point* to the *score* of the images they belong to. At the end, we normalize these scores by dividing each image's score by the sum of descriptors in the image itself and the query image. We put the images in descending order by their scores and pick the top k results. To enhance the retrieval precision, we apply a results reranking step, in which, the top k retrieved images are exhaustively re-matched with the query image using NOM. This method is shown in Algorithm 2.

4. RESULTS AND DISCUSSION

In this section, we firstly state our evaluation protocol and benchmarking dataset, then we present and discuss our experimental evaluation results.

4.1. Evaluation Protocol

We evaluated our proposed method over the famous University of Kentucky Benchmarking (UKB) dataset [1]. This dataset consists of 10200 images grouped into 2550 categories, each category contains four images that are considered matches to each other. Given a query image, it is required to get the image itself and the other three images in the same category as the top four results. So, the precision measure is a floating-point *score* in the range from 0 to 4.

We applied our method on LSH and SH. For LSH, we run another variation of the algorithm that requires to center the data vectors around the origin vector, i.e. zero-centered, this variation is denoted LSHZC. We generated BRISK local image descriptors of size 512 bits.

All experiments were run on an Intel Core™ i7-950 processor with 8 MB cache and 3.06 GHz clock speed, and 8 GB memory. We took advantage of the newly introduced POPCNT instruction which was introduced with the Nehalem-microarchitecture-based Core i7 processors. This instruction efficiently computes the population count of binary strings. We used it in measuring Hamming distances between bins or vectors and in the offline pre-computing of the population counts for all bins and vectors.

In all experiments, we ran all 10200 images as queries and took the average score. We tested hashing methods with binary code lengths ranging from 12 to 40. For Multi-bin search, we used inter-bin distance threshold as a percent of the binary code length, we chose 0.125, i.e. 1 bit for each 8 bits in the binary code. This threshold empirically showed the best trade-off between search time and accuracy. In the reranking step, we reranked only the top 50 results to make a

Table 1. Average query times (milliseconds)

	Binary code length							
Method	12	16	20	24	28	32	36	40
SH	10.044	1.752	0.646	0.290	0.232	0.060	0.187	0.126
LSH	157.278	147.728	46.663	24.335	11.250	4.303	1.292	0.237
LSHZC	5.216	0.846	0.219	0.144	0.141	0.121	0.175	0.178
SH-MB	50.925	74.996	78.213	73.335	74.305	81.444	28.933	172.792
SH-MB-R	98.773	122.445	124.856	119.882	119.780	126.975	75.580	220.559
LSH-MB	324.395	1778.833	1450.745	2854.049	3901.451	2375.265	2089.725	537.185
LSH-MB-R	375.055	1826.268	1498.360	2902.558	3950.555	2424.009	2375.980	584.126
LSHZC-MB	10.919	19.546	22.363	34.692	69.251	62.591	1.960	0.821
LSHZC-MB-R	60.113	69.730	71.759	82.765	116.497	110.434	48.107	82.564

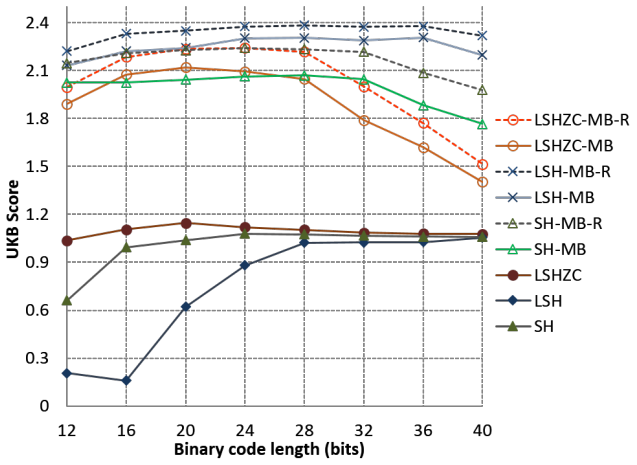
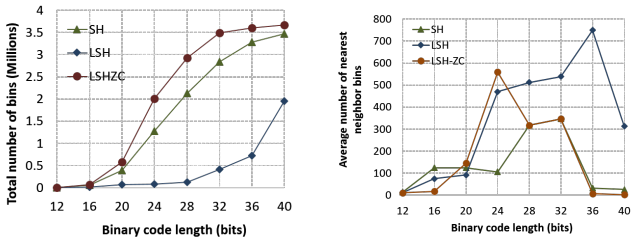


Fig. 1. Comparison of retrieval precision (UKB score) between hashing methods before and after applying Multi-bin serach and reranking steps.



(a) (b)

Fig. 2. (a) Total number of bins generated by each hashing method. (b) Average number of nearest neighbor bins for each binary code length.

trade-off between accuracy and time. We denoted all evaluated methods as follows:

- SH: Spherical Hashing.
- LSH: Locality Sensitive Hashing.
- LSHZC: LSH with Zero-Centered vectors.
- MB: Applying Multi-Bin search.
- R: Applying a Reranking step.

4.2. Experimental Results

Figure 1 shows the scores of the evaluated methods SH, LSH and LSHZC, and the improvement in score resulting from applying Multi-bin search method. Also, the improvement introduced by the reranking step is shown with dashed lines. It is shown that Multi-Bin search improves the retrieval precision (UKB score) of all evaluated methods by about 100%. The reranking step adds some precision to all Multi-bin methods by about 10-20%.

Table 1 shows average query times over 10200 queries. Of course, original hashing methods SH and LSHZC, especially with larger binary code lengths, have the shortest query times but with very low precision values as shown above in Fig. 1. The LSH, LSH-MB and LSH-MB-R query times is too long compared to other methods, this is due to the large sizes of bins generated by LSH. This is shown in Fig. 2 (a), the total number of bins generated by each hashing method for various binary code lengths. It is clear that LSH generates the smallest number of bins. This yields larger bins that require more time to examine. LSHZC-MB has the shortest times among all Multi-bin methods, ranging around 50 milliseconds. Figure 2 (b) shows the average number of nearest neighbor bins found within threshold for each binary code.

4.3. Conclusions

In this work, we proposed a large-scale image retrieval method based on binary hashing methods and binary local image descriptors. In this method, we not only search a target bin, but also we search the nearest neighbor bins to a target bin. The proposed approach showed significant improvement, about 100% over original hashing methods. In our future work, we would like to evaluate our method on many other hashing methods and binary descriptors. Also, we would like to use larger datasets for exploiting scalability issues.

5. REFERENCES

[1] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR06)*, vol. 2, pp. 2161–2168, 2006.

[2] Z. Wu, Q. Ke, M. Isard, and J. Sun, "Bundling features for large scale partial-duplicate web image search," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 25–32.

[3] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *International Conference on Computer Vision (ICCV)*. IEEE, 2003, pp. 1470–1477.

[4] David G Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[5] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European Conference on Computer Vision (ECCV)*. 2010, pp. 778–792, Springer.

[6] J. Cronje, "Bfrost: binary features from robust orientation segment tests accelerated on the gpu," in *Proceedings of the 22nd Annual Symposium of the Pattern Recognition Society of South Africa*, 2011.

[7] S. Leutenegger, M. Chli, and R.Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2548–2555.

[8] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *Computer Vision (ICCV), International Conference on*. IEEE, 2011, pp. 2564–2571.

[9] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *Computer Vision and Pattern Recognition (CVPR), Conference on*. IEEE, 2012, pp. 510–517.

[10] M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

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

[12] Jae-Pil Heo, YoungWoon Lee, Junfeng He, Shih-Fu Chang, and Sung-eui Yoon, "Spherical hashing," in *International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012.

[13] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[14] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision (ECCV)*. 2006, pp. 430–443, Springer.

[15] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of Computing*. ACM, 1998, pp. 604–613.

[16] L.J. Liu, X.B. Shen, and X.C. Zou, "An improved fast encoding algorithm for vector quantization," *Journal of the American Society for Information Science and Technology*, vol. 55, no. 1, pp. 81–87, 2004.

[17] Y. Hel-Or and H. Hel-Or, "Real-time pattern matching using projection kernels," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 9, pp. 1430–1445, 2005.

[18] M. Gharavi-Alkhansari, "A fast globally optimal algorithm for template matching using low-resolution pruning," *Image Processing, IEEE Transactions on*, vol. 10, no. 4, pp. 526–533, 2001.

[19] Mohamed Yousef and Khaled F. Hussain, "Fast exhaustive-search equivalent pattern matching through norm ordering," *Journal of Visual Communication and Image Representation*, vol. 24, no. 5, pp. 592–601, 2013.