Unsupervised Rank-Preserving Hashing for Large-Scale Image Retrieval

Svebor Karaman
Columbia University
New York, NY, USA
sk4089@columbia.edu

Xudong Lin
Columbia University
New York, NY, USA
xl2798@columbia.edu

Xuefeng Hu
University of Southern California
Los Angeles, CA, USA
xuefengh@usc.edu

Shih-Fu Chang
Columbia University
New York, NY, USA
sc250@columbia.edu

ABSTRACT

We propose an unsupervised hashing method, exploiting a shallow neural network, that aims to produce binary codes that preserve the ranking induced by an original real-valued representation. This is motivated by the emergence of small-world graph-based approximate search methods that rely on local neighborhood ranking. We formalize the training process in an intuitive way by considering each training sample as a query and aiming to obtain a ranking of a random subset of the training set using the hash codes that is the same as the ranking using the original features. We also explore the use of a decoder to obtain an approximated reconstruction of the original features. At test time, we retrieve the most promising database samples using only the hash codes and perform re-ranking using the reconstructed features, thus allowing the complete elimination of the original real-valued features and the associated high memory cost. Experiments conducted on publicly available large-scale datasets show that our method consistently outperforms all compared state-of-the-art unsupervised hashing methods and that the reconstruction procedure can effectively boost the search accuracy with a minimal constant additional cost.

CCS CONCEPTS

• **Information systems → Learning to rank; Top-k retrieval in databases; Search index compression; Image search; Retrieval efficiency.**

KEYWORDS

Hashing; Binary representation; Content-based Indexing; Image Retrieval; Indexing; Reconstruction; Large-Scale Retrieval

ACM Reference Format:
Svebor Karaman, Xudong Lin, Xuefeng Hu, and Shih-Fu Chang. 2019. Unsupervised Rank-Preserving Hashing for Large-Scale Image Retrieval. In *International Conference on Multimedia Retrieval (ICMR '19), June 10–13, 2019, Ottawa, ON, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3323873.3325038

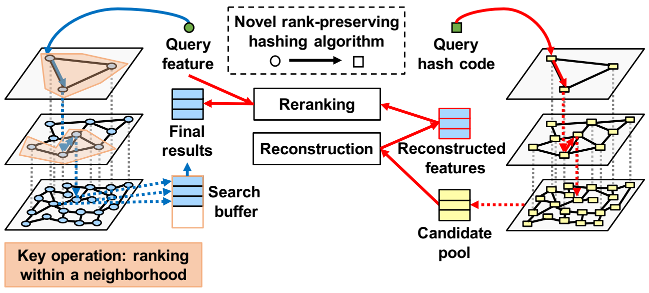

Figure 1: The search process with HNSW using real-valued features (left, blue pathway) and our URPH hashing model and its decoder (right, red pathway). The search process with real-valued features relies heavily on ranking features in a neighborhood. We therefore propose to learn a rank-preserving hashing method to compress the real-valued features into binary codes, and a decoder to perform re-ranking without needing to access the original features.

1 INTRODUCTION

Researchers in computer vision have developed many methods [22, 25, 27] to extract representations to capture the most important information of an image. These representations are commonly used in content-based retrieval systems enabling searching for similar images in a large database. However, for a large database, the cost of an exhaustive search and of storing all the uncompressed features both become prohibitive. Therefore, the development of efficient and high-accuracy approximate nearest neighbour search (ANN) methods as well as compression methods that preserve the original representation discriminability are active research topics.

The ANN problem has been studied for decades [3, 6], but an emerging trend in the last few years is to exploit a graph structure to perform the search. The Hierarchical Navigable Small-World (HNSW) method [28] is now dominating the ANN benchmarks[1]. The main drawback of such graph-based method is that they require

[1]https://github.com/erikbern/ann-benchmarks

storing all the original features. The recent work [9] applies product quantization coding to the HNSW indexing. Their problem setting is considering mostly the compression ratio over search performance, and the product quantization method doesn't have the advantage of fast Hamming distance computation of hash codes. Due to these differences, we consider it as a related but different problem. The search process with HNSW, illustrated in the left of Figure 1, relies on ranking within the neighborhood of each node in the graph in the top layers and completely ranking a search buffer in the bottom layer[2]. We therefore explore the idea of learning a hashing model, to compress each sample representation into binary codes, preserving the ranking obtained with the real-valued features. Our goal is to reduce the feature storage cost by using hashing while maintaining the effective search performance of HNSW. We further propose to learn a decoder enabling to perform re-ranking using reconstructed features and thus allow to discard the original features.

Many approaches have been proposed to learn to produce binary codes [6, 11, 12, 26, 29, 32, 38], the works being most relevant to our proposed method are rank-preserving (either supervised [31, 33, 36, 39] or unsupervised [8, 34]) and reconstruction-related methods [5, 10, 15, 16, 19, 20, 23, 30]. Hashing is often used as a first step in the search process. Namely, computing exhaustively the Hamming distances between a query hash code and all database hash codes and gathering a small candidate set of samples that have the smallest Hamming distances. A second step, often referred to as *re-ranking*, is to get the corresponding original features of the candidate samples and compute the exact distances with the query feature. This re-ranking step can be slow if the database samples features are read from disk and require a lage amount of memory to store all the features for faster access.

Low dimensional (i.e. 64 bits or less) hash codes are commonly used in the literature but suffer from limited discriminative power compared to the original features. Prior rank-preserving methods usually aim to train hash codes that preserve the ranking defined according to labels, while our goal is to preserve the ranking obtained with original features that we assume have been heavily optimized for the target task. Reconstruction based methods most of the time exploit reconstruction in the optimization process to learn the hash model but not at test time to re-rank as we do.

In this work, our goal is to totally get rid of the original features and take advantage of both the graph-based indexing method like HNSW and the compact hash codes. We therefore propose to learn:

- an unsupervised hashing model producing medium length hash codes, that is optimized directly for the search task by aiming to preserve the rank order within any set of samples;
- a decoder to reconstruct approximation of the original features from hash codes, that can be exploited to perform re-ranking without requiring the original features.

Our approach is an effective solution to solve the large-scale content-based retrieval problem while allowing to fully discard the original features. In our search process illustrated in Figure 1 in the right side as the red pathway, we first collect a candidate pool of the most promising database samples using a HNSW graph built only with the hash codes, then reconstruct an approximation of their features on-the-fly and re-rank the candidates by comparing the query feature to their reconstructed features. Experiments conducted on million-scale datasets show the consistent superiority of our method compared to state-of-the-art hashing methods (with relative performance gains of up to 35%). Our method integrated with the graph-based search method [28] overcomes its main limitation which is its memory usage. For example, for SIFT1M [18], the proposed method using 256 bits can reduce the memory requirement to store the samples from 488MB to 31MB only (plus 2MB to store the decoder weights), and achieves a 1-Recall@10 of 0.77.

[2]Due to space limitation, we refer the reader to [28] for additional details.

2 OUR METHOD

Formally, the goal of learning to hash is to learn a mapping $H : \mathbb{R}^n \to \{0, 1\}^m$. Ideally, the hashing model is supposed to be

$$H(x; \theta) = \text{sign}(F(x; \theta)), \tag{1}$$

where sign is the sign function and $F(x; \theta) : R^n \to R^m$ is a transformation (e.g. a neural network) and θ are the parameters of that transformation. However, the sign function being not differentiable forbids the optimization of the hashing model by gradient descent. To tackle this problem, we follow [35] and relax the sign function with the hyperbolic tangent (tanh)

$$\tilde{H}(x; \theta) = \tanh(F(x; \theta)). \tag{2}$$

For simplicity, we will omit the parameters θ and use $H(x)$ and $\tilde{H}(x)$ instead of $H(x; \theta)$ and $\tilde{H}(x; \theta)$ in the rest of the paper. Since we relax the hashing model in the training time, we also need to approximate the Hamming distance and we use the Euclidean distance, denoted as $d(\cdot, \cdot)$ in the following, as a surrogate.

2.1 Unsupervised Rank-Preserving Hashing

Unlike supervised ranking-based hashing methods, no semantic labels are provided in our setting. Therefore, to obtain a ranking to preserve, we propose to simulate a search process with the original features. As shown in Figure 2, during training we construct a mini-batch of batch size N by: (i) randomly selecting N samples from the training set; (ii) randomly selecting one sample from them to be the query q; (iii) ranking the rest of the samples in ascending distance to the query. Our main optimization goal is to preserve the ranking of the batch according to the query.

The order is preserved if and only if, for any triplet composed of the query q, and two candidates i and j, i and j remain in the same order in the Hamming space. Therefore, we want to penalize any triplets in the Hamming space that are not in the original order with our ranking loss defined as

$$L_R = \sum_{i=1}^{N-2} \sum_{j=i+1}^{N-1} w_i [d(\tilde{H}(x_i), \tilde{H}(q)) - d(\tilde{H}(x_j), \tilde{H}(q))]_+, \tag{3}$$

where q is the original feature of the query, and x_i is the feature of the i^{th} candidate, and $[\cdot]_+ = \max(\cdot, 0)$. w_i is a weight to ensure that the model penalizes more any misordering happening at smaller ranks and it is defined as

$$w_i = \exp\left(-\frac{i-1}{N-1}\right). \tag{4}$$

We also adopt two widely-used regularization terms [35] in learning to hash methods: bit uncorrelation loss and binarization loss.

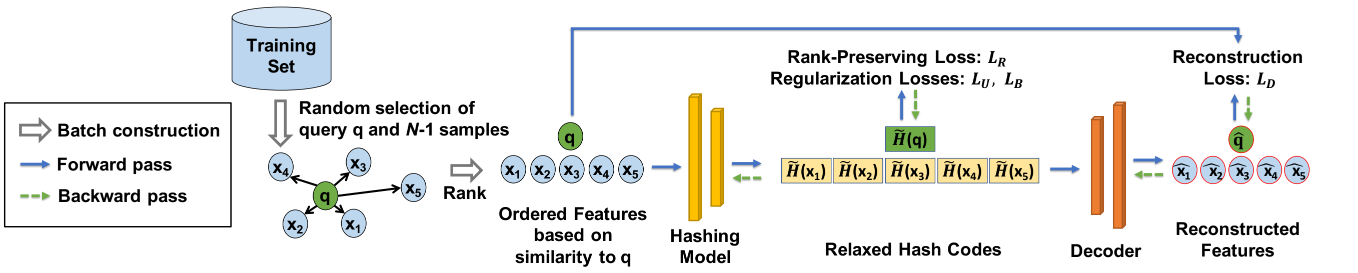
Training Set
Batch construction
Forward pass
Backward pass
Random selection of query q and N-1 samples
Rank
Ordered Features based on similarity to q
Hashing Model
Rank-Preserving Loss: L_R
Regularization Losses: L_U, L_B
Relaxed Hash Codes
Decoder
Reconstruction Loss: L_D
Reconstructed Features

Figure 2: Illustration of our URPH training procedure. We first construct training batches by randomly selecting a query q and $N-1$ samples from the training set, where N is the batch size. We order the features based on their similarity to q and compute their hash codes. We utilize the proposed rank-preserving loss and regularization losses to learn a relaxed hashing model. Then we use the binarized hash codes to learn a decoder which reconstructs an approximation of the original features, e.g. $\hat{x_1}$, from the hash codes, e.g. $H(x_1)$. The reconstruction loss is only used to optimize the decoder *not* the hashing model.

The bit uncorrelation loss L_U is aiming to make every bit useful by enforcing every pair of bit to be uncorrelated, and is computed as

$$L_U = ||\bar{B}^T \bar{B} - I||_2, \tag{5}$$

$\bar{B} = [\bar{H}(q); \bar{H}(x_1); \bar{H}(x_2); ...; \bar{H}(x_{N-1})]$ is a $N \times M$ matrix of N M-bit l2-normalized relaxed hash codes, and I is the $M \times M$ identity matrix. Note that here we apply l2-normalization to the relaxed hash codes before calculating the uncorrelation loss, as without l2-normalization a trivial solution is to have all bits close to zero.

The binarization loss L_B enforces the hash codes to be binary

$$L_B = 1 - \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \tilde{B}_{ij}^2, \tag{6}$$

where $\tilde{B}_{ij}$ denotes the element in the i^{th} row and j^{th} column of the matrix $\tilde{B} = [\tilde{H}(q); \tilde{H}(x_1); \tilde{H}(x_2); ...; \tilde{H}(x_{N-1})]$.

Our complete objective for learning our hashing model is

$$L = \lambda_R * L_R + \lambda_U * L_U + \lambda_B * L_B, \tag{7}$$

with λ_R, λ_U and λ_B being weights to control each loss contribution.

2.2 Reconstructing features from binary codes

Due to a small number of possible distances [8], there can be ambiguity or ties [13] in the candidate pool retrieved by Hamming distance comparisons. Usually, the re-ranking performed with the original features would alleviate that problem, but as we want to fully discard the original features, we propose to learn to reconstruct an approximation of the original features.

Formally, we learn a decoder $D(b; \phi) : \{0, 1\}^m \to \mathbb{R}^n$, where ϕ are the parameters of the model. Let us denote $\hat{x}_i = D(b_i; \phi)$ the reconstructed feature of x_i, with $b_i = H(x_i)$. We use a l2-norm loss as the optimization objective for the reconstruction

$$L_D = \frac{1}{N} \sum_{i=1}^{N} ||x_i - \hat{x}_i||_2. \tag{8}$$

Note that b_i is a binarized code without relaxation, and there is no gradient back-propagated from the decoder to the hashing model. An illustration of the training of our unsupervised rank-preserving hashing (URPH) method with its decoder is shown in Figure 2.

At test time, as illustrated in Figure 1 as the red pathway, we use the hashing model trained as detailed in the previous section to compress the original features into binary codes, build a HNSW index using these codes, and perform a search on the graph by computing Hamming distances between the query hash code and the hash codes of the samples encountered along the search path to obtain a candidate pool. Then we reconstruct real-valued features for the candidates obtained with the hash codes, and re-rank the pool by computing the asymmetric distance defined as

$$d_a(q, b_i) = d(q, \hat{x}_i), \tag{9}$$

where q is the query feature, b_i is the binary code of sample x_i and $\hat{x}_i$ its reconstructed feature, and $d(\cdot, \cdot)$ is the Euclidean distance.

2.3 Implementation and training

Our hashing model is a shallow neural network with a single hidden layer with 8× the number of dimensions of the input nodes using an exponential linear unit (elu) [7] as an activation function and followed by a batch normalization layer [17]. The decoder is also a shallow neural network with a mirrored architecture of the hashing model. We train our model with the stochastic gradient descent with Tensorflow [1] . To ensure that the rank-preserving loss dominates the optimization process, we set λ_R to the inverse of the first batch ranking loss value, while the λ_U, and λ_B are set 0.5 and 0.3 respectively. The batch size N is set to 512. We observed that the hashing model converges faster than the decoder, so we first train the hashing model and the decoder simultaneously for 50, 000 iterations then only the decoder for another 50, 000 iterations, as the hash codes will be stable it helps the decoder to converge.

3 EXPERIMENTS

In this section, we evaluate and compare our method with state-of-the-art methods on two public large-scale datasets:

- SIFT1M [18] consists of a training set, a query set and a base set, which are not overlapped. There are respectively 100, 000, 10, 000, and 1, 000, 000 samples in these sub-sets. Each sample is a 128-dimension SIFT [27] feature. We use l2-normalization as a pre-processing step for this dataset.
- Deep1M [2] consists of a training set of 1, 000, 000 samples, a query set of 10, 000 samples and a base set of 1, 000, 000

	Deep1M															SIFT1M											
	1-Recall@1					1-Recall@10					1-Recall@100					1-Recall@1				1-Recall@10				1-Recall@100			
# of bits	64	96	128	256	512	64	96	128	256	512	64	96	128	256	512	64	128	256	512	64	128	256	512	64	128	256	512
ITQ [12]	0.05	0.09	-	-	-	0.20	0.31	-	-	-	0.49	0.65	-	-	-	0.03	0.09	-	-	0.10	0.26	-	-	0.28	0.55	-	-
IsoH [21]	0.05	**0.10**	-	-	-	0.21	0.31	-	-	-	0.51	0.67	-	-	-	**0.10**	**0.18**	-	-	**0.27**	**0.45**	-	-	**0.57**	0.76	-	-
SH [37]	**0.06**	0.08	-	-	-	0.20	0.25	-	-	-	0.49	0.56	-	-	-	**0.10**	0.16	-	-	0.26	0.39	-	-	0.54	0.68	-	-
BRE [23]	0.03	0.06	0.08	0.16	0.22	0.13	0.22	0.29	0.46	0.62	0.37	0.52	0.63	0.81	0.92	0.02	0.07	0.12	0.19	0.09	0.20	0.32	0.48	0.25	0.46	0.63	0.79
SpH [14]	0.04	0.07	0.09	0.18	0.27	0.15	0.24	0.31	0.51	0.68	0.40	0.56	0.66	0.85	0.94	0.07	0.15	0.24	0.32	0.21	0.38	0.56	0.70	0.48	0.71	0.86	0.93
LSH [6]	0.04	0.07	0.10	0.19	0.32	0.15	0.24	0.32	0.55	0.77	0.39	0.54	0.66	0.89	0.98	0.03	0.10	0.19	0.30	0.11	0.27	0.45	0.67	0.27	0.53	0.76	0.93
KLSH [24]	0.04	0.07	0.10	0.19	0.30	0.14	0.24	0.32	0.54	0.74	0.37	0.53	0.66	0.87	0.96	0.06	0.15	0.23	0.31	0.20	0.38	0.54	0.70	0.44	0.69	0.86	0.94
OPH [34]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.20	0.31	0.37	-	0.51	0.68	0.81	-
URPH	**0.06**	**0.10**	**0.13**	**0.24**	**0.36**	**0.23**	**0.33**	**0.43**	**0.65**	**0.83**	**0.56**	**0.70**	**0.79**	**0.94**	**0.99**	0.09	**0.18**	**0.28**	**0.39**	0.26	0.44	**0.63**	**0.79**	**0.57**	**0.77**	**0.91**	**0.98**
URPH-RE	**0.10**	**0.16**	**0.21**	**0.36**	**0.49**	**0.35**	**0.51**	**0.61**	**0.82**	**0.94**	**0.56**	**0.70**	**0.79**	**0.94**	**0.99**	**0.12**	**0.23**	**0.34**	**0.45**	**0.37**	**0.58**	**0.77**	**0.88**	**0.57**	**0.77**	**0.91**	**0.98**

Table 1: 1-Recall@1, 10, 100 on Deep1M and SIFT1M with varying number of hash bits in [64, 96, 128, 256, 512].

samples, which are not overlapped. Each sample is a 96-dimension normalized deep feature produced by a deep neural network. Note that Deep1M is a sub-set of the originally proposed Deep1B [2] subsampled as described in [9].

3.1 Compared Methods and Evaluation Metrics

We compare our methods with widely-used unsupervised hashing methods: ITQ [12], SH [37], IsoH [21], BRE [23], KLSH [24], LSH [6], SpH [14]. We obtain their implementations from [4]. Note that ITQ, SH and IsoH all rely on an eigenvalue decomposition of the features' covariance matrix and thus cannot produce hash codes with more bits than the original features' dimension. We also compare with the results reported by the rank-preserving method OPH [34] on SIFT1M. We denote our hashing model as *URPH*, and our hashing model with reconstruction and asymmetric search as *URPH-RE*. For each method, we first train a model on the training set with hash codes length of 64, 96 only for Deep1M, 128, 256 or 512 bits. We then compress the database features into binary codes. Then, to speed up the search process compared to an exhaustive search, we build a HNSW graph using the binary codes[3]. We perform searches with the HNSW index built using the binary codes to retrieve a candidate pool of the K ($K \in \{1, 10, 100\}$ in our experiments) closest samples. These are the returned samples evaluated for all hashing methods with the widely-used 1-Recall-at-K metric defined as

$$1\text{-}Recall@K = \frac{1}{|Q|}\sum_{q=1}^{|Q|} |R(q,K) \cap NN(q)|, \qquad (10)$$

where Q is the query set, $R(q,K)$ are the top K retrieved samples and $NN(q)$ is the true nearest sample of query q in the database.

For UPRH-RE only, we re-rank the candidate pool, obtained with $K = 100$, by reconstructing the features of these candidates and comparing them to the query feature, as detailed in Section 2.2.

3.2 Discussions

Results on Deep1M and SIFT1M are reported in Table 1. Our URPH significantly outperforms previous hashing methods, especially when the number of bits is sufficient. URPH outperforms the best baseline methods on Deep1M by relative gains of 35.7%, 31.3%, and 19.3% for 1-Recall@(1,10,100) with 128 bits and of 25.3%, 17.4%, and 6.2% for 1-Recall@(1,10,100) with 256 bits. URPH outperforms the best baseline method on SIFT1M by relative gains of 16.0%, 12.3%, and 5.76% for 1-Recall@(1, 10, 100) with 256 bits and of 22.8%, 12.6%, and 3.7% for 1-Recall@(1, 10, 100) with 512 bits. Our method also outperforms the rank-preserving method OPH [34].

The asymmetric search (URPH-RE) can significantly boost the retrieval performance on both datasets with the 1-Recall@1 approaching 50% and the 1-Recall@10 of about 90% when using 512 bits, while adding a small constant cost of about 1ms to the search process that already takes about 7ms when using HNSW and would take up to 30ms using an exhaustive search process. The results obtained with HNSW for all hashing methods (including ours) are within 1% of the exhaustive search results.

4 CONCLUSIONS

We have presented an unsupervised hashing method aiming to produce binary codes that preserve the ranking ability of an original real-valued representation. Our method used with the hierarchical navigable small world graph indexing can produce accurate search results at a fast search speed overcoming the high storage memory requirement. Our method clearly outperforms other hashing methods on two million-scale datasets Deep1M and SIFT1M, especially when hash code lengths are sufficient. Furthermore, we show that learning a decoder to reconstruct an approximation of original features and perform re-ranking can significantly boosting the retrieval performance without storing the original features. Our proposed method allows to compress real-valued features into binary codes while preserving most of their retrieval ability.

ACKNOWLEDGMENTS

This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R&D Contract No. 2014-14071600012. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

[3]We evaluate all methods with HNSW and set $maxM = 32$, $maxM0 = 64$ and $efSearch = 1024$ to have a very good approximation of the exhaustive search

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

[2] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.

[3] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[4] Deng Cai. 2016. A revisit of hashing algorithms for approximate nearest neighbor search. *arXiv preprint arXiv:1612.07545* (2016).

[5] Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. 2015. Hashing with binary autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 557–566.

[6] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 380–388.

[7] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[8] Matthijs Douze, Hervé Jégou, and Florent Perronnin. 2016. Polysemous codes. In *European Conference on Computer Vision*. Springer, 785–801.

[9] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2018. Link and code: Fast indexing with graphs and compact regression codes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3646–3654.

[10] Ling-yu Duan, Yuwei Wu, Yicheng Huang, Zhe Wang, Junsong Yuan, and Wen Gao. 2018. Minimizing Reconstruction Bias Hashing via Joint Projection Learning and Quantization. *IEEE Transactions on Image Processing* 27, 6 (2018), 3127–3141.

[11] Yueqi Duan, Ziwei Wang, Jiwen Lu, Xudong Lin, and Jie Zhou. 2018. GraphBit: Bitwise Interaction Mining via Deep Reinforcement Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8270–8279.

[12] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2013), 2916–2929.

[13] Kun He, Fatih Cakir, Sarah Adel Bargal, and Stan Sclaroff. 2018. Hashing as tie-aware learning to rank. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 4023–4032.

[14] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. 2012. Spherical hashing. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2957–2964.

[15] Di Hu, Feiping Nie, and Xuelong Li. 2018. Deep binary reconstruction for cross-modal hashing. *IEEE Transactions on Multimedia* (2018).

[16] Mengqiu Hu, Yang Yang, Fumin Shen, Ning Xie, and Heng Tao Shen. 2018. Hashing with angular reconstructive embeddings. *IEEE Transactions on Image Processing* 27, 2 (2018), 545–555.

[17] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[18] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2011), 117–128.

[19] Hervé Jégou, Teddy Furon, and Jean-Jacques Fuchs. 2012. Anti-sparse coding for approximate nearest neighbor search. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2029–2032.

[20] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 861–864.

[21] Weihao Kong and Wu-Jun Li. 2012. Isotropic hashing. In *Advances in neural information processing systems*. 1646–1654.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[23] Brian Kulis and Trevor Darrell. 2009. Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*. 1042–1050.

[24] Brian Kulis and Kristen Grauman. 2009. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2130–2137.

[25] Hongzhi Li, Joseph G Ellis, Lei Zhang, and Shih-Fu Chang. 2018. PatternNet: Visual Pattern Mining with Deep Neural Network. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ACM, 291–299.

[26] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. 2012. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2074–2081.

[27] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60, 2 (2004), 91–110.

[28] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[29] Vidyadhar Rao, Prateek Jain, and CV Jawahar. 2016. Diverse yet efficient retrieval using locality sensitive hashing. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*. ACM, 189–196.

[30] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Semantic hashing. *International Journal of Approximate Reasoning* 50, 7 (2009), 969–978.

[31] Dongjin Song, Wei Liu, Rongrong Ji, David A Meyer, and John R Smith. 2015. Top rank supervised binary coding for visual search. In *Proceedings of the IEEE International Conference on Computer Vision*. 1922–1930.

[32] Dongjin Song, Wei Liu, David A Meyer, Dacheng Tao, and Rongrong Ji. 2015. Rank preserving hashing for rapid image search. In *Data Compression Conference (DCC), 2015*. IEEE, 353–362.

[33] Jun Wang, Wei Liu, Andy X Sun, and Yu-Gang Jiang. 2013. Learning hash codes with listwise supervision. In *Proceedings of the IEEE International Conference on Computer Vision*. 3032–3039.

[34] Jianfeng Wang, Jingdong Wang, Nenghai Yu, and Shipeng Li. 2013. Order preserving hashing for approximate nearest neighbor search. In *Proceedings of the 21st ACM international conference on Multimedia*. ACM, 133–142.

[35] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2018. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2018), 769–790.

[36] Qifan Wang, Zhiwei Zhang, and Luo Si. 2015. Ranking Preserving Hashing for Fast Similarity Search.. In *IJCAI*. 3911–3917.

[37] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Advances in neural information processing systems*. 1753–1760.

[38] Rui Yang, Yuliang Shi, and Xin-Shun Xu. 2017. Discrete Multi-view Hashing for Effective Image Retrieval. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM, 175–183.

[39] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. 2015. Deep semantic ranking based hashing for multi-label image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1556–1564.