To Appear in Proc. IEEE International Conference on Image Processing (ICIP), 2015

ONLINE SUPERVISED HASHING

Fatih Cakir and Stan Sclaroff

Department of Computer Science
Boston University
{fcakir,sclaroff}@cs.bu.edu

ABSTRACT

Fast similarity search is becoming more and more critical given the ever growing sizes of datasets. Hashing approaches provide both fast search mechanisms and compact indexing structures to address this critical need. In image retrieval problems where labeled training data is available, supervised hashing methods prevail over unsupervised methods. However, most supervised hashing methods are batch-learners; this hinders their ability to adapt to changes as a dataset grows and diversifies. In this work, we propose an online supervised hashing technique that is based on Error Correcting Output Codes. Given an incoming stream of training data with corresponding labels, our method learns and adapts its hashing functions in a discriminative manner. Our method makes no assumption about the number of possible class labels, and accommodates new classes as they are presented in the incoming data stream. In experiments with three image retrieval benchmarks, the proposed method yields state-of-the-art retrieval performance as measured in Mean Average Precision, while also being orders-of-magnitude faster than competing batch methods for supervised hashing.

Index Terms— Hashing, retrieval, indexing, similarity search.

1. INTRODUCTION

Similarity search lies at the heart of many applications and, as datasets continue to grow and diversify, expediting this search is becoming increasingly challenging and important. One promising family of approaches is based on *hashing*: data is mapped to binary vectors in Hamming space where the binary representations permit fast search mechanisms with a very low memory footprint. Recent applications utilizing this approach include: image annotation [1], visual tracking [2], 3D reconstruction [3], video segmentation [4], object detection [5], etc.

Hashing methods can be categorized broadly as data-independent and data-dependent techniques. Locality Sensitive Hashing methods [6, 7, 8] are prime examples of data-independent methods; they give guarantees on the approximation to particular metrics, without regard to the dataset to be indexed. For certain application settings, distances are defined only on the available data set; thus, solutions are required for learning the hashings directly from data. Such data-dependent solutions include: methods that approximate pairwise distances via quantization or spectral graph analysis [9, 10, 11], works that leverage available label information for semantic retrieval [12, 13, 14, 15], and semi-supervised methods that utilize both the data distribution and the available label information such as [16].

Hashing schemes constructed via data-dependent techniques tend to yield superior retrieval performance, primarily through the learning phase where properties like compactness and informativeness of the resulting binary codes are enforced. However, the computational cost of learning is crucial when large-scale datasets are considered, but most methods are batch-learners and slow. Moreover, with batch-learning it is difficult to adapt to variations in the dataset as it grows. Many applications require that the hash mappings be versatile given such changes as it would be extremely costly to do learning from scratch.

Weiss et al. [9] give two important properties a 'good' hash code must have: (1) it should easily be computed for a novel data point and (2) a small number of bits should be adequate enough in representing the data. Given the above discussion we propose a third important property: (3) a 'good' code must be amenable to the continued growth of a dataset. In this work, we propose an online supervised method for learning hash codes that satisfies all three properties. Our formulation is based on Error Correcting Output Codes (ECOCs). Our online method is orders-of-magnitude faster than batch-learning of hashing parameters and, more importantly, being online it is adaptable to variations in the incoming data. We consider a stochastic environment where the hash mappings are updated according to sequentially arriving data, and the number of classes is not known *a priori*.

We focus on the task of retrieving semantically similar neighbors for a given query. This task has its use in many applications including label-based image retrieval and annotation [17, 18], semantic segmentation [19], image super resolution [20] etc. For this task, supervised hashing methods tend to offer superior performance, mainly due to leveraging label information in learning the hash functions. Specifically, we deal with labels having distinct categorical identities. During testing, the goal is then to retrieve instances that share the same label(s) with the query.

As in [21, 22], we consider a stochastic environment where the hash mappings are updated according to sequentially incoming data with the use of ECOCs. However, unlike previous work, we assume no prior information on the label space. The arriving data can be associated with possibly unseen labels and the hash functions are accommodated to such variations. To our knowledge, this is the first supervised hashing method that allows the label space to grow.

2. ONLINE SUPERVISED HASHING

We are given a set of data points $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ where $\mathbf{x} \in \mathcal{X}$ denotes an instance in the feature space $\mathcal{X}$ and $\mathbf{y} \subseteq \mathcal{Y}$ is a subset of the label space $\mathcal{Y}$. The goal of hashing is to learn a mapping $\Phi : \mathcal{X} \rightarrow \mathcal{H}^B$ such that similarities in the original space $\mathcal{X}$ are preserved in the $B-$dimensional Hamming space $\mathcal{H}^B$. The similarity can be induced from a particular metric or it can be derived

from label information. Following recent work, we utilize a set of hash functions for the mapping, i.e., $\Phi(\mathbf{x}) = [h_1(\mathbf{x}), \ldots, h_B(\mathbf{x})]^T$ where each hash function $h_i(\cdot;\theta_i) : \mathcal{X} \rightarrow \{-1, 1\}$ is responsible for the generation of one bit and θ_i is its associated parameter vector.

In [22], a Boosting algorithm based on ECOCs is used for learning $h_1, \ldots, h_B$ achieving state-of-the-art performance. They utilize an ECOC matrix $\Lambda = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}^T$ for this purpose where each row or codeword $\mathbf{c}_i \in \{-1, 1\}^B$ corresponds to a label and each column embodies the bipartitioning on which h_i is trained. The base learner h_i is user-specified but ought to be simple enough for good generalization and fast training. However, the batch-learning takes a considerable amount of time even with simple learners and $\mathcal{Y}$ is assumed to be known *a priori*.

In contrast, we employ ECOCs in an online setting, in which the hash functions are updated sequentially with incoming data. Moreover, we assume no prior information on the label space $\mathcal{Y}$, the incoming instances can be associated with previously observed labels or not. The method has to accommodate newly arrived data with its possibly never-seen labels. This is an essential feature given the ever-growing sizes of datasets and the inclusions of initially unknown classes.

Given an instance $(\mathbf{x}^t, \mathbf{y}^t)$, we would like to update our mapping Φ. For simplicity, assume $\left|\mathbf{y}^t\right| = 1$, i.e, each data point is associated with a single label only. Let $\Lambda_{\mathbf{y}}$ denote the corresponding codeword for $\mathbf{y}^t$. $\Lambda_{yi} \in \{-1, 1\}$ then denotes the binary label of $\mathbf{y}^t$ in the i^{th} bipartitioning. Though any type of hash functions can be used, in this we consider hyperplanes of the following form

$$h(\mathbf{x};\theta) = \text{sgn}(\mathbf{w}^T\mathbf{x} - w_0), \tag{1}$$

where $\theta = [\mathbf{w}^T; w_0]$. Our goal is to solve the following problem:

$$\min_{\Theta=[\theta_1;\ldots;\theta_B]} \sum_{i=1}^{B} [\Lambda_{yi} \neq h_i(\mathbf{x}^t;\theta_i)]_{0\backslash 1}. \tag{2}$$

where Θ is the concetanation of parameter vectors θ_i. Replacing the $0\backslash 1$ loss with a convex function l such as the exponential loss and dropping the non-differentiable sgn in Eq. 1, the objective function becomes convex and we consider stochastic gradient descent (SGD) to minimize it. SGD has been successfully applied to large-scale learning problems as it provides huge memory savings and substantial computational time improvements.

As we will see, it is helpful to consider independently updating each hash function h_i for minimizing Eq. 2. Specifically, hashing h_i is updated according to the following rule:

$$\theta_i^{t+1} \leftarrow \theta_i^t - \eta^t \nabla_{\theta_i} l(h_i(\mathbf{x}^t;\theta_i^t), \Lambda_{yi}), \tag{3}$$

where the learning rate η^t is a positive real number. Although the choice of Λ is important, the error-correlation among individual hashings is crucial for the retrieval performance. Even if we find the Θ that minimizes the objective function, errors made by the hash functions will typically be correlated; therefore, reducing this correlation is crucial for the success of an ECOC based algorithm [23]. In Boosting this problem is tackled by reweighting the probability distribution associated with the training data, thus enabling the learner to focus on incorrectly mapped (i.e., misclassified) instances. For our online setting, we handle the error-correlation in a similar manner and take into account previous mappings when updating each hash function. Formally, h_i is updated as follows:

$$\theta_i^{t+1} \leftarrow \theta_i^t - \eta^t \nabla_{\theta_i} l(H_{i-1} + \ h_i(\mathbf{x}^t;\theta_i^t); \Lambda_{\mathbf{y}}), \tag{4}$$

input : Streaming data $\{(\mathbf{x}^t, \mathbf{y}^t)\}_{t=1}^T$, Codebook $\mathcal{C}$, η
Initialize $\Theta = [\theta_1, \ldots, \theta_B], k = 1$;
for $t \leftarrow 1$ **to** T **do**
  if $\mathbf{y}^t \nsubseteq \mathcal{Y}$ **then**
    for *each new label* $\overline{y} \in \mathbf{y}^t$ **do**
      $\mathcal{Y} \leftarrow \{\ \mathcal{Y},\ \overline{y}\}$;
      $\Lambda(k,:) \leftarrow$ random codeword c^* from $\mathcal{C}$ `// denote it as` $\Lambda_{\mathbf{y}}$;
      $\mathcal{C} \leftarrow \mathcal{C} \setminus c^*$;
      $k \leftarrow k + 1$;
    end
  end
  for $i \leftarrow 1$ **to** B **do**
    $\theta_i^{t+1} \leftarrow \theta_i^t - \eta^t \nabla_\theta l(H_{i-1} + \ h_i(\mathbf{x}^t;\theta_i^t); \Lambda_{\mathbf{y}})$
  end
end

Algorithm 1: Online learning for hashing

where $H_{i-1} = \sum_{k=1}^{i-1} \Lambda_{yk} h_k(\mathbf{x}^t;\theta_k^t)$. With this approach, we can handle the error-correlation problem in a way that is not possible when applying SGD on Θ directly. Eq. 4 inspired by [24], but our formulation differs from [24] in that we incorporate ECOCs in learning.

When a new label is observed, we assign a new codeword to it and proceed with the update as usual. The codeword can be generated on-the-fly, but to further reduce the computational overhead it is helpful to construct a sufficiently large set of codewords or a codebook, beforehand. The performance of the method also depends on this construction, e.g., the distance between the codewords must be large enough to ensure error-correction. In practice, randomly constructing the binary codebook performs better than using construction heuristics [25]. Therefore, in our implementation, we use random construction for codebook generation. Our online method for learning the hash functions is summarized in Alg. 1.

To populate the hash table after learning, we index a training point $\mathbf{x}$ by using either the codeword $\mathbf{c}$ corresponding to its label (if available) or the output $\Phi(\mathbf{x})$. If the data points to be indexed have label information it is more convenient to use the corresponding codewords as the binary codes of the points since it is shown to compensate a number of hash function errors during retrieval and thus provide improved performance [22]. Given a query $\mathbf{x}_q$ during testing, $\Phi(\mathbf{x}_q)$ is computed and the instances are retrieved via Hamming ranking between the binary codes. In the next section, we give experimental results based on this retrieval scheme.

3. EXPERIMENTS

We evaluate our approach on three widely used datasets; CIFAR-10, SUN397 and NUSWIDE. We compare our method against Locality Sensitive Hashing (LSH) [6], Binary Reconstructive Embedding (BRE) [12], Minimal Loss Hashing (MLH) [13], Supervised Hashing with Kernels [14], Fast Hashing (FastHast) [26], Supervised Hashing with Error Correcting Codes (ECC) [22] and Online Kernel Hashing (OKH) [21]. These methods have shown to outperform earlier hashing techniques such as [10, 9, 16]. We refer to our method as OECC in the following sections.

3.1. Evaluation Protocol

For all experiments we follow the protocol used in [14, 16, 21]. We consider the Hamming ranking in which instances are ranked based on Hamming distances to the query. This retrieval scheme has linear complexity but owing to the binary representations it is extremely fast in modern CPUs. We consider Mean Average Precision (mAP) scores for a set of queries evaluated at varying bit lengths. For further analysis we also report mAP values vs. CPU time for all the benchmarks. All experiments are repeated five times and the average is reported for all metrics. We set all algorithmic parameters of all the methods via cross validation. We choose performance over learning time when selecting the type of base learners in the hashing methods. Specifically, for ECC we use the linear SVM as the base learner for all our experiments. Similarly, we always select the best performing learner despite the possibility of being much slower for the FastHash technique. As for the loss function in OECC we utilize the exponential loss and select a constant step size for η^t.

All experiments were conducted on a workstation with 2.4 GHz Intel Xeon CPU and 512 GB RAM.

3.2. Datasets

CIFAR-10 The CIFAR-10 benchmark contains 60K samples from 10 different categories represented as 512-dimensional Gist descriptors. We randomly partition the dataset into two; a training and a test consisting of 59K and 1K samples (100 per class), respectively. 2K instances (20 per class) are sampled from the training set to learn the hash functions, and the remaining training data is used to populate the hash table.

SUN397 The SUN397 dataset contains over 100K samples from 397 categories represented with 512-dimensional Gist descriptors. We sample 10 instances from each class to construct our test set. We sample 7.9K instances (20 instances per class) from the remaining points to learn the hash functions while the rest of the training data is used to populate the hash table.

NUSWIDE This datasets contains over 270K instances with 81 ground truth concepts in which each point can be associated with multiple labels. We use the available 500-dimensional BoW descriptors [27] as the representation. Similarly, we partition the data into two parts; 269K and 1K samples for the training and test sets, respectively. We use 1.6K points (20 instances per concept) selected from the training set to learn the hash functions while the rest is again used to populate the hash table. Following [16], the precision metric is evaluated based on whether the retrieved points share at least one label with the query. Regarding the ECC method, it requires an update to a probability distribution over the data points based on previously incorrect mappings but it is not clear how to do so in a multilabelled dataset. For such reasons, we omit ECC in the results for NUSWIDE.

For the online methods OKH and OECC, LSH is used for initialization of the hashing parameters. The sets of samples used to learn the hash functions are selected randomly without any class consideration. Additionally, when reporting the mAP vs CPU time, the online learning is continued until 59K, 100K and 100K points are observed for the CIFAR-10, SUN397 and NUSWIDE datasets, respectively. For all datasets, we mean-center the data and do unit normalization. As stated, the training data used for populating the hash tables have associated label information; thus, in populating the hash tables for the CIFAR-10 and SUN 397 datasets we index each training point with the codeword corresponding to its label. For NUSWIDE, since multiple labels can correspond to a training point we simply use Φ as the index for initially populating the hash table.

4. RESULTS

Table 1 shows mAP values for CIFAR-10. We observe that ECC performs best for all length codes while our OECC technique is a close runner-up. More importantly, our method achieves these results with substantial time improvements. For example, it takes only 2.9 seconds to learn the hash function parameters compared to 355 seconds of ECC, while attaining comparable results. Another significant improvement is the memory footprint of the binary codes. Even with 4 bits, our method achieves comparable performance to state-of-the-art-techniques with 64-bits (excluding ECC). Similarly, Table 2 shows results for the SUN397 dataset. Here we observe that our OECC method either is the best performing method or the closest runner-up for all length codes. Again our method achieves these results orders of magnitude faster compared to other solutions. Table 3 shows results for the NUSWIDE dataset in which our technique achieves state-of-the-art performance when # of bits is > 12 with the fastest learning time excluding the baseline LSH.

For all benchmarks we observe state-of-the-art performance with substantial time and memory savings. Our OECC method either achieves top performance or is a close runner-up, while being orders-of-magnitude faster than competing methods and using more compact codes. Being online OKH also has the similar advantages with respect to computational efficiency, but only performs slightly better than LSH in terms of retrieval accuracy.

The graphs in Fig. 1 give a more detailed comparison between our method and the recent online hashing technique, OKH. We report the mAP value vs. CPU time for both algorithms. The test sets of datasets CIFAR-10 and SUN397 contain points sampled from all classes; therefore, for evaluation purposes, we do indexing after all possible codewords have been observed. This occurs early in the online process. Regarding the results, the performance of OECC surpasses nearly all other techniques in all three benchmarks within a fraction of their learning time. For OKH, although we observe a consistent minor improvement in performance at the early stages, it yields inferior results as the learning process continues. Compared to OKH the performance of OECC either improves as more training examples are observed or oscillates around a particular value. Oscillation may be due to the constant step size selection in Eq. 4. The selection of η is mostly application-specific and related to the knowledge of whether the instances are sampled from a stationary or non-stationary distribution, e.g, if the data points are believed to be sampled from a non-stationary distribution, a diminishing step size will not allow the hash functions to adapt to such a variation.

5. CONCLUSION

We proposed an online supervised hashing technique that is adaptable to continuing growth and diversification of datasets. Our OECC method does not assume any prior knowledge on the label space of the data. OECC achieves state-of-the-art performance in three image retrieval benchmarks and it is orders-of-magnitude faster than batch methods. Our method attains mean average precision (mAP) that is comparable to state-of-the-art, but using more compact codes. Our method significantly outperforms the previous online supervised hashing approach, while also being faster in its computation.

We believe the topic of designing hashing methods amenable to variations that occur in growing datasets will be a fruitful research area, giving direct benefit to many applications. One limitation of our current study is fixing the length of the binary codes. In addition, a theoretical analysis of our online framework is lacking. In future work we plan to analyze and investigate these issues.

Method	Mean Average Precision (Random 0.1)						Training time (seconds)
	4 bits	8 bits	12 bits	24 bits	32 bits	64 bits	24 bits
LSH [6]	0.11	0.12	0.12	0.13	0.13	0.14	0.1
BRE [12]	0.15	0.16	0.15	0.15	0.15	0.16	295.4
MLH [13]	0.14	0.16	0.16	0.15	0.16	0.15	280
SHK [14]	0.21	0.24	0.26	0.29	0.30	0.32	149.8
FastHash [26]	0.21	0.26	0.28	0.31	0.32	0.34	899
ECC [22]	**0.33**	**0.39**	**0.44**	**0.53**	**0.55**	**0.58**	355.3
OKH [21]	0.13	0.13	0.13	0.14	0.15	0.15	6.5
OECC	0.32	0.38	0.41	0.48	0.48	0.52	**2.9**

Table 1: Mean Average Precision for the CIFAR-10 dataset. For all methods, 2K points are used in learning the hash functions.**Bold** denotes the best performing method while red represents the best online technique. The training time includes time for learning and populating the hash table.

Method	Mean Average Precision $\times 10^{-1}$ (Random 0.02)						Training time (seconds)
	4 bits	8 bits	12 bits	24 bits	32 bits	64 bits	24 bits
LSH [6]	0.031	0.033	0.038	0.044	0.047	0.056	0.2
BRE [12]	0.046	0.051	0.056	0.058	0.061	0.077	146.4
MLH [13]	0.041	0.046	0.050	0.057	0.060	0.073	149.5
SHK [14]	0.050	0.059	0.064	0.068	0.068	0.065	2400
FastHash [26]	0.034	0.043	0.045	0.050	0.054	0.060	4424.6
ECC [22]	**0.061**	**0.098**	0.103	**0.144**	**0.145**	0.198	22741
OKH [21]	0.034	0.035	0.039	0.045	0.050	0.059	7.2
OECC	**0.061**	0.094	**0.110**	0.135	0.141	**0.201**	**2.9**

Table 2: Mean Average Precision for the SUN397 dataset. For all methods, 7.9K points are used in learning the hash functions. **Bold** denotes the best performing method while red represents the best online technique. The training time includes time for learning and populating the hash table.

Method	Mean Average Precision (Random $\sim$ 0.10)						Training time (seconds)
	4 bits	8 bits	12 bits	24 bits	32 bits	64 bits	24 bits
LSH [6]	0.111	0.112	0.113	0.116	0.115	0.122	0.5
BRE [12]	**0.130**	**0.126**	0.118	0.125	0.125	0.135	296
MLH [13]	0.119	0.125	0.123	0.126	**0.131**	0.126	304.1
SHK [14]	0.114	0.121	0.117	0.118	0.118	0.122	43.9
FastHash [26]	0.116	0.122	0.121	0.130	0.129	0.127	199.6
OKH [21]	0.111	0.113	0.114	0.118	0.118	0.120	8
OECC	0.119	0.122	**0.127**	**0.134**	**0.131**	**0.136**	**6.6**

Table 3: Mean Average Precision for the NUSWIDE dataset. For all methods, 1.6K points are used in learning the hash functions. **Bold** denotes the best performing method while red represents the best online technique. The training time includes time for learning and populating the hash table.

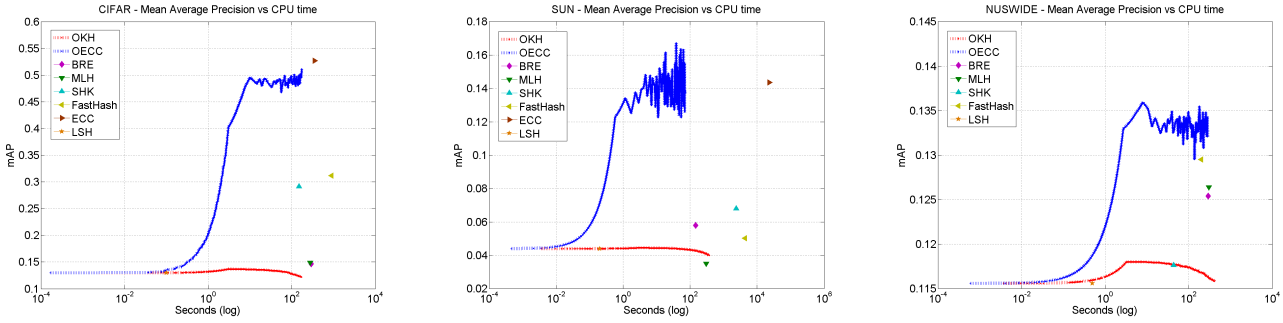

Fig. 1: Mean Average Precision with respect to CPU time for CIFAR-10 (left), SUN397 (middle) and NUSWIDE (right) datasets in which for OKH and OECC the online learning is continued for 59K, 100K and 100K points, respectively. For all other methods the dots represents the training time with 2K, 7.9K and 1.6K samples.

6. REFERENCES

[1] Qifan Wang, Bin Shen, Shumiao Wang, Liang Li, and Luo Si, "Binary codes embedding for fast image tagging with incomplete labels," in *Proc. European Conf. on Computer Vision (ECCV)*, 2014.

[2] Xi Li, Chunhua Shen, A. Dick, and A. van den Hengel, "Learning compact binary codes for visual tracking," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[3] Jian Cheng, Cong Leng, Jiaxiang Wu, Hainan Cui, and Hanqing Lu, "Fast and accurate image matching with cascade hashing for 3d reconstruction," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[4] Xiao Liu, Dacheng Tao, Mingli Song, Ying Ruan, Chun Chen, and Jiajun Bu, "Weakly supervised multiclass video segmentation," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[5] Thomas Dean, Mark A. Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, and Jay Yagnik, "Fast, accurate detection of 100,000 object classes on a single machine," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. of Symposium on Computational Geometry (SCG)*, 2004.

[7] Aristides Gionis, Piotr Indyk, and Rajeev Motwani, "Similarity search in high dimensions via hashing," in *Proc. International Conf. on Very Large Data Bases (VLDB)*, 1999.

[8] Brian Kulis and Kristen Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proc. IEEE International Conf. on Computer Vision (ICCV)*, 2009.

[9] Yair Weiss, Antonio Torralba, and Robert Fergus, "Spectral hashing," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2008.

[10] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu, "Self-taught hashing for fast similarity search," in *Proc. ACM SIGIR Conf. on Research & Development in Information Retrieval (SIGIR)*, 2010.

[11] Yunchao Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[12] Brian Kulis and Trevor Darrell, "Learning to hash with binary reconstructive embeddings," in *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2009.

[13] Mohammad Norouzi and David J. Fleet, "Minimal loss hashing for compact binary codes.," in *Proc. International Conf. on Machine Learning (ICML)*, 2011.

[14] Jun Wang Liu, Wei and, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang, "Supervised hashing with kernels.," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[15] G. Lin, C. Shen, D. Suter, and A. van den Hengel, "A general two-step approach to learning-based hashing," in *Proc. IEEE International Conf. on Computer Vision (ICCV)*, 2013.

[16] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang, "Semi-supervised hashing for large-scale search.," *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2012.

[17] Gustavo Carneiro, Antoni B. Chan, Pedro J. Moreno, and Nuno Vasconcelos, "Supervised learning of semantic classes for image annotation and retrieval," *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2007.

[18] Matthieu Guillaumin, Thomas Mensink, Jakob Verbeek, and Cordelia Schmid, "Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation," in *Proc. IEEE International Conf. on Computer Vision (ICCV)*, 2009.

[19] Ce Liu, J. Yuen, and A. Torralba, "Nonparametric scene parsing via label transfer," *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2011.

[20] Huanjing Yue, Xiaoyan Sun, Jingyu Yang, and Feng Wu, "Landmark image super-resolution by retrieving web images," *IEEE Trans. on Image Processing*, 2013.

[21] Long-Kai Huang, Qiang Yang 0010, and Wei-Shi Zheng, "Online hashing," in *Proc. International Joint Conf. on Artificial Intelligence (IJCAI)*, 2013.

[22] Fatih Cakir and Stan Sclaroff, "Supervised hashing with error correcting codes," in *Proc. ACM Conf. on Multimedia*, 2014.

[23] Venkatesan Guruswami and Amit Sahai, "Multiclass learning, boosting, and error-correcting codes," in *COLT*, 1999.

[24] B. Babenko, Ming-Hsuan Yang, and S. Belongie, "A family of online boosting algorithms," in *IEEE International Conf. on Computer Vision Workshops (ICCV Workshops)*, 2009.

[25] Ling Li, "Multiclass boosting with repartitioning," in *Proc. International Conf. on Machine Learning (ICML)*, 2006.

[26] Guosheng Lin, Chunhua Shen, Qinfeng Shi, A. van den Hengel, and D. Suter, "Fast supervised hashing with decision trees for high-dimensional data," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[27] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao. Zheng, "Nus-wide: A real-world web image database from national university of singapore," in *In Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, 2009.