Computer Vision and Image Understanding 000 (2016) 1–12

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

Online supervised hashing

Fatih Cakir*, Sarah Adel Bargal, Stan Sclaroff

Department of Computer Science, Boston University, Boston, MA 02215, United States

ARTICLE INFO

Article history:
Received 15 January 2016
Revised 19 September 2016
Accepted 19 October 2016
Available online xxx

Keywords:
Hashing
Fast similarity search
Approximate nearest neighbors
Retrieval

ABSTRACT

Fast nearest neighbor search is becoming more and more crucial given the advent of large-scale data in many computer vision applications. Hashing approaches provide both fast search mechanisms and compact index structures to address this critical need. In image retrieval problems where labeled training data is available, supervised hashing methods prevail over unsupervised methods. Most state-of-the-art supervised hashing approaches employ batch-learners. Unfortunately, batch-learning strategies may be inefficient when confronted with large datasets. Moreover, with batch-learners, it is unclear how to adapt the hash functions as the dataset continues to grow and new variations appear over time. To handle these issues, we propose OSH: an Online Supervised Hashing technique that is based on Error Correcting Output Codes. We consider a stochastic setting where the data arrives sequentially and 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, our 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. Also, our method significantly outperforms recently introduced online hashing solutions.

1. Introduction

Given a query, finding similar points in a corpus is a central problem in many computer vision applications. The ever-growing size of available data collections and the increasing use of high-dimensional representations in describing data, have increased the computational complexity for performing similarity search, urging researchers to develop search strategies that can be used to explore such collections in an efficient and effective manner.

Various similarity search techniques have been proposed to address these challenges. Such techniques include tree-based construction algorithms (Arya et al., 1998; Jagadish et al., 2005), which partition the search space so that only a subset of data points is considered for a query. Another group of techniques employ dimensionality reduction methods (Roweis and Saul, 2000; Tenenbaum et al., 2000), which map the data to a lower-dimensional space while preserving the neighborhood structure. A speedup is achieved in distance computations with the more compact, lower-dimensional representations.

However, these approaches do not scale well with higher-dimensional data representations and larger datasets. One promising family of approaches is based on hashing, in which the data is mapped to binary vectors in Hamming space. The binary vector representations permit fast search mechanisms with a very small memory footprint. Example applications that utilize hashing include: image annotation (Wang et al., 2014b), visual tracking (Li et al., 2013), 3D reconstruction (Cheng et al., 2014), video segmentation (Liu et al., 2014), object detection (Dean et al., 2013) and multimedia retrieval (Gao et al., 2015; Song et al., 2015; 2013).

Hashing methods can be broadly categorized as data-independent and data-dependent techniques. Data-independent methods (Datar et al., 2004; Gionis et al., 1999; Kulis and Grauman, 2009) give guarantees on the approximation to particular metrics, without regard to the dataset that is to be indexed. However, for certain application settings, distances are defined only on the available data set; thus, data-dependent solutions (Gong and Lazebnik, 2011; Kulis and Darrell, 2009; Lin et al., 2013; Liu et al., 2012; Wang et al., 2012; Weiss et al., 2008) are formulated to learn the hashings directly from data.

Data-dependent methods generally outperform data-independent solutions in retrieval tasks primarily due to the training phase where desirable properties such as compactness and informativeness of the hash mapping are imposed. Consequently, the resulting binary codes better capture the data-distribution. However, this training phase usually involves solving a complex optimization problem in which the optimum is gener-

* Corresponding author.
E-mail addresses: fcakir@bu.edu (F. Cakir), sbargal@bu.edu (S.A. Bargal), sclaroff@bu.edu (S. Sclaroff).

http://dx.doi.org/10.1016/j.cviu.2016.10.009
1077-3142/

ally found via batch learning. This batch learning usually has time complexity that scales as a quadratic function of the number of items in the dataset. As a result, it is very costly to re-run the batch learning with each update of the corpus, in order to adapt the hash mapping for evolving data distributions. A static corpus is rarely observed in practice; on the contrary, expansions and diversifications of the data are very common. Data points associated to previously observed or unobserved classes may arrive, necessitating an update in the hash mapping to accommodate to this non-stationarity. In such cases, it would be extremely costly to repeatedly do batch learning from scratch.

Many hashing studies conform to two important properties initially stated by Weiss et al. (2008): (1) the domain of the hash mapping should cover the entire input space and (2) compact codes should be adequate enough in representing the data. Given the above discussion, we suggest a third important property: (3) a hash mapping must be amenable to the variation of a dataset. In this work, we propose an online supervised method for learning hash codes that satisfies all three properties. We specifically consider the problem of retrieving semantically similar neighbors where the semantics is induced from label information. This problem is central in many vision tasks including, but not limited to, label-based image retrieval and annotation (Carneiro et al., 2007; Guillaumin et al., 2009), semantic segmentation (Liu et al., 2011), image super resolution (Yue et al., 2013), etc. Supervised hashing methods have shown to outperform unsupervised methods in semantic retrieval mainly due to leveraging available label information.

Our formulation is based on Error Correcting Output Codes (ECOCs). ECOCs have their origins in coding theory and have been successfully used to solve many computer vision problems (Jiang and Tu, 2009; Kittler et al., 2001; Schapire, 1997; Zhao and Xing, 2013). The general theme is to use a distributed representation for the output space. These representations are carefully selected so they partition the output space into distant target regions. Errors made in the system (e.g., *channel* or *classifiers*) then can be recovered to a certain extent. In the hashing context, an ECOC formulation has several advantages. It allows one to be more specific regarding the *range* of the hash mapping. Prior work usually enforce properties to the hash mapping Φ through binary constraints resulting in complex (NP-hard) optimization problems. Instead, we directly construct the elements of the range (set) as desired and then do minimization. ECOCs also enables compensation for a number of hash function errors during retrieval when the range of Φ is carefully constructed. Finally, it provides a constant time hash-lookup complexity during retrieval.

We consider a stochastic setting in which data items sequentially arrive and the hash mapping is updated accordingly. The data items may be associated with previously unobserved labels; thus, we assume that the number of labels is not known *a priori*. In experimental evaluation, our proposed method yields accuracy that is at least comparable to (sometimes even better than) state-of-the-art batch solutions but is orders-of-magnitude faster in learning the hash mapping. Most importantly, our method is adaptable to data variations. This is critical for diversifying and expanding datasets (please observe Fig. 1). We also significantly outperform two competing recent online hashing methods (Cakir and Sclaroff, 2015; Huang Long-Kai and Wei-Shi, 2013).

In summary, our contributions are twofold:

1. We introduce an adaptive supervised hashing technique. It is orders-of-magnitude faster than state-of-the-art batch methods, while providing comparable or better accuracy. Also, compared to recently proposed online techniques, our method shows significant improvements.
2. Our learning formulation does not require any prior assumptions on the label space and is well-suited for expanding datasets that have new label inclusions. Our learning algorithm has linear time complexity with respect to number of items in the dataset. To the best of our knowledge, it is the first supervised hashing technique that allows the label space to grow.

The remainder of the paper is organized as follows. Section 2 briefly surveys related work. Section 3 gives the formulation of our methodology. Section 4 provides experiments followed by concluding remarks in Section 5.

2. Related work

In this section, we briefly provide a review of related hashing techniques.

2.1. Hashing

Many hashing methods have been introduced over the years. Notable earlier examples include Locality Sensitive Hashing methods (Datar et al., 2004; Gionis et al., 1999; Kulis and Grauman, 2009) where metric functions such as the Euclidean, Jaccard and Cosine distances are approximated. These methods usually have theoretical guarantees on the approximation quality and conform with sub-linear retrieval mechanisms. However, they are confined to certain metrics as they ignore the data distribution and/or related meta-data.

Contrary to earlier methods, recent approaches are data-dependent such that hash functions are directly learned from the data. These methods can be considered as binary embeddings that map the data into the Hamming space while preserving a specific neighborhood structure. Such a neighborhood is induced from the meta-data (e.g., labels) or is completely determined by the user (e.g., via similarity-dissimilarity indicators of data pairs). With the new binary representations, distance computations can be efficiently carried out allowing even a linear search to be done very efficiently for large-scale data. These data-dependent methods can be grouped as follows: rank preserving methods (Norouzi and Fleet, 2011; Shakhnarovich et al., 2003), similarity alignment techniques (Kulis and Darrell, 2009; Lin et al., 2014; Liu et al., 2012; Wang et al., 2012), quantization/PCA based methods (Gong and Lazebnik, 2011; He et al., 2013; Jegou et al., 2011), spectral and graph based solutions (Ge et al., 2014; Strecha et al., 2012; Weiss et al., 2008; Zhang et al., 2010), and very recently, deep-learning methods (Lai et al., 2015; Lin et al., 2015; Xia et al., 2014). We now review a few of the prominent methods. For a more comprehensive survey we refer readers to Wang et al. (2014a).

Among similarity alignment solutions, Minimal Loss Hashing (MLH) (Norouzi and Fleet, 2011) considers minimizing a hinge-type loss function motivated from structural SVMs. In Binary Reconstructive Embeddings (BRE), (Kulis and Darrell, 2009), a kernel-based solution is proposed where the goal is to construct hash functions by minimizing an empirical loss between the input and Hamming space distances via a coordinate descent type algorithm. Supervised Hashing with Kernels (SHK) (Liu et al., 2012) is similar to BRE such that a kernel-based solution is proposed; but, instead of preserving the equivalence of the input and Hamming space distances, the kernel function weights are learned by minimizing an objective function based on the binary code inner products.

Another notable line of work includes quantization/PCA based techniques. Among these, Semi-Supervised Hashing (Wang et al., 2012) learns the hash functions by maximizing the empirical accuracy on labeled data and also the entropy of the generated hash functions on any unlabeled data. This is shown to be very similar to doing a PCA analysis where the hash functions are the eigenvectors of the biased covariance matrix (biased due to the supervised

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

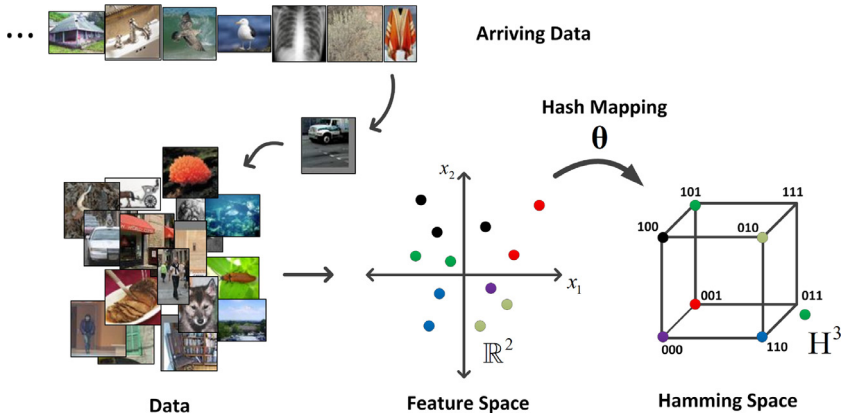


Fig. 1. A toy example with a hash mapping $\Phi : \mathbb{R}^2 \rightarrow \mathcal{H}^3$. Supervised learning of this mapping is usually done in batch mode in state-the-art methods. If the dataset grows and diversifies, the hash mapping that is computed in batch mode become outdated and must be recomputed by re-running the batch optimization from scratch. The batch learning usually has time complexity that scales as a quadratic function of the number of items in the dataset. As a result, it is very costly to adapt the hash mapping for evolving data distributions. We introduce an online technique for learning the hash mapping that is easily amenable to such variations and expansions of the dataset.

information). Other noteworthy work includes PCA inspired methods where the principal components are taken as the hash functions. If "groups" exist within the data (e.g. suitable for clustering) then further refining the principal components for better binarization has shown to be beneficial, e.g., as in Iterative Quantization (Gong and Lazebnik, 2011).

Spectral Hashing (Weiss et al., 2008) and Self-Taught Hashing (Zhang et al., 2010) are notable spectral hashing techniques where the similarity of the instances is preserved as binary codes by solving a graph Laplacian problem.

Deep learning based approaches have recently gained prominence (Lai et al., 2015; Lin et al., 2015; Xia et al., 2014). Among deep learning based hashing methods, one notable example is Lai et al. (2015) where the hash mapping and images features are jointly learned with a triplet loss formulation. This triplet loss ensures that an images is more similar to the second image than to the third one with respect to their binary codes.

Although these recent data-dependent solutions generally perform better than their data-independent counterparts, the learning phase takes a considerable amount of time as the time complexity usually grows quadratically with respect to dataset size and involves solving a complex integer programming problem. Consequently, many of these solutions sample only a subset of the training data to learn the hash mapping. Also these solutions are batch methods; therefore, given any new variation in the dataset the mapping must be re-learned from scratch. However, expansions and variations in datasets are common in practice. As a dataset grows, it seems natural that new classes and data items will appear. A hash mapping should be adaptive to such diversification.

A noteworthy study with similar observations tackles this issue by proposing a strategy to update the hash functions in a selective manner with the arrival of new data (Yang et al., 2013). At each step, a set of hash functions is selected by considering the hash functions' consistency or contribution in preserving the similarities of the input data. However, this strategy is not online in that previously seen examples are stored and learning is done in batch mode.

2.2. Online hashing

Online hashing methods have recently been introduced that are adaptive to data variations by updating the hash mapping in an online manner with streaming input.

Huang Long-Kai and Wei-Shi (2013) introduced Online Kernel Hashing, in which an online passive-aggressive algorithm is used to update the hash functions. The data items sequentially arrive in pairs with a supervision indicator denoting the similarity of the items. The hash functions to be updated are selected based on a Hamming loss that reflects the number of bit flips that are needed in the binary codes. An update is then performed via gradient descent on the selected hashing parameters.

Cakir and Sclaroff (2015) also consider a similar framework. They argue that it is difficult to assess which hash function to update in an online setting as it is the collective effort of all the hash functions that yields good retrieval performance. A squared error loss function is minimized, but, due to the discrepancy between the retrieval accuracy and the objective function, another criterion is used to infer which hash functions to update.

Leng et al. (2015) propose approximating the properties of the incoming data in a "sketch" matrix. The incoming data arrives in small batches and a data-sketch matrix maintains the properties of this data while offering significant memory savings. A PCA-based method on the data-sketch is then used to derive the hash functions. One crucial limitation of this method is that it discards any label information in its formulation.

Although (Cakir and Sclaroff, 2015; Huang Long-Kai and Wei-Shi, 2013) leverage label information, the data items arrive in pairs with a similarity indicator denoting the similarity of the items, i.e., whether or not the pair share the same class. We argue that capturing the semantic neighborhood via a simple indicator is difficult, and, instead, we directly utilize any available label information. With the usage of ECOCs this allows us to assign target codes in the Hamming space to labels directly and to learn the hash mapping accordingly, *i.e.*, we explicitly structure the range of the hash mapping providing significant improvements in retrieval performance.

3. Online supervised hashing

In this section, we first provide the problem setting and define the ECOC framework. Afterwards, we formulate our online supervised hashing technique.

3.1. Framework

Assume the joint space $\mathcal{Z} \triangleq \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ denote the input and label spaces, respectively. The goal of hashing is to learn a mapping $\Phi : \mathcal{X} \rightarrow \mathcal{H}^b$ such that a neighborhood structure is preserved in the b-dimensional Hamming space $\mathcal{H}$. This neighborhood is usually obtained from a particular metric associated with $\mathcal{X}$, label information defined on $\mathcal{Y}$ or can be derived jointly from $\mathcal{Z}$.

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

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 $h_i(\cdot; \mathbf{w}_i) : \mathcal{X} \rightarrow \{-1, 1\}$ is responsible for the generation of one bit and $\mathbf{w}_i$ is its associated parameter vector.

Our formulation is based on Error Correcting Output Codes (ECOCs). An ECOC based formulation has several advantages. First, it allows being specific regarding the range of Φ. Prior work usually enforces desirable properties such as compactness and informativeness on the range (set) without explicitly specifying its elements. Such properties are generally imposed through integer constraints in the objective function leading to a high-complexity optimization problem. Instead, we consider specific elements in the range of Φ as target codes, thereby enabling an easy imposition of any desirable property for the mapping. Also, the resulting optimization procedure is less demanding without the complex constraints. Secondly, the target codes can directly be used as hash bins when constructing an index. Since the number of target codes is much smaller than the co-domain of Φ, this allows a constant $\mathcal{O}(|\mathcal{T}|)$ retrieval complexity where $\mathcal{T}$ is the set of target codes. Lastly, if the target codes are selected to have ample bit differences, a number of hash function errors can be compensated for during retrieval, thus providing further robustness.

A hashing method also based on ECOCs has been considered in Cakir and Sclaroff (2014) achieving state-of-the-art performance. However, the method is batch -making it intractable to re-learn the hash mapping for each data variation. Also, it is not clear how the method works for multi-label datasets and most importantly the label space $\mathcal{Y}$ 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.

3.2. Methodology

We consider the hash functions to be hyperplanes of the following form:

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

where $\mathbf{w}$ and $\mathbf{x}$ are given in homogeneous form. We assume a stochastic environment in which data items $(\mathbf{x}, \mathbf{y}) \in \mathcal{Z}$ arrives sequentially. Although our method is applicable to multi-label datasets, for mathematical brevity, assume $|\mathbf{y}| = 1$, *i.e.*, each data point is associated only with a single label. Each label $\mathbf{y}$ is assigned a specific target code in the b-dimensional Hamming space $\mathcal{H}^b$. Let $\mathbf{c_y} \in \mathcal{T}$ denote the target code for label y. Intuitively, we should find a mapping Φ such that $d_h(\Phi(\mathbf{x}), \mathbf{c_y})$ is minimized, where $d_h = \sum_{i=1}^{b} [\![\mathbf{c_y} \neq h_i(\mathbf{x}; \mathbf{w}_i)]\!]$ is the Hamming loss. The objective function then can be formulated as:

$$\begin{aligned} J(\Phi) &= \int_{\mathcal{Z}} d_h(\Phi(\mathbf{x}), \mathbf{c}_y) dP(\mathbf{z}) \\ &= \int_{\mathcal{Z}} \sum_{i=1}^{b} \underbrace{[\![\mathbf{c}_{yi} \neq h_i(\mathbf{x}; \mathbf{w}_i)]\!]}_{\leq l(\mathbf{c}_{yi} \mathbf{w}_i^T \mathbf{x})} dP(\mathbf{z}). \end{aligned} \tag{2}$$

Replacing the 0\1 loss in d_h with a margin-based convex upper-bound l such as the exponential loss, the objective function becomes convex. Let us denote this new objective function as $\widetilde{J}(\Phi)$. We consider stochastic gradient descent (SGD) to minimize it:

$$\Phi^{t+1} \leftarrow \Phi^t - \eta^t \nabla_\Phi \widetilde{J}(\Phi^t) \tag{3}$$

where $\Phi = [\mathbf{w}_1, \cdots, \mathbf{w}_b]$ and the learning rate η^t is a positive real number. SGD has been successfully applied to large-scale learning problems as it provides huge memory savings and substantial computational time improvements. In our case, it also enables an online leaning approach for supervised learning of the hash mapping Φ.

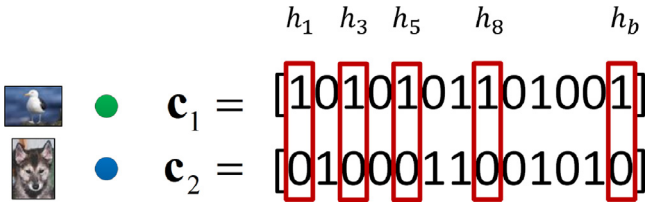


Fig. 2. Assume two class labels: *green* and *blue*. The two target codes $\mathbf{c}_1$ and $\mathbf{c}_2$ are assigned to these classes, respectively. If the codes are lengthy then identical bi-partitions can occur, possibly resulting in highly-correlated hash functions for these bits. In this example, when the first class is assigned value 1, and the second class is assigned value 0, we get the identical bi-partitions marked using red boxes. The corresponding hashings h_1, h_3, h_5, h_8 and h_b will be correlated if initialized similarly. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Notice that the objective can be written as $\widetilde{J}(\Phi) = \mathbb{E}_{\mathcal{Z}}[\sum_{i=1}^{T} l(\mathbf{c}_{yi} \mathbf{w}_i^T \mathbf{x})] = \sum_{i=1}^{T} \mathbb{E}_{\mathcal{Z}}[l(\mathbf{c}_{yi} \mathbf{w}_i^T \mathbf{x})]$; thus, Eq. (3) can be considered as finding dichotomizers of a linear form, each parametrized by $\mathbf{w}$, which minimizes the sum of expected margin-based losses. Since the expected losses are in sum form, this is equivalent to independently minimizing each hash function h_i. Specifically, we can update hashing h_i according to the following rule:

$$\mathbf{w}_i^{t+1} \leftarrow \mathbf{w}_i^t - \eta^t \nabla_{\mathbf{w}} l(h_i(\mathbf{x}; \mathbf{w}_i^t), \mathbf{c}_{yi}). \tag{4}$$

The choice of $\mathcal{T}$ is important. The target codes $\mathbf{c}$ should be distant enough in $\mathcal{H}^b$ to increase robustness. The error-correlation among individual hash functions is also crucial for the retrieval performance. Even if we find Φ that minimizes the objective function, errors made by the hash functions may be correlated. Each bit position in a target code $\mathbf{c}$ denotes a binary label for the data points; thus, identical bi-partitions may be produced giving rise to highly-correlated dichotomizers as illustrated in Fig. 2. Therefore, reducing this correlation is crucial for the success of an ECOC based algorithm (Guruswami and Sahai, 1999). In Boosting this problem is tackled by re-weighting 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:

$$\mathbf{w_i^{t+1}} \leftarrow \mathbf{w}_i^t - \eta^t \nabla_{\mathbf{w}_i} l(H_{i-1} + h_i(\mathbf{x}; \mathbf{w}_i^t); \mathbf{c}_y), \tag{5}$$

where $H_{i-1} = \sum_{i=1}^{i-1} \mathbf{c}_{yk} h_k(\mathbf{x}^t; \mathbf{w}_k^t)$. With this approach, we can handle the error-correlation problem in a way that is not possible when applying SGD on Φ directly. Eq. (4) is inspired by Babenko et al. (2009), but our formulation differs from Babenko et al. (2009) in that we incorporate ECOCs in learning.

When a new label is observed, we assign a new target code to it and proceed with the update as usual. The target code can be generated on-the-fly, but to further reduce the computational overhead it is helpful to construct a sufficiently large set of codes or a codebook $\mathcal{C}$, beforehand. The performance of the method also depends on this codebook's construction, *e.g.*, the distance between the target codes must be large enough to ensure error-correction. In practice, randomly constructing the binary codebook performs better than using construction heuristics (Li, 2006). Therefore, in our implementation, we use random construction for code generation. We summarize our Online Supervised Hashing (OSH) learning algorithm in Algorithm 1.

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

Table 1
Difference between the two hash mappings for indexing data. $\mathcal{T}$ is the set containing the target codes $\mathbf{c}$. N is the number of hashed data items. b is the dimensionality of the Hamming space $\mathcal{H}$. In practice, typically $|\mathcal{T}| \ll N \ll 2^b$. When target codes $\mathbf{c}$ is used to index data items, it is enough to compare the target codes to the hashed query $\Phi(\mathbf{x}_q)$. This results in an $\mathcal{O}(|\mathcal{T}|)$ time retrieval complexity. On the other hand, when the output of Φ is used to populate the index, Hamming ranking on the hashed data items can be done having linear time complexity ($\mathcal{O}(N)$).

Hash using	# possible hash bins	Retrieval Complexity	Hash unlabeled data
Target codes $\mathbf{c}_y$	$\lvert\mathcal{T}\rvert$	Constant $\mathcal{O}(\lvert\mathcal{T}\rvert)$	No
Mapping Φ	2^b	Linear $\mathcal{O}(N)$	Yes

input : Streaming data $\{(\mathbf{x}^t, \mathbf{y}^t)\}_{t=1}^{T}$, Codebook $\mathcal{C}$, $\tilde{\eta}$,
Procedure `find`$(\mathcal{T}, \mathbf{y})$ to obtain the target code(s) of $\mathbf{y}$ from target set $\mathcal{T}$,
Initialize $\Phi = [w_1, \ldots, w_b], k = 1$;

for $t \leftarrow 1$ **to** T **do**
  if $\mathbf{y}^t \not\subset \mathcal{Y}$ **then**
    for *each new label* $\bar{y} \in \mathbf{y}^t$ **do**
      $\mathcal{Y} \leftarrow \{\tilde{\mathcal{Y}}, \tilde{\bar{y}}\}$;
      $\mathcal{T} \leftarrow$ random codeword c^* from $\tilde{\mathcal{C}}$ // ;
      $\mathcal{C} \leftarrow \tilde{\mathcal{C}} \backslash \tilde{c}^*$;
      $k \leftarrow k + 1$;
    end
  end
  for ***each*** y *in* $\mathbf{y}^t$ **do**
    $\mathbf{c}_y \leftarrow$ `find`(Λ, y);
    for $i \leftarrow 1$ **to** b **do**
      $\mathbf{w}_i^{t+1} \leftarrow \mathbf{w}_i^t - \eta^t \nabla_{\mathbf{w}} l(H_{i-1} + \tilde{h}_i(\mathbf{x}^t; \mathbf{w}_i^t); c_{\mathbf{y}})$
    end
  end
end

Algorithm 1: OSH: Online supervised hashing.

3.3. Populating the index table and retrieval

In this work, the use of ECOCs enables two different ways to index a data item. A data item $\mathbf{x}$ can be indexed by using either the target code $\mathbf{c}$ corresponding to its label y (if available) or the output of the hash mapping $\Phi(\mathbf{x})$.

OSH-$\mathcal{T}$. If the data instance to be indexed has label information, it is beneficial to use the corresponding target code $\mathbf{c}$ as its hash bin when populating the hash table with instances. This indexing approach has been shown to compensate for a number of hash function errors during retrieval, and thus, provides improved performance (Cakir and Sclaroff, 2014) when labels are available. The hash bins correspond with labels and since, in practice, the number of hash bins $|\mathcal{T}|$ is much smaller than the number of indexed data items, the hash structure will typically be dense. Therefore, to retrieve similar items for a query $\mathbf{x}_q$ computing a Hamming rank for all the data items in the index is unnecessary. Instead, it is enough to consider the bin with Hamming distance closest to $\Phi(\mathbf{x}_q)$. This retrieval procedure has $\mathcal{O}(|\mathcal{T}|)$ time complexity. Rather than ranking all items, one would typically wants to rank data items in a specific bin of the hash table. Thus, after the closest hash bin is identified, the items located in that bin can be ranked according to similarity to the query, based on their $\Phi(\mathbf{x})$.

OSH-Φ. The hash mapping Φ can also be used as bins, *i.e.*, to hash the data item $\mathbf{x}$, the output $\Phi(x)$ can be used as an index. This may be done when label information is not provided. The co-domain of Φ is the b-dimensional Hamming space; thus, the resulting index structure built with Φ will typically be sparse with many empty bins and some items will be associated with unique indices (if b is not so small). Consequently, given a query, $\Phi(x)$ can be computed and hashed items can be retrieved via Hamming ranking. This retrieval has $\mathcal{O}(N)$ time complexity where N is the number of indexed items, but owing to fast distance computations in the Hamming space, it is extremely fast.

In practice, it is convenient to use both schemes. In OSH-$\mathcal{T}$, the main index structure is constructed from the target codes $\mathbf{c}$, and the items in specific hash bins are accessed through a secondary hash table that is populated with Φ. Given a query, the closest bin in the main index can be located via Hamming decoding, and the items in that bin can be Hamming ranked using $\Phi(\mathbf{x})$. In OSH-Φ, the main index structure is populated using Φ computed for each data instance. The closest bins in the index can be retrieved in sequence of proximity to the query $\Phi(\mathbf{x})$, until the desired number of items is obtained and ranked. Table 1 summarizes the differences between these two approaches.

To make our work comparable with competing methods, we assign each item a binary code that is its hash bin index, irrespective of the hashing scheme. Given a query, we retrieve all items via Hamming ranking to compute the mean Average Precision (mAP), as explained in the next section.

4. Experiments

We evaluate our approach on four widely used datasets: CIFAR-10, SUN397, NUSWIDE and PLACES205. We compare our method against Locality Sensitive Hashing (LSH) (Datar et al., 2004), Binary Reconstructive Embedding (BRE) (Kulis and Darrell, 2009), Minimal Loss Hashing (MLH) (Norouzi and Fleet, 2011), Supervised Hashing with Kernels (Liu et al., 2012), Fast Hashing (FastHash) (Lin et al., 2014), Supervised Hashing with Error Correcting Codes (ECC) (Cakir and Sclaroff, 2014), Online Kernel Hashing (OKH) (Huang Long-Kai and Wei-Shi, 2013), Adaptive Hashing (AdaptHash) (Cakir and Sclaroff, 2015) and deep learning based hashing methods (Lai et al., 2015; Lin et al., 2015; Xia et al., 2014). These methods have shown to outperform earlier hashing techniques such as Weiss et al. (2008), Wang et al. (2012), Zhang et al. (2010), and Gong and Lazebnik (2011). We refer to our method as OSH in the following sections: **OSH-$\mathcal{T}$** when the target codes c are used as the hash bins, and **OSH-Φ** when Φ is used to populate the hash table.

4.1. Evaluation protocol

For all experiments we follow the protocol used in Liu et al. (2012), Wang et al. (2012), and Huang Long-Kai and Wei-Shi (2013). 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 and/or mAP values vs. CPU time analysis. Algorithmic parameters are set via cross-validation on a small vali-

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

Table 2
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 (batch or online) while underline denotes the best online method. The training time (in seconds) includes time for learning and populating the hash table. Results shown in the CNN section of the table use the 4096-dimensional CNN features, as described in the text. For the CNN comparison, results for the online methods and the best performing batch method (ECC) are reported.

Method	Mean Average Precision *(Random 0.1)*						Training time *(sec)*
	4 bits	8 bits	12 bits	24 bits	32 bits	64 bits	24 bits
Batch							
LSH (Datar et al., 2004)	0.11	0.12	0.12	0.13	0.13	0.14	0.1
BRE (Kulis and Darrell, 2009)	0.15	0.16	0.15	0.15	0.15	0.16	295
MLH (Norouzi and Fleet, 2011)	0.14	0.16	0.16	0.15	0.16	0.15	280
SHK (Liu et al., 2012)	0.21	0.24	0.26	0.29	0.30	0.32	149
FastHash (Lin et al., 2014)	0.21	0.26	0.28	0.31	0.32	0.34	899
ECC-Φ (Cakir and Sclaroff, 2014)	0.15	0.18	0.20	0.24	0.25	0.27	355
ECC-$\mathcal{T}$ (Cakir and Sclaroff, 2014)	**0.33**	**0.39**	**0.44**	**0.53**	**0.55**	**0.58**	355
Online							
OKH (Huang Long-Kai and Wei-Shi, 2013)	0.13	0.13	0.13	0.14	0.15	0.15	8.3
AdaptHash (Cakir and Sclaroff, 2015)	0.10	0.10	0.12	0.13	0.14	0.14	6.2
SketchHash (Leng et al., 2015)	0.14	0.15	0.15	0.16	0.16	0.17	1.3
OSH-Φ	0.15	0.19	0.20	0.21	0.21	0.22	3.2
OSH-$\mathcal{T}$	0.32	0.38	0.41	0.48	0.48	0.52	**2.9**
CNN							
ECC-Φ	0.31	0.43	0.55	0.64	0.66	0.68	1.6K
ECC-$\mathcal{T}$	0.57	0.71	0.74	0.80	0.81	0.83	1.6K
OKH	0.14	0.15	0.18	0.21	0.21	0.26	31
AdaptHash	0.10	0.15	0.15	0.19	0.22	0.26	7.0
SketchHash	0.23	0.24	0.24	0.24	0.26	0.27	6.5
OSH-Φ	0.30	0.33	0.38	0.45	0.45	0.47	13
OSH-$\mathcal{T}$	0.45	0.69	0.71	0.72	0.75	0.76	11

dation set sampled from each dataset (specifically, 4K data points are used). 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 OSH we utilize the exponential loss and select a constant step size for $\eta^t = \{0.2, 0.5, 0.1, 0.1\}$ for CIFAR-10, SUN397, NUSWIDE and PLACES205, respectively.

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

4.2. Datasets and features

In this section we introduce the benchmark image datasets used in our experimental evaluation, and the descriptors used to represent the images in these datasets. We consider four benchmark 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 3.7K instances (10 instances per class) from the remaining instances to learn the hash functions while the rest of the training data is used to populate the hash table.

NUSWIDE: This dataset contains over 270K samples. Each sample can be associated with multiple labels, corresponding with 81 ground truth concepts. We use the 500-dimensional BoW descriptors provided with the dataset (Chua et al., 2009) as the feature representation. As in Chua et al. (2009), we partition the data into two parts: 269K and 1K samples for the training and test sets, respectively. We use 2K samples selected at random from the training set to learn the hash functions, while the remaining data is used to populate the hash table. Following (Wang et al., 2012), the precision metric is evaluated based on whether the retrieved instances share at least one label with the query.

PLACES205: For large-scale experiments we use the Places dataset. This dataset is a 2.5 million image subset of the recently introduced Places benchmark (Zhou et al., 2014). Images in this dataset belong to one of 205 scene categories. A test set of 4.1K points is constructed by sampling 20 images per class. The rest of the dataset constitutes the training data used to populate the hash table. A random subset of 100K images from this training data is used to learn the hash functions. Nearly all batch methods were not trainable for this benchmark; thus, we report results for online hashing solutions only. We use CNN features as image representations for PLACES205. These CNN features are pre-computed from the *fc7* layer of an AlexNet (Krizhevsky et al., 2012) trained on the ImageNet dataset (Deng et al., 2009), and are reduced to 128 dimensions by PCA. The network is *not* fine-tuned on this target dataset.

Similarly to PLACES205, we also use deep feature representations of the images in CIFAR-10, SUN397 and NUSWIDE. The features are extracted from the fully connected layer *fc7* of the 16-layer VGG16 (Simonyan and Zisserman, 2015) Convolutional Neural Network (CNN) pre-trained on the ImageNet dataset (Deng et al., 2009). The features are extracted from a network that is also not fine-tuned for these experimental datasets. Each image is repre-

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

Table 3
Mean Average Precision for the SUN397 dataset. For all methods, 3.7K points are used in learning the hash functions. **Bold** denotes the best performing method (batch or online) while underline denotes the best online method. The training time (in seconds) includes time for learning and populating the hash table. Results shown in the CNN section of the table use the 4096-dimensional CNN features, as described in the text. For the CNN comparison, results for the online methods and the best performing batch method (ECC) are reported.

Method	Mean Average Precision $\times 10^{-1}$ *(Random 0.02)*						Training time *(sec)*
	4 bits	8 bits	12 bits	24 bits	32 bits	64 bits	24 bits
Batch							
LSH (Datar et al., 2004)	0.031	0.033	0.038	0.044	0.047	0.056	0.2
BRE (Kulis and Darrell, 2009)	0.046	0.051	0.056	0.058	0.061	0.077	146
MLH (Norouzi and Fleet, 2011)	0.041	0.046	0.050	0.057	0.060	0.073	149
SHK (Liu et al., 2012)	0.050	0.059	0.064	0.068	0.068	0.065	2.4K
FastHash (Lin et al., 2014)	0.034	0.043	0.045	0.050	0.054	0.060	4.4K
ECC-Φ (Cakir and Sclaroff, 2014)	0.02	0.04	0.04	0.06	0.07	0.08	22K
ECC-$\mathcal{T}$ (Cakir and Sclaroff, 2014)	**0.061**	**0.098**	0.103	**0.144**	**0.145**	0.198	22K
Online							
OKH (Huang Long-Kai and Wei-Shi, 2013)	0.034	0.035	0.039	0.045	0.050	0.059	32
AdaptHash (Cakir and Sclaroff, 2015)	0.02	0.026	0.040	0.044	0.047	0.057	10
SketchHash (Leng et al., 2015)	0.046	0.052	0.057	0.063	0.067	0.076	2.3
OSH-Φ	0.033	0.039	0.042	0.057	0.062	0.078	3.1
OSH-$\mathcal{T}$	**0.061**	0.094	**0.110**	0.135	0.141	**0.201**	**3.2**
CNN							
ECC-Φ	0.03	0.06	0.09	0.158	0.19	0.29	16K
ECC-$\mathcal{T}$	0.07	0.14	0.152	0.238	0.324	0.575	14K
OKH	0.027	0.028	0.029	0.033	0.034	0.051	33
AdaptHash	0.026	0.029	0.031	0.033	0.036	0.053	11
SketchHash	0.027	0.029	0.035	0.041	0.049	0.072	13
OSH-Φ	0.041	0.072	0.071	0.133	0.162	0.250	19
OSH-$\mathcal{T}$	0.079	0.133	0.181	0.298	0.402	0.557	21

Table 4
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 (batch or online) while underline denotes the best online method. The training time (in seconds) includes time for learning and populating the hash table. Results shown in the CNN section of the table use the 4096-dimensional CNN features, as described in the text. For the CNN comparison, results for the online methods and the best performing batch method (FastHash) are reported. Note: ECC and OSH-$\mathcal{T}$ are not applicable because NUSWIDE is a multi-label dataset.

Method	Mean Average Precision (*Random* $\sim$ 0.21)						Training time *(sec)*
	4 bits	8 bits	12 bits	24 bits	32 bits	64 bits	24 bits
Batch							
LSH (Datar et al., 2004)	0.215	0.220	0.225	0.237	0.232	0.268	0.5
BRE (Kulis and Darrell, 2009)	0.255	0.261	0.265	0.274	0.277	0.286	288
MLH (Norouzi and Fleet, 2011)	0.243	0.259	0.264	0.267	0.271	0.279	310
SHK (Liu et al., 2012)	0.248	0.265	0.263	0.281	0.287	0.29	68
FastHash (Lin et al., 2014)	**0.28**	**0.291**	**0.293**	**0.296**	**0.302**	**0.304**	201
Online							
OKH (Huang Long-Kai and Wei-Shi, 2013)	0.219	0.22	0.231	0.232	0.236	0.24	13
AdaptHash (Cakir and Sclaroff, 2015)	0.215	0.216	0.232	0.234	0.236	0.237	10
SketchHash (Leng et al., 2015)	0.01	0.01	0.01	0.01	0.01	0.01	1
OSH-Φ	0.261	0.266	0.274	0.281	0.283	0.293	**4.6**
CNN							
FastHash	0.477	0.533	0.549	0.564	0.568	0.577	6.8K
OKH	0.224	0.245	0.248	0.256	0.279	0.304	30
AdaptHash	0.225	0.239	0.248	0.266	0.288	0.314	14
SketchHash	0.01	0.01	0.01	0.01	0.01	0.01	5
OSH-Φ	0.414	0.438	0.465	0.489	0.504	0.51	17

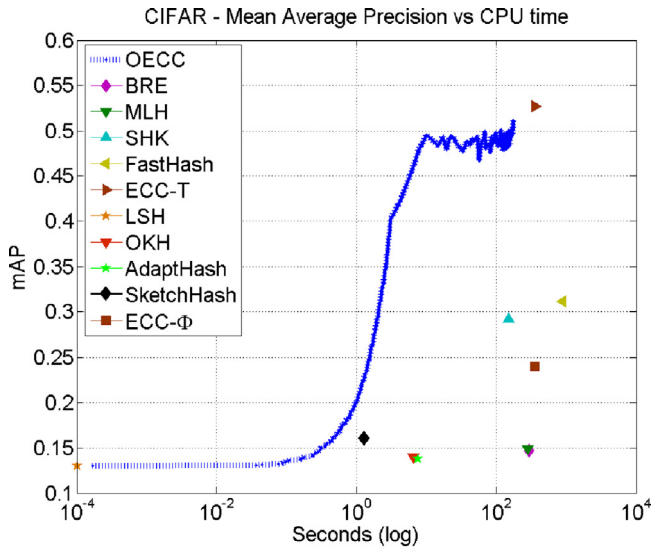
sented by a 4096-dimensional feature vector. These features are then used in place of the Gist and BoWs features in experiments that test OSH, the other online hashing methods, and the best performing batch method for each benchmark dataset using the deep features as input.

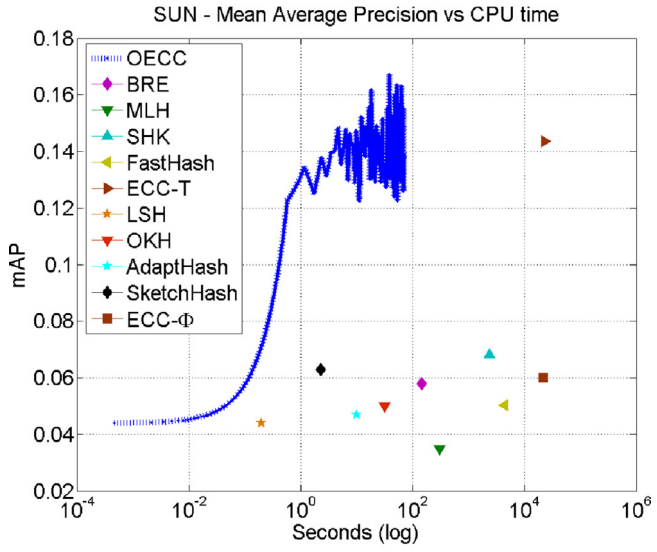
4.2.1. Initialization and populating the hash table

For the online methods OKH, AdaptHash and OSH, 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. In addition, when reporting the mAP vs. CPU time, the online learning is continued until 59K, 100K, 100K and 100K samples are observed for the CIFAR-10, SUN397, NUSWIDE and PLACES205 datasets, respectively. Some methods mean-center and unit-normalize the data; thus, to put all methods on equal footing, we also mean-center the data and do unit normalization as preprocessing.

As stated in Section 3.3, two different schemes are possible to populate a hash table. In OSH-$\mathcal{T}$, each data item $\mathbf{x}$ is indexed using the target code $\mathbf{c}$ corresponding to its label (if label information is available). In OSH-Φ, each data item is indexed using the output of the mapping $\Phi(\mathbf{x})$. We use an analogous annotation of ECC-$\mathcal{T}$ and ECC-Φ for ECC. For CIFAR-10, SUN397 and PLACES205 datasets, we can evaluate and compare performance of OSH and ECC using both schemes. For NUSWIDE, since multiple labels can be associated with a data item, we simply use OSH-Φ as the index when

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009





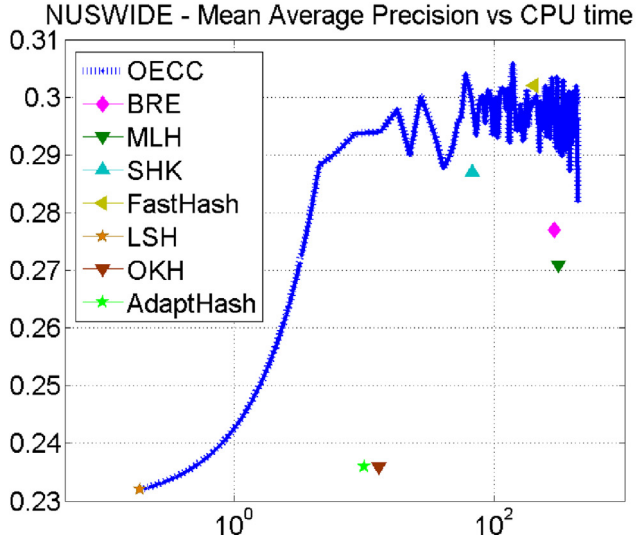


Fig. 3. Mean Average Precision with respect to CPU time for CIFAR-10 (top), SUN397 (middle) and NUSWIDE (bottom) datasets in which for OSH the online learning is continued for 59K, 100K and 100K points, respectively. For all other methods the dots represents the training time with 2K, 3.7K and 2K samples. Note that SketchHash is not included in final figure.

populating the hash table. Note that ECC (both $-\Phi$ and $-\mathcal{T}$) is not applicable on multi-label datasets.

4.3. Results

In this section we present the results of our experiments. We compare the performance of OSH, our proposed online hashing method, to state-of-the-art online and batch methods. We then analyze the precision/time tradeoff of our method, and compare it against other online and batch methods. Finally, we examine queries and their corresponding retrieved images using our two indexing schemes: OSH-Φ and OSH-$\mathcal{T}$.

4.3.1. Performance comparisons

We present Mean Average Precision (mAP) performance and time performance comparisons between our online hashing method and other state-of-the-art online and batch hashing methods. This is done for the three benchmark datasets: CIFAR-10, SUN397, and NUSWIDE.

Table 2 reports the mAP values for CIFAR-10. The top section of the table reports results for the batch methods. We observe that ECC-$\mathcal{T}$ performs best among the batch methods for all length codes. The middle section of Table 2 reports results for the online methods. We can see that our method, OSH attains best performance among the online methods. More importantly, our method achieves these results with substantial time improvements. For example, it takes only **2.9** s to learn the hash function parameters compared to **355** s of the best performing batch method ECC-$\mathcal{T}$, 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-$\mathcal{T}$).

ECC-$\mathcal{T}$ performs better than ECC-Φ and other competing methods because ECC-$\mathcal{T}$ leverages class labels while populating the hash table, i.e., when indexing a data item, if label information is available, the corresponding ECOC is used. This allows compensation for hash function errors during retrieval. All other competing work (excluding OSH) cannot leverage class labels to assign hash codes while populating the hash table, since there is no unique code for a particular label. Our OSH technique also can leverage label information while populating the hash table, if that label information is available (corresponding to OSH-$\mathcal{T}$ in the results). Note that, even when label information is not used in populating the hash table (*i.e.*, OSH-Φ) superior results are achieved vs. competing online hashing methods.

We also evaluated performance when CNN features are used to represent the images instead of Gist. These results are reported in the bottom section of Table 2. Here we observe that using CNN features results in a significant boost in the mAP performance. We compare the best performing batch method (ECC-$\mathcal{T}$) against all online methods. Our method, OSH, remains the best online hashing solution while attaining mAP that is competitive with the best performing batch method for this dataset, ECC-$\mathcal{T}$. The training time increases when using CNN features due to the higher dimensionality of the feature representation, as expected. However, our method again has learning times that are a fraction of the best batch method: **1.6K** vs. **11** s.

Table 3 reports results for the SUN397 dataset. From the batch and online sections of the table, we observe that our method, OSH, is the best online hashing method. It is a strong competitor for the batch method ECC-$\mathcal{T}$, sometimes even exceeding its performance. Again our method achieves these results orders-of-magnitude faster compared to other solutions. When CNN features are used, we again observe a significant boost in performance. We compare performance of the best-performing batch method vs. all

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

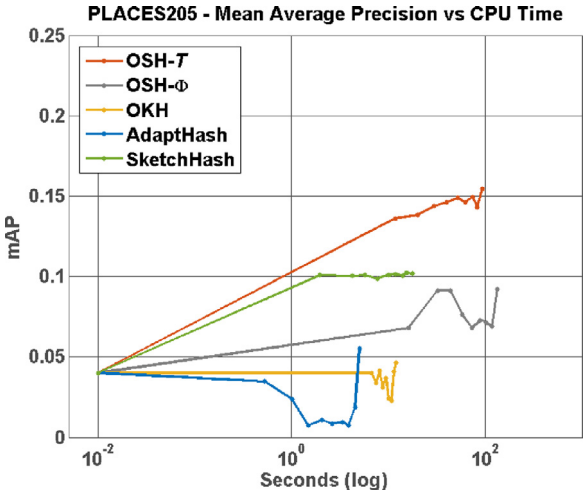


Fig. 4. Mean Average Precision with respect to CPU time for PLACES205 datasets in which online learning is continued for 100K points. Random performance value for mAP is ∼ 0.004.

of the online methods. Using CNN features, our OSH technique either achieves comparable results with ECC-$\mathcal{T}$, the best competing batch method or exceeds it. Again these results are achieved orders-of-magnitude faster: **14K** vs. **21** s, for the ECC-$\mathcal{T}$ vs. our OSH method, respectively.

Table 4 reports results for the NUSWIDE dataset. For this benchmark, our OSH technique surpasses all methods, batch or online, for all bit lengths, excluding FastHash. When using CNN features, we compare the best batch method against all online methods. OSH remains the best online hashing solution and a close runner-up to the best batch method. Training time increases due to the higher dimensionality of feature representation, as expected. However, our method again has learning times that are a fraction of the best method; **0.57** mAP in **6.8K** seconds for FastHash-CNN compared to **0.51** mAP in **17** s. We observed that SketchHash performs significantly worse, even with lengthier codes or with CNN descriptors. We believe this might be due to NUSWIDE being a multi-label dataset and that SketchHash's unsupervised formulation is incapable of capturing the semantics of a multi-labeled dataset adequately. Note that ECC and OSH-$\mathcal{T}$ are not applicable for NUSWIDE since it is a multi-label dataset.

For all benchmarks we observe state-of-the-art performance with substantial time and memory savings. Our OSH 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 and AdaptHash also offer similar advantages with respect to computational efficiency, but only perform slightly better than LSH in terms of retrieval accuracy.

4.3.2. mAP vs. tme tradeoff

In this section we study more closely the tradeoff between mean Average Precision (mAP) and CPU time.

The graphs in Fig. 3 report the mAP value vs. CPU time for the online and batch hashing methods, for the three benchmark datasets. The test sets of the CIFAR-10 and SUN397 datasets contain instances sampled from all classes; therefore, for evaluation purposes, we do indexing after all possible labels have been observed. This occurs early in the online process.

As can be observed in the graphs of Fig. 3, the mAP performance of OSH surpasses nearly all other techniques in all three benchmarks within a fraction of their learning time. The performance of OSH either improves as more training examples are observed or oscillates around a particular value. Oscillation may be

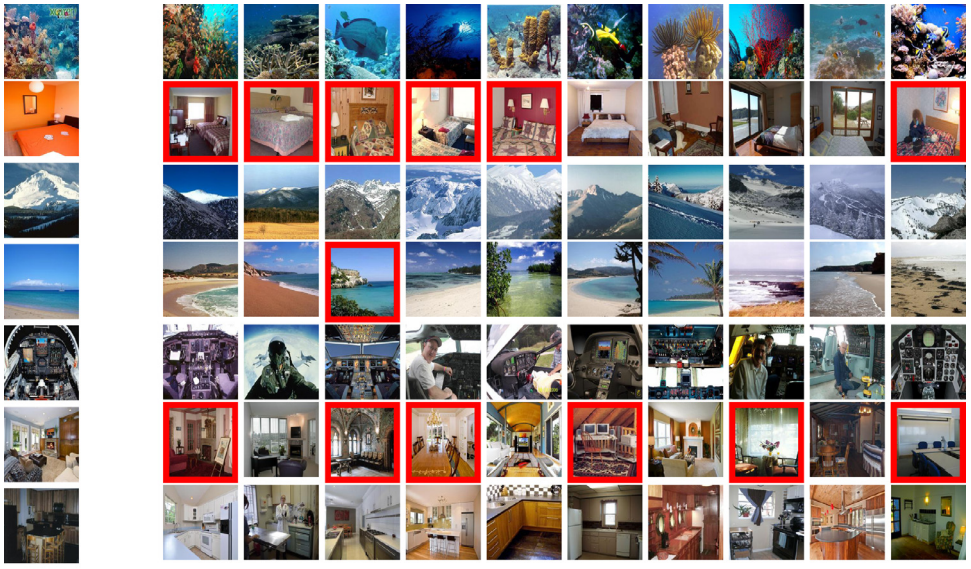
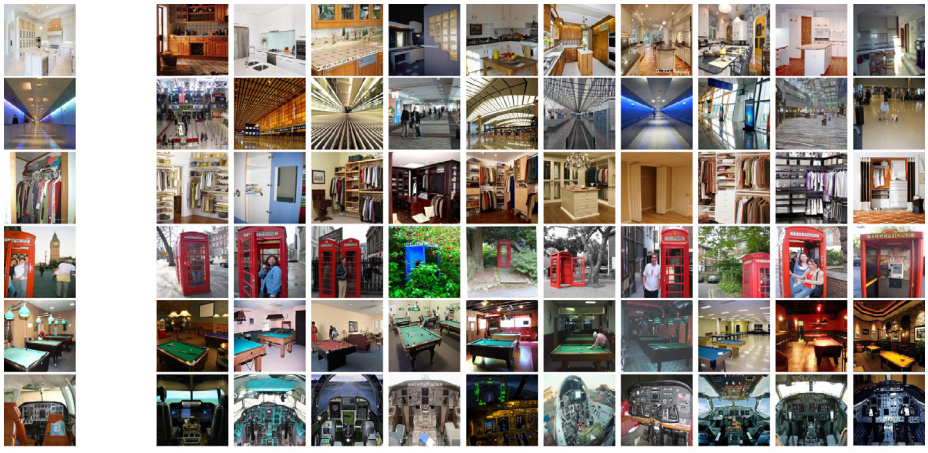
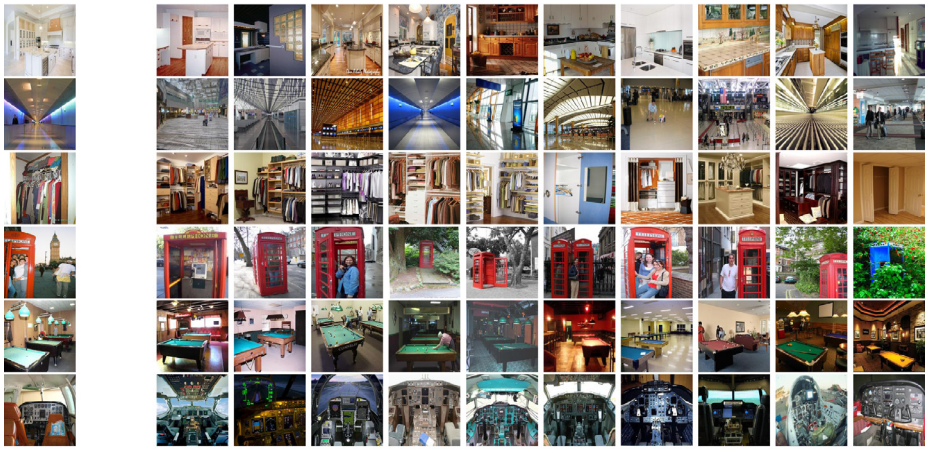


Fig. 5. Retrieval when Φ is used to populate the hash table: OSH-Φ. Seven sample test images of different classes for the **SUN397** dataset are shown. (a) shows the 7 query images: *UnderwaterCoralReef, Bedroom, SnowyMountain, Beach, Cockpit, LivingRoom, Kitchen*, respectively. (b) shows 10 retrieved images for each class that possess highest similarity to the query image; going from left to write similarity with query image decreases. Retrieved images that are marked in red belong to a different class than the query image. *Bedroom* query retrievals include 6/10 images labeled *HotelRoom. Beach* query retrievals include 1/10 images labeled *coast. LivingRoom* query retrievals include 2/10 images labeled *Parlor*, 1/10 image labeled *DiningRoom*, 1/10 labeled *HomeOffice*, and 1/10 labeled *ConferenceRoom*. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009



Query (a) Retrieval when hash bins correspond to target codes: all retrievals in the bin have equal similarity to the query.



Query (b) The images in the hash bins are re-ranked according to Φ, ordered left to right in descending similarity.

Fig. 6. Retrieval when the target codes from the set $\mathcal{T}$ are used as the hash bins for the **SUN397** dataset: OSH-$\mathcal{T}$. (a) The images residing unordered in the hash bins; (b) the top 10 images retrieved using OSH-$\mathcal{T}$ are now ranked according to Φ. The images belong to these classes: *Kitchen, AirportTerminal, Closet, PhoneBooth, PoolroomEstablishment, Cockpit*, respectively. 10 retrieved images for each query that possess highest similarity to the query image. Note that all images retrieved belong to the correct class of the query image.

due to the constant step size selection in Eq. (5).[1] 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. For example, 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.

4.3.3. Experiments with the Places205 dataset

Having established our methods superiority in previous benchmarks we now conduct further tests on a much larger dataset: PLACES205. This dataset has 2.5 million images over 205 categories. We use 100K training points to learn our hash mapping and plot the retrieval performance in terms of mean Average Precision (mAP) over CPU Time. It is impractical to train most batch methods on such a large dataset, due to the space complexity of these batch methods. For this reason, we compare against online solutions only.

In Fig. 4 we observe OSH-$\mathcal{T}$ is the best online solution while SketchHash is the runner-up. Though SketchHash converges early in the learning process, our method's performance increases as more data is seen. Learning time for SketchHash is less compared to our solution as it processes 100K instances faster; however, at any CPU time value our method outperforms SketchHash in retrieval performance. OSH-Φ also performs well nearing, 0.1 mAP value at 100K points, albeit it falls short against OSH-$\mathcal{T}$ and SketchHash. Finally, OKH and AdaptHash perform poorly.

4.3.4. Retrieval results

We now examine images retrieved for particular image queries, so that we may gain some insight into where and why retrieval errors may occur. We do this for OSH-Φ and OSH-$\mathcal{T}$.

In Fig. 5, we present example retrieval results for OSH-Φ for several image queries from the SUN397 dataset. The top 10 retrievals of seven query images from seven distinct categories are presented. Most of the retrieved images belong to the same class

[1] Also, the log scale used for the x-axis is another reason. We used log-scale to visually shrink the large discrepancy of CPU times between the methods.

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

as the query image. Interestingly, many of the retrieved images that do not belong to the same class appear visually similar, and can be from semantically related classes. Examples include: an image retrieved from class *Coast* for a query of class *Beach*, and images retrieved from the class *HotelRoom* to the class *Bedroom*.

In Fig. 6, we present retrieval results for our method OSH-$\mathcal{T}$. Fig. 6(a) shows the top 10 retrieved images for some example query images in the SUN397 dataset. Since the index is populated using OSH-$\mathcal{T}$, all retrieved images have the same similarity score and so they are presented in random rank-order in the figure.

In order to further improve the ranking of the OSH-$\mathcal{T}$ retrievals, we can rank them according to Φ. Note that ranking the OSH-$\mathcal{T}$ retrievals using Φ does not change the mAP of OSH-$\mathcal{T}$ as all retrievals are from the same class.

Fig. 6(b), shows the same 10 retrieved images for each query, but this time re-ranked according to Φ. While it is subjective to eyeball the image similarities, some interesting observations emerge: In row 1 of Fig. 6(b) the first image has the most similar colors and orientation compared to the query image. In row 2 of Fig. 6(b) the least similar image is a closeup on people rather than depicting an airport terminal space as in the query image and other retrievals that are ranked higher. In row 3 of Fig. 6(b), filled open closets appear to have higher similarity than empty or closed ones. In row 4 of Fig. 6(b), the blue phone booth is being ranked least similar compared to all the red ones. In row 5 of Fig. 6(b), with the exception of one image, green pool tables are retrieved with higher similarity that others. Finally, in row 6 of Fig. 6(b) all frontal view cockpits are retrieved with higher similarity than the final two, which are not frontal views.

4.4. Comparison against deep learning hashing methods

Deep learning based hashing methods have been introduced recently in which features specific to a target domain and the hash mapping are simultaneously learned (Lai et al., 2015; Lin et al., 2015; Xia et al., 2014). We compare our technique against such recently introduced methods. We use deep features extracted from the fully connected layer *fc7* of the 16-layer VGG16 (Simonyan and Zisserman, 2015) CNN pre-trained on the ImageNet dataset (Deng et al., 2009). The features we use are extracted from a network that is *not fine-tuned* for the experimental datasets listed in this work. The length of the binary codes is selected to be 48 to match experiments of these works. [33] is based on the AlexNet architecture with an introduced latent layer between *fc7* and *fc8*, [34] is based on the AlexNet architecture, and [39] use their own architecture of eight stacked convolution layers, which feed into divide-and-encode modules to divide intermediate image features into multiple branches, with each branch corresponding to one hash bit. In all these networks, training occurs on the target task. In contrast, we compute CNN features from a VGG architecture pre-trained on ImageNet and do not fine-tune using images from our dataset or for our target task. As a result of the different architectures and fine-tuning vs. not fine-tuning, these methods are not directly comparable with experiments presented in Tables 2–4. However, these results are discussed further here.

In the experiments with the CIFAR-10 dataset, we obtain mAP values of **0.77** and **0.795** when 5K and 50K images are used, respectively. Compare this to **0.58** (**32%** improvement) and **0.52** (**48%** improvement) over Lai et al. (2015) and Xia et al. (2014), respectively. The performance increases to **0.795** when trained with 50K images. Our method does not demonstrate better performance than Lin et al. (2015) which reports a 0.89 mAP value; but, our technique is orders-of-magnitude faster than these deep learning solutions, achieving these performance values with a hash mapping learned in 50 and 500 seconds for 5K and 50K points, respectively. Whereas these methods require hours of GPU training.

5. Conclusion

We proposed an online supervised hashing technique that is adaptable to continuing growth and diversification of datasets. Our OSH method does not assume any prior knowledge on the label space of the data. OSH achieves state-of-the-art performance on 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 previous online hashing approaches, while also being occasionally faster in its computation.

Online hashing methods are important due to their ability to adapt to variations in datasets as they grow and diversify. Equally important, online hashing methods offer superior time complexity vs. batch methods for learning with respect to dataset input size. Nevertheless, such schemes have their own challenges. With frequent updates in hash functions, the index must also be frequently kept up-to-date. This may cause inefficiencies in the system; therefore, solutions must be developed to alleviate this particular problem. Also, it could be advantageous if ECOC assignment were to account for compound or hierarchical structure of the label space. These are topics for future work.

References

Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y., 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. of the ACM.
Babenko, B., Yang, M.-H., Belongie, S., 2009. A family of online boosting algorithms. In: IEEE International Conf. on Computer Vision Workshops (ICCV Workshops).
Cakir, F., Sclaroff, S., 2014. Supervised hashing with error correcting codes. In: Proc. ACM Conf. on Multimedia.
Cakir, F., Sclaroff, S., 2015. Adaptive hashing for fast similarity search. In: IEEE International Conf. on Computer Vision (ICCV). IEEE.
Carneiro, G., Chan, A.B., Moreno, P.J., Vasconcelos, N., 2007. Supervised learning of semantic classes for image annotation and retrieval. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI).
Cheng, J., Leng, C., Wu, J., Cui, H., Lu, H., 2014. Fast and accurate image matching with cascade hashing for 3d reconstruction. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z., Zheng, Y.-T., 2009. 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).
Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S., 2004. Locality-sensitive hashing scheme based on p-stable distributions. In: Proc. of Symposium on Computational Geometry (SCG).
Dean, T., Ruzon, M.A., Segal, M., Shlens, J., Vijayanarasimhan, S., Yagnik, J., 2013. Fast, accurate detection of 100,000 object classes on a single machine. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., 2009. Imagenet: a large-scale hierarchical image database. CVPR.
Gao, L., Song, J., Zou, F., Zhang, D., Shao, J., 2015. Scalable multimedia retrieval by deep learning hashing with relative similarity learning. In: Proceedings of the 23rd ACM International Conference on Multimedia.
Ge, T. He, K., Sun, J., 2014. Graph cuts for supervised binary coding.
Gionis, A., Indyk, P., Motwani, R., 1999. Similarity search in high dimensions via hashing. In: Proc. International Conf. on Very Large Data Bases (VLDB).
Gong, Y., Lazebnik, S., 2011. Iterative quantization: a procrustean approach to learning binary codes. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Guillaumin, M., Mensink, T., Verbeek, J., Schmid, C., 2009. Tagprop: discriminative metric learning in nearest neighbor models for image auto-annotation. In: Proc. IEEE International Conf. on Computer Vision (ICCV).
Guruswami, V., Sahai, A., 1999. Multiclass learning, boosting, and error-correcting codes. COLT.
He, K., Wen, F., Sun, J., 2013. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Huang Long-Kai, Q.Y., Wei-Shi, Z., 2013. Online hashing. In: Proc. International Joint Conf. on Artificial Intelligence (IJCAI).
Jagadish, H.V., Ooi, B.C., Tan, K.-L., Yu, C., Zhang, R., 2005. iDistance: an adaptive b+-tree based indexing method for nearest neighbor search. ACM Trans. on Database Systems.
Jegou, H., Douze, M., Schmid, C., 2011. Product quantization for nearest neighbor search. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), 33.
Jiang, J., Tu, Z., 2009. Efficient scale space auto-context for image segmentation and labeling. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009

Kittler, J., Ghaderi, R., Windeatt, T., Matas, J., 2001. Face verification using error correcting output codes. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105.
Kulis, B., Darrell, T., 2009. Learning to hash with binary reconstructive embeddings. In: Proc. Advances in Neural Information Processing Systems (NIPS).
Kulis, B., Grauman, K., 2009. Kernelized locality-sensitive hashing for scalable image search. In: Proc. IEEE International Conf. on Computer Vision (ICCV).
Lai, H., Pan, Y., Liu, Y., Yan, S., 2015. Simultaneous feature learning and hash coding with deep neural networks. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Leng, C., Wu, J., Cheng, J., Bai, X., Lu, H., 2015. Online sketching hashing. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Li, L., 2006. Multiclass boosting with repartitioning. In: Proc. International Conf. on Machine Learning (ICML).
Li, X., Shen, C., Dick, A., van den Hengel, A., 2013. Learning compact binary codes for visual tracking. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Lin, G., Shen, C., Shi, Q., van den Hengel, A., Suter, D., 2014. Fast supervised hashing with decision trees for high-dimensional data. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Lin, G., Shen, C., Suter, D., van den Hengel, A., 2013. A general two-step approach to learning-based hashing. In: Proc. IEEE International Conf. on Computer Vision (ICCV).
Lin, K., Yang, H.-F., Hsiao, J.-H., Chen, C.-S., 2015. Deep learning of binary hash codes for fast image retrieval. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops.
Liu, C., Yuen, J., Torralba, A., 2011. Nonparametric scene parsing via label transfer. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI).
Liu, J.W., Wei, Ji, R., Jiang, Y.-G., Chang, S.-F., 2012. Supervised hashing with kernels.. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Liu, X., Tao, D., Song, M., Ruan, Y., Chen, C., Bu, J., 2014. Weakly supervised multiclass video segmentation. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Norouzi, M., Fleet, D.J., 2011. Minimal loss hashing for compact binary codes.. In: Proc. International Conf. on Machine Learning (ICML).
Roweis, S.T., Saul, L.K., 2000. Nonlinear dimensionality reduction by locally linear embedding. Science.
Schapire, R.E., 1997. Using output codes to boost multiclass learning problems. In: Proc. International Conf. on Machine Learning (ICML).
Shakhnarovich, G., Viola, P., Darrell, T., 2003. Fast pose estimation with parameter sensitive hashing. In: Proc. IEEE International Conf. on Computer Vision (ICCV).
Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition. ICLR.
Song, J., Gao, L., Yan, Y., Zhang, D., Sebe, N., 2015. Supervised hashing with pseudo labels for scalable multimedia retrieval. In: Proceedings of the 23rd ACM International Conference on Multimedia.
Song, J., Yang, Y., Huang, Z., Shen, H.T., Luo, J., 2013. Effective multiple feature hashing for large-scale near-duplicate video retrieval. IEEE Transactions on Multimedia.
Strecha, C., Bronstein, A.M., Bronstein, M.M., Fua, P., 2012. Ldahash: improved matching with smaller descriptors. IEEE Trans. Pattern Anal. Mach. Intell. (PAMI) 34 (1), 66–78.
Tenenbaum, J.B., Silva, V., Langford, J.C., 2000. A global geometric framework for nonlinear dimensionality reduction. Science.
Wang, J., Kumar, S., Chang, S.-F., 2012. Semi-supervised hashing for large-scale search.. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI).
Wang, J., Shen, H. T., Song, J., Ji, J., 2014a. Hashing for similarity search: a survey. CoRR abs/1408.2927.
Wang, Q., Shen, B., Wang, S., Li, L., Si, L., 2014. Binary codes embedding for fast image tagging with incomplete labels. In: Proc. European Conf. on Computer Vision (ECCV).
Weiss, Y., Torralba, A., Fergus, R., 2008. Spectral hashing. In: Proc. Advances in Neural Information Processing Systems (NIPS).
Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S., 2014. Supervised hashing for image retrieval via image resentation learning.. In: Conference on Artificial Intelligence (AAAI).
Yang, Q., Huang, L.-K., Zheng, W.-S., Ling, Y., 2013. Smart hashing update for fast response.. In: Proc. International Joint Conf. on Artificial Intelligence (IJCAI).
Yue, H., Sun, X., Yang, J., Wu, F., 2013. Landmark image super-resolution by retrieving web images. IEEE Trans. on Image Processing.
Zhang, D., Wang, J., Cai, D., Lu, J., 2010. Self-taught hashing for fast similarity search. In: Proc. ACM SIGIR Conf. on Research & Development in Information Retrieval (SIGIR).
Zhao, B., Xing, E.P., 2013. Sparse output coding for large-scale visual recognition. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).
Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., Oliva, A., 2014. Learning deep features for scene recognition using places database. In: Advances in Neural Information Processing Systems, pp. 487–495.

Please cite this article as: F. Cakir et al., Online supervised hashing, Computer Vision and Image Understanding (2016), http://dx.doi.org/10.1016/j.cviu.2016.10.009