Adaptive Hashing for Fast Similarity Search

Fatih Cakir Stan Sclaroff
Department of Computer Science
Boston University, Boston, MA
{fcakir,sclaroff}@cs.bu.edu

Abstract

With the staggering growth in image and video datasets, algorithms that provide fast similarity search and compact storage are crucial. Hashing methods that map the data into Hamming space have shown promise; however, many of these methods employ a batch-learning strategy in which the computational cost and memory requirements may become intractable and infeasible with larger and larger datasets. To overcome these challenges, we propose an online learning algorithm based on stochastic gradient descent in which the hash functions are updated iteratively with streaming data. In experiments with three image retrieval benchmarks, our online algorithm attains retrieval accuracy that is comparable to competing state-of-the-art batch-learning solutions, while our formulation is orders of magnitude faster and being online it is adaptable to the variations of the data. Moreover, our formulation yields improved retrieval performance over a recently reported online hashing technique, Online Kernel Hashing.

1. Introduction

While the current ease of collecting images and videos has made huge repositories available for the computer vision community, it has also challenged researchers to address the problems associated with such large-scale data. At the core of these problems lies the challenge of fast similarity search and the issue of storing these massive collections. One promising strategy for expediting similarity search involves mapping the data items into Hamming space via a set of hash functions where linear search is fast and often sublinear solutions perform well. Furthermore, the resulting binary representations allow compact indexing structures with a very low memory footprint.

Relatively early work in this domain include the data-independent Locality Sensitive Hashing and its variants [3, 5, 13] in which the retrieved nearest neighbors of a query are ensured to be within a scale of the true neighbors. Many subsequent studies have utilized training data to learn the hash functions. These data-dependent methods can be categorized as unsupervised, semi-supervised and supervised techniques. Ignoring any label information, unsupervised studies [23, 24, 7] are often used to preserve a metric-induced distance in the Hamming space. When the goal is the retrieval of semantically similar neighbors, supervised methods [12, 19, 18, 17] have shown to outperform these unsupervised techniques. Finally, semi-supervised methods like [22] leverage label information while constraining the solution space with the use of unlabeled data.

Although data-dependent solutions tend to yield higher accuracy in retrieval tasks than their data-independent counterparts, the computational cost of the learning phase is a critical issue when large datasets are considered. Large scale data are the norm for hashing applications, and as datasets continue to grow and include variations that were not originally present, hash functions must also accommodate this deviation. However, this may necessitate retraining from scratch for many data-dependent solutions employing a batch learning strategy. To overcome these challenges, we propose an online learning algorithm in which the hash functions are updated swiftly in an iterative manner with streaming data. The proposed formulation employs stochastic gradient descent (SGD). SGD algorithms provide huge memory savings and can provide substantial performance improvements in large-scale learning. The properties of SGD have been extensively studied [1, 14] and SGD has been successfully applied to a wide variety of applications including but not limited to; tracking [10], recognition [15], learning [21] etc.

Despite its feasibility and practicality, it is not straightforward to apply SGD for hashing. Please observe the setup in Fig. 1. Considering semantic based retrieval, an appropriate goal would be to assign the same binary vector to instances sharing the same label. The two hash functions of form $f = \text{sgn}(\mathbf{w}^T\mathbf{x})$ are ample to yield perfect empirical Mean Average Precision (mAP) scores based on Hamming rankings for the synthetic dataset in Fig. 1. However, in an online learning framework where pairs of points are available at each iteration, f_2 will produce identical mappings

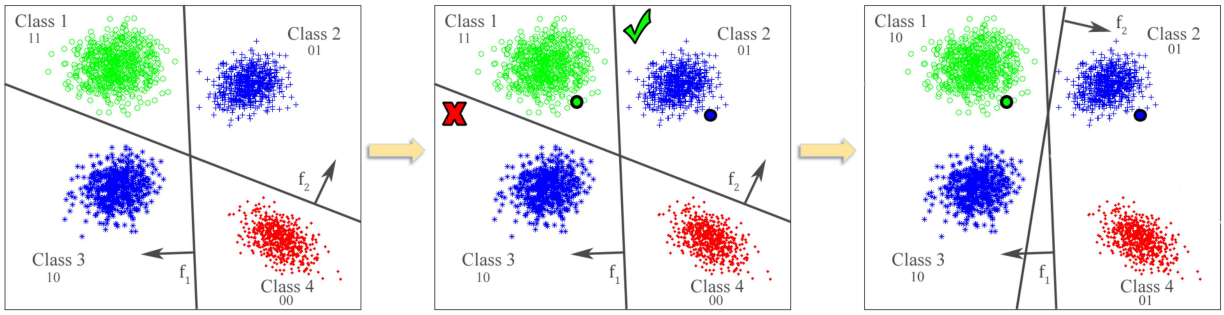

Figure 1: A setup in $\mathbb{R}^2$ with four classes and two hash functions f_1 and f_2. The feature space is divided into four regions and a binary code is assigned to instances of each class. Given the circled pair of points sampled from two distinct classes C_1 and C_2, the hash function f_2 assigns the same bit. Correcting this 'error' will give rise to the case where identical binary codes are assigned to different classes, a far cry from what is desired.

for a pair sampled from the two distinct classes C_1 and C_2, but it will be erroneous to "correct" this hashing. Simply put, the *collective effort* of the hash functions towards the end goal can lead to difficulty in assessing which hash functions to update or whether to update any at all.

In this study, we thus propose an SGD based solution for hashing. Being online, our method is amenable to subsequent variations of the data. Moreover, the method is orders of magnitude faster than state-of-the-art batch solutions while attaining comparable retrieval accuracy on three standard image retrieval benchmarks. We also provide improved retrieval accuracy over a recently reported online hashing method [9].

The remainder of the paper is organized as follows. Section 2 gives the framework for our method. In Section 3 we report experiments followed by concluding remarks in Section 4.

2. Adaptive Hashing

In this section, we first define the notation and the feature representation used in our formulation. We then devise an online method for learning the hash functions via Stochastic Gradient Descent (SGD), along with an update strategy that selects the hash functions to be updated and a regularization formulation that discourages redundant hash functions.

2.1. Notation and Feature Representation

We are given a set of data points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ in which each point is in a $d-$dimensional feature space, i.e., $\mathbf{x} \in \mathbb{R}^d$. In addition, let S denote a similarity matrix in which its elements $s_{ij} \in \{-1, 1\}$ define non-similarity or similarity for pairs $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively. S can be derived from a metric defined on $\mathbb{R}^d$ or from label information if available. Let $\mathcal{H}$ denote the Hamming space. In hashing, the ultimate goal is to assign binary codes to instances such that their proximity in feature space $\mathbb{R}^d$, as embodied in S, are preserved in Hamming space $\mathcal{H}$. A collection of hash functions $\{f_1, f_2, \ldots, f_b\}$ is utilized for this purpose where each function $f : \mathbb{R}^d \rightarrow \{-1, 1\}$ accounts for the generation of one bit in the binary code.

Following [18, 9], we use hash functions of the form

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) - w_0) \tag{1}$$

where $\boldsymbol{\phi}(\mathbf{x}) = [K(\mathbf{x}_1, \mathbf{x}), \ldots, K(\mathbf{x}_m, \mathbf{x})]^T$ is a non-linear mapping, $K(\cdot, \cdot)$ is a kernel function and points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ are uniformly sampled from $\mathcal{X}$ beforehand. This kernelized representation has shown to be effective [13] especially for inseparable data while also being efficient when $m \ll N$.

For compact codes, maximizing the entropy of the hash functions has shown to be beneficial [23, 22]. This implies $\int f(\mathbf{x}) dP(\mathbf{x}) = 0$ and we approximate it by setting the bias term w_0 to be $\frac{1}{N}\sum_{i=1}^{N} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i)$. $f(\mathbf{x})$ can then be compactly written as $\text{sgn}(\mathbf{w}^T \boldsymbol{\psi}(\mathbf{x}))$ where $\boldsymbol{\psi}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x}) - \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\phi}(\mathbf{x}_i)$. Finally, the binary code of $\mathbf{x}$ is computed and denoted by $\mathbf{f}(\mathbf{x}) = \text{sgn}(W^T \boldsymbol{\psi}(\mathbf{x})) = [f_1(\mathbf{x}), ..., f_b(\mathbf{x})]^T$ where sgn is an element-wise operation and $W = [\mathbf{w}_1, ..., \mathbf{w}_b] \in \mathbb{R}^{d \times b}$ with d now denoting the dimensionality of $\boldsymbol{\psi}(\mathbf{x})$.

After defining the feature representation, hashing studies usually approach the problem of learning the parameters $\mathbf{w}_1, ..., \mathbf{w}_b$ by minimizing a certain loss measure [12, 19, 18] or through optimizing an objective function and subsequently inferring these parameters [23, 24]. In the next section, we will present our SGD based algorithm for learning the hash functions. From this perspective, we assume the pairs of points $\{\mathbf{x}_i, \mathbf{x}_j\}$ arrive sequentially in an i.i.d manner from an underlying distribution and the similarity indicator s_{ij} is computed on the fly.

2.2. Learning Formulation

Following [17], which compared different loss functions, we employ the squared error loss:

$$l(\mathbf{f}(\mathbf{x}_i), \mathbf{f}(\mathbf{x}_j); W) = \left(\mathbf{f}(\mathbf{x}_i)^T \mathbf{f}(\mathbf{x}_j) - bs_{ij}\right)^2 \quad (2)$$

where b is the length of the binary code. This loss function is mathematically attractive for gradient computations and has been empirically shown to offer superior performance [18]. If $F = [\mathbf{f}(\mathbf{x_1}), ..., \mathbf{f}(\mathbf{x_N})]^T = \left[\text{sgn}(W^T \boldsymbol{\psi}(\mathbf{x}_1)), \ldots, \text{sgn}(W^T \boldsymbol{\psi}(\mathbf{x}_N))\right]^T = \text{sgn}(\left(W^T \Phi\right)^T)$ where $\Phi = [\boldsymbol{\psi}(\mathbf{x_1}), ..., \boldsymbol{\psi}(\mathbf{x_N})]$ then learning the hash function parameters can be formulated as the following least-squares optimization problem

$$\min_{W \in \mathbb{R}^{m \times b}} J(W) = \sum_{ij} l(\mathbf{f}(\mathbf{x}_i), \mathbf{f}(\mathbf{x}_j); W) = \left\| F^T F - bS \right\|_{\mathcal{F}}^2 \quad (3)$$

where $\| \cdot \|_{\mathcal{F}}$ is the Frobenius norm. In [18], this problem is solved in a sequential manner, by finding a parameter vector $\mathbf{w}$ at each iteration. Instead, we would like to minimize it via online gradient descent, where at each iteration a pair $\{\mathbf{x}_i, \mathbf{x}_j\}$ is chosen at random, and parameter W is updated according to the following rule

$$W^{t+1} \leftarrow W^t - \eta^t \nabla_W l(\mathbf{f}(\mathbf{x}_i), \mathbf{f}(\mathbf{x}_j); W^t) \quad (4)$$

where the learning rate η^t is a positive real number. To compute $\nabla_W l$, we approximate the non-differentiable *sgn* function with the sigmoid $\sigma(x) = 2/(1 + e^{-x}) - 1$. Assuming $\mathbf{v}_g = \mathbf{w}_g^T \boldsymbol{\psi}(\mathbf{x}_i)$ and $\mathbf{u}_g = \mathbf{w}_g^T \boldsymbol{\psi}(\mathbf{x}_j)$, the derivative of l with respect to w_{fg} is

$$\begin{aligned} \frac{\partial l(\mathbf{x_i}, \mathbf{x_j}, W)}{\partial w_{fg}} &= 2[\sigma(W^T \boldsymbol{\psi}(\mathbf{x}_i))^T \sigma(W^T \boldsymbol{\psi}(\mathbf{x}_j)) - bs_{ij}] \times \\ &\quad [\sigma(\mathbf{v}_g) \frac{\partial \sigma(\mathbf{u}_g)}{\partial \mathbf{u}_g} \frac{\partial \mathbf{u}_g}{\partial w_{fg}} + \sigma(\mathbf{u}_g) \frac{\partial \sigma(\mathbf{v}_g)}{\partial \mathbf{v}_g} \frac{\partial \mathbf{v}_g}{\partial w_{fg}}] \end{aligned} \quad (5)$$

where $\frac{\partial \sigma(\mathbf{u}_g)}{\partial \mathbf{u}_g} = \frac{2e^{-\mathbf{u}_g}}{(1+e^{-\mathbf{u}_g})^2}$ and $\frac{\partial \mathbf{u}_g}{\partial w_{fg}} = \psi_f(\mathbf{x}_j)$[1]. Subsequently, we obtain $\partial l / \partial W = OP + QR$ where $O = [\boldsymbol{\psi}(\mathbf{x}_i), \ldots, \boldsymbol{\psi}(\mathbf{x}_i)] \in \mathbb{R}^{m \times b}$ and $P = \text{diag}[\sigma(\mathbf{u}_1) \partial \sigma(\mathbf{v}_1) / \partial \mathbf{v}_1, \ldots, \sigma(\mathbf{u}_b) \partial \sigma(\mathbf{v}_b) / \partial \mathbf{v}_b] \in \mathbb{R}^{b \times b}$ [2]. Updating W solely based on the squared error loss would be erroneous since this function incurs a penalty if the Hamming distance between non-similar (similar) points is not maximized (minimized). As illustrated in Fig. 1, a perfect retrieval can still be achieved when this criterion is not satisfied. Thus, an additional step before applying Eq. 4 is required to determine what parameters to update, as will be described next.

[1] $\frac{\partial \sigma(\mathbf{v}_g)}{\partial \mathbf{v}_g}$ and $\frac{\partial \mathbf{v}_g}{\partial w_{fg}}$ are similarly evaluated.

[2] Q and R are obtained by replacing $\mathbf{x}_i$ with $\mathbf{x}_j$ in O and by swapping $\mathbf{u}_g$ and $\mathbf{v}_g$ in R.

2.3. Update Strategy

The squared error function in Eq. 2 is practically convenient and performs well, but its gradient may be nonzero even when there is no need for an update (e.g., the case shown in Fig. 1). Thus, at each step of online learning, we need to decide whether or not to update any hash function at all and if so, we seek to determine the amount and which of the hash functions need to be corrected. For this purpose, we employ the hinge-like loss function of [19, 9] defined as

$$l_h(\mathbf{f}(\mathbf{x}_i), \mathbf{f}(\mathbf{x}_j)) = \begin{cases} \max(0, d_H - (1-\alpha)b) & s_{ij} = 1 \\ \max(0, \alpha b - d_H) & s_{ij} = -1 \end{cases} \quad (6)$$

where $d_H \equiv \| \mathbf{f}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_j) \|_H$ is the Hamming distance and $\alpha \in [0, 1]$ is a user-defined parameter designating the extent to which the hash functions may produce a loss. If $l_h = 0$ then we do not perform any update, otherwise, $\lceil l_h \rceil$ indicates the number of bits in the binary code to be corrected. In determining which of the $\lceil l_h \rceil$ functions to update, we consider updating the functions for which the hash mappings are the most erroneous. Geometrically, this corresponds to the functions for which similar (dissimilar) points are incorrectly mapped to different (identical) bits with a high margin. Formally, let $\Lambda = \left\{ \max\left(\frac{|f_1(\mathbf{x}_i)|}{\|\overline{\mathbf{w}_1}\|}, \frac{|f_1(\mathbf{x}_j)|}{\|\overline{\mathbf{w}_1}\|} \right), \ldots, \max\left(\frac{|f_b(\mathbf{x}_i)|}{\|\overline{\mathbf{w}_b}\|}, \frac{|f_b(\mathbf{x}_j)|}{\|\overline{\mathbf{w}_b}\|} \right) \right\}$ where $\overline{\mathbf{w}} = [\mathbf{w}, w_0]^T$. We select the hash functions to be updated by sorting set Λ in descending order and identifying the indices of the first $\lceil l_h \rceil$ elements that incorrectly map $\{\mathbf{x}_i, \mathbf{x}_j\}$.

2.4. Regularization

Decorrelated hash functions are important for attaining good performance with compact codes and also for avoiding redundancy in the mappings [22]. During online learning, the decision boundaries may become progressively correlated, subsequently leading to degraded performance. In order to alleviate this issue, we add an orthogonality regularizer to Eq. 1. This provides an alternative to strictly constraining the hashings to be orthogonal, which may result in mappings along directions that have very low variance in the data. Eq. 1 then becomes

$$\begin{aligned} l(\mathbf{f}(\mathbf{x}_i), \mathbf{f}(\mathbf{x}_j); W) &= \left(\mathbf{f}(\mathbf{x}_i)^T \mathbf{f}(\mathbf{x}_j) - Bs_{ij}\right)^2 \\ &\quad + \tfrac{\lambda}{4} \left\| W^T W - I \right\|_{\mathcal{F}}^2 \end{aligned} \quad (7)$$

where λ is the regularization parameter. Consequently, $\partial l / \partial W$ is now equal to $OP + QR + \lambda(WW^T - I)W$.

The objective function in Eq. 7 is non-convex, and solving for the global minimum is difficult in practice. To get around this, a surrogate convex function could considered for which a global minimum can be found [20] and convergence of learning can be analyzed. However, because of the strong assumptions that must be made, use of a convex

input : Streaming pairs $\{(\mathbf{x}_i^t, \mathbf{x}_j^t)\}_{t=1}^T$, α, λ, η^t
Initialize W_0;
for $t \leftarrow 1$ **to** T **do**
  Compute binary codes $\mathbf{f}(\mathbf{x}_i^t)$, $\mathbf{f}(\mathbf{x}_j^t)$ and similarity indicator s_{ij};
  Compute loss $l_h(\mathbf{f}(\mathbf{x}_i), \mathbf{f}(\mathbf{x}_j))$ according to Eq. 6 ;
  if $\lceil l_h \rceil \neq 0$ **then**
    Compute $\nabla_W l(\mathbf{x}_i, \mathbf{x}_j, W^t) \in \mathbb{R}^{d \times b}$;
    Compute and sort Λ;
    $j \leftarrow$ the incorrect first $\lceil l_h \rceil$ indices of sorted Λ;
    $\nabla_W l(:, j) \leftarrow 0$;
    $W^{t+1} \leftarrow$
    $W^t - \eta^t \nabla_W l(\mathbf{x}_i, \mathbf{x}_j, W^t)$// Update;
  else
    $W^{t+1} \leftarrow W^t$
  end
end

Algorithm 1: Adaptive hashing algorithm based on SGD for fast similarity search

surrogate can lead to poor approximation quality in representing the "real problem," resulting in inferior performance in practice [4, 2]. In the computer vision literature, many have advocated for direct minimization of non-convex objective functions, oftentimes using stochastic gradient descent (SGD), e.g., [11, 8, 6], yielding state-of-the-art results. In this work, we wish to faithfully represent the actual adaptive hashing learning task; therefore, we eschew the use of a convex surrogate for Eq. 7 and directly minimize via SGD. We summarize our adaptive hashing algorithm in Alg. 1. We find that directly minimizing Eq. 7 via SGD yields excellent performance in adaptively learning the hashing functions in our experiments.

3. Experiments

For comparison with [9] we evaluate our approach on the 22K LabelMe and PhotoTourism-Half Dome datasets. In addition, we also demonstrate results on the large-scale Tiny 1M benchmark. We compare our method with five state-of-the-art solutions; Kernelized Locality Sensitive Hashing (KLSH) [13], Binary Reconstructive Embeddings (BRE) [12], Minimal Loss Hashing (MLH) [19], Supervised Hashing with Kernels (SHK) [18] and Fast Hashing (FastHash) [16]. These methods have outperformed earlier supervised and unsupervised techniques such as [3, 24, 23, 22]. In addition, we compare our approach against Online Kernel Hashing (OKH) [9].

3.1. Evaluation Protocol

We consider two large-scale retrieval schemes: one scheme is based on Hamming ranking and the other is based on hash lookup. The first scheme ranks instances based on Hamming distances to the query. Although it requires a linear scan over the corpus it is extremely fast owing to the binary representations of the data. To evaluate retrieval accuracy, we measure Mean Average Precision (mAP) scores for a set of queries evaluated at varying bit lengths (up to 256 bits). The second retrieval scheme, which is based on hash lookup, involves retrieving instances within a Hamming ball; specifically, we set the Hamming radius to 3 bits. This procedure has constant time complexity. To quantify retrieval accuracy under this scheme, we compute the Average Precision. If a query returns no neighbors within the Hamming ball, it is considered as zero precision. We also report mAP with respect to CPU time. Furthermore, when comparing against OKH we also report the cumulative mean mAP and area under curve (AUC) metric. For all experiments, we follow the protocol used in [18, 9] to construct training and testing sets.

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

3.2. Datasets

22K LabelMe The 22K LabelMe dataset has 22,019 images represented as 512-dimensional Gist descriptors. As pre-processing, we normalize each instance to have unit length. The dataset is randomly partitioned into two: a training and testing set with 20K and 2K samples, respectively. A 2K subset of the training points is used as validation data for tuning algorithmic parameters, while another 2K samples are used to learn the hash functions; the remaining samples are used to populate the hash table. The l_2 norm is used in determining the nearest neighbors. Specifically, $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered similar pairs if their Euclidean norm is within the smallest 5% of the 20K distances. S is constructed accordingly and the closest 5% distances of a query are also used to determine its true neighbors.

Half Dome This dataset contains a series of patches with matching information obtained from the Photo Tourism reconstruction of Half Dome. The dataset includes 107,732 patches in which the matching ones are assumed to be projected from the same 3D point into different images. We extract Gist descriptors for each patch and as previously, we normalize each instance to have unit length. The dataset is then partitioned into a training and testing set with 105K and 2K samples, respectively. A 2K subset of the training samples is used as validation data for tuning algorithmic parameters, while another 2K samples from the training set are used to learn the hash functions, and

the remaining training samples are then used to populate the hash table. The match information associated with the patches is used to construct the matrix S and to determine the true neighbors of a query.

Tiny 1M For this benchmark, a set of million images are sampled from the Tiny image dataset in which 2K distinct samples are used as test, training and validation data while the rest is used for populating the hash table. The data is similarly preprocessed to have unit norm length. Similarly to the 22K LabelMe benchmark, the l_2 norm is used in determining the nearest neighbors, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered similar pairs if their Euclidean distance is within the smallest 5% of the 1M distances. S is constructed accordingly and the closest 5% distances of a query are also used to determine its true neighbors.

As described in the next section, when evaluating our method against OKH, the online learning process is continued until 10K, 50K and 25K pairs of training points are observed for the LabelMe, Half Dome and Tiny1M datasets, respectively. This is done to provide a more detailed and fair comparison between the OKH online method and our online method for adaptively learning the hashing functions.

3.3. Comparison with Online Hashing [9]

Fig. 2 gives comparison between our method and OKH. We report mAP and cumulative mean mAP values with respect to iteration number for 96 bit binary codes. We also report the Area Under Curve (AUC) score for the latter metric. As stated, the parameters for both methods have been set via cross-validation and the methods are initialized with KLSH. The online learning is continued for 10K, 50K and 25K points for the LabelMe, Half Dome and Tiny1M datasets. The experiments are repeated five times with different randomly chosen initializations and orderings of pairs. Ours and Ours$_r$ denote the un-regularized and regularized versions of our method, respectively.

Analyzing the results, we do not observe significant improvements in performance for the 22K LabelMe dataset. Our method has a benign AUC score increase with respect to OKH. On the other hand, the performance improvements for the Half Dome and Tiny 1M datasets are much more noticeable. Our method converges faster in terms of iterations and shows a significant boost in AUC score (0.67 compared to 0.60 for Half Dome and 0.43 compared to 0.40 for Tiny 1M, specifically). Although both methods perform similarly at the early stages of learning, OKH yields inferior performance as the learning process continues compared to our method. Moreover our technique learns better hash functions in achieving higher retrieval accuracy for both the Halfdome and Tiny 1M benchmarks. The regularization also has a positive albeit slight effect in improving the performance. This validates a former claim that the regularization term alleviates the possible gradual correlation among hash functions as the algorithm progresses and thus helps in avoiding inferior performance.

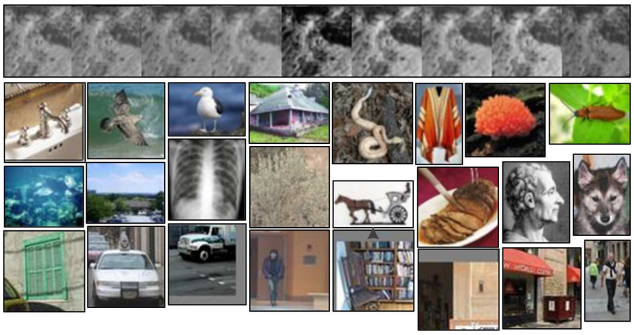

Figure 3: Sample pictures from the 22K LabelMe, Halfdome and Tiny 1M datasets.

3.4. Comparison with Batch Solutions

We also conducted experiments to compare our online solution against leading batch methods. For these comparisons, we show performance values for hash codes of varying length, in which the number of bits is varied from 12 to 256. All methods are trained and tested with the same data splits. Specifically, as a standard setup, 2K training points are used for learning the hash functions for both batch and online solutions. Thus, for online techniques, the performance scores reported in the tables and figures are derived from an early stage of the algorithm in which both our method and OKH achieve similar results, as will be shown. This is different when reporting mAP values with respect to CPU time, where the online learning is continued for 10K (LabelMe), 50K (Halfdome), and 25K (Tiny1M) pairs of points as is done in Section 3.3. The mAP vs CPU time figures is thus important to demonstrate a more fair comparison. Other algorithmic parameters have been set via cross validation. Finally, *Ours* describe our regularized method.

Table 1 shows mAP values for the 22K LabelMe benchmark. Analyzing the results, we observe that while BRE performs the best in most cases, our method surpasses other batch solutions especially when # of bits is > 96, e.g., for 96 and 128 bit binary codes our method achieves 0.64 and 0.67 mAP values, respectively, surpassing SHK, MLH, FastHash KLSH. More importantly, these results are achieved with drastic training time improvements. For example, it takes only ~ 4 secs to learn the hash functions that give retrieval accuracy that is comparable with state-of-the-art batch solutions that take at best 600 secs to train.

Results for the hash lookup based retrieval scheme are shown in Fig. 4 (center-right). Here, we again observe that the online learning techniques demonstrate similar hash lookup precision values and success rates. Not surprisingly

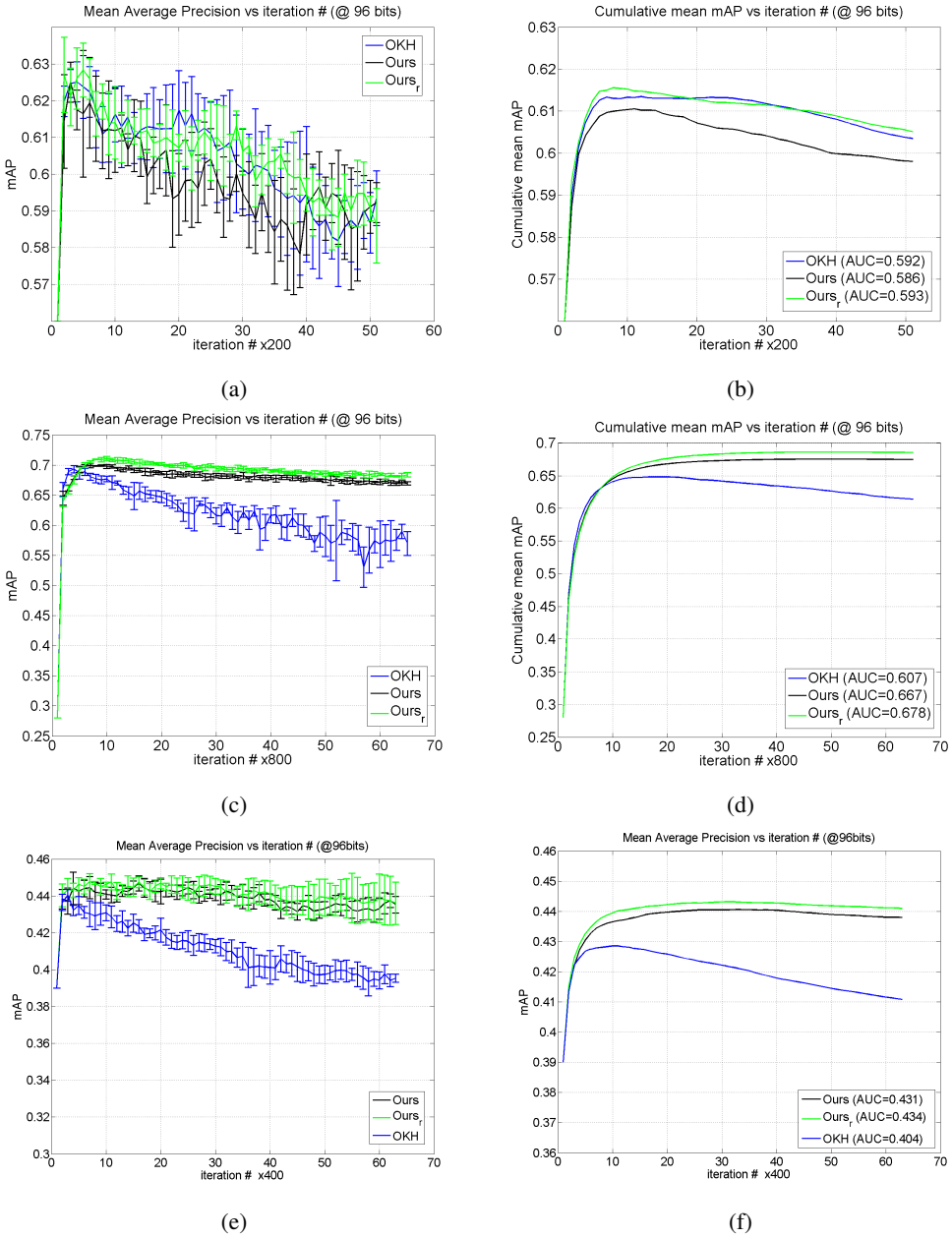

Figure 2: Mean average precision and its cumulative mean with respect to iteration number for comparison with OKH [9]. Ours and Ours$_r$ denote the unregularized and regularized versions of our method, respectively. (Top row) 22K LabelMe dataset. (Middle row) Half dome dataset. (Bottom row) Tiny 1M dataset.

both the precision values and success rates drop near zero for lengthier codes due to the exponential decrease of instances in the hash bins. With fewer bits the hash lookup precision is also low, most likely due to the decrease in hashing quality with small binary codes. This suggests that implementers must appropriately select the Hamming ball radius and the code size for optimal retrieval performance.

Similarly, Table 2 reports mAP at varying numbers of bits for the Halfdome dataset. We observe that SHK performs the best for most cases while our method is a close runner-up, e.g., SHK achieves 0.74 and 0.79 mAP values compared to our 0.66 and 0.76 for 96 and 128 bits, respectively. Again, our method achieves these results with hash functions learned orders of magnitude faster than SHK and all other batch methods. Though KLSH also has a very low training time it demonstrates poor retrieval performance

Method	Mean Average Precision (Random ~ 0.05)						Training time (seconds)
	12 bits	24 bits	48 bits	96 bits	128 bits	256 bits	96 bits
BRE [12]	0.26	0.48	**0.58**	**0.67**	**0.70**	**0.74**	929
MLH [19]	0.31	0.37	0.46	0.59	0.61	0.63	967
SHK [18]	0.39	0.48	0.56	0.61	0.63	0.67	1056
FastHash [16]	**0.40**	**0.49**	0.56	0.61	0.63	0.66	672
KLSH [13]	0.31	0.42	0.48	0.56	0.59	0.64	5×10^{-4}
OKH [9] (@2K)	0.32	0.43	0.53	0.63	0.67	0.73	3.7
Ours (@2K)	0.33	0.43	0.54	0.64	0.67	0.71	3.8

Table 1: Mean Average Precision for the 22K LabelMe dataset. For all methods, 2K points are used in learning the hash functions.

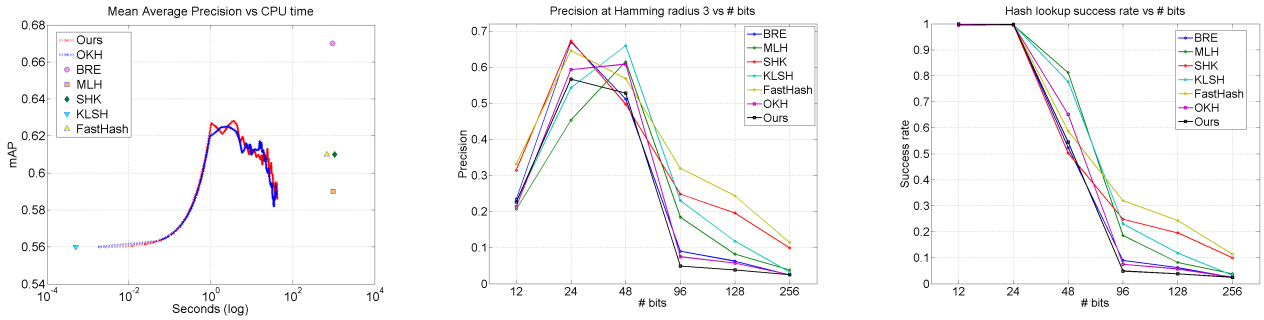

Figure 4: Results for the 22 LabelMe dataset. (Left) Mean Average Precision (at 96 bits) with respect to CPU time in which for OKH and our method, the online learning is continued for 10K pairs of points. (Center) Mean hash lookup precision with Hamming radius 3. (Right) Hash lookup success rate.

Method	Mean Average Precision (Random $\sim 1\times10^{-4}$)						Training time (seconds)
	12 bits	24 bits	48 bits	96 bits	128 bits	256 bits	96 bits
BRE [12]	0.003	0.078	0.40	0.62	0.69	0.80	1022
MLH [19]	0.012	0.13	0.38	0.62	0.67	0.80	968
SHK [18]	**0.024**	**0.23**	0.46	**0.74**	**0.79**	**0.87**	631
FastHash [16]	**0.024**	0.16	0.43	0.69	0.75	0.84	465
KLSH [13]	0.012	0.05	0.11	0.28	0.33	0.40	5×10^{-4}
OKH [9] (@2K)	0.022	0.18	0.45	0.69	0.73	0.81	6
Ours (@2K)	**0.024**	0.18	**0.47**	0.66	0.76	0.83	7.8

Table 2: Mean Average Precision for the Halfdome dataset. For all methods, 2K points are used in learning the hash functions.

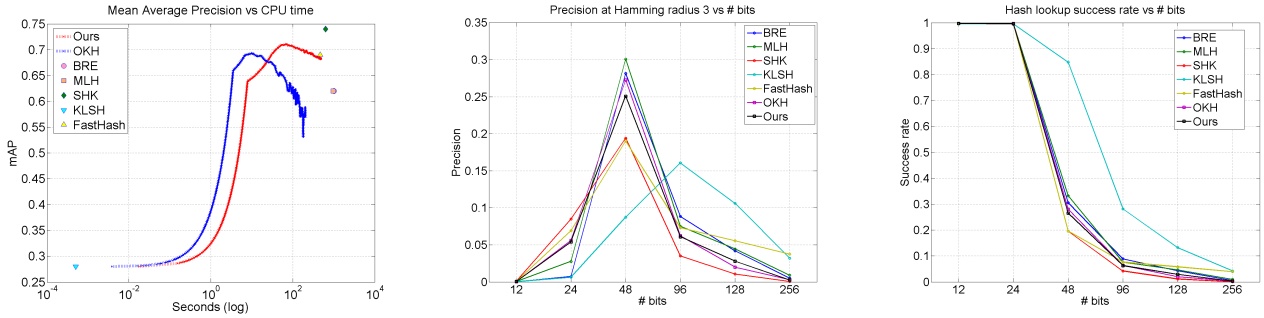

Figure 5: Results for the Halfdome dataset. (Left) Mean Average Precision (at 96 bits) with respect to CPU time in which for OKH and our method, the online learning is continued for 50K pairs of points. (Center) Mean hash lookup precision with Hamming radius 3. (Right) Hash lookup success rates.

Method	Mean Average Precision (Random)						Training time (seconds)
	12 bits	24 bits	48 bits	96 bits	128 bits	256 bits	96 bits
BRE [12]	0.24	**0.32**	**0.38**	**0.47**	**0.50**	**0.59**	669
MLH [19]	0.21	0.28	0.35	0.41	0.45	0.55	672
SHK [18]	0.24	0.30	0.36	0.41	0.42	0.45	3534
FastHash [16]	**0.26**	**0.32**	**0.38**	0.44	0.46	0.52	772
KLSH [13]	0.2	0.25	0.32	0.39	0.42	0.45	5×10^{-4}
OKH [9] (@2K)	0.23	0.28	0.36	0.44	0.47	0.55	1.9
Ours (@2K)	0.24	0.29	0.36	0.44	0.48	0.52	2.4

Table 3: Mean Average Precision for the Tiny 1M dataset. For all methods, 2K points are used in learning the hash functions.

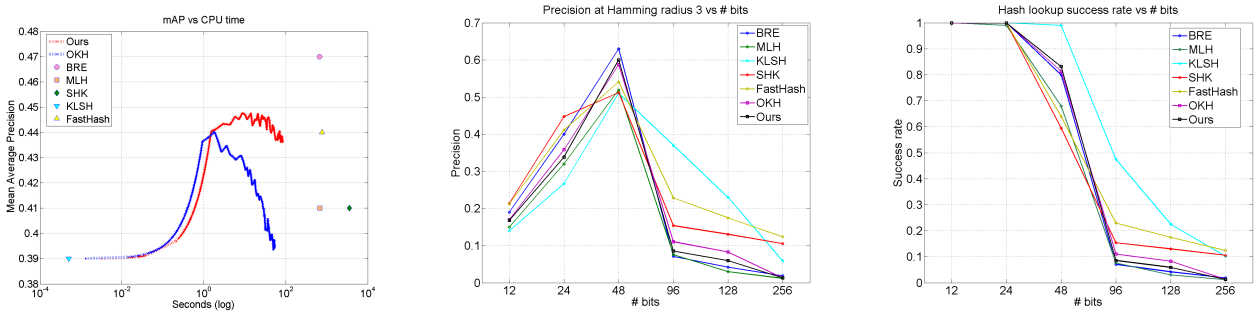

Figure 6: Results for the Tiny 1M dataset. (Left) Mean Average Precision (at 96 bits) with respect to CPU time in which for OKH and our method, the online learning is continued for 25K pairs of points. (Center) Mean hash lookup precision with Hamming radius 3. (Right) Hash lookup success rates.

compared vs. other solutions. Fig. 5 (center-right) show results for the hash lookup precision and success rate in which we observe similar patterns and values compared to state-of-the-art techniques.

Table 3 reports mAP values for the large scale Tiny 1M dataset. Here we observe that FastHash and BRE performs best with compact codes and lengthier codes, respectively. Our online technique performs competitively against these batch methods. The hash lookup precision and success rates for the online techniques are again competitive as shown in Fig. 6 (center-right).

Figs. 4-6 (left) show mAP values of all the techniques with respect to CPU time. The batch learners correspond to single points in which 2K samples are used for learning. For the 22K LabelMe dataset, we observe no significant difference in performance between our technique and OKH. However, both of these online methods outperform state-of-the-art (except for BRE) with computation time significantly reduced. For the Halfdome and Tiny 1M benchmarks, the OKH method is slightly faster than ours; however, in both benchmarks the accuracy of OKH degrades as the learning process continues. Overall, our method demonstrates higher retrieval accuracy compared to OKH and is generally the runner-up method overall. Yet, our results are obtained from hash functions learned orders of magnitude faster compared to the state-of-the-art batch techniques.

4. Conclusion

In this study, we have proposed an algorithm based on stochastic gradient descent to efficiently learn hash functions for fast similarity search. Being online it is adaptable to variations and growth in datasets. Our online algorithm attains retrieval accuracy that is comparable with state-of-the-art batch-learning methods on three standard image retrieval benchmarks, while being orders of magnitude faster than competing state-of-the-art batch-learning methods. In addition, our proposed formulation gives improved retrieval performance over the only competing online hashing technique, OKH, as demonstrated in experiments.

Adaptive online hashing methods are essential for large-scale applications where the dataset is not static, but continues to grow and diversify. This application setting presents important challenges for hashing-based fast similarity search techniques, since the hashing functions would need to continue to evolve and improve over time. While our results are promising, we believe further improvements are possible. In particular, we would like to investigate methods that can augment and grow the hash codes over time, while at the same time not revisiting (or relearning) hashes for previously seen items that are already stored in an index.

References

[1] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 161–168, 2008. 1

[2] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proc. International Conf. on Machine Learning (ICML)*, 2006. 4

[3] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry SCG*, 2004. 1, 4

[4] T.-M.-T. Do and T. Artières. Regularized bundle methods for convex and non-convex risks. *Journal of Machine Learning Research*, 2012. 4

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases VLDB*, 1999. 1

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014. 4

[7] Y. 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. 1

[8] J. He, L. Balzano, and A. Szlam. Incremental gradient on the grassmannian for online foreground and background separation in subsampled video. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012. 4

[9] L.-K. Huang, Q. Y. 0010, and W.-S. Zheng. Online hashing. In *International Joint Conferences on Artificial Intelligence IJCAI*, 2013. 2, 3, 4, 5, 6, 7, 8

[10] R. Kehl, M. Bray, and L. Van Gool. Full body tracking from multiple views using stochastic sampling. In *IEEE conference on Computer Vision and Pattern Recognition CVPR*, 2005. 1

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2012. 4

[12] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2009. 1, 2, 4, 7, 8

[13] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proc. IEEE International Conf. on Computer Vision (ICCV)*, 2009. 1, 2, 4, 7, 8

[14] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag. 1

[15] Y. Lecun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2004. 1

[16] G. Lin, C. Shen, Q. 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. 4, 7, 8

[17] 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. 1, 3

[18] J. W. Liu, Wei and, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012. 1, 2, 3, 4, 7, 8

[19] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proc. International Conf. on Machine Learning (ICML)*, 2011. 1, 2, 3, 4, 7, 8

[20] S. Shalev-Shwartz. Online learning and online convex optimization. *Journal Foundations and Trends in Machine Learning*, 2012. 3

[21] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proc. International Conf. on Machine Learning (ICML)*, 2007. 1

[22] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Trans. Pattern Anal. Mach. Intell. TPAMI*, 2012. 1, 2, 3, 4

[23] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2008. 1, 2, 4

[24] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval SIGIR*, 2010. 1, 2, 4