MIHash: Online Hashing with Mutual Information

Fatih Cakir* Kun He* Sarah Adel Bargal Stan Sclaroff
Department of Computer Science
Boston University, Boston, MA
{fcakir,hekun,sbargal,sclaroff}@cs.bu.edu

Abstract

Learning-based hashing methods are widely used for nearest neighbor retrieval, and recently, online hashing methods have demonstrated good performance-complexity trade-offs by learning hash functions from streaming data. In this paper, we first address a key challenge for online hashing: the binary codes for indexed data must be recomputed to keep pace with updates to the hash functions. We propose an efficient quality measure for hash functions, based on an information-theoretic quantity, mutual information*, and use it successfully as a criterion to eliminate unnecessary hash table updates. Next, we also show how to optimize the mutual information objective using stochastic gradient descent. We thus develop a novel hashing method,* MIHash*, that can be used in both online and batch settings. Experiments on image retrieval benchmarks (including a 2.5M image dataset) confirm the effectiveness of our formulation, both in reducing hash table recomputations and in learning high-quality hash functions.*

1. Introduction

Hashing is a widely used approach for practical nearest neighbor search in many computer vision applications. It has been observed that adaptive hashing methods that learn to hash from data generally outperform data-independent hashing methods such as Locality Sensitive Hashing [4]. In this paper, we focus on a relatively new family of adaptive hashing methods, namely *online* adaptive hashing methods [1, 2, 6, 11]. These techniques employ online learning in the presence of streaming data, and are appealing due to their low computational complexity and their ability to adapt to changes in the data distribution.

Despite recent progress, a key challenge has not been addressed in online hashing, which motivates this work: the computed binary representations, or the "hash table", may become outdated after a change in the hash mapping. To reflect the updates in the hash mapping, the hash table may need to be recomputed frequently, causing inefficiencies in the system such as successive disk I/O, especially when dealing with large datasets. We thus identify an important question for online adaptive hashing systems: *when to update the hash table?* Previous online hashing solutions do not address this question, as they usually update both the hash mapping and hash table concurrently.

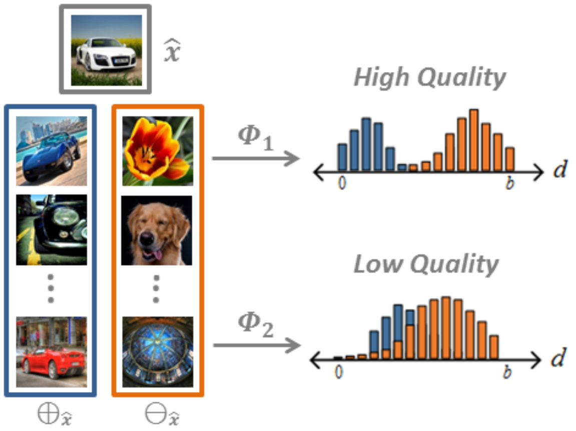

Figure 1: We study online hashing for efficient nearest neighbor retrieval. Given a hash mapping Φ, an image $\hat{\mathbf{x}}$, along with its neighbors in $\oplus_{\hat{\mathbf{x}}}$ and non-neighbors in $\ominus_{\hat{\mathbf{x}}}$, are mapped to binary codes, yielding two distributions of Hamming distances. In this example, Φ_1 has higher quality than Φ_2 since it induces more separable distributions. The information-theoretic quantity Mutual Information can be used to capture the separability, which gives a good quality indicator and learning objective for online hashing.

We make the observation that achieving high quality nearest neighbor search is an ultimate goal in hashing systems, and therefore any effort to limit computational complexity should preserve, if not improve, that quality. Therefore, another important question is: *how to quantify quality?* Here, we briefly describe our answer to this question, but first introduce some necessary notation. We would like to learn a hash mapping Φ from feature space $\mathcal{X}$ to the b-dimensional Hamming space $\mathcal{H}^b$, whose outputs are b-bit

[*] First two authors contributed equally.

binary codes. The goal of hashing is to preserve a neighborhood structure in $\mathcal{X}$ after the mapping to $\mathcal{H}^b$. Given $\hat{\mathbf{x}} \in \mathcal{X}$, the neighborhood structure is usually given in terms of a set of its neighbors $\oplus_{\hat{\mathbf{x}}}$, and a set of non-neighbors $\ominus_{\hat{\mathbf{x}}}$. We discuss how to derive the neighborhood structure in Sec. 3.

As shown in Fig. 1, the distributions of the Hamming distances from $\hat{\mathbf{x}}$ to its neighbors and non-neighbors are histograms over $\{0, 1, \ldots, b\}$. Ideally, if there is no overlap between these two distributions, we can recover $\oplus_{\hat{\mathbf{x}}}$ and $\ominus_{\hat{\mathbf{x}}}$ by simply thresholding the Hamming distance. A nonzero overlap results in ambiguity, as observing the Hamming distance is no longer sufficient to determine neighbor relationships. Our discovery is that this overlap can be quantified using an information-theoretic quantity, *mutual information*, between two random variables induced by Φ. We then use mutual information to define a novel measure to quantify quality for hash functions in general.

With a quality measure defined, we answer the motivating question of when to update the hash table. We propose a simple solution by restricting updates to times when there is an estimated improvement in hashing quality, based on an efficient estimation method in the presence of streaming data. Notably, since mutual information is a good general-purpose quality measure for hashing, this results in a general plug-in module for online hashing that does not require knowledge of the learning method.

Inspired by this strong result, we next ask, *can we optimize mutual information as an objective to learn hash functions?* We propose a novel hashing method, MIHash, by deriving gradient descent rules on the mutual information objective, which can be applied in online stochastic optimization, as well as on deep architectures. The mutual information objective is free of tuning parameters, unlike others that may require thresholds, margins, *etc*.

We conduct experiments on three image retrieval benchmarks, including the Places205 dataset [32] with 2.5M images. For four recent online hashing methods, our mutual information based update criterion consistently leads to over an order of magnitude reduction in hash table recomputations, while maintaining retrieval accuracy. Moreover, our novel MIHash method achieves state-of-the-art retrieval results, in both online and batch learning settings.

2. Related Work

In this section, we mainly review hashing methods that adaptively update the hash mapping with incoming data, given that our focus is on online adaptive hashing. For a more general survey on hashing, please refer to [25].

Huang *et al.* [6] propose Online Kernel Hashing, where a stochastic environment is considered with pairs of points arriving sequentially. At each step, a number of hash functions are selected based on a Hamming loss measure and parameters are updated via stochastic gradient descent (SGD).

Cakir and Sclaroff [1] argue that, in a stochastic setting, it is difficult to determine which hash functions to update as it is the collective effort of all the hash functions that yields a good hash mapping. Hamming loss is considered to infer the hash functions to be updated at each step and a squared error loss is minimized via SGD.

In [2], binary Error Correcting Output Codes (ECOCs) are employed in learning the hash functions. This work follows a more general two-step hashing framework [14], where the set of ECOCs are generated beforehand and are assigned to labeled data as they appear, allowing the label space to grow with incoming data. Then, hash functions are learned to fit the binary ECOCs using Online Boosting.

Inspired by the idea of "data sketching", Leng *et al.* introduce Online Sketching Hashing [11] where a small fixed-size sketch of the incoming data is maintained in an online fashion. The sketch retains the Frobenius norm of the full data matrix, which offers space savings, and allows to apply certain batch-based hashing methods. A PCA-based batch learning method is applied on the sketch to obtain hash functions.

None of the above online hashing methods offer a solution to decide whether or not the hash table shall be updated given a new hash mapping. However, such a solution is crucial in practice, as limiting the frequency of updates will alleviate the computational burden of keeping the hash table up-to-date. Although [2] and [6] include strategies to select individual hash functions to recompute, they still require computing on all indexed data instances.

Recently, some methods employ deep neural networks to learn hash mappings, *e.g.* [12, 15, 27, 30] and others. These methods use minibatch-based stochastic optimization, however, they usually require multiple passes over a given dataset to learn the hash mapping, and the hash table is only computed when the hash mapping has been learned. Therefore, current deep learning based hashing methods are essentially batch learning methods, which differ from the online hashing methods that we consider, *i.e.* methods that process streaming data to learn and update the hash mappings on-the-fly while continuously updating the hash table. Nevertheless, when evaluating our mutual information based hashing objective, we compare against state-of-the-art batch hashing formulations as well, by contrasting different objective functions on the same model architecture.

Lastly, Ustinova *et al.* [23] recently proposed a method to derive differentiation rules for objective functions that require histogram binning, and apply it in learning deep embeddings. When optimizing our mutual information objective, we utilize their differentiable histogram binning technique for deriving gradient-based optimization rules. Note that both our problem setup and objective function are quite different from [23].

3. Online Hashing with Mutual Information

As mentioned in Sec. 1, the goal of hashing is to learn a hash mapping $\Phi : \mathcal{X} \rightarrow \mathcal{H}^b$ such that a desired neighborhood structure is preserved. We consider an online learning setup where Φ is continuously updated from input streaming data, and at time t, the current mapping Φ_t is learned from $\{\mathbf{x}_1, \dots, \mathbf{x}_t\}$. We follow the standard setup of learning Φ from pairs of instances with similar/dissimilar labels [9, 6, 1, 12]. These labels, along with the neighborhood structure, can be derived from a metric, *e.g.* two instances are labeled similar (*i.e.* neighbors of each other) if their Euclidean distance in $\mathcal{X}$ is below a threshold. Such a setting is often called unsupervised hashing. On the other hand, in supervised hashing with labeled data, pair labels are derived from individual class labels: instances are similar if they are from the same class, and dissimilar otherwise.

Below, we first derive the mutual information quality measure and discuss its use in determining when to update the hash table in Sec. 3.1. We then describe a gradient-based approach for optimizing the same quality measure, as an objective for learning hash mappings, in Sec. 3.2. Finally, we discuss the benefits of using mutual information in Sec. 3.3.

3.1. MI as Update Criterion

We revisit our motivating question: *When to update the hash table in online hashing?* During the online learning of Φ_t, we assume a retrieval set $\mathcal{S} \subseteq \mathcal{X}$, which may include the streaming data after they are received. We define the hash table as the set of hashed binary codes: $\mathcal{T}(\mathcal{S}, \Phi) = \{\Phi(\mathbf{x}) | \mathbf{x} \in \mathcal{S}\}$. Given the adaptive nature of online hashing, $\mathcal{T}$ may need to be recomputed often to keep pace with Φ_t; however, this is undesirable if $\mathcal{S}$ is large or the change in Φ_t's quality does not justify the cost of an update.

We propose to view the learning of Φ_t and computation of $\mathcal{T}$ as separate events, which may happen at different rates. To this end, we introduce the notion of a *snapshot*, denoted Φ^s, which is occasionally taken of Φ_t and used to recompute $\mathcal{T}$. Importantly, this happens only when the nearest neighbor retrieval quality of Φ_t has improved, and we now define the quality measure.

Given hash mapping $\Phi : \mathcal{X} \rightarrow \{-1, +1\}^b$, Φ induces Hamming distance $d_\Phi : \mathcal{X} \times \mathcal{X} \rightarrow \{0, 1, \dots, b\}$ as

$$d_\Phi(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{2}\left(b - \Phi(\mathbf{x})^\top \Phi(\hat{\mathbf{x}})\right). \tag{1}$$

Consider some instance $\hat{\mathbf{x}} \in \mathcal{X}$, and the sets containing neighbors and non-neighbors, $\oplus_{\hat{\mathbf{x}}}$ and $\ominus_{\hat{\mathbf{x}}}$. Φ induces two conditional distributions, $P(d_\Phi(\mathbf{x}, \hat{\mathbf{x}}) | \mathbf{x} \in \oplus_{\hat{\mathbf{x}}})$ and $P(d_\Phi(\mathbf{x}, \hat{\mathbf{x}}) | \mathbf{x} \in \ominus_{\hat{\mathbf{x}}})$ as seen in Fig. 1, and it is desirable to have low overlap between them. To formulate the idea, for Φ and $\hat{\mathbf{x}}$, define random variable $\mathcal{D}_{\hat{\mathbf{x}},\Phi} : \mathcal{X} \rightarrow \{0, 1, \dots, b\}, \mathbf{x} \mapsto d_\Phi(\mathbf{x}, \hat{\mathbf{x}})$, and let $\mathcal{C}_{\hat{\mathbf{x}}} : \mathcal{X} \rightarrow \{0, 1\}$ be the membership indicator for $\oplus_{\hat{\mathbf{x}}}$. The two conditional distributions can now be expressed as $P(\mathcal{D}_{\hat{\mathbf{x}},\Phi} | \mathcal{C}_{\hat{\mathbf{x}}} = 1)$ and $P(\mathcal{D}_{\hat{\mathbf{x}},\Phi} | \mathcal{C}_{\hat{\mathbf{x}}} = 0)$, and we can write the *mutual information* between $\mathcal{D}_{\hat{\mathbf{x}},\Phi}$ and $\mathcal{C}_{\hat{\mathbf{x}}}$ as

$$\mathcal{I}(\mathcal{D}_{\hat{\mathbf{x}},\Phi}; \mathcal{C}_{\hat{\mathbf{x}}}) = H(\mathcal{C}_{\hat{\mathbf{x}}}) - H(\mathcal{C}_{\hat{\mathbf{x}}} | \mathcal{D}_{\hat{\mathbf{x}},\Phi}) \tag{2}$$

$$= H(\mathcal{D}_{\hat{\mathbf{x}},\Phi}) - H(\mathcal{D}_{\hat{\mathbf{x}},\Phi} | \mathcal{C}_{\hat{\mathbf{x}}}) \tag{3}$$

where H denotes (conditional) entropy. In the following, for brevity we will drop subscripts Φ and $\hat{\mathbf{x}}$, and denote the two conditional distributions and the marginal $P(\mathcal{D}_{\hat{\mathbf{x}},\Phi})$ as $p_\mathcal{D}^+$, $p_\mathcal{D}^-$, and $p_\mathcal{D}$, respectively.

By definition, $\mathcal{I}(\mathcal{D}; \mathcal{C})$ measures the decrease in uncertainty in the neighborhood information $\mathcal{C}$ when observing the Hamming distances $\mathcal{D}$. We claim that $\mathcal{I}(\mathcal{D}; \mathcal{C})$ also captures how well Φ preserves the neighborhood structure of $\hat{\mathbf{x}}$. If $\mathcal{I}(\mathcal{D}; \mathcal{C})$ attains a high value, which means $\mathcal{C}$ can be determined with low uncertainty by observing $\mathcal{D}$, then Φ must have achieved good separation (*i.e.* low overlap) between $p_\mathcal{D}^+$ and $p_\mathcal{D}^-$. $\mathcal{I}$ is maximized when there is no overlap, and minimized when $p_\mathcal{D}^+$ and $p_\mathcal{D}^-$ are exactly identical. Recall, however, that $\mathcal{I}$ is defined with respect to a single instance $\hat{\mathbf{x}}$; therefore, for a general quality measure, we integrate $\mathcal{I}$ over the feature space:

$$Q(\Phi) = \int_\mathcal{X} \mathcal{I}(\mathcal{D}_{\hat{\mathbf{x}},\Phi}; C_{\hat{\mathbf{x}}}) p(\hat{\mathbf{x}}) d\hat{\mathbf{x}}. \tag{4}$$

$Q(\Phi)$ captures the expected amount of separation between $p_\mathcal{D}^+$ and $p_\mathcal{D}^-$ achieved by Φ, over all instances in $\mathcal{X}$.

In the online setting, given the current hash mapping Φ_t and previous snapshot Φ^s, it is then straightforward to pose the update criterion as

$$Q(\Phi_t) - Q(\Phi^s) > \theta, \tag{5}$$

where θ is a threshold; a straightforward choice is $\theta = 0$. However, Eq. 4 is generally difficult to evaluate due to the intractable integral; in practice, we resort to Monte-Carlo approximations to this integral, as we describe next.

Monte-Carlo Approximation by Reservoir Sampling

We give a Monte-Carlo approximation of Eq. 4. Since we work with streaming data, we employ the Reservoir Sampling algorithm [24], which enables sampling from a stream or sets of large/unknown cardinality. With reservoir sampling, we obtain a *reservoir set* $\mathcal{R} \triangleq \{\mathbf{x}_1^r, \dots, \mathbf{x}_K^r\}$ from the stream, which can be regarded as a finite sample from $p(\mathbf{x})$. We estimate the value of Q on $\mathcal{R}$ as:

$$Q_\mathcal{R}(\Phi) = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{x^r} \in \mathcal{R}} \mathcal{I}_\mathcal{R}(\mathcal{D}_{\mathbf{x^r},\Phi}; \mathcal{C}_{\mathbf{x^r}}). \tag{6}$$

We use subscript $\mathcal{R}$ to indicate that when computing the mutual information $\mathcal{I}$, the $p_\mathcal{D}^+$ and $p_\mathcal{D}^-$ for a reservoir instance $\mathbf{x^r}$ are estimated from $\mathcal{R}$. This can be done in $\mathcal{O}(|\mathcal{R}|)$

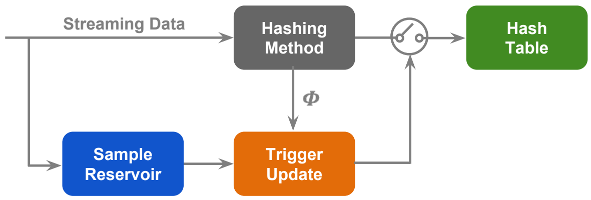

Figure 2: We present the general plug-in module for online hashing methods: Trigger Update (TU). We sample a reservoir $\mathcal{R}$ from the input stream, and estimate the mutual information criterion $Q_{\mathcal{R}}$. Based on its value, TU decides whether a hash table update should be executed.

time for each $\mathbf{x}^{\mathbf{r}}$, as the discrete distributions can be estimated via histogram binning.

Fig. 2 summarizes our approach. We use the reservoir set to estimate the quality $Q_{\mathcal{R}}$, and "trigger" an update to the hash table only when $Q_{\mathcal{R}}$ improves over a threshold. Notably, our approach provides a general *plug-in module* for online hashing techniques, in that it only needs access to streaming data and the hash mapping itself, independent of the hashing method's inner workings.

3.2. MI as Learning Objective

Having shown that mutual information is a suitable measure of neighborhood quality, we consider its use as a learning objective for hashing. Following the notation in Sec. 3.1, we define a loss $\mathcal{L}$ with respect to $\hat{\mathbf{x}} \in \mathcal{X}$ and Φ as

$$\mathcal{L}(\hat{\mathbf{x}}, \Phi) = -\mathcal{I}(\mathcal{D}_{\hat{\mathbf{x}},\Phi}; \mathcal{C}_{\hat{\mathbf{x}}}). \tag{7}$$

We model Φ as a collection of parameterized hash functions, each responsible for generating a single bit: $\Phi(\mathbf{x}) = [\phi_1(\mathbf{x}; W), ..., \phi_b(\mathbf{x}; W)]$, where $\phi_i : \mathcal{X} \to \{-1, +1\}, \forall i$, and W represents the model parameters. For example, linear hash functions can be written as $\phi_i(\mathbf{x}) = \mathrm{sgn}(w_i^\top \mathbf{x})$, and for deep neural networks the bits are generated by thresholding the activations of the output layer.

Inspired by the online nature of the problem and recent advances in stochastic optimization, we derive gradient descent rules for $\mathcal{L}$. The entropy-based mutual information is differentiable with respect to the entries of $p_{\mathcal{D}}$, $p^+_{\mathcal{D}}$ and $p^-_{\mathcal{D}}$, and, as mentioned before, these discrete distributions can be estimated via histogram binning. However, it is not clear how to differentiate histogram binning to generate gradients for model parameters. We describe a differentiable histogram binning technique next.

Differentiable Histogram Binning

We borrow ideas from [23] and estimate $p^+_{\mathcal{D}}$, $p^-_{\mathcal{D}}$ and $p_{\mathcal{D}}$ using a differentiable histogram binning technique. For b-bit Hamming distances, we use $(K+1)$-bin normalized histograms with bin centers $v_0 = 0, ..., v_K = b$ and uniform bin width $\Delta = b/K$, where $K = b$ by default. Consider, for example, the k-th entry in $p^+_{\mathcal{D}}$, denoted as $p^+_{\mathcal{D},k}$. It can be estimated as

$$p^+_{\mathcal{D},k} = \frac{1}{|\oplus|} \sum_{\mathbf{x}\in\oplus} \delta_{\mathbf{x},k}, \tag{8}$$

where $\delta_{\mathbf{x},k}$ records the contribution of $\mathbf{x}$ to bin k. It is obtained by interpolating $d_\Phi(\mathbf{x}, \hat{\mathbf{x}})$ using a triangular kernel:

$$\delta_{\mathbf{x},k} = \begin{cases} (d_\Phi(\mathbf{x},\hat{\mathbf{x}}) - v_{k-1})/\Delta, & d_\Phi(\mathbf{x},\hat{\mathbf{x}}) \in [v_{k-1}, v_k], \\ (v_{k+1} - d_\Phi(\mathbf{x},\hat{\mathbf{x}}))/\Delta, & d_\Phi(\mathbf{x},\hat{\mathbf{x}}) \in [v_k, v_{k+1}], \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

This binning process admits subgradients:

$$\frac{\partial \delta_{\mathbf{x},k}}{\partial d_\Phi(\mathbf{x},\hat{\mathbf{x}})} = \begin{cases} 1/\Delta, & d_\Phi(\mathbf{x},\hat{\mathbf{x}}) \in [v_{k-1}, v_k], \\ -1/\Delta, & d_\Phi(\mathbf{x},\hat{\mathbf{x}}) \in [v_k, v_{k+1}], \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Gradients of MI

We now derive the gradient of $\mathcal{I}$ with respect to the output of the hash mapping, $\Phi(\hat{\mathbf{x}})$. Using standard chain rule, we can first write

$$\frac{\partial \mathcal{I}}{\partial \Phi(\hat{\mathbf{x}})} = \sum_{k=0}^{K} \left[\frac{\partial \mathcal{I}}{\partial p^+_{\mathcal{D},k}} \frac{\partial p^+_{\mathcal{D},k}}{\partial \Phi(\hat{\mathbf{x}})} + \frac{\partial \mathcal{I}}{\partial p^-_{\mathcal{D},k}} \frac{\partial p^-_{\mathcal{D},k}}{\partial \Phi(\hat{\mathbf{x}})} \right]. \tag{11}$$

We focus on terms involving $p^+_{\mathcal{D},k}$, and omit derivations for $p^-_{\mathcal{D},k}$ due to symmetry. For $k = 0, \dots, K$, we have

$$\frac{\partial \mathcal{I}}{\partial p^+_{\mathcal{D},k}} = -\frac{\partial H(\mathcal{D}|\mathcal{C})}{\partial p^+_{\mathcal{D},k}} + \frac{\partial H(\mathcal{D})}{\partial p^+_{\mathcal{D},k}} \tag{12}$$

$$= p^+(\log p^+_{\mathcal{D},k} + 1) - (\log p_{\mathcal{D},k} + 1)\frac{\partial p_{\mathcal{D},k}}{\partial p^+_{\mathcal{D},k}} \tag{13}$$

$$= p^+(\log p^+_{\mathcal{D},k} - \log p_{\mathcal{D},k}), \tag{14}$$

where we used the fact that $p_{\mathcal{D},k} = p^+ p^+_{\mathcal{D},k} + p^- p^-_{\mathcal{D},k}$, with p^+ and p^- being shorthands for the priors $P(\mathcal{C} = 1)$ and $P(\mathcal{C} = 0)$. We next tackle the term $\partial p^+_{\mathcal{D},k} / \partial \Phi(\hat{\mathbf{x}})$ in Eq. 11. From the definition of $p^+_{\mathcal{D},k}$ in Eq.8, we have

$$\frac{\partial p^+_{\mathcal{D},k}}{\partial \Phi(\hat{\mathbf{x}})} = \frac{1}{|\oplus|} \sum_{\mathbf{x}\in\oplus} \frac{\partial \delta_{\mathbf{x},k}}{\partial \Phi(\hat{\mathbf{x}})} \tag{15}$$

$$= \frac{1}{|\oplus|} \sum_{\mathbf{x}\in\oplus} \frac{\partial \delta_{\mathbf{x},k}}{\partial d_\Phi(\mathbf{x},\hat{\mathbf{x}})} \frac{\partial d_\Phi(\mathbf{x},\hat{\mathbf{x}})}{\partial \Phi(\hat{\mathbf{x}})} \tag{16}$$

$$= \frac{1}{|\oplus|} \sum_{\mathbf{x}\in\oplus} \frac{\partial \delta_{\mathbf{x},k}}{\partial d_\Phi(\mathbf{x},\hat{\mathbf{x}})} \frac{-\Phi(\mathbf{x})}{2}. \tag{17}$$

Note that $\partial \delta_{\mathbf{x},k} / \partial d_\Phi(\mathbf{x}, \hat{\mathbf{x}})$ is already given in Eq. 10. For the last step, we used the definition of d_Φ in Eq. 1.

Lastly, to back-propagate gradients to Φ's inputs and ultimately model parameters, we approximate the discontinuous sign function with sigmoid, which is a standard technique in hashing, *e.g.* [1, 12, 16].

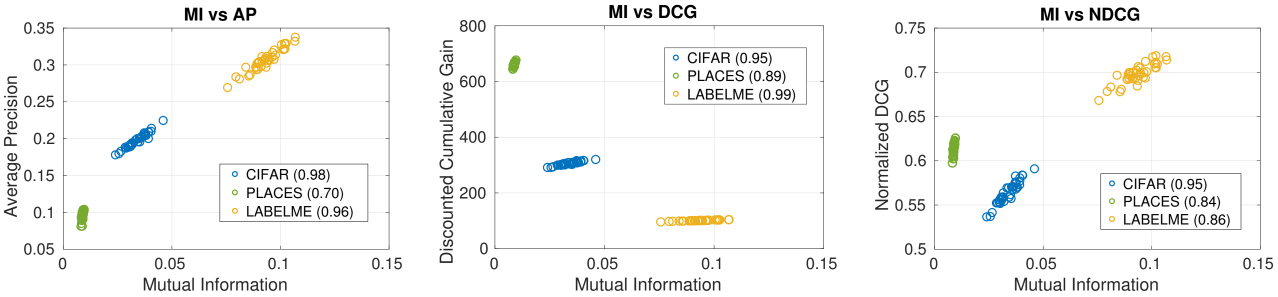

Figure 3: We show Pearson correlation coefficients between mutual information (MI) and AP, DCG, and NDCG, evaluated on the CIFAR-10, LabelMe, and Places205 datasets. We sample 100 instances to form the query set, and use the rest to populate the hash table. The hash mapping parameters are randomly sampled from a Gaussian, similar to LSH [4]. Each experiment is conducted 50 times. There exist strong correlations between MI and the standard metrics.

3.3. Benefits of MI

For monitoring the performance of hashing algorithms, it appears that one could directly use standard ranking metrics, such as Average Precision (AP), Discounted Cumulative Gain (DCG), and Normalized DCG (NDCG) [17]. Here, we discuss the benefits of instead using mutual information. First, we note that there exist strong correlations between mutual information and standard ranking metrics. Fig. 3 demonstrates the Pearson correlation coefficients between MI and AP, DCG, and NDCG, on three benchmarks. Although a theoretical analysis is beyond the scope of this work, empirically we find that MI serves as an efficient and general-purpose ranking surrogate.

We also point out the lower computational complexity of mutual information. Let n be the reservoir set size. Computing Eq. 6 involves estimating discrete distributions via histogram binning, and takes $\mathcal{O}(n)$ time for each reservoir item, since $\mathcal{D}$ only takes discrete values from $\{0, 1, \ldots, b\}$, In contrast, ranking measures such as AP and NDCG have $\mathcal{O}(n \log n)$ complexity due to sorting, which render them disadvantageous.

Finally, Sec. 3.2 showed that the mutual information objective is suitable for direct, gradient-based optimization. In contrast, optimizing metrics like AP and NDCG is much more challenging as they are non-differentiable, and existing works usually resort to optimizing their surrogates [13, 26, 29] rather than gradient-based optimization. Furthermore, mutual information itself is essentially parameter-free, whereas many other hashing formulations require (and can be sensitive to) tuning parameters, such as thresholds or margins [18, 27], quantization strength [12, 15, 20], *etc*.

4. Experiments

We evaluate our approach on three widely used image benchmarks. We first describe the datasets and experimental setup in Sec. 4.1. We evaluate the mutual information update criterion in Sec. 4.2 and the mutual information based objective function for learning hash mappings in Sec. 4.3. Our implementation is publicly available at `https://github.com/fcakir/mihash`.

4.1. Datasets and Experimental Setup

CIFAR-10 is a widely-used dataset for image classification and retrieval, containing 60K images from 10 different categories [7]. For feature representation, we use CNN features extracted from the $fc7$ layer of a VGG-16 network [21] pre-trained on ImageNet.

Places205 is a subset of the large-scale Places dataset [32] for scene recognition. Places205 contains 2.5M images from 205 scene categories. This is a very challenging dataset due to its large size and number of categories, and it has not been studied in the hashing literature to our knowledge. We extract CNN features from the $fc7$ layer of an AlexNet [8] pre-trained on ImageNet, and reduce the dimensionality to 128 using PCA.

LabelMe. The 22K LabelMe dataset [19, 22] has 22,019 images represented as 512-dimensional GIST descriptors. This is an unsupervised dataset without labels, and standard practice uses the Euclidean distance to determine neighbor relationships. Specifically, $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered neighbor pairs if their Euclidean distance is within the smallest 5% in the training set. For a query, the closest 5% examples are considered true neighbors.

All datasets are randomly split into a retrieval set and a test set, and a subset from the retrieval set is used for learning hash functions. Specifically, for **CIFAR-10**, the test set has 1K images and the retrieval set has 59K. A random subset of 20K images from the retrieval set is used for learning, and the size of the reservoir is set to 1K. For **Places205**, we sample 20 images from each class to construct a test set of 4.1K images, and use the rest as the retrieval set. A random subset of 100K images is used to for learning, and the reservoir size is 5K. For **LabelMe**, the dataset is split into retrieval and test sets with 20K and 2K samples, respectively. Similar to CIFAR-10, we use a reservoir of size 1K.

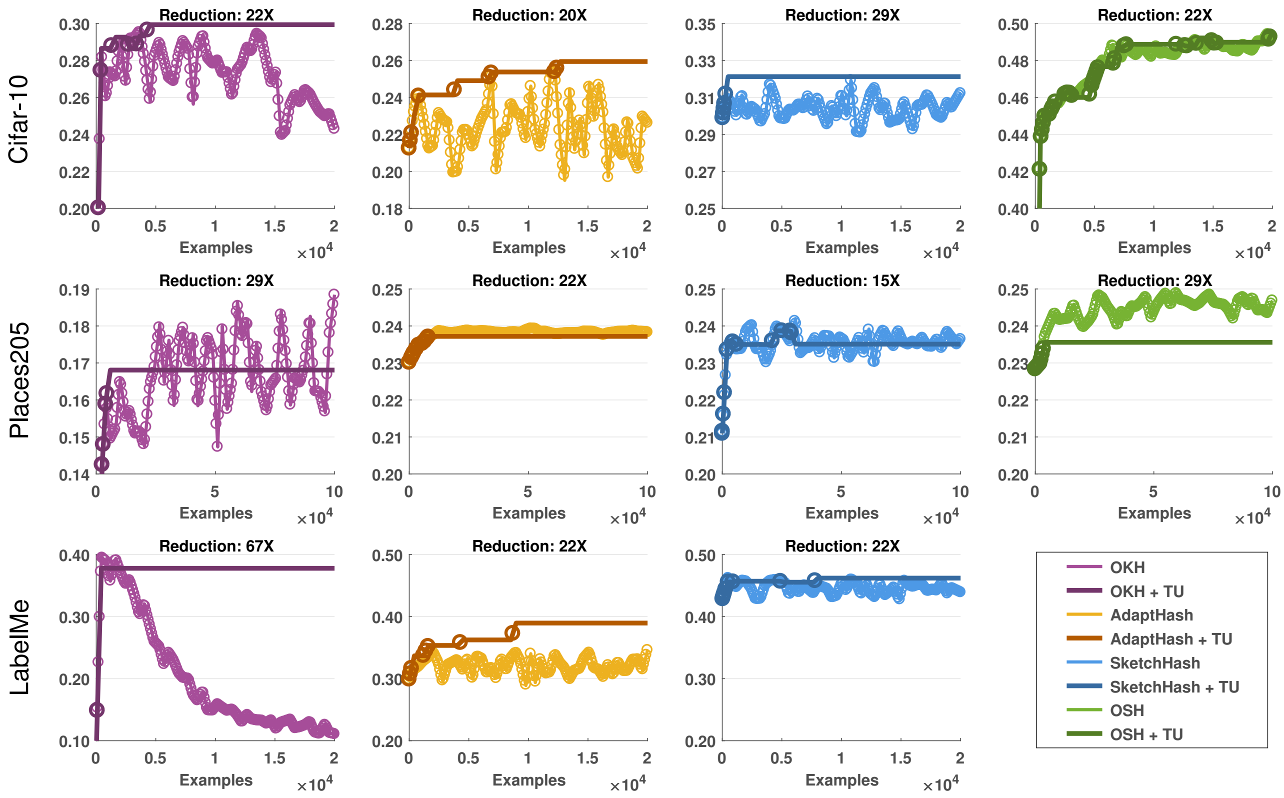

Figure 4: Retrieval mAP *vs.* number of processed training examples for four hashing methods on the three datasets, with and without Trigger Update (TU). We use default threshold $\theta = 0$ for TU. Circles indicate hash table updates, and the ratio of reduction in the number of updates is shown in the titles. TU substantially reduces the number of updates while having a stabilizing effect on the retrieval performance. Note: since OSH [2] assumes supervision in terms of class labels, it is not applicable to the unsupervised LabelMe dataset.

For online hashing experiments, we run three randomized trials for each experiment and report averaged results. To evaluate retrieval performances, we adopt the widely-used mean Average Precision (mAP). Due to the large size of Places205, mAP is very time-consuming to compute, and we compute mAP on the top 1000 retrieved examples (mAP@1000), as done in [15].

4.2. Evaluation: Update Criterion

We evaluate our mutual information based update criterion, the Trigger Update module (TU). We apply TU to all existing online hashing methods known to us: Online Kernel Hashing (OKH) [6], Online Supervised Hashing (OSH) [2], Adaptive Hashing (AdaptHash) [1] and Online Sketching Hashing (SketchHash) [11]. We use publicly available implementations of all methods. The hash code length is fixed at 32 bits.

As our work is the first in addressing the hash table update criterion for online hashing, we compare to a data-agnostic baseline, which updates the hash table at a fixed rate. The rate is controlled by a parameter U, referred to as the “update interval”: after processing every U examples, the baseline unconditionally triggers an update, while TU makes a decision using the mutual information criterion. For each dataset, U is set such that the baseline updates 201 times in total. This ensures that the baseline is never too outdated, but updates are still fairly infrequent: in all cases, the smallest U is 100.

Results for the Trigger Update module. Fig. 4 depicts the retrieval mAP over time for all four online hashing methods considered, on three datasets, with and without incorporating TU. We can clearly observe a significant reduction in the number of hash table updates, between one and two orders of magnitude in all cases. For example, the number of hash table updates is reduced by a factor of 67 for the OKH method on LabelMe.

The quality-based update criterion is particularly important for hashing methods that may yield inferior hash mappings due to noisy data and/or imperfect learning techniques. In other words, TU can be used to filter updates to the hash mapping with negative or small improvement. This has a stabilizing effect on the mAP curve, notably for OKH

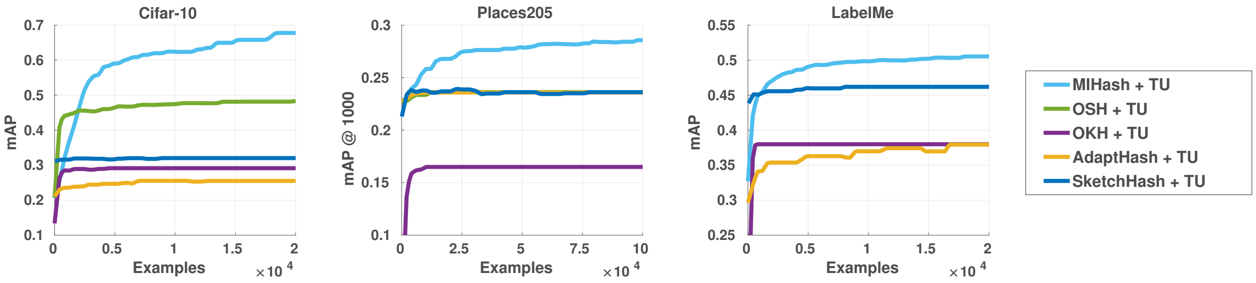

Figure 5: Online hashing performance comparison on three datasets, where all methods use the Trigger Update module (TU) with default threshold $\theta = 0$. MIHash clearly outperforms other competing methods. OSH, AdaptHash, and SketchHash perform very similarly on Places205, thus their curves overlap.

and AdaptHash. For OSH, which appears to stably improve over time, TU nevertheless significantly reduces revisits to the hash table while maintaining its performance.

All results in Fig. 4 are obtained using the default threshold parameter $\theta = 0$, defined in Eq. 5. We do not tune θ in order to show general applicability. We also discuss the impact of the reservoir set $\mathcal{R}$. There is a trade-off regarding the size of $\mathcal{R}$: a larger $\mathcal{R}$ leads to better approximation but increases the overhead. Nevertheless, we observed robust and consistent results with $|\mathcal{R}|$ not exceeding 5% of the size of the training stream.

4.3. Evaluation: Learning Objective

We evaluate the mutual information based hashing objective. We name our method MIHash, and train it using stochastic gradient descent (SGD). This allows it to be applied to both the online setting and batch setting in learning hash functions.

During minibatch-based SGD, to compute the mutual information objective in Eq. 7 and its gradients, we need access to the sets $\oplus_{\hat{\mathbf{x}}}$, $\ominus_{\hat{\mathbf{x}}}$ for each considered $\hat{\mathbf{x}}$, in order to estimate $p^+_\mathcal{D}$ and $p^-_\mathcal{D}$. For the online setting in Sec. 4.3.1, a standalone reservoir set $\mathcal{R}$ is assumed as in the previous experiment, and we partition $\mathcal{R}$ into $\{\oplus_{\hat{\mathbf{x}}}, \ominus_{\hat{\mathbf{x}}}\}$ with respect to each incoming $\hat{\mathbf{x}}$. In this case, even a batch size of 1 can be used. For the batch setting in Sec. 4.3.2, $\{\oplus_{\hat{\mathbf{x}}}, \ominus_{\hat{\mathbf{x}}}\}$ are defined within the same minibatch as $\hat{\mathbf{x}}$.

4.3.1 Online Setting

We first consider an online setting that is the same as in Sec. 4.2. We compare against other online hashing methods: OKH, OSH, AdaptHash and SketchHash. All methods are equipped with the TU module with the default threshold $\theta = 0$, which has been demonstrated to work well.

Results for Online Setting. We first show the mAP curve comparisons in Fig. 5. For competing online hashing methods, the curves are the same as the ones with TU in Fig. 4, and we remove markers to avoid clutter. MIHash clearly outperforms other online hashing methods on all three datasets, and shows potential for further improvement with more training data. The combination of TU and MIHash gives a complete online hashing system that enjoys a superior learning objective with a plug-in update criterion that improves efficiency.

We next give insights into the distribution-separating effect from optimizing mutual information. In Fig. 6, we plot the conditional distributions $p^+_\mathcal{D}$ and $p^-_\mathcal{D}$ averaged on the CIFAR-10 test set, before and after learning MIHash with the 20K training examples. Before learning, with a randomly initialized hash mapping, $p^+_\mathcal{D}$ and $p^-_\mathcal{D}$ exhibit high overlap. After learning, MIHash achieves good separation between $p^+_\mathcal{D}$ and $p^-_\mathcal{D}$: the overlap reduces significantly, and the mass of $p^+_\mathcal{D}$ is pushed towards 0. This separation is reflected in the large improvement in mAP (0.68 vs. 0.22).

In contrast with the other methods, the mutual information formulation is parameter-free. For instance, there is no threshold parameter that requires separating $p^+_\mathcal{D}$ and $p^-_\mathcal{D}$ at a certain distance value. Likewise, there is no margin parameter that dictates the amount of separation in absolute terms. Such parameters usually need to be tuned to fit to data, whereas the optimization of mutual information is automatically guided by the data itself.

4.3.2 Batch Setting

To further demonstrate the potential of MIHash, we consider the batch learning setting, where the entire training set is available at once. We compare against state-of-the-art batch formulations, including: Supervised Hashing with Kernels (SHK) [16], Fast Supervised Hashing with Decision Trees (FastHash) [14], Supervised Discrete Hashing (SDH) [20], Efficient Training of Very Deep Neural Networks (VDSH) [30], Deep Supervised Hashing with Pairwise Labels (DPSH) [12] and Deep Supervised Hashing with Triplet Labels (DTSH) [27]. These competing methods have shown to outperform earlier and other work such as [5, 9, 18, 28, 10, 31]. We focus on comparisons on the CIFAR-10 dataset, which is the canonical benchmark for supervised hashing. Similar to [27], we consider two exper-

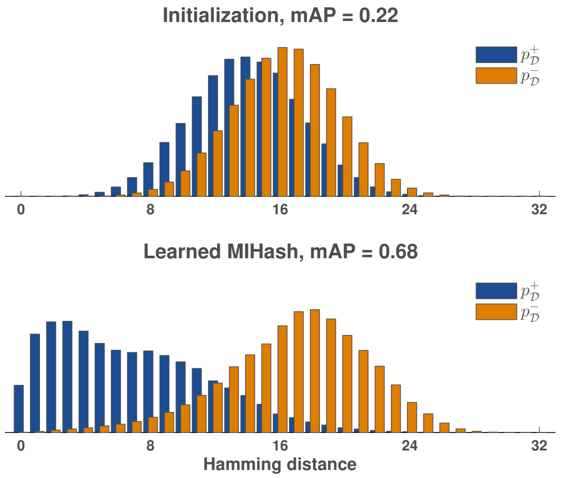

Figure 6: We plot the distributions $p_{\mathcal{D}}^{+}$ and $p_{\mathcal{D}}^{-}$, averaged on the CIFAR-10 test set, before and after learning MIHash with 20K training examples. Optimizing the mutual information objective substantially reduces the overlap between them, resulting in state-of-the-art mAP for the online setting, as shown in Fig. 5.

		Code Length			
	Method	12	24	32	48
Setting 1	SHK	0.497	0.615	0.645	0.682
	SDH	0.521	0.576	0.589	0.592
	VDSH	0.523	0.546	0.537	0.554
	DPSH	0.420	0.518	0.538	0.553
	DTSH	0.617	0.659	0.689	0.702
	FastHash	0.632	0.700	0.724	0.738
	MIHash[1]	0.524	0.563	0.597	0.609
	MIHash	**0.683**	**0.720**	**0.727**	**0.746**
	Method	16	24	32	48
Setting 2	DPSH[2]	0.763	0.781	0.795	0.807
	DTSH[2]	0.915	0.923	0.925	0.926
	DPSH	0.908	0.909	0.917	0.932
	DTSH	0.916	0.924	0.927	0.934
	MIHash	**0.929**	**0.933**	**0.938**	**0.942**

[1] Results after a single training epoch.

[2] Results as reported in DPSH [12] and DTSH [27].

Table 1: Comparison against state-of-the-art hashing methods on CIFAR-10. We report mean Average Precision (mAP) on the test set, with best results in **bold**. See text for the details of the two experimental settings.

imental settings, which we detail below.

Setting 1: 5K training examples are sampled for learning hash mappings, and 1K examples are used as the test set. All methods learn shallow models on top of *fc7* features from a VGG-16 network [21] pretrained on ImageNet. For three gradient-based methods (DPSH, DTSH, and MIHash), this means learning linear hash functions. Note that VDSH uses customized architectures consisting of only fully-connected layers, and it is unclear how to adapt it to use standard architectures; we used its full model with 16 layers and 1024 nodes per layer.

Setting 2: We use the full training set of size 50K and test set of size 10K. We focus on comparing the end-to-end performance of MIHash against two recent leading methods: DPSH and DTSH, using the same VGG-F network architecture [3] that they are trained on.

We use publicly available implementations for the compared methods, and exhaustively search parameter settings for them. For MIHash, the minibatch size is set to 100 and 250 in Settings 1 and 2, respectively. We use SGD with momentum, and decrease the learning rate when the training loss saturates. See supplementary material for more details.

Results for Batch Setting. In Table 1, we list batch learning results for all methods. In Setting 1, MIHash outperforms all competing methods in terms of mAP, in some cases with only a single training epoch (*e.g.* against VDSH, DPSH). This suggests that mutual information is a more effective learning objective for hashing. MIHash learns a linear layer on the input features, while some other methods can learn non-linear hash functions: for instance, the closest competitor, FastHash, is a two-step hashing method based on sophisticated binary code inference and boosted trees.

In Setting 2, with end-to-end finetuning, MIHash significantly outperforms DPSH and DTSH, the two most competitive deep hashing methods, and sets the current state-of-the-art for CIFAR-10. Again, note that MIHash has no tuning parameters in its objective function. In contrast, both DPSH and DTSH have parameters to control the quantization strength that need to be tuned.

5. Conclusion

We advance the state-of-the-art for online hashing in two aspects. In order to resolve the issue of hash table updates in online hashing, we define a quality measure using the mutual information between variables induced by the hash mapping. This measure is efficiently computable, correlates well with standard evaluation metrics, and leads to consistent computational savings for existing online hashing methods while maintaining their retrieval accuracy. Inspired by these strong results, we further propose a hashing method MIHash, by optimizing mutual information as an objective with stochastic gradient descent. In both online and batch settings, MIHash achieves superior performance compared to state-of-the-art hashing techniques.

Acknowledgements

This research was supported in part by a BU IGNITION award, US NSF grant 1029430, and gifts from NVIDIA.

References

[1] F. Cakir and S. Sclaroff. Adaptive hashing for fast similarity search. In *Proc. IEEE International Conf. on Computer Vision (ICCV)*, 2015.

[2] F. Cakir and S. Sclaroff. Online supervised hashing. In *Proc. IEEE International Conf. on Image Processing (ICIP)*, 2015.

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference (BMVC)*, 2014.

[4] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. International Conf. on Very Large Data Bases (VLDB)*, 1999.

[5] 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.

[6] L.-K. Huang, Q. Y. Yang, and W.-S. Zheng. Online hashing. In *Proc. International Joint Conf. on Artificial Intelligence (IJCAI)*, 2013.

[7] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

[8] 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.

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

[10] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[11] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu. Online sketching hashing. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[12] W.-J. Li, S. Wang, and W.-C. Kang. Feature learning based deep supervised hashing with pairwise labels. In *Proc. International Joint Conf. on Artificial Intelligence (IJCAI)*, 2016.

[13] G. Lin, F. Liu, C. Shen, J. Wu, and H. T. Shen. Structured learning of binary codes with column generation for optimizing ranking measures. *International Journal of Computer Vision (IJCV)*, pages 1–22, 2016.

[14] 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.

[15] K. Lin, J. Lu, C.-S. Chen, and J. Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] 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.

[17] C. D. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval. 2008.

[18] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proc. International Conf. on Machine Learning (ICML)*, 2011.

[19] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 2008.

[20] F. Shen, C. S. Wei, L. Heng, and T. Shen. Supervised discrete hashing. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[22] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008.

[23] E. Ustinova and V. Lempitsky. Learning deep embeddings with histogram loss. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 4170–4178, 2016.

[24] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[25] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *CoRR*.

[26] Q. Wang, Z. Zhang, and L. Si. Ranking preserving hashing for fast similarity search. In *Proc. International Joint Conf. on Artificial Intelligence (IJCAI)*, 2015.

[27] Y. Wang, Xiaofang Shi and K. M. Kitani. Deep supervised hashing with triplet labels. In *Proc. Asian Conf. on Computer Vision (ACCV)*, 2016.

[28] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, volume 1, page 2, 2014.

[29] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proc. ACM Conf. on Research & Development in Information Retrieval (SIGIR)*, pages 271–278. ACM, 2007.

[30] Z. Zhang, Y. Chen, and V. Saligrama. Efficient training of very deep neural networks for supervised hashing. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[31] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[32] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2014.