Hashing as Tie-Aware Learning to Rank

Kun He, Fatih Cakir, Sarah A. Bargal, Stan Sclaroff
Department of Computer Science, Boston University
Boston, MA 02215
{hekun, fcakir, sbargal, sclaroff}@cs.bu.edu

Abstract

We formulate the problem of supervised hashing, or learning binary embeddings of data, as a learning to rank problem. Specifically, we optimize two common ranking-based evaluation metrics, Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG). Observing that ranking with the discrete Hamming distance naturally results in ties, we propose to use tie-aware versions of ranking metrics in both the evaluation and the learning of supervised hashing. For AP and NDCG, we derive continuous relaxations of their tie-aware versions, and optimize them using stochastic gradient ascent with deep neural networks. Our results establish the new state-of-the-art for tie-aware AP and NDCG on common hashing benchmarks.

1 Introduction

In this paper, we tackle the problem of supervised hashing, which is concerned with learning binary embeddings of data in order to enable fast nearest neighbor search. Our goal is to design and optimize appropriate learning objectives that match the evaluation metrics used at test time, which has been recognized as difficult in the literature. Supervised hashing is usually evaluated assuming an information retrieval setup, where a set of queries is retrieved against a database using the learned binary representation. Retrieval performance can be measured using ranking-based metrics like Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG) [15]. Thus, we are interested in optimizing such ranking metrics for supervised hashing.

In the context of hashing, an important observation is that retrieval results ranked by discrete-valued Hamming distances generally contain many *ties*, and different tie-breaking strategies can lead to different, or even unfair, results (see Figure 1 and Section 3.2). Although this issue is known in the information retrieval community [17], the learning to hash literature still relatively lacks tie-awareness, and current evaluation criteria do not guard against exploitation of tie-breaking strategies. Thus, we first advocate incorporating tie-awareness in evaluating supervised hashing, and give evaluation procedures for tie-aware ranking metrics. By taking advantage of the discreteness of Hamming distances, such metrics can be evaluated efficiently in linear time.

Our natural next step is to learn hash functions by optimizing tie-aware ranking metrics. This can be seen as an instance of learning to rank, but restricted to the hashing problem. To solve the resulting discrete (NP-hard) optimization problems, we apply continuous relaxation, which enables us to perform gradient-based optimization with deep architectures. We specifically study the optimization of tie-aware AP and NDCG, and our results establish the new state-of-the-art for these metrics on common hashing benchmarks.

2 Related Work

Hashing is a widely used approach for practical nearest neighbor search, thanks to the efficiency of evaluating Hamming distances using bitwise operations. It has been theoretically demonstrated [1]

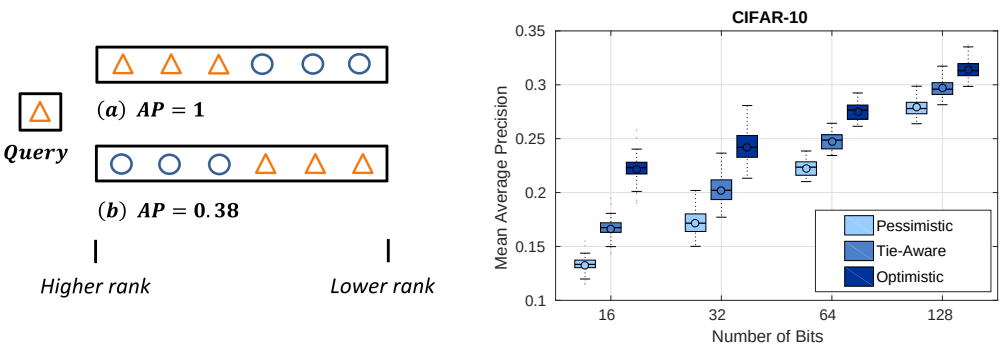

Figure 1: Different tie-breaking strategies in evaluating supervised hashing. Left: toy example where all items are tied. The optimistic tie-breaking strategy achieves perfect AP of 1, while the pessimistic achieves only 0.38. Right: on the CIFAR-10 dataset, we show box plots of mean AP values from 50 random trials, using the LSH method [5]. The gap between pessimistic and optimistic versions can be high, especially for short codes, while tie-aware AP (Section 3.2) gives faithful estimates.

that data-dependent hashing methods outperform data-independent ones such as Locality Sensitive Hashing [5]. This phenomenon is also widely observed in the growing literature on learning to hash. Our work falls into the category of affinity-based hashing [8, 14, 19] as we consider supervision in the form of pairwise affinities. Regarding optimization, the discrete nature of hashing usually results in NP-hard optimization problems. Our solution uses continuous relaxations, which is in line with relaxation-based methods, *e.g.* [8, 14], but differs from alternating methods that attempt to preserve the discrete constraints, *e.g.* [13, 18, 19], and two-step methods [12].

Supervised hashing can be considered a special case of metric learning [18], which itself can be formulated as a learning to rank problem [11, 16]. Optimizing ranking metrics such as AP and NDCG has received much attention in the information retrieval and learning to rank literature. For instance, it is well-known that a surrogate on AP can be optimized with the structural SVM [26], and a bound optimization method exists specifically for NDCG [23]. Alternatively, there are gradient-based methods based on smoothing or approximating the metrics [2, 6, 9]. These methods did not consider applications in hashing.

Retrieving with the Hamming distance naturally results in tied rankings, but this fact is not widely taken into account in the hashing literature. One existing solution is to sidestep this issue using weighted Hamming distances [27], but at the cost of reduced efficiency. Fortunately, tie-aware versions of common ranking metrics have been derived in the information retrieval community [17]. We show how to efficiently evaluate tie-aware ranking metrics for discrete hashing, based on the use of counting sort on Hamming distances.

Several works have considered optimizing ranking metrics in the learning to hash literature. However, none of them are tie-aware, and not all use gradient-based optimization. For example, [13] optimizes ranking surrogates using the structural SVM, [21] optimizes precision at the top of the ranking, and [24] optimizes approximated NDCG. Notably, [13] also exploits counting sort on Hamming distances to speed up loss-augmented inference for the NDCG surrogate loss. Although many recent hashing methods employ gradient optimization and/or deep neural networks, *e.g.* [10, 25], their learning objectives are usually not designed to match the evaluation metrics. To enable gradient optimization for tie-aware metrics, we use a recent result on differentiable histogram binning [22].

3 Hashing as Tie-Aware Learning to Rank

3.1 Preliminaries

Learning to hash. In the supervised hashing problem, we wish to learn a hash mapping $\Phi : \mathcal{X} \to \mathcal{H}^b$, where $\mathcal{X}$ is the feature space, and $\mathcal{H}^b = \{-1, +1\}^b$ is the b-dimensional Hamming space. A hash mapping Φ induces the Hamming distance $d_\Phi(\cdot, \cdot)$, which takes values in $\{0, 1, \ldots, b\}$. The form of supervision we consider is *pairwise affinities*, which is standard in supervised hashing. Formally, we

assume an affinity oracle $\mathcal{A}$, where $\mathcal{A}(x_i, x_j) > 0$ if $x_i, x_j \in \mathcal{X}$ are similar (the value indicates the strength of similarity), and $\mathcal{A}(x_i, x_j) = 0$ otherwise.

In this paper, we restrict $\mathcal{A}$ to take values from a finite set $\mathcal{V}$, which includes two important special cases. First, *binary affinities*, *i.e.*, $\mathcal{V} = \{0, 1\}$, are extensively studied in the current hashing literature, which in practice can be derived from thresholding the Euclidean distance in $\mathcal{X}$ or from agreement of class labels. The second case is when $\mathcal{V}$ consists of non-negative integers, or *multi-level affinities*, which is frequently considered in information retrieval tasks, including in web search engines.

Throughout this paper we assume the setup where a query $x_q \in \mathcal{X}$ is retrieved against some retrieval set $S \subseteq \mathcal{X}$. Retrieval is performed by ranking the instances in S by increasing distance to x_q, where the distance we use is the Hamming distance d_Φ induced from hash mapping Φ. Unless otherwise noted, we implicitly assume dependency on x_q, S, and Φ, and omit them in our notation.

Ranking-based evaluation. The ranking can be represented by an index vector R, whose elements form a permutation of $\{1, \ldots, |S|\}$. Ranking-based metrics usually measure some form of agreement between the ranking and ground truth affinities. We use the shorthand $\mathcal{A}_q(i) = \mathcal{A}(x_q, x_i)$ below.

First, in the case of binary affinity, let the number of x_q's *neighbors* be $N^+ = |\{x_i \in S | \mathcal{A}_q(i) = 1\}|$. Average Precision (AP) averages the precision at cutoff k over all cutoffs:

$$\text{AP}(R) = \frac{1}{N^+} \sum_{k=1}^{|S|} \mathcal{A}_q(R_k) \left[\frac{1}{k} \sum_{j=1}^{k} \mathcal{A}_q(R_j) \right]. \tag{1}$$

Next, for integer-valued affinities, Discounted Cumulative Gain (DCG) is defined as

$$\text{DCG}(R) = \sum_{k=1}^{|S|} G(\mathcal{A}_q(R_k)) D(k), \quad \text{where } G(a) = 2^a - 1, D(k) = \frac{1}{\log_2(k+1)}. \tag{2}$$

G and D are called gain and discount, respectively. The normalized variant (NDCG) normalizes to the range of $[0, 1]$, where the normalizing factor is $\max_R \text{DCG}(R)$.

3.2 The Issue of Ties, and Tie-Aware Metrics

When evaluating information retrieval systems, special attention is required when there exist ties in the distances [17]. In this case, the ranking R is not unique as the tied items can be ordered arbitrarily, and the tie-breaking strategy may have a sizable impact on the result. We demonstrate a toy example with binary affinity in Figure 1, where all retrievals have the same distance to a query. The *optimistic* tie-breaking strategy ranks all neighbors in front and achieves AP of 1, while the opposite *pessimistic* strategy's AP value can be very low if the non-neighbors outnumber the neighbors.

Ties appear ubiquitously in hashing, since the Hamming distance only takes integer values in $\{0, 1, \ldots, b\}$. The difference between optimistic and pessimistic tie-breaking strategies is especially large when b is small compared to the dataset's size. In Figure 1, we also show a numerical experiment comparing different versions of AP on the CIFAR-10 dataset [7]. We sample 1000 examples as the test set, retrieve them against the rest using the LSH method [5], and repeat 50 times. The absolute difference in the mean AP value can be as high as 0.1 for short codes (*e.g.* 16 bits), and remains noticeable for longer codes. Unfortunately, current hashing evaluation protocols do not guard against the potential exploitation of reporting optimistic values when ground truth affinities are known, although we have not found evidence of such exploitation in the literature.

To make ranking metrics tie-aware, one needs to average the value of the metric over all possible permutations of tied items. Tie-aware versions of common ranking metrics can indeed be derived in closed form [17]; we refer readers there for detailed derivations. Here, we describe how to efficiently compute them by exploiting the structure of Hamming distance, focusing on AP and NDCG. Recall that we rank a retrieval set S with respect to a query x_q. With integer-valued Hamming distances, we redefine the ranking R to be a collection of $(b+1)$ "ties", *i.e.* $R = \{R^{(0)}, \ldots, R^{(b)}\}$, where $R^{(d)} = \{i | d_\Phi(x_q, x_i) = d\}$. Below, we will make use of discrete histograms conditioned on affinity values, $(n_{0,v}, \ldots, n_{b,v})$, where $n_{d,v} = |R^{(d)} \cap \{i | \mathcal{A}_q(i) = v\}|, \forall v \in \mathcal{V}$, and their cumulative sums $(N_{0,v}, \ldots, N_{b,v})$ where $N_{d,v} = \sum_{i \le d} n_{i,v}$. We also define the total histograms as $n_d = \sum_{v \in \mathcal{V}} n_{d,v}$ and $N_d = \sum_{i \le d} n_i$.

Tie-aware AP. We denote AP_T as the tie-aware version of AP. For binary affinities $\mathcal{V} = \{0, 1\}$, let $n_d^+ = n_{d,1}, n_d^- = n_{d,0}$, and N_d^+, N_d^- be their cumulative sums. AP_T decomposes additively over each tie $R^{(d)}$, whose contribution $\text{AP}_\text{T}(R^{(d)})$ can be written as

$$\text{AP}_\text{T}(R^{(d)}) \;=\; \sum_{t=N_{d-1}+1}^{N_d} \frac{N_{d-1}^+ + (t - N_{d-1} - 1)\frac{n_d^+ - 1}{n_d - 1} + 1}{t} \frac{n_d^+}{n_d} \;\triangleq\; \frac{n_d^+}{n_d} \sum_{t=N_{d-1}+1}^{N_d} \beta_d(t). \tag{3}$$

And overall, $\text{AP}_\text{T}(R) = \frac{1}{N^+} \sum_{d=0}^{b} \text{AP}_\text{T}(R^{(d)})$.

Tie-aware NDCG. We first consider the un-normalized DCG. Like AP_T, its tie-aware version (denoted DCG_T) also decomposes additively over the ties,

$$\text{DCG}_\text{T}(R^{(d)}) \;=\; \sum_{i \in R^{(d)}} \frac{G(\mathcal{A}_q(i))}{n_d} \sum_{t=N_{d-1}+1}^{N_d} D(t) \;=\; \sum_{v \in \mathcal{V}} \frac{G(v) n_{d,v}}{n_d} \sum_{t=N_{d-1}+1}^{N_d} D(t). \tag{4}$$

And $\text{DCG}_\text{T}(R) = \sum_{d=0}^{b} \text{DCG}_\text{T}(R^{(d)})$. For NDCG_T, the normalizing factor is unaffected by ties.

Time complexity Analysis. Let $|S| = N$. Given the Hamming distances $\{d_\Phi(x_q, x) | x \in S\}$, the first step is to generate the ranking R, or populate the ties $\{R^{(d)}\}$. This step is essentially the *counting sort* for integers, which has $O(bN)$ time complexity. Computing either AP_T or DCG_T then takes $O(\sum_d n_d) = O(N)$ time, which makes the total time complexity $O(bN)$. In our formulation, the number of bits b is a constant, and therefore the complexity is linear in N. In contrast, for real-valued distances, sorting generally takes $O(N \log N)$ time and is the dominating factor.

For NDCG, computing the normalizing factor requires sorting the gain values in descending order. Under the assumption that the affinity values are integers from $\mathcal{V}$, the number of unique gain values is $|\mathcal{V}|$, and counting sort can be applied in $O(|\mathcal{V}|N)$ time. The total time complexity is thus $O((b + |\mathcal{V}|)N)$, which is also linear in N provided that $|\mathcal{V}|$ is known.

4 Optimizing Tie-Aware Metrics

In this section, we describe our approach to optimizing tie-aware ranking metrics. We are interested in performing gradient-based optimization to take advantage of end-to-end deep architectures.

4.1 Continuous Relaxations

Gradient-based optimization for supervised hashing is inherently challenging, due to the discrete (and usually NP-hard) nature of the problem. Specifically, in our case, there are two sources of discreteness. First, as is universal in hashing formulations, the bits in the hash code are binary. Second, the tie-aware metrics involve integer-valued histogram bin counts $\{n_{d,v}\}$.

We first tackle the discreteness of the bits. A commonly used formulation for the Hamming distance d_Φ and hash mapping Φ is

$$d_\Phi(x, x') = \frac{1}{2}\left(b - \Phi(x)^\top \Phi(x')\right), \tag{5}$$

$$\Phi(x) = (\phi_1(x), \ldots, \phi_b(x)),\ \phi_i(x) = \text{sgn}(f_i(x; w)) \in \{-1, +1\}, \forall i, \tag{6}$$

where f_i are the logits for each bit, parameterized by w, *e.g.* a neural network. We adopt a standard approximation technique in hashing [10, 14, 24, 25] and approximate the sgn function using sigmoid, *i.e.*, we replace $\phi_i(x)$ with $\tilde{\phi}_i(x) = 2\sigma(\alpha f_i(x; w)) - 1$, where α is a tuning parameter.

As a result of this relaxation, both the hash mapping and the distance function (5) are now real-valued, and will be denoted $\tilde{\Phi}$ and $\tilde{d}_\Phi$, respectively. The remaining discreteness is from the histogram bin counts $\{n_{d,v}\}$. We also relax them into real-valued "soft histograms" $\{c_{d,v}\}$, whose cumulative sums are denoted $\{C_{d,v}\}$, but we face another difficulty: the definitions of AP_T (3) and DCG_T (4) both involve a finite sum with lower and upper limits $N_{d-1} + 1$ and N_d, preventing a direct relaxation. We approximate these finite sums with continuous integrals, thus removing the second source of discreteness. We give results in the next proposition.

Proposition 1. *Denote the continuous relaxations of AP_T and DCG_T as AP_r and DCG_r, respectively. We have the following result:*

$$AP_r(R^{(d)}) = \frac{c_d^+(c_d^+-1)}{c_d-1} + \frac{c_d^+}{c_d}\left[C_{d-1}^+ + 1 - \frac{c_d^+-1}{c_d-1}(C_{d-1}+1)\right]\ln\frac{C_d}{C_{d-1}+1} \tag{7}$$

$$DCG_r(R^{(d)}) = \ln 2 \sum_{v\in\mathcal{V}} \frac{G(v)c_{d,v}}{c_d}\left[Li(C_d+1) - Li(C_{d-1}+2)\right] \tag{8}$$

where Li is the logarithmic integral function $Li(x) = \int_0^x dx/\ln x$.

Proof. First, recall that $\mathrm{AP_T}(R^{(d)})$ is defined as $\frac{n_d^+}{n_d}\sum_{t=N_{d-1}+1}^{N_d}\beta_d(t)$, where

$$\beta_d(t) = \frac{N_{d-1}^+ + (t-N_{d-1}-1)\frac{n_d^+-1}{n_d-1}+1}{t} = \frac{n_d^+-1}{n_d-1} + \frac{N_{d-1}^+ + 1 - \frac{n_d^+-1}{n_d-1}(N_{d-1}+1)}{t}. \tag{9}$$

The right hand side is of the form $A + B/t$ where A, B are constant in t. Summing the first term is trivial. For the second term, ignoring the constant scaling, summing gives a partial sum of the harmonic series, which is well approximated by the natural logarithm:

$$\sum_{t=N_{d-1}+1}^{N_d}\frac{1}{t} \approx \int_{N_{d-1}+1}^{N_d}\frac{dt}{t} = \ln(N_d) - \ln(N_{d-1}+1). \tag{10}$$

Then, we can do relaxation by substituting $\{n_{d,v}\}$ and $\{N_{d,v}\}$ with $\{c_{d,v}\}$ and $\{C_{d,v}\}$.

Next, the sum of discount values in $\mathrm{DCG_T}(R^{(d)})$ is handled in the same way.

$$\sum_{t=N_{d-1}+1}^{N_d} D(t) \approx \int_{N_{d-1}+1}^{N_d}\frac{dt}{\log_2(t+1)} = \ln 2\int_{N_{d-1}+2}^{N_d+1}\frac{dt}{\ln t} \tag{11}$$

$$= \ln 2\left[Li(N_d+1) - Li(N_{d-1}+2)\right]. \tag{12}$$

□

Both relaxations (7)(8) have closed-form derivatives. The differentiation for $\mathrm{AP_r}$ is straightforward, while for $\mathrm{DCG_r}$ it recovers the logarithm.

4.2 End-to-End Learning

We perform end-to-end learning with gradient ascent. First, we can compute the partial derivatives of the continuous relaxations $\mathrm{AP_r}$ and $\mathrm{DCG_r}$ with respect to $\{c_{d,v}\}$, the soft histograms. Next, we differentiate the histogram binning process. Note that the un-relaxed discrete histogram $(n_{0,v},\ldots,n_{b,v})$ for $\forall v\in\mathcal{V}$ can be estimated as follows:

$$n_{d,v} = \sum_{i|\mathcal{A}_q(i)=v}\mathbf{1}[d_\Phi(x_q,x_i)=d],\ d=0,\ldots,b. \tag{13}$$

When d_Φ is relaxed into $\tilde{d}_\Phi$, we employ a technique similar to [22], and replace the binary indicator $\mathbf{1}[\cdot]$ with a (sub-)differentiable function $\delta(\tilde{d}_\Phi(x_q,x_i),d)$, where the (sub-)gradient $\partial\delta(x,d)/\partial x$ has some simple form. Specifically, $\delta(\tilde{d}_\Phi(x_q,x_i),d)$ linearly interpolates the relaxed $\tilde{d}_\Phi(x_q,x_i)$ into the d-th bin with slope $\Delta>0$:

$$\delta(\tilde{d}_\Phi(x_q,x_i),d) = \begin{cases} 1-\frac{|\tilde{d}_\Phi(x_q,x_i)-d|}{\Delta}, & |\tilde{d}_\Phi(x_q,x_i)-d|\le\Delta, \\ 0, & \text{otherwise.}\end{cases} \tag{14}$$

As the result, we now have the soft histogram $c_{d,v}$, which we can differentiate with respect to $\tilde{\Phi}$ using chain rule:

$$\frac{\partial c_{d,v}}{\partial\tilde{\Phi}(x_i)} = \mathbf{1}[\mathcal{A}_q(i)=v]\frac{\partial\delta(\tilde{d}_\Phi(x_q,x_i),d)}{\partial\tilde{d}_\Phi(x_q,x_i)}\frac{-\tilde{\Phi}(x_q)}{2}, \forall x_i\in S, \tag{15}$$

$$\frac{\partial c_{d,v}}{\partial\tilde{\Phi}(x_q)} = \sum_{x_i\in S}\mathbf{1}[\mathcal{A}_q(i)=v]\frac{\partial\delta(\tilde{d}_\Phi(x_q,x_i),d)}{\partial\tilde{d}_\Phi(x_q,x_i)}\frac{-\tilde{\Phi}(x_i)}{2}. \tag{16}$$

CIFAR-10 Setting 1, 32 bits		
Method	**AP_T**	**AP (O, P)**
BRE	0.502	0.500 (0.543, 0.465)
StructHash	0.691	0.692 (0.727, 0.660)
MACHash	0.726	0.725 (0.755, 0.690)
FastHash	0.742	0.742 (0.770, 0.710)
Ours-AP	0.800	0.799 (0.827, 0.762)

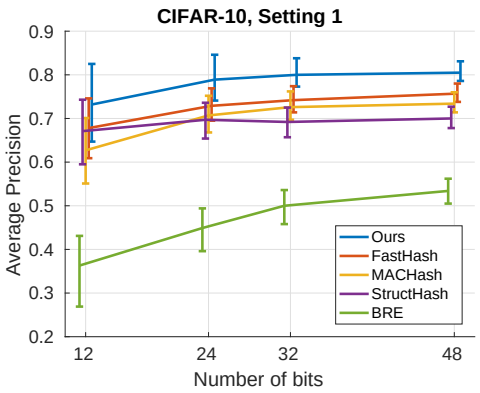

Figure 2: Left: we present the tie-aware AP_T, the tie-unaware AP, the optimistic AP (O), and the pessimistic AP (P) for several methods, on CIFAR-10. Ours-AP is our method optimized using the AP_r objective. Right: we plot AP values for the same methods, where error bars indicate the range spanned by different tie-breaking strategies.

The next and final step is to back-propagate gradients to the parameters of the relaxed hash mapping $\tilde{\Phi}$, which amounts to differentiating the sigmoid approximation.

We train our models using minibatch-based stochastic gradient ascent. Within a minibatch, each example is retrieved against the rest of the minibatch. That is, each example in a minibatch of size M is used as the query x_q once, and participates in the retrieval set for some other example $M-1$ times. Then, the objective is averaged over the M queries.

5 Experiments

5.1 Experimental Setup

We conduct experiments on three image retrieval datasets that are commonly used in supervised hashing: CIFAR-10 [7], NUS-WIDE [4], and 22K LabelMe [20]. CIFAR-10 contains 60K single-labeled images from 10 classes. NUS-WIDE is a multi-label dataset with 270K Flickr images. Following common practice [10, 25], we use a subset of 196K images associated with the most frequent 21 labels. Lastly, LabelMe is an unlabeled dataset consisting of 22K images.

Each dataset is split into a test set and a retrieval set, and examples from the retrieval set are used in training. At test time, each example from the test set is retrieved against the retrieval set, and the mean value of the metric over the test set is reported. Similar to [25], we consider two experimental settings for CIFAR-10. In the first setting, the test set is constructed with 100 random images from each class (total: 1K), the rest is used as retrieval set, and 500 images per class are used for training (total: 5K). The second setting uses a standard 10K/50K split and the entire retrieval set is used in training. For NUS-WIDE, we sample 100 images per label to construct a test set of size 2.1K, and use 500 images per label from the retrieval set for training (total: 10.5K). We randomly split LabelMe into a test test of size 2K and retrieval set of 20K, and sample 5K examples for training.

We compare against a range of supervised hashing methods. First, we consider hashing methods that learn from pairwise affinities, including Binary Reconstructive Embeddings (BRE) [8], Fast Supervised Hashing (FastHash) [12], Hashing using Auxiliary Coordinates (MACHash) [19], and Deep Pair-Supervised Hashing (DPSH) [10]. We also include Structured Hashing (StructHash) [13] and Deep Triplet-Supervised Hashing (DTSH) [25], two triplet-based methods that are inspired by learning to rank. Since tie-aware versions of ranking metrics have not been reported in the hashing literature, we re-train and evaluate all methods using publicly available implementations. For our method, we perform finetuning using our objectives on the ImageNet-pretrained VGG-F network [3], which is used in DPSH and DTSH, two recent top-performing methods (in terms of AP). The other methods are not amenable to end-to-end training, and for fairness we train them on fc7-layer features from VGG-F, unless otherwise noted.

	CIFAR-10					NUS-WIDE				
Method	12 Bits	24 Bits	32 Bits	48 Bits		12 Bits	24 Bits	32 Bits	48 Bits	
BRE	0.361	0.448	0.502	0.533	S1 (AP_T)	0.561	0.578	0.589	0.607	AP_T
MACHash	0.628	0.707	0.726	0.734		0.361	0.361	0.361	0.361	
FastHash	0.678	0.729	0.742	0.757		0.646	0.686	0.698	0.712	
StructHash	0.664	0.693	0.691	0.700		0.639	0.645	0.666	0.669	
DPSH*	0.713	0.727	0.744	0.757		0.658	0.674	0.695	0.700	
DTSH	0.710	0.750	0.765	0.774		0.660	0.700	0.707	0.723	
Ours-AP	**0.732**	**0.789**	**0.800**	**0.805**		**0.709**	**0.734**	**0.745**	**0.752**	
Method	16 Bits	24 Bits	32 Bits	48 Bits	S2 (AP_T)	12 Bits	24 Bits	32 Bits	48 Bits	AP@5K
DPSH*	0.908	0.909	0.917	0.932		0.758	0.793	0.818	0.830	
DTSH	0.916	0.924	0.927	0.934		0.773	0.813	0.820	0.838	
Ours-AP	**0.922**	**0.935**	**0.939**	**0.941**		**0.795**	**0.835**	**0.848**	**0.862**	

* Trained using parameters recommended by DTSH [25].

Table 1: Comparison of AP_T on CIFAR-10 and NUS-WIDE. For CIFAR-10, we compare all methods in the first setting (S1), and end-to-end methods in the second setting (S2). For NUS-WIDE, we additionally report AP@5K for three methods. Ours-AP is trained with the VGG-F network and AP_r as objective, and consistently achieves state-of-the-art performance on both datasets.

5.2 Evaluation of Tie-Awareness

Before reporting and comparing the results, we first evaluate the effect of tie-awareness, as it is part of the motivation for this work. Similar to Figure 1, we present the actual values of AP_T, the optimistic AP, and the pessimistic AP for various methods on CIFAR-10 (first setting) in Figure 2. As expected, the gap between optimistic and pessimistic versions of AP can be quite large for short hash codes, and decreases as the code length increases, since each tie would decrease in size as there are more bits. We again emphasize that the evaluation of hashing algorithms should use tie-aware ranking metrics to eliminate ambiguity in the results.

However, there is another interesting observation. The tie-unaware AP values are usually very close to the tie-aware AP_T, matching up to two or three digits, for all methods we tested. The reason is that with a randomly ordered retrieval set and a sufficiently large test set, averaging the tie-unaware AP over the test set empirically behaves similarly to explicitly averaging over all permutations of ties. Since randomizing the ordering of the retrieval set before sorting is common practice, we believe the current AP values reported in the literature are indeed quite fair, and we are generally able to reproduce them for the tested methods with small variations. Still, this is not robust as the variability is not eliminated, and using general-purpose sorting algorithms on Hamming distances is not efficient. In the following experiments, we focus on reporting tie-aware metrics AP_T and $NDCG_T$.

5.3 AP Optimization

We evaluate AP optimization on the two labeled datasets, CIFAR-10 and NUS-WIDE. For labeled data, it is common to infer affinities from label agreements. In CIFAR-10, two examples are neighbors (*i.e.* have pairwise affinity 1) if they share the same label. In the multi-labeled NUS-WIDE, two examples are neighbors if they share at least one label. On CIFAR-10, we compare all methods in the first setting, and in the second setting we compare the end-to-end methods (DPSH, DTSH, and ours), which generally outperform others.

Results. We present results in Table 1. By optimizing the relaxation of AP_T in an end-to-end fashion, our method achieves the new state-of-the-art in AP on both datasets with pairwise supervision, outperforming both the pair-based methods, and the triplet-based methods (StructHash and DTSH). Note that triplets provide stronger supervision than pairs, as there are $O(N^3)$ triplets *vs.* $O(N^2)$ pairs for N examples. Triplet-based losses can also be viewed as local ranking losses, but they are not directly related to the test metrics.

For NUS-WIDE, it is customary in existing works to report AP evaluated at maximum cutoff of 5K (AP@5K), the reasoning being that ranking the full retrieval set is inefficient using general-purpose sorting algorithms. Our method also performs well in terms of AP@5K, but we note that focusing on the top of the ranking would overestimate the true AP, as can be seen in Table 1. Using counting sort, we are able to evaluate AP_T on the full ranking efficiently.

Method	NUS-WIDE				LabelMe			
	16 Bits	32 Bits	48 Bits	64 Bits	16 Bits	32 Bits	48 Bits	64 Bits
BRE*	0.805	0.817	0.827	0.834	0.807	0.848	0.871	0.880
MACHash	0.821	0.821	0.821	0.821	0.683	0.683	0.683	0.687
FastHash	0.885	0.896	0.899	0.902	0.844	0.868	0.855	0.864
StructHash	0.889	0.893	0.894	0.898	0.857	0.888	0.904	0.915
DPSH	0.895	0.905	0.909	0.909	0.844	0.856	0.871	0.874
DTSH	0.896	0.905	0.911	0.913	0.838	0.852	0.859	0.862
Ours-NDCG	**0.903**	**0.910**	**0.916**	**0.927**	**0.866**	**0.895**	**0.908**	**0.917**

* Evaluated on the the 5K training subset of the retrieval set.

Table 2: Comparison of NDCG_T on NUS-WIDE and LabelMe. Ours-NDCG is trained with the NDCG_r objective, using the VGG-F network on NUS-WIDE and single-layer linear hash functions on LabelMe, and again consistently outperforms competing methods.

5.4 NDCG Optimization

We evaluate NDCG optimization with a multi-level affinity setup, *i.e.* the set of affinity values $\mathcal{V}$ is a finite set of non-negative integers. Using multi-level affinities is common in some information retrieval tasks, and offers more fine-grained specification of the desired structure of the learned Hamming space. To our knowledge, this setup has not been considered in the hashing literature.

The NUS-WIDE dataset provides a natural testbed for this experiment as it is a multi-label dataset. We compute affinities by counting the number of labels that two examples share in NUS-WIDE, and keep other settings the same as in the AP experiments. Next, on the unlabeled LabelMe dataset we derive affinities by thresholding the Euclidean distances between examples. An existing setup for binary affinities on LabelMe is to treat pairs as neighbors if their Euclidean distance is within the top 5% on the training set. Similarly, we use four thresholds (5%, 1%, 0.2%, 0.1%) and assign affinity values 1, 2, 5, and 10, respectively, to emphasize assigning high ranks to the closest neighbors in the original feature space. We learn shallow models on precomputed GIST features on LabelMe; for gradient-based methods, this means using linear hash functions, *i.e.* $f_i(x; w) = \text{sgn}(w_i^\top x)$, in (6). For methods designed to use binary pairwise affinities (FastHash, MACHash, DPSH), we convert the affinities into binary values; note that this would reduce to the standard binary affinity setup on both datasets.

Results. We summarize NDCG optimization results in Table 2. NDCG is less position-sensitive than AP, since all items in the ranking contribute to the overall score, while AP only counts the contributions from the neighbors. As a result, for a given method, the NDCG score is generally higher than AP. Nevertheless, our method with the NDCG objective again outperforms all competing methods on both datasets. On LabelMe, all methods are restricted to learn shallow models, and DPSH and DTSH appear to perform less competitively in this case, indicating a mismatch between their objectives and the evaluation metric. A close competitor to our method on LabelMe is StructHash, which optimizes a surrogate on NDCG using boosted decision trees; our method still outperforms StructHash with lower-capacity linear hash functions. We believe this highlights the benefit of optimizing direct relaxations of the evaluation metric over optimizing surrogates.

6 Conclusion

In this work, we propose new formulations to both learn and evaluate supervised hashing models. Our original motivation is to learn hash functions by optimizing appropriate objective functions, which we believe should match the test-time evaluation metrics. By studying commonly used ranking-based metrics, we first raise awareness on the issue of ties in evaluating supervised hashing algorithms, and advocate using tie-aware versions of ranking metrics during evaluation. These metrics can be efficiently computed in linear time, thanks to counting sort. We then make the novel contribution of optimizing tie-aware ranking metrics for hashing, focusing on the two most widely-used examples: AP and NDCG. To tackle the resulting discrete optimization problems, we derive their continuous relaxations which have closed-form gradients, and perform end-to-end learning with stochastic gradient ascent and deep neural networks. This results in the new state-of-the-art for AP and NDCG on three benchmarks commonly used in supervised hashing.

References

[1] A. Andoni and I. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 793–801, 2015.

[2] O. Chapelle and M. Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval*, 13(3):216–235, 2010.

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

[4] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng. NUS-WIDE: A real-world web image database from National University of Singapore. In *Proc. ACM CIVR*, 2009.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pages 518–529, 1999.

[6] I. Kim and C.-H. Lee. An efficient gradient-based approach to optimizing average precision through maximal figure-of-merit learning. *Journal of Signal Processing Systems*, 74(3):285–295, 2014.

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

[8] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1042–1050, 2009.

[9] A. Kustarev, Y. Ustinovsky, Y. Logachev, E. Grechnikov, I. Segalovich, and P. Serdyukov. Smoothing NDCG metrics using tied scores. In *Proc. ACM CIKM*, pages 2053–2056, 2011.

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

[11] D. Lim and G. Lanckriet. Efficient learning of mahalanobis metrics for ranking. In *Proc. Int'l Conf. on Machine Learning (ICML)*, pages 1980–1988, 2014.

[12] 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.

[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] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2074–2081. IEEE, 2012.

[15] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*. Cambridge university press, 2008.

[16] B. McFee and G. R. Lanckriet. Metric learning to rank. In *Proc. Int'l Conf. on Machine Learning (ICML)*, pages 775–782, 2010.

[17] F. McSherry and M. Najork. Computing information retrieval performance measures efficiently in the presence of tied scores. In *European Conf. on Information Retrieval*, pages 414–421, 2008.

[18] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1061–1069, 2012.

[19] R. Raziperchikolaei and M. A. Carreira-Perpinán. Optimizing affinity-based binary hashing using auxiliary coordinates. In *Advances in Neural Information Processing Systems (NIPS)*, pages 640–648, 2016.

[20] 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 (IJCV)*, 77(1):157–173, 2008.

[21] D. Song, W. Liu, R. Ji, D. A. Meyer, and J. R. Smith. Top rank supervised binary coding for visual search. In *Proc. IEEE Int'l Conf. on Computer Vision (ICCV)*, pages 1922–1930, 2015.

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

[23] H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to rank by optimizing NDCG measure. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1883–1891, 2009.

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

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

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

[27] L. Zhang, Y. Zhang, J. Tang, K. Lu, and Q. Tian. Binary code ranking with weighted hamming distance. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1586–1593, 2013.