Hashing as Tie-Aware Learning to Rank

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

Abstract

Hashing, or learning binary embeddings of data, is frequently used in nearest neighbor retrieval. In this paper, we develop learning to rank formulations for hashing, aimed at directly optimizing ranking-based evaluation metrics such as Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG). We first observe that the integer-valued Hamming distance often leads to tied rankings, and propose to use tie-aware versions of AP and NDCG to evaluate hashing for retrieval. Then, to optimize tie-aware ranking metrics, we derive their continuous relaxations, and perform gradient-based optimization with deep neural networks. Our results establish the new state-of-the-art for image retrieval by Hamming ranking in common benchmarks.

1. Introduction

In this paper, we consider the problem of hashing, which is concerned with learning binary embeddings of data in order to enable fast approximate nearest neighbor retrieval. We take a task-driven approach, and seek to optimize learning objectives that closely match test-time performance measures. Nearest neighbor retrieval performance is frequently measured using ranking-based evaluation metrics, such as Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG) [26], but the optimization of such metrics has been deemed difficult in the hashing literature [30]. We propose a novel learning to rank formulation to tackle these difficult optimization problems, and our main contribution is a gradient-based method that directly optimizes ranking metrics for hashing. Coupled with deep neural networks, this method achieves state-of-the-art results.

Our formulation is inspired by a simple observation. When performing retrieval with binary vector encodings and the integer-valued Hamming distance, the resulting ranking usually contains *ties*, and different tie-breaking strategies can lead to different results (Fig. 1). In fact, ties are a common problem in ranking, and much attention has been paid to it, including in Kendall's classical work on rank correlation [15], and in the modern information retrieval literature [3, 28]. Unfortunately, the learning to hash literature largely lacks tie-awareness, and current evaluation protocols rarely take tie-breaking into account. Thus, we advocate using *tie-aware* ranking evaluation metrics, which implicitly average over all permutations of tied items, and permit efficient closed-form evaluation.

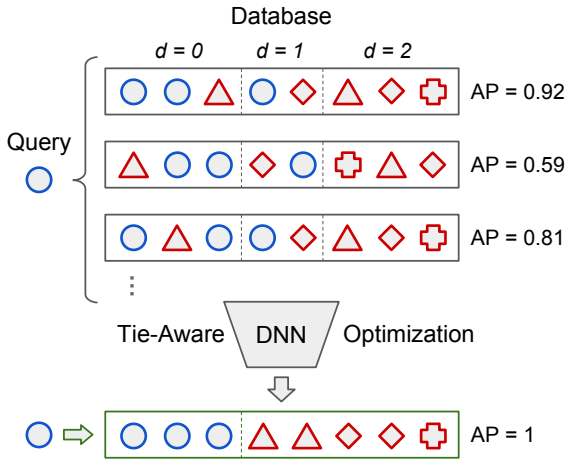

Figure 1: When applying hashing for nearest neighbor retrieval, the integer-valued Hamming distance produces *ties* (items that share the same distance). If left uncontrolled, different tie-breaking strategies could give drastically different values of the evaluation metric, *e.g.* AP. We address this issue by using *tie-aware* ranking metrics that implicitly average over all the permutations in closed form. We further use tie-aware ranking metrics as optimization objectives in deep hashing networks, leading to state-of-the-art results.

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 with listwise loss functions, which is advantageous compared to many other ranking-inspired hashing formulations. To solve the associated discrete and NP-hard optimization problems, we relax the problems into their continuous counterparts where closed-form gradients are available, and then perform gradient-based optimization with deep neural networks. We specifically study the optimization of AP and NDCG, two ranking metrics that are

widely used in evaluating nearest neighbor retrieval performance. Our results establish the new state-of-the-art for these metrics in common image retrieval benchmarks.

2. Related Work

Hashing is a widely used approach for practical nearest neighbor retrieval [39], thanks to the efficiency of evaluating Hamming distances using bitwise operations, as well as the low memory and storage footprint. It has been theoretically demonstrated [1] that data-dependent hashing methods outperform data-independent ones such as Locality Sensitive Hashing [14]. We tackle the supervised hashing problem, also known as affinity-based hashing [18, 25, 30], where supervision is given in the form of pairwise affinities. Regarding optimization, the discrete nature of hashing usually results in NP-hard problems. Our solution uses continuous relaxations, which is in line with relaxation-based methods, *e.g.* [4, 18, 25], but differs from alternating methods that preserve the discrete constraints [22, 29, 30] and two-step methods [6, 23, 48].

Supervised hashing can be cast as a distance metric learning problem [29], which itself can be formulated as learning to rank [21, 27]. Optimizing ranking metrics such as AP and NDCG has received much attention in the learning to rank literature. For instance, surrogates of AP and NDCG can be optimized in the structural SVM framework [10, 45], and bound optimization algorithms exist for NDCG [38]. Alternatively, there are gradient-based methods based on smoothing or approximating these metrics [2, 11, 19, 35]. Recently, [36] tackles few-shot classification by optimizing AP using the direct loss minimization framework [34]. These methods did not consider applications in hashing.

In the learning to hash literature, different strategies have been proposed to handle the difficulties in optimizing listwise ranking metrics. For example, [40] decomposes listwise supervision into local triplets, [22, 44] use structural SVMs to optimize surrogate losses, [33] maximizes precision at the top, and [41, 47] optimize NDCG surrogates. In other recent methods using deep neural networks, the learning objectives are not designed to match ranking evaluation metrics, *e.g.* [4, 20, 42, 48]. In contrast, we directly optimize listwise ranking metrics using deep neural networks.

Key to our formulation is the observation that the integer-valued Hamming distance results in rankings with ties. However, this fact is not widely taken into consideration in previous work. Ties can be sidestepped by using weighted Hamming distance [22, 46], but at the cost of reduced efficiency. Fortunately, tie-aware versions of common ranking metrics have been found in the information retrieval literature [28]. Inspired by such results, we propose to optimize tie-aware ranking metrics on Hamming distances. Our gradient-based optimization uses a recent differentiable histogram binning technique [4, 5, 37].

3. Hashing as Tie-Aware Ranking

3.1. Preliminaries

Learning to hash. In learning to hash, 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 $d_\Phi : \mathcal{X} \times \mathcal{X} \to \{0, 1, \dots, b\}$ as[1]

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

We consider a supervised learning setting, or *supervised hashing*, where supervision is specified using pairwise affinities. Formally, we assume access to an affinity oracle $\mathcal{A}$, whose value indicates a notion of similarity: two examples $x_i, x_j \in \mathcal{X}$ are called similar if $\mathcal{A}(x_i, x_j) > 0$, and dissimilar when $\mathcal{A}(x_i, x_j) = 0$. In this paper, we restrict $\mathcal{A}$ to take values from a finite set $\mathcal{V}$, which covers two important special cases. First, $\mathcal{V} = \{0, 1\}$, or *binary affinities*, are extensively studied in the current literature. Binary affinities can be derived from agreement of class labels, or by thresholding the original Euclidean distance in $\mathcal{X}$.[2] The second case is *multi-level affinities*, where $\mathcal{V}$ consists of non-negative integers. This more fine-grained model of similarity 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 database $S \subseteq \mathcal{X}$. Retrieval is performed by ranking the instances in S by increasing distance to x_q, using d_Φ as the distance metric. This is termed "retrieval by Hamming ranking" in the hashing literature. The ranking can be represented by an index vector R, whose elements form a permutation of $\{1, \dots, |S|\}$. Below, let R_i be the i-th element in R, and $\mathcal{A}_q(i) = \mathcal{A}(x_q, x_i)$. Unless otherwise noted, we implicitly assume dependency on x_q, S, and Φ in our notation.

Ranking-based evaluation. Ranking-based metrics usually measure some form of agreement between the ranking and ground truth affinities, capturing the intuition that retrievals with high affinity to the query should be ranked high. First, in the case of binary affinity, we define $N^+ = |\{x_i \in S | \mathcal{A}_q(i) = 1\}|$. Average Precision (AP) averages the precision at cutoff k over all cutoffs:

$$\mathrm{AP}(R) = \frac{1}{N^+} \sum_{k=1}^{|S|} \mathcal{A}_q(R_k) \left[\frac{1}{k} \sum_{j=1}^{k} \mathcal{A}_q(R_j)\right]. \tag{2}$$

Next, for integer-valued affinities, Discounted Cumulative

[1] Although the usual implementation is by counting bit differences, this equivalent formulation has the advantage of being differentiable.

[2] The latter is sometimes referred to as "unsupervised hashing" in the literature due to the absence of class labels.

Gain (DCG) is defined as

$$\mathrm{DCG}(R) = \sum_{k=1}^{|S|} G(\mathcal{A}_q(R_k))D(k), \tag{3}$$

$$\text{where } G(a) = 2^a - 1,\ D(k) = \frac{1}{\log_2(k+1)}. \tag{4}$$

G and D are called gain and (logarithmic) discount, respectively. Normalized DCG (NDCG) divides DCG by its maximum possible value, ensuring a range of $[0, 1]$:

$$\mathrm{NDCG}(R) = \frac{\mathrm{DCG}(R)}{\max_{R'} \mathrm{DCG}(R')}. \tag{5}$$

3.2. Tie-Awareness in Hashing

When evaluating information retrieval systems, special attention is required when there exist ties in the distances [3, 28]. 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 have given an example in Fig. 1. Surprisingly, we found that current ranking-based hashing evaluation protocols usually do not take tie-breaking into account, which could result in ambiguous comparisons or even unfair exploitation. Perhaps more importantly, ties render the formulation of direct optimization unclear: what tie-breaking strategy should we assume when using AP or NDCG as optimization objectives? Thus, we believe that it is important to seek *tie-aware* evaluation metrics for hashing.

Rather than picking a fixed tie-breaking strategy or relying on randomization, the tie-aware solution that we propose is to average the value of the ranking metric over all possible permutations of tied items. This solution is appealing in several ways: it is deterministic, it is unambiguous and cannot be exploited, and it reduces to the ordinary version when there are no ties. However, there is one caveat: generating all permutations for n tied items requires $O(n!)$ time, which is super-exponential and prohibitive. Fortunately, [28] observes that the average can be computed implicitly for commonly used ranking metrics, and gives their tie-aware versions in closed form. Based on this result, we further describe how to efficiently compute tie-aware ranking metrics by exploiting the structure of the Hamming distance.

We focus on AP and NDCG, and denote the tie-aware versions of AP and (N)DCG as AP_T and (N)DCG_T, respectively. First, we define some notation. 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 $R^{(d)} = \{i | d_\Phi(x_q, x_i) = d\}$ is the set of retrievals having Hamming distance d to the query. We define a set 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 $N_{d,v} = \sum_{j \le d} n_{j,v}$. We also define the total histograms as $n_d = \sum_{v \in \mathcal{V}} n_{d,v}$ with cumulative sum $N_d = \sum_{j \le d} n_j$.

Next, Proposition 1 gives the closed forms of AP_T and DCG_T. We give proof in the appendix.

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 the normalized NDCG_T, the normalizing factor is unaffected by ties, but computing it still requires sorting the gain values in descending order. Under the assumption that the set of affinity values $\mathcal{V}$ consists of non-negative integers, 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

---

Proposition 1. *Both* AP_T *and* DCG_T *decompose additively over the ties. For* $\mathcal{V} = \{0, 1\}$, *let* $n_d^+ \triangleq n_{d,1}, N_d^+ \triangleq N_{d,1}$, *and* $N^+ = \sum_d n_d^+$, *the contribution of each tie* $R^{(d)}$ *to* AP_T *is computed as*

$$\mathrm{AP}_\mathrm{T}(R^{(d)}) = \frac{n_d^+}{n_d N^+} \sum_{t=N_{d-1}+1}^{N_d} \frac{N_{d-1}^+ + (t - N_{d-1} - 1)\frac{n_d^+ - 1}{n_d - 1} + 1}{t}. \tag{6}$$

For DCG_T, *the contribution of* $R^{(d)}$ *is*

$$\mathrm{DCG}_\mathrm{T}(R^{(d)}) = \sum_{i \in R^{(d)}} \frac{G(\mathcal{A}_q(i))}{n_d} \sum_{t=N_{d-1}+1}^{N_d} D(t) = \sum_{v \in \mathcal{V}} \frac{G(v) n_{d,v}}{n_d} \sum_{t=N_{d-1}+1}^{N_d} D(t). \tag{7}$$

Proof. See appendix. □

that $|\mathcal{V}|$ is known. We note that counting sort on Hamming distances is also used by Lin *et al*. [22] to speed up loss-augmented inference for their NDCG surrogate loss.

3.3. The Learning to Rank View

Since we focus on optimizing ranking metrics, our work has connections to learning to rank [24]. Many supervised hashing formulations use loss functions defined on pairs or triplets of training examples, which correspond to *pointwise* and *pairwise* approaches in learning to rank terminology. We collectively refer to these as local ranking losses. Since we optimize evaluation metrics defined on a ranked list, our approach falls into the *listwise* category, and it is well-known [9, 40, 44] that listwise ranking approaches are generally superior to pointwise and pairwise approaches.

We further note that there exists a mismatch between optimizing local ranking losses and optimizing for evaluation performance. This is because listwise evaluation metrics are *position-sensitive*: errors made on individual pairs/triplets impact results differently depending on the position in the list, and more so near the top. To address this mismatch, local ranking methods often need nontrivial weighting or sampling heuristics to focus on errors made near the top. In fact, the sampling is especially crucial in triplet-based methods, *e.g*. [22, 42, 48], since the set of possible triplets is of size $O(N^3)$ for N training examples, which can be prohibitive to enumerate. Triplet-based methods are also popular in the metric learning literature, and it is similarly observed [43] that careful sampling and weighting are key to stable learning. In contrast, we directly optimize listwise ranking metrics, without requiring sampling or weighting heuristics: the minibatches are sampled at random, and no weighting on training instances is used.

4. Optimizing Tie-Aware Ranking Metrics

In this section, we describe our approach to optimizing tie-aware ranking metrics. For discrete hashing, such optimization is NP-hard, since it involves combinatorial search over all configurations of binary bits. Instead, we are interested in a relaxation approach using gradient-based deep neural networks. Therefore, we apply continuous relaxation to the discrete optimization problems.

4.1. Continuous Relaxations

Our continuous relaxation needs to address two types of discrete variables. 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 binary bits. Commonly, bits in the hash code are generated by a thresholding operation using the sgn function,

$$\Phi(x) = (\phi_1(x), \dots, \phi_b(x)), \tag{8}$$

$$\phi_i(x) = \operatorname{sgn}(f_i(x; w)) \in \{-1, 1\}, \forall i, \tag{9}$$

where in our case f_i are neural network activations, parameterized by w. We smoothly approximate the sgn function using the $\tanh$ function, which is a standard technique in hashing [4, 8, 20, 25, 41, 42]:

$$\phi_i(x) \approx \hat{\phi}_i(x) = \tanh(\alpha f_i(x; w)) \in (-1, 1). \tag{10}$$

The constant α is a scaling parameter.

As a result of this relaxation, both the hash mapping and the distance function (1) are now real-valued, and will be denoted $\hat{\Phi}$ and $\hat{d}_\Phi$, respectively. The remaining discreteness is from the histogram bin counts $\{n_{d,v}\}$. We also relax them into real-valued "soft histograms" $\{c_{d,v}\}$ (described below), whose cumulative sums are denoted $\{C_{d,v}\}$. However, we face another difficulty: the definitions of $\mathrm{AP_T}$ (6) and $\mathrm{DCG_T}$ (7) both involve a finite sum with lower and upper limits $N_{d-1} + 1$ and N_d, which themselves are variables to be relaxed. We approximate these finite sums by continuous integrals, removing the second source of discreteness. We outline the results in Proposition 2, and leave proof and error analysis to the appendix.

Importantly, both relaxations have closed-form derivatives. The differentiation for $\mathrm{AP_r}$ (11) is straightforward, while for $\mathrm{DCG_r}$ it removes the integral in (12).

4.2. End-to-End Learning

We perform end-to-end learning with gradient ascent. First, as mentioned above, the continuous relaxations $\mathrm{AP_r}$ and $\mathrm{DCG_r}$ have closed-form partial derivatives with respect

---

Proposition 2. *The continuous relaxations of* $\mathrm{AP_T}$ *and* $\mathrm{DCG_T}$*, denoted as* $\mathrm{AP_r}$ *and* $\mathrm{DCG_r}$ *respectively, are as follows:*

$$\mathrm{AP_r}(R^{(d)}) = \frac{c_d^+(c_d^+ - 1)}{N^+(c_d - 1)} + \frac{c_d^+}{N^+ c_d}\left[C_{d-1}^+ + 1 - \frac{c_d^+ - 1}{c_d - 1}(C_{d-1} + 1)\right] \ln \frac{C_d}{C_{d-1}}, \tag{11}$$

$$\mathrm{DCG_r}(R^{(d)}) = \ln 2 \sum_{v \in \mathcal{V}} \frac{G(v) c_{d,v}}{c_d} \int_{C_{d-1}+1}^{C_d+1} \frac{dt}{\ln t}. \tag{12}$$

Proof. See appendix. □

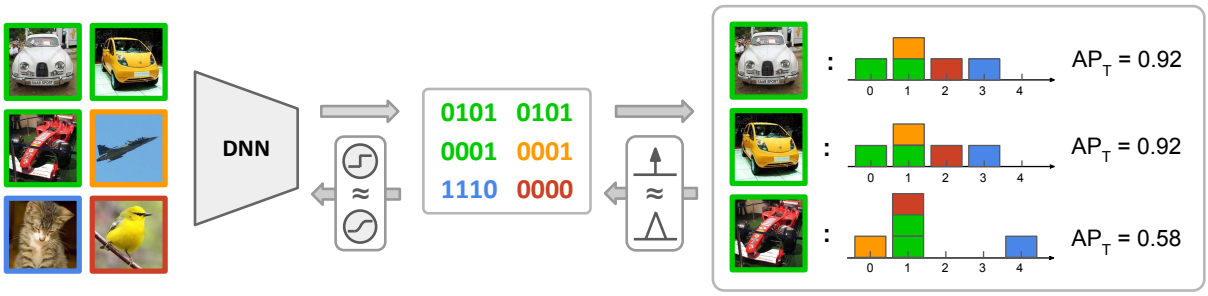

Figure 2: The flow of computation in our model. Input images are mapped to b-bit binary codes by a deep neural network ($b = 4$ in this example). During training, in a minibatch, each example is used as query to rank the rest of the batch, producing a histogram of Hamming distances with $(b + 1)$ bins. Tie-aware ranking metrics ($\mathrm{AP_T}$ shown here) are computed on these histograms, and averaged over the batch. To maintain end-to-end differentiability, we derive continuous relaxations for $\mathrm{AP_T}$ and $\mathrm{NDCG_T}$, and employ two differentiable approximations to non-differentiable operations (backward arrows).

to the soft histograms $\{c_{d,v}\}$. Next, we consider differentiating the histogram entries. Note that before relaxation, the discrete histogram $(n_{0,v}, \ldots, n_{b,v})$ for $\forall v \in \mathcal{V}$ is constructed as follows:

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

To relax $n_{d,v}$ into $c_{d,v}$, we employ a technique from [4, 37], where the binary indicator $\mathbf{1}[\cdot]$ is replaced by a differentiable function $\delta(\hat{d}_\Phi(x_q, x_i), d)$ with easy-to-compute gradients. Specifically, $\delta(\hat{d}_\Phi(x_q, x_i), d)$ linearly interpolates $\hat{d}_\Phi(x_q, x_i)$ into the d-th bin with slope $\Delta > 0$:

$$\forall z \in \mathbb{R}, \delta(z, d) = \begin{cases} 1 - \frac{|z-d|}{\Delta}, & |z - d| \leq \Delta, \\ 0, & \text{otherwise.} \end{cases} \quad (14)$$

Note that δ approaches the indicator function as $\Delta \to 0$. We now have the soft histogram $c_{d,v}$ as

$$c_{d,v} = \sum_{x_i | \mathcal{A}_q(i) = v} \delta(\hat{d}_\Phi(x_q, x_i), d), \quad (15)$$

and we differentiate $c_{d,v}$ using chain rule, *e.g.*

$$\frac{\partial c_{d,v}}{\partial \hat{\Phi}(x_q)} = \sum_{x_i | \mathcal{A}_q(i) = v} \frac{\partial \delta(\hat{d}_\Phi(x_q, x_i), d)}{\partial \hat{d}_\Phi(x_q, x_i)} \frac{-\hat{\Phi}(x_i)}{2}. \quad (16)$$

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

As shown in Fig. 2, 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 database 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 image retrieval datasets that are commonly used in the hashing literature: CIFAR-10 [16], NUS-WIDE [13], 22K LabelMe [31], and ImageNet100 [8]. Each dataset is split into a test set and a database, and examples from the database are used in training. At test time, queries from the test set are used to perform Hamming ranking on the database, and the performance metric is averaged over the test set.

- **CIFAR-10** is a canonical benchmark for image classification and retrieval, with 60K single-labeled images from 10 classes. Following [42], we consider two experimental settings. In the first setting, the test set is constructed with 100 random images from each class (total: 1K), the rest is used as database, and 500 images per class are used for training (total: 5K). The second setting uses the standard 10K/50K split and the entire database is used in training.
- **NUS-WIDE** is a multi-label dataset with 270K Flickr images. For the database, we use a subset of 196K images associated with the most frequent 21 labels as in [20, 42]. 100 images per label are sampled to construct a test set of size 2.1K, and the training set contains 500 images per label (total: 10.5K).
- **LabelMe** is an unlabeled dataset of 22K images. As in [7], we randomly split LabelMe into a test test of size 2K and database of 20K. We sample 5K examples from the database for training.
- **ImageNet100** is a subset of ImageNet, containing all the images from 100 classes. We use the same setup as in [8]: 130 images per class, totaling 130K images, are sampled for training, and all images in the selected classes from the ILSVRC 2012 validation set are used as queries.

Retrieval-based evaluation of supervised hashing was recently put into question by [32], which points out that for multi-class datasets, binary encoding of classifier outputs is already a competitive solution. While this is an important point, deriving pairwise affinities from multi-class label agreement is a special case in our formulation. As mentioned in Sec. 3.1, our formulation uses a general pairwise affinity oracle $\mathcal{A}$, which may or may not be derived from labels, and can be either binary or multi-level. In fact, the datasets we consider range from multi-class/single-label (CIFAR-10, ImageNet100) to multi-label (NUS-WIDE) and unlabeled (LabelMe), and only the first case can be addressed by multi-class classification. For multi-level affinities, we also propose a new evaluation protocol using NDCG.

We term our method TALR (**T**ie-**A**ware **L**earning to **R**ank), and compare it against a range of classical and state-of-the-art hashing methods. Due to the vast hashing literature, an exhaustive comparison is unfortunately not feasible. Focusing on the learning to rank aspect, we select representative methods from all three categories:

- **Pointwise (pair-based).** Methods that define loss functions on instance pairs: Binary Reconstructive Embeddings (BRE) [18], Fast Supervised Hashing (FastHash) [23], Hashing using Auxiliary Coordinates (MACHash) [30], Deep Pair-Supervised Hashing (DPSH) [20], and Hashing by Continuation (HashNet) [8].
- **Pairwise (triplet-based).** We include a recent method, Deep Triplet-Supervised Hashing (DTSH) [42].
- **Listwise (list-based).** We compare to two listwise ranking methods: Structured Hashing (StructHash) [22] which optimizes an NDCG surrogate, and Hashing with Mutual Information (MIHash) [4] which optimizes mutual information as a ranking surrogate for binary affinities.

These selected methods include recent ones that achieve state-of-the-art results on CIFAR-10 (MIHash, DTSH), NUS-WIDE (DTSH, HashNet) and ImageNet100 (HashNet).

Since tie-aware evaluation of Hamming ranking performance has not been reported in the hashing literature, we re-train and evaluate all methods using publicly available implementations.

5.2. AP Optimization

We evaluate AP optimization on the three labeled datasets, CIFAR-10, NUS-WIDE, and ImageNet100. As we mentioned earlier, for labeled data, affinities can be inferred from label agreements. Specifically, in CIFAR-10 and ImageNet100, two examples are neighbors (*i.e.* have pairwise affinity 1) if they share the same class label. In the multi-labeled NUS-WIDE, two examples are neighbors if they share at least one label.

5.2.1 CIFAR-10 and NUS-WIDE

We first carry out AP optimization experiments on the two well-studied datasets, CIFAR-10 and NUS-WIDE. For these experiments, we perform finetuning using the ImageNet-pretrained VGG-F network [12], which is used in DPSH and DTSH, two recent top-performing methods. For methods that are not amenable to end-to-end training, we train them on fc7-layer features from VGG-F. 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, MIHash, and ours. We do not include HashNet as it uses a different network architecture (AlexNet), but will compare to it later on ImageNet100.

We present AP optimization results in Table 1. By optimizing the relaxation of AP_T in an end-to-end fashion, our method (TALR-AP) achieves the new state-of-the-art in AP on both datasets, outperforming all the pair-based and triplet-based methods by significant margins. Compared to listwise ranking solutions, TALR-AP outperforms StructHash significantly by taking advantage of deep learning, and outperforms MIHash by matching the training objective to the evaluation metric. A side note is that for NUS-WIDE, it is customary in previous work [20, 42] to report AP evaluated at maximum cutoff of 5K (AP@5K), since ranking the full database is inefficient using general-purpose sorting algorithms. However, focusing on the top of the ranking overestimates the true AP, as seen in Table 1. Using counting sort, we are able to evaluate AP_T on the full database efficiently, and TALR-AP also outperforms other methods in terms of AP@5K.

5.2.2 ImageNet100

For ImageNet100 experiments, we closely follow the setup in HashNet [8] and fine-tune the AlexNet architecture [17] pretrained on ImageNet. Due to space limitations, we report comparisons against recent state-of-the-art methods on ImageNet100. The first competitor is HashNet, which is empirically superior to a wide range of classical and recent methods, and was previously the state-of-the-art method on ImageNet100. We also compare to MIHash, as it is the second-best method on CIFAR-10 and NUS-WIDE in the previous experiment. As in [8], the minibatch size is set to 256 for all methods, and the learning rate for the pre-trained convolution and fully connected layers are scaled down, since the model is fine-tuned on the same dataset that it was originally trained on. AP at cutoff 1000 (AP@1000) is used as the evaluation metric.

ImageNet100 results are summarized in Table 2. TALR-AP outperforms both competing methods, and the improvement is especially significant with short hash codes (16 and 32 bits). This indicates that our direct optimization approach produces better compact binary representations that preserve desired rankings. The state-of-the-art performance with com-

Method	CIFAR-10					NUS-WIDE				
	12 Bits	24 Bits	32 Bits	48 Bits		12 Bits	24 Bits	32 Bits	48 Bits	
BRE [18]	0.361	0.448	0.502	0.533	S1 ($\mathrm{AP_T}$)	0.561	0.578	0.589	0.607	$\mathrm{AP_T}$
MACHash [30]	0.628	0.707	0.726	0.734		0.361	0.361	0.361	0.361	
FastHash [23]	0.678	0.729	0.742	0.757		0.646	0.686	0.698	0.712	
StructHash [22]	0.664	0.693	0.691	0.700		0.639	0.645	0.666	0.669	
DPSH [20]*	0.720	0.757	0.757	0.767		0.658	0.674	0.695	0.700	
DTSH [42]	0.725	0.773	0.781	0.810		0.660	0.700	0.707	0.723	
MIHash [4]	0.687	0.775	0.786	0.822		0.652	0.693	0.709	0.723	
TALR-AP	**0.732**	**0.789**	**0.800**	**0.826**		**0.709**	**0.734**	**0.745**	**0.752**	
Method	16 Bits	24 Bits	32 Bits	48 Bits		12 Bits	24 Bits	32 Bits	48 Bits	
DPSH [20]*	0.908	0.909	0.917	0.932	S2 ($\mathrm{AP_T}$)	0.758	0.793	0.818	0.830	AP@5K
DTSH [42]	0.916	0.924	0.927	0.934		0.773	0.813	0.820	0.838	
MIHash [4]	0.929	0.933	0.938	0.942		0.767	0.784	0.809	0.834	
TALR-AP	**0.939**	**0.941**	**0.943**	**0.945**		**0.795**	**0.835**	**0.848**	**0.862**	

* Trained using parameters recommended by authors of DTSH.

Table 1: AP comparison on CIFAR-10 and NUS-WIDE with VGG-F architecture. On CIFAR-10, we compare all methods in the first setting (S1), and deep learning methods in the second (S2). We report the tie-aware $\mathrm{AP_T}$, and additionally AP@5K for NUS-WIDE. TALR-AP optimizes tie-aware AP using stochastic gradient ascent, and achieves state-of-the-art performance.

Method	16 Bits	32 Bits	48 Bits	64 Bits
HashNet [8]	0.5059	0.6306	0.6633	0.6835
MIHash [4]	0.5688	0.6608	0.6852	0.6947
TALR-AP	**0.5892**	**0.6689**	**0.6985**	**0.7053**

Table 2: AP@1000 results on ImageNet100 with AlexNet. TALR-AP outperforms state-of-the-art solutions using mutual information [4] and continuation methods [8].

pact codes has important implications for cases where memory and storage resources are restricted (*e.g.* mobile applications), and for indexing large-scale databases.

5.3. 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. Multi-level affinities are common in information retrieval tasks, and offer 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.

In the multi-label NUS-WIDE dataset, we define the affinity value between two examples as the number of labels they share, and keep other settings the same as in the AP experiment. For the unlabeled LabelMe dataset, we derive affinities by thresholding the Euclidean distances between examples. Inspired by an existing binary affinity setup [7] that defines neighbors as having Euclidean distance within the top 5% on the training set, we use four thresholds $\{5\%, 1\%, 0.2\%, 0.1\%\}$ and assign affinity values $\{1, 2, 5, 10\}$. This emphasizes 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.* $f_i(x; w) = w_i^\top x$, in (9). For methods that are not designed to use multi-level affinities (FastHash, MACHash, DPSH, MIHash), we convert the affinities into binary values; this reduces to the standard binary affinity setup on both datasets.

We give NDCG results in Table 3. Again, our method with the tie-aware NDCG objective (TALR-NDCG) outperforms all competing methods on both datasets. Interestingly, on LabelMe where all methods are restricted to learn shallow models on GIST features, we observe slightly different trends compared to other datasets. For example, without learning deep representations, DPSH and DTSH appear to perform less competitively, indicating a mismatch between their objectives and the evaluation metric. The closest competitors to TALR-NDCG on LabelMe are indeed the two listwise ranking methods: StructHash which optimizes a NDCG surrogate using boosted decision trees, and MIHash which is designed for binary affinities. TALR-NDCG outperforms both methods, and notably does so with linear hash functions, which have lower learning capacity compared StructHash's boosted decision trees. This highlights the benefit of our direct optimization formulation.

5.4. Effects of Tie-Breaking

We lastly discuss the effect of tie-breaking in evaluating hashing algorithms. As mentioned in Sec. 3.2, tie-breaking is an uncontrolled parameter in current evaluation protocols, which can affect results, and even be exploited. To demonstrate this, we consider for example the AP experiment in CIFAR-10's first setting, presented in Sec. 5.2. For each

Method	NUS-WIDE				LabelMe			
	16 Bits	32 Bits	48 Bits	64 Bits	16 Bits	32 Bits	48 Bits	64 Bits
BRE [18]*	0.805	0.817	0.827	0.834	0.807	0.848	0.871	0.880
MACHash [30]	0.821	0.821	0.821	0.821	0.683	0.683	0.683	0.687
FastHash [23]	0.885	0.896	0.899	0.902	0.844	0.868	0.855	0.864
DPSH [20]	0.895	0.905	0.909	0.909	0.844	0.856	0.871	0.874
DTSH [42]	0.896	0.905	0.911	0.913	0.838	0.852	0.859	0.862
StructHash [22]	0.889	0.893	0.894	0.898	0.857	0.888	0.904	0.915
MIHash [4]	0.886	0.903	0.909	0.912	0.860	0.889	0.907	0.914
TALR-NDCG	**0.903**	**0.910**	**0.916**	**0.927**	**0.866**	**0.895**	**0.908**	**0.917**

* Evaluated on the the 5K training subset due to kernel-based formulation.

Table 3: NDCG comparison on NUS-WIDE (VGG-F architecture) and LabelMe (shallow models on GIST features). TALR-NDCG optimizes tie-aware NDCG using stochastic gradient ascent, and consistently outperforms competing methods.

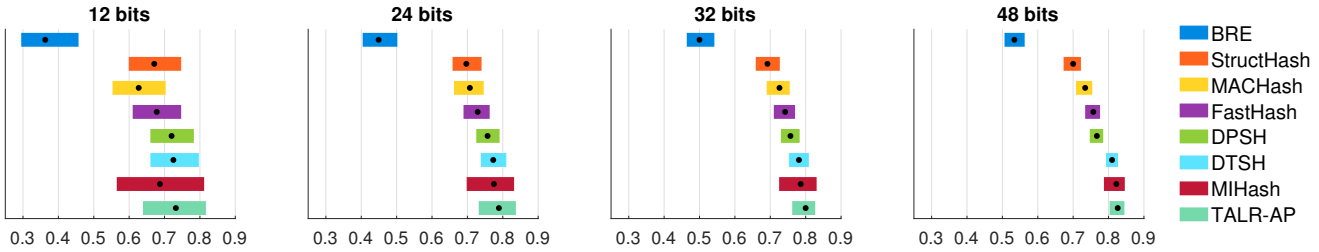

Figure 3: Effects of tie-breaking: we plot the ranges of test-time mAP values spanned by all possible tie-breaking strategies, for all methods considered in the CIFAR-10 experiment (first setting). Horizontal axis: mAP. Black dots: values of tie-aware AP_T. Without controlling for tie-breaking, relative performance comparison between different methods can be ambiguous. The ambiguity is eliminated by tie-awareness.

method included in this experiment, we plot the range of test set mAP spanned by all possible tie-breaking strategies. As can be seen in Fig. 3, the ranges corresponding to different methods generally overlap; therefore, without controlling for tie-breaking, relative performance comparison between different methods is essentially ambiguous. The ranges shrink as code length increases, since the number of ties generally decreases when there are more bins in the histogram.

Current hashing methods usually compute test-time AP and NDCG using random tie-breaking and general-purpose sorting algorithms. Interestingly, in all of our experiments, we observe that this produces values very close to the tie-aware AP_T and NDCG_T. The reason is that with a randomly ordered database, averaging the tie-unaware metric over a sufficiently large test set behaves similarly to the tie-aware solution of averaging over all permutations. Therefore, the results reported in the current literature are indeed quite fair, and so far we have found no evidence of exploitation of tie-breaking strategies. Still, we recommend using tie-aware ranking metrics in evaluation, as they completely eliminate ambiguity, and counting sort on Hamming distances is much more efficient than general-purpose sorting.

We note that although random tie-breaking is an approximation to tie-awareness at *test time*, it does not answer the question of how to *optimize* the ranking metrics during training. Our original motivation is to optimize ranking metrics for hashing, and the existence of closed-form tie-aware ranking metrics is what makes direct optimization feasible.

6. Conclusion

We have proposed a new approach to hashing for nearest neighbor retrieval, with an emphasis on directly optimizing evaluation metrics used at test-time. A study into the commonly used retrieval by Hamming ranking setup led us to consider the issue of ties, and we advocate for using tie-aware versions of ranking metrics. We then make the novel contribution of optimizing tie-aware ranking metrics for hashing, focusing on the important special cases of AP and NDCG. To tackle the resulting discrete and NP-hard optimization problems, we derive their continuous relaxations. Then, we perform end-to-end stochastic gradient ascent with deep neural networks. This results in the new state-of-the-art for common image retrieval benchmarks.

Acknowledgements

The authors would like to thank Qinxun Bai, Peter Gacs, and Dora Erdos for helpful discussions. This work is supported in part by a BU IGNITION award, NSF grant 1029430, and gifts from Nvidia.

References

[1] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proc. ACM Symposium on Theory of Computing (STOC)*, 2015.

[2] Christopher J. Burges, Robert Ragno, and Quoc V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[3] Guillaume Cabanac, Gilles Hubert, Mohand Boughanem, and Claude Chrisment. Tie-breaking bias: Effect of an uncontrolled parameter on information retrieval evaluation. In *International Conference of the Cross-Language Evaluation Forum*, 2010.

[4] Fatih Cakir, Kun He, Sarah Adel Bargal, and Stan Sclaroff. MIHash: Online Hashing with Mutual Information. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[5] Fatih Cakir, Kun He, Sarah Adel Bargal, and Stan Sclaroff. Hashing with mutual information. *arXiv preprint arXiv:1803.00974*, 2018.

[6] Fatih Cakir and Stan Sclaroff. Supervised hashing with error correcting codes. In *Proc. ACM International Conference on Multimedia*. ACM, 2014.

[7] Fatih Cakir and Stan Sclaroff. Adaptive hashing for fast similarity search. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2015.

[8] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. HashNet: Deep learning to hash by continuation. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[9] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proc. International Conference on Machine Learning (ICML)*, 2007.

[10] Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. Structured learning for non-smooth ranking losses. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2008.

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

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

[13] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao Zheng. NUS-WIDE: A real-world web image database from National University of Singapore. In *Proc. ACM CIVR*, 2009.

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

[15] Maurice G Kendall. *Rank correlation methods*. Griffin, 1948.

[16] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

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

[19] Andrey Kustarev, Yury Ustinovsky, Yury Logachev, Evgeny Grechnikov, Ilya Segalovich, and Pavel Serdyukov. Smoothing NDCG metrics using tied scores. In *Proc. ACM CIKM*, 2011.

[20] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

[21] Daryl Lim and Gert Lanckriet. Efficient learning of mahalanobis metrics for ranking. In *Proc. International Conference on Machine Learning (ICML)*, 2014.

[22] Guosheng Lin, Fayao Liu, Chunhua Shen, Jianxin Wu, and Heng Tao Shen. Structured learning of binary codes with column generation for optimizing ranking measures. *International Journal of Computer Vision (IJCV)*, 2016.

[23] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[24] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

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

[26] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*. Cambridge university press, 2008.

[27] Brian McFee and Gert R Lanckriet. Metric learning to rank. In *Proc. International Conference on Machine Learning (ICML)*, 2010.

[28] Frank McSherry and Marc Najork. Computing information retrieval performance measures efficiently in the presence of tied scores. In *Proc. European Conference on Information Retrieval*, 2008.

[29] Mohammad Norouzi, David J Fleet, and Ruslan R Salakhutdinov. Hamming distance metric learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[30] Ramin Raziperchikolaei and Miguel A Carreira-Perpinán. Optimizing affinity-based binary hashing using auxiliary coordinates. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[31] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision (IJCV)*, 77(1):157–173, 2008.

[32] Alexandre Sablayrolles, Matthijs Douze, Nicolas Usunier, and Hervé Jégou. How should we evaluate supervised hashing? In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[33] Dongjin Song, Wei Liu, Rongrong Ji, David A Meyer, and John R Smith. Top rank supervised binary coding for visual search. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2015.

[34] Yang Song, Alexander G. Schwing, Richard S. Zemel, and Raquel Urtasun. Training deep neural networks via direct loss minimization. In *Proc. International Conference on Machine Learning (ICML)*, 2016.

[35] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proc. ACM International Conference on Web Search and Data Mining (WSDM)*, 2008.

[36] Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. Few-shot learning through an information retrieval lens. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2252–2262, 2017.

[37] Evgeniya Ustinova and Victor Lempitsky. Learning deep embeddings with histogram loss. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[38] Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao. Learning to rank by optimizing NDCG measure. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[39] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

[40] Jun Wang, Wei Liu, Andy X Sun, and Yu-Gang Jiang. Learning hash codes with listwise supervision. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2013.

[41] Qifan Wang, Zhiwei Zhang, and Luo Si. Ranking preserving hashing for fast similarity search. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[42] Yi Wang, Xiaofang Shi and Kris M Kitani. Deep supervised hashing with triplet labels. In *Proc. Asian Conference on Computer Vision (ACCV)*, 2016.

[43] Chao-Yuan Wu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. Sampling matters in deep embedding learning. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[44] Zhou Yu, Fei Wu, Yin Zhang, Siliang Tang, Jian Shao, and Yueting Zhuang. Hashing with list-wise learning to rank. In *Proc. ACM SIGIR Conference on Research & Development in Information Retrieval*, 2014.

[45] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proc. ACM SIGIR Conference on Research & Development in Information Retrieval*, 2007.

[46] Lei Zhang, Yongdong Zhang, Jinhu Tang, Ke Lu, and Qi Tian. Binary code ranking with weighted hamming distance. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[47] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[48] Bohan Zhuang, Guosheng Lin, Chunhua Shen, and Ian Reid. Fast training of triplet-based deep binary embedding networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.