Deep Triplet Supervised Hashing

Xiaofang Wang

May 2017

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Kris Kitani, Co-Chair
Martial Hebert, Co-Chair
Abhinav Gupta
Xinlei Chen

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

CMU-RI-TR-17-19

For my family

Abstract

Hashing is one of the most popular and powerful approximate nearest neighbor search techniques for large-scale image retrieval. Most traditional hashing methods first represent images as off-the-shelf visual features and then produce hash codes in a separate stage. However, off-the-shelf visual features may not be optimally compatible with the hash code learning procedure, which may result in sub-optimal hash codes. Recently, deep hashing methods have been proposed to simultaneously learn image features and hash codes using deep neural networks and have shown superior performance over traditional hashing methods. The current state-of-the-art deep hashing method DPSH [14] has demonstrated great performance but it suffers the problem of unbalanced supervision signals. To address this issue, we propose two novel deep hashing methods DPSH-Weighted and DTSH. At the heart of our methods DPSH-Weighted and DTSH is the novel formulation of weighted pairwise label likelihood and triplet label likelihood. Our methods learn images features and hash codes by maximizing the label likelihood. One common problem that most deep hashing methods face is how to train a network to output binary codes. We provide in-depth analysis of two typical methods to train such a network. Extensive experiments on four public benchmark datasets, including CIFAR-10 [10], NUS-WIDE [3], Standford Cars [9] and UKBench [23], show that both DPSH-Weighted and DTSH outperform the state-of-the-art method DPSH [14], and DTSH can obtain the highest performance among all the current deep hashing methods.

Acknowledgments

First and foremost, I would like to offer my special thanks to my advisors Kris Kitani and Martial Hebert, for their guidance and support during my master study. They give me such a great environment to do research. Their enthusiasm, insight and wisdom keep inspiring and guiding me throughout my master study. I could not have imagined having better advisors for my master.

I also would like to thank Abhinav Gupta and Xinlei Chen for being my committee members. Thanks Abhinav for giving critical and insightful suggestions on my work. Thanks Xinlei for helping me prepare the thesis presentation and giving insightful comments. The thesis would not have been possible without their help.

Dozens of people have helped me during my master study. I wish to thank all my labmates, friends and fellow students for the help they gave me and loads of fun we have had together. I especially want to thank Wei-Chiu Ma, Chenyang Li, Yi Shi, Anqi Li, Shen Li and Mengtian Li for all their support.

Last but far from the least, I would like to thank my parents and my family for their continuous love and support throughout my life.

Chapter 1

Introduction

With the rapid growth of image data on the Internet, much attention has been devoted to approximate nearest neighbor (ANN) search. Hashing is one of the most popular and powerful techniques for ANN search due to its computational and storage efficiencies. Hashing aims to map high dimensional image features into compact hash codes or binary codes so that the Hamming distance between hash codes approximates the Euclidean distance between image features.

Many hashing methods have been proposed and they can be categorized into data-independent and data-dependent methods. Compared with the data-dependent methods, data-independent methods need longer codes to achieve satisfactory performance [5]. Data-dependent methods can be further categorized into unsupervised and supervised methods. Compared to unsupervised methods, supervised methods usually can achieve competitive performance with fewer bits due to the help of supervised information, which is advantageous for search speed and storage efficiency [13].

Most existing hashing methods first represent images as off-the-shelf visual features such as GIST [25], SIFT [22] and the hash code learning procedure is independent of the features of images. However, off-the-shelf visual features may not be optimally compatible with hash code learning procedure. In other words, the similarity between two images may not be optimally preserved by the visual features and thus the learned hash codes are sub-optimal [13]. Therefore, those hashing methods may not be able to achieve satisfactory performance in practice. To address the drawbacks of hashing methods that rely on off-the-shelf visual features, feature learning based deep hashing methods [13, 14, 18, 34, 35] have been proposed to simultaneously learn image feature and hash codes with deep neural networks and have demonstrated superior performance over traditional hashing methods. Most proposed deep hashing methods fall into the category of supervised hashing methods. Supervised information is given in the form of pairwise labels or triplet labels, a special case of ranking labels.

Deep Pairwise-Supervised Hashing (DPSH) [14] is the current state-of-the-art deep hashing method, which is supervised by pairwise labels. Similar to LFH [34], DPSH aims to maximize the likelihood of the pairwise similarities, which is modeled as a function of the Hamming distance between the corresponding data points.

Pairwise labels consist of two forms of labels: positive image pairs and negative image pairs. Two similar images form a positive image pair and two dissimilar images form a negative image pair. DPSH [14] uniformly sample image pairs, which usually results in that much fewer positive

image pairs are sampled than negative image pairs, *i.e.*, the supervision signals are unbalanced. The model will see much more negative image pairs than positive image pairs during training. However, in their formulation of the likelihood of the given pairwise labels, positive and negative image pairs are considered as equally important. To sum up, DPSH [14] feeds the model with unbalanced supervision signals but does not take that unbalance into account in their objective function when training the model. As a result, the model will pay much attention to pulling negative image pairs further but ignore pushing positive image pairs closer during training.

Ignoring the unbalance of supervision signals will greatly harm the performance of the model. To overcome the unbalance of supervision signals, we propose the following two solutions:

- *DPSH-Weighted*: revise the objective function to consider the unbalance;
- *DTSH*: balance the supervision signals by employing triplet labels as supervision.

DPSH-Weighted: We take the unbalance of supervision signals into consideration in the objective function. To that end, we propose a novel formulation of the *weighted* likelihood of the given pairwise labels, which is modeled as the weighted product of the likelihood of positive and negative image pairs. Our model is trained by maximizing the weighted likelihood of the given pairwise labels. We name this method as *Weighted Deep Pairwise-Supervised Hashing* (DPSH-Weighted).

DTSH: Different from pairwise labels, triplet labels only consist one single form: (query, positive, negative). Using triplet labels as supervision eliminates the need to worry about the balance of supervision signals. More importantly, triplet labels inherently contain richer information than pairwise labels. Each triplet label can be naturally decomposed into two pairwise labels. Whereas, a triplet label can be constructed from two pairwise labels only when the same query image is used in a positive pairwise label and a negative pairwise label simultaneously. A triplet label ensures that in the learned hash code space, the query image is close to the positive image and far from the negative image simultaneously. However, a pairwise label can only ensure that one constraint is observed. Triplet labels explicitly provide a notion of relative similarities between images while pairwise labels can only encode that implicitly. Therefore, we propose a novel triplet label based deep hashing method, which learns image features and hash codes by maximizing the likelihood of the given triplet labels. We name this method as *Deep Triplet-Supervised Hashing* (DTSH).

Both DPSH-Weighted and DTSH perform image feature learning and hash code learning simultaneously and are end-to-end trainable. As shown in Fig. 3.1, the proposed model has three key components: (1) image feature learning component: convolution layers and fully connected layers to learn visual features from images, (2) hash code learning component: one fully connected layer to learn hash codes from image features and (3) objective function component: an objective function to measure how well the given labels (pairwise labels or triplet labels) are satisfied by the learned hash codes by computing the label likelihood. All the components can be seamlessly integrated to a convolution neural network.

Since the target of deep hashing is to obtain binary codes from the network, our proposed methods and previous deep hashing methods [13, 14, 18, 34, 35] all face the common problem of how to train a network to output binary codes. Typically, deep hashing methods obtain binary codes by adopting the sign function as the activation function for the last layer of the network. The sign function quantizes the continuous output of the network to binary codes. However, this

enhances the difficulty of training the network. Importantly, the gradient of the sign function equals to zero (ignoring the point discontinuity at $x = 0$). This means when back-propagating gradients from the last layer, all the gradients will be zero and back-propagation fails to train the network. Generally speaking, there are two methods to solve this problem: (1) approximate the sign function with a sigmoid function such that the gradient is non-zero, or (2) relax the binary codes and add a quantization error term in the objective function during training. We perform in-depth analysis of the two methods to train the network and demonstrate that the second method performs much better than the first one.

Extensive experiments on standard benchmark datasets such as CIFAR-10 [10] and NUS-WIDE [3] show that our proposed deep hashing methods DPSH-Weighted and DTSH outperform all the baselines, including the state-of-the-art method DPSH [14] and all the previous triplet label based deep hashing methods. We also compare our proposed methods to DPSH [14] on another two datasets Standford Cars [9] and UKBench [23]. Experiments show that DPSH-Weighted and DTSH also significantly outperform DPSH on Standford Cars and UKBench. In particular, DTSH achieves the highest performance among all the methods. The four datasets represent three different application scenarios respectively: (1) category level similarity, (2) fine-grained category level similarity and (3) instance level similarity. The four dataets also have different characteristics in terms the number of classes and the number of images per class. We believe evaluating on the four benchmark datasets can provide a comprehensive understanding of the performance of our methods.

Our main contributions are outlined as follows: (1) To overcome the unbalance of supervision signals in DPSH [14], we present two novel deep hashing methods DPSH-Weighted and DTSH. (2) In DPSH-Weighted, we propose a novel formulation of the weighted likelihood of the given pairwise labels. (3) In DTSH, we propose a novel formulation of the likelihood of the given triplet labels. We also introduce a margin parameter in DTSH to speed up the training. (4) We provide in-depth analysis of two typical methods to train a network with binary output. (5) We obtain state-of-the-art performance on four benchmark datasets.

The rest of the thesis is organized as follows. We review the related work in Chapter 2. Chapter 3 introduces our proposed deep hashing methods. We give the experimental results in Chapter 4 and conclude the thesis in Chapter 5.

Chapter 2

Related Work

Hashing methods can be categorized into data-independent and data-dependent methods, based on whether they are independent of training data. Representative data-independent methods include Locality Sensitive Hashing (LSH) [1] and Shift-Invariant Kernels Hashing (SIKH) [26]. Data-independent methods generally need longer hash codes for satisfactory performance, compared to data-dependent methods. Data-dependent methods can be further divided into unsupervised and supervised methods, based on whether supervised information is provided or not.

Unsupervised hashing methods only utilize the training data points to learn hash functions, without using any supervised information. For example, Spectral Hashing (SH) [31] seeks compact binary codes by solving graph partitioning problem. Binary Reconstructive Embedding (BRE) [12] learns hash functions based on explicitly minimizing the reconstruction error between the original distances and the Hamming distances of the corresponding hash codes. Iterative Quantization (ITQ) [5] formulate the problem of learning a good binary code in terms of directly minimizing the quantization error of mapping this data to vertices of the binary hypercube. Some other notable examples of unsupervised hashing methods include Isotropic Hashing(IsoHash) [8], some graph-based hashing methods [7, 19, 21] and two deep hashing methods Semantic Hashing [27] and the hashing method proposed in [4].

Supervised hashing methods leverage labeled data to learn hash codes. Typically, the supervised information are provided in one of three forms: point-wise labels, pairwise labels or ranking labels [14]. Representative point-wise label based methods include CCA-ITQ [5] and the deep hashing method proposed in [17]. Representative pairwise label based hashing methods include Minimal Loss Hashing (MLH) [24], Supervised Hashing with Kernels (KSH) [20], Latent Factor Hashing (LFH) [33], Fash Supervised Hashing (FASTH) [16] and two deep hashing methods: CNNH [32] and DPSH [14]. Representative ranking label based hashing methods include Ranking-based Supervised Hashing(RSH) [30], Column Generation Hashing (CGHASH) [15] and the deep hashing methods proposed in [4, 13, 34, 35]. One special case of ranking labels is triplet labels.

Most existing supervised hashing methods represent images as off-the-shelf visual features and perform hash code learning independent of visual features, including some deep hashing methods [4, 27]. However, off-the-shelf features may not be optimally compatible with hash code learning procedure and thus results in sub-optimal hash codes. Therefore, deep hashing methods are proposed.

Most deep hashing methods are supervised by pairwise labels or triplet labels. Pairwise label based deep hashing methods include CNNH [32], DSH [18], and DPSH [14]. CNNH is the first proposed deep hashing method without using off-the-shelf features. However, CNNH cannot learn image features and hash codes simultaneously and still has limitations. This has been verified by the authors of CNNH themselves in a follow-up work [13]. Deep Pairwise-Supervised Hashing (DPSH) [14] is the current state-of-the-art deep hashing method, which is able to simultaneously perform image feature learning and hash code learning with pairwise labels and achieves highest performance compared to other deep hashing methods. Our method DPSH-Weighted is the first to consider the weight between the positive image pairs and negative image pairs among all the pairwise label based deep hashing method.

Ranking label or triplet label based deep hashing methods include Network in Network Hashing (NINH) [13], Deep Semantic Ranking based Hashing (DSRH) [35], Deep Regularized Similarity Comparison Hashing (DRSCH) [34] and Deep Similarity Comparison Hashing (DSCH) [34]. While these methods can simultaneously perform image feature learning and hash code learning given supervision of triplet labels, we present a novel formulation of the likelihood of the given triplet labels to evaluate the quality of learned hash codes in our proposed method DTSH.

Chapter 3

Method

We first give the formal problem definition of supervised hashing with pairwise labels and supervised hashing with triplet labels. Then, we introduce the framework of our proposed methods DPSH-Weighted and DTSH. We then describe our formulation of the weighted pairwise label likelihood used in DPSH-Weighted and the triplet label likelihood in used in DTSH. Finally, we describe how we learn the model.

3.1 Problem Definition

The problem definition of supervised hashing with pairwise labels and supervised hashing with triplet labels are the same with each other except the provided labels, so we describe them together.

We are given N training images $\mathcal{I} = \{I_1, \ldots, I_N\}$, as well as M pairwise labels or M triplet labels[1]. We denote the given M pairwise labels as $\mathcal{S} = \{s_{q_{11}q_{12}}, \ldots, s_{q_{M1}q_{M2}}\}$, where $s_{q_{m1}q_{m2}}$ indicates whether the image of index q_{m1} ($I_{q_{m1}}$) and the image of index q_{m2} ($I_{q_{m2}}$) are similar to each other. To be more concrete, $s_{q_{m1}q_{m2}} = 1$ means $I_{q_{m1}}$ and $I_{q_{m2}}$ are similar to each other, and $s_{q_{m1}q_{m2}} = 0$ means $I_{q_{m1}}$ and $I_{q_{m2}}$ are dissimilar from each other. The M triplet labels are represented as $\mathcal{T} = \{(q_1, p_1, n_1), \ldots, (q_M, p_M, n_M)\}$, where the triplet of image indices (q_m, p_m, n_m) denotes that the query image of index q_m (I_{q_m}) is more similar to the positive image of index p_m (I_{p_m}) than to the negative image of index n_m (I_{n_m}).

One possible way of determining the value of $s_{q_{m1}q_{m2}}$ is to see if image $I_{q_{m1}}$ and image $I_{q_{m2}}$ are in the same semantic class or not. Similarly, one possible way of generating triplet labels is by selecting the query and positive images (I_{q_m} and I_{p_m}) from the same semantic class and selecting the negative image (I_{n_m}) from a different semantic class.

Our goal is to learn a hash code $\mathbf{b}_n$ for each image I_n, where $\mathbf{b} \in \{+1, -1\}^L$ and L is the length of hash codes. The hash codes $\mathcal{B} = \{\mathbf{b}_n\}_{n=1}^N$ should satisfy all the pairwise labels $\mathcal{S}$ or the triplet labels $\mathcal{T}$ as much as possible in the Hamming space. More specifically, for pairwise labels, $\text{dist}_H(\mathbf{b}_{q_{m1}}, \mathbf{b}_{p_{m2}})$ should be as small as possible if $s_{q_{m1}q_{m2}} = 1$, otherwise $\text{dist}_H(\mathbf{b}_{q_{m1}}, \mathbf{b}_{p_{m2}})$ should be as large as possible. For triplet labels, $\text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{p_m})$ should be

[1] We only consider the case that we are given pairwise labels alone or triplet labels alone. We do not consider the case that both pairwise labels and triplet labels are provided.

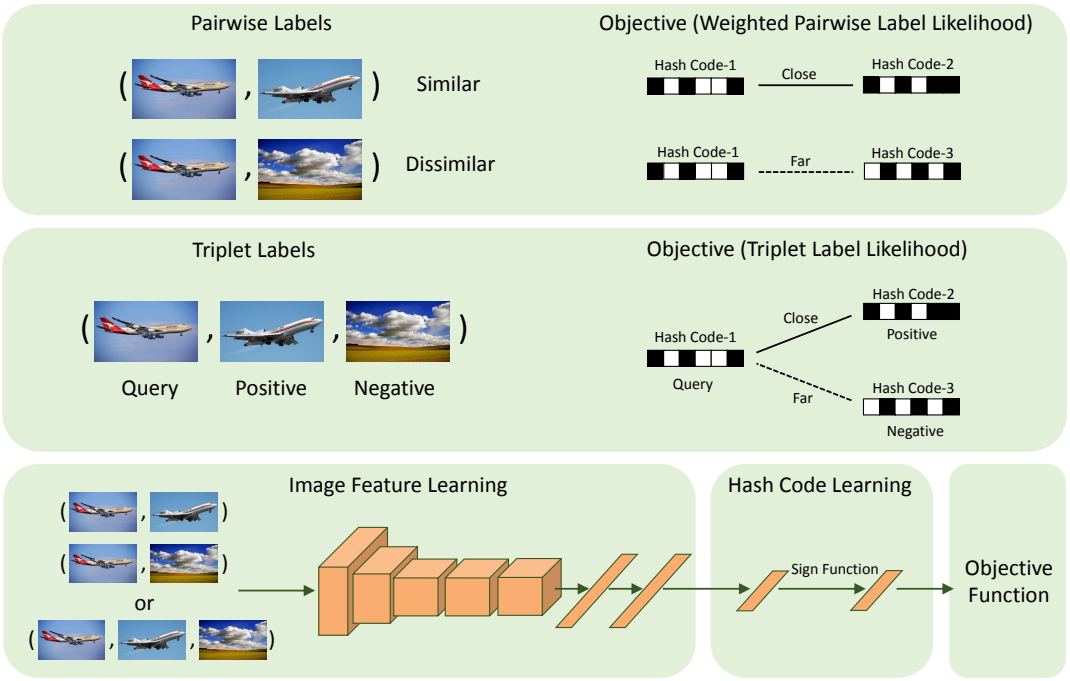


Figure 3.1: Framework of the proposed end-to-end deep hashing methods. DPSH-Weighted is supervised by pairwise labels and DTSH is supervised by triplet labels. In the hash code learning component, we explicitly show that the sign function is adopted as the activation function to obtain binary output. The objective function component evaluates how well the given labels are satisfied by the learned hash codes.

smaller than $\text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{n_m})$ as much as possible. Here, $\text{dist}_H(\cdot, \cdot)$ denotes the Hamming distance between two hash codes.

Generally speaking, we aim to learn a hash function $h(\cdot)$ to map images to hash codes. We can write $h(\cdot)$ as $[h_1(\cdot), ..., h_L(\cdot)]$ and for image I_n, its hash code can be denoted as $\mathbf{b}_n = h(I_n) = [h_1(I_n), ..., h_L(I_n)]$.

3.2 Framework

We now introduce the framework of our proposed methods DPSH-Weighted and DTSH. Most previous hashing methods rely on off-the-shelf visual features, which may not be optimally compatible with the hash code learning procedure. Thus deep hashing methods[13, 14, 18, 34, 35] are proposed to simultaneously learn image features and hash codes from images. We propose two novel deep hashing methods: DPSH-Weighted and DTSH, that utilizes pairwise labels and triplet labels respectively. Both methods can simultaneously perform image feature learning and hash code learning in an end-to-end manner. As shown in Fig. 3.1, our methods consist of three key components: (1) an image feature learning component, (2) a hash code learning component and (3) a objective function component. We introduce the image feature learning and hash code learning component in this section, and the objective function component in the next section.

3.2.1 Image Feature Learning

This component is designed to employ several convolution layers and fully connected layers to learn visual features from images. We adopt the CNN-F network architecture [2] for this component. CNN-F has eight layers, where the last layer is designed to learn the probability distribution over category labels. So only the first seven layers of CNN-F are used in this component. We use the same network architecture with DPSH [14]. Other networks like AlexNet [11], residual network [6] can also be used in this component.

3.2.2 Hash Code Learning

This component is designed to learn hash codes of images. We employ one fully connected layer with the element-wise sign function as the activation function. The sign function is adopted to obtain binary codes. In particular, the number of neurons of the fully connected layers equals the length of targeted hash codes. Multiple fully connected layers or other architectures like the divide-and-encode module proposed by [13] can also be applied here. We do not focus on this in this paper and leave this for future study.

3.3 Objective Function

The objective function component measures how well the given labels (pairwise or triplet labels) are satisfied by the learned hash codes by computing the likelihood of the given labels. In this section, we first motivate our methods by introducing the formulation of the pairwise label likelihood used in DPSH [14]. Then, we explain our formulation of the weighted pairwise label likelihood used in DPSH-Weighted and the triplet label likelihood used in DTSH. We call them *weighted pairwise label likelihood* and *triplet label likelihood* respectively throughout the text.

3.3.1 DPSH: Pairwise Label Likelihood

Let Θ_{ij} denote half of the inner product between two hash codes $\mathbf{b}_i, \mathbf{b}_j \in \{+1, -1\}^L$:

$$\Theta_{ij} = \frac{1}{2}\mathbf{b}_i^T \mathbf{b}_j. \tag{3.1}$$

Then the pairwise label likelihood is formulated as:

$$p(\mathcal{S} \mid \mathcal{B}) = \prod_{m=1}^{M} p(s_{q_{m1}q_{m2}} \mid \mathcal{B}), \tag{3.2}$$

with

$$p(s_{q_{m1}q_{m2}} \mid \mathcal{B}) = \begin{cases} \sigma(\Theta_{q_{m1}q_{m2}}), & s_{q_{m1}q_{m2}} = 1 \\ 1 - \sigma(\Theta_{q_{m1}q_{m2}}), & s_{q_{m1}q_{m2}} = 0 \end{cases}, \tag{3.3}$$

where $\sigma(x)$ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and $\mathcal{B}$ is the set of all hash codes.

We now show how maximizing the pairwise label likelihood matches the goal to preserve the similarity relationships between images. We first prove the following relationship between the Hamming distance between two binary codes and their inner product:

$$\text{dist}_H(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{2}(L - 2\Theta_{ij}), \tag{3.4}$$

where L is the length of the hash codes.

According to Eq. 3.3, we know that if image $I_{q_{m1}}$ and image $I_{q_{m2}}$ are similar to each other, *i.e.*, $s_{q_{m1}q_{m2}} = 1$, the larger $p(s_{q_{m1}q_{m2}} \mid \mathcal{B})$ is, the larger $\Theta_{q_{m1}q_{m2}}$ will be. According to Eq. 3.4, the larger $\Theta_{q_{m1}q_{m2}}$ is, the smaller $\text{dist}_H(\mathbf{b}_{q_{m1}}, \mathbf{b}_{q_{m2}})$ will be. Thus, by maximizing the pairwise label likelihood $p(\mathcal{S} \mid \mathcal{B})$, we can enforce the Hamming distance between similar images to be small. Similarly, we can show that, by maximizing $p(\mathcal{S} \mid \mathcal{B})$, we can also enforce the Hamming distance between dissimilar images to be large.

DPSH [14] aims to minimize the negative log pairwise label likelihood, which is defined as follows,

$$\begin{aligned} L_p &= -\log p(\mathcal{S} \mid \mathcal{B}) \\ &= -\sum_{m=1}^{M} \log p(s_{q_{m1}q_{m2}} \mid \mathcal{B}), \end{aligned} \tag{3.5}$$

where the subscript p denotes that L_p is derived from the pairwise label likelihood. Plus Eq. 3.3, we can derive that

$$L_p = -\sum_{m=1}^{M} (s_{q_{m1}q_{m2}}\Theta_{q_{m1}q_{m2}} - \log(1 + e^{\Theta_{q_{m1}q_{m2}}})). \tag{3.6}$$

DPSH [14] uniformly sample image pairs, thus usually much fewer positive image pairs are sampled than negative image pairs during the training stage. For example, CIFAR-10 [10] consists of 10 categories and each category has an equal number of images. Uniformly sampling can only obtain a single positive image pair out of 10 image pairs. UKBench [23] contains thousands of different object instances and each instance has 4 examples in the dataset. Uniformly sampling will give us only one single positive image pair out of thousands of image pairs. Uniform sampling feeds the model with unbalanced supervision signals so that the model sees much more negative image pairs than positive image pairs during the training stage. However, DPSH [14] does not take the unbalance of supervision signals into account in their objective function when training the model. As a result, the model will pay much attention to pulling negative image pairs further but ignore pushing positive image pairs closer. This will greatly degrade the performance of the model.

To overcome the unbalance of supervision signals, we propose two novel deep hashing methods DPSH-Weighted and DTSH, which are detailed in the following sections. In principle, DTSH-Weighted solves the problem of unbalance by modifying the objective function to consider the unbalance of supervision signals. DTSH deals with the problem by balancing the supervision signals.

3.3.2 DPSH-Weigtehd: Weighted Pairwise Label Likelihood

DPSH [14] ignores the unbalance of supervision signals in their objective function. One natural way to solve this problem is to modify the objective function to take the unbalance into consideration. This motivates us to propose the following formulation of the weighted pairwise label likelihood, which is modeled as the weighted product of the likelihood of positive and negative image pairs:

$$\begin{aligned} p(\mathcal{S} \mid \mathcal{B}) = & \prod_{m \in M_P} p(s_{q_{m1}q_{m2}} \mid \mathcal{B})^{\lambda} \\ & \cdot \prod_{m \in M_N} p(s_{q_{m1}q_{m2}} \mid \mathcal{B}), \end{aligned} \tag{3.7}$$

where λ is the weight parameter for positive image pairs, $M_P = \{m \mid 1 \leq m \leq M, s_{q_{m1}q_{m2}} = 1\}$ and $M_N = \{m \mid 1 \leq m \leq M, s_{q_{m1}q_{m2}} = 0\}$. M_P is the set of positive image pairs and M_N is the set of negative image pairs. $p(s_{q_{m1}q_{m2}} \mid \mathcal{B})$ is already defined in Eq. 3.3.

DPSH-Weigthed trains the model by minimizing the negative log weighted pairwise label likelihood, which is equivalent to the weighted sum of the negative log likelihood of positive and negative image pairs:

$$\begin{aligned} L_{wp} = & -\log p(\mathcal{S} \mid \mathcal{B}) \\ = & -\lambda \sum_{m \in M_P} \log p(s_{q_{m1}q_{m2}} \mid \mathcal{B}) \\ & - \sum_{m \in M_N} \log p(s_{q_{m1}q_{m2}} \mid \mathcal{B}) \\ = & -\lambda \sum_{m \in M_P} (s_{q_{m1}q_{m2}} \Theta_{q_{m1}q_{m2}} - \log(1 + e^{\Theta_{q_{m1}q_{m2}}})) \\ & - \sum_{m \in M_N} (s_{q_{m1}q_{m2}} \Theta_{q_{m1}q_{m2}} - \log(1 + e^{\Theta_{q_{m1}q_{m2}}})), \end{aligned} \tag{3.8}$$

where the subscript wp emphasizes that different from L_p in Eq. 3.6, L_{wp} puts an weight λ on positive image pairs. The larger the weight λ is, the more attention the model will pay to pushing positive image pairs closer.

3.3.3 DTSH: Triplet Label Likelihood

Another way to overcome the unbalance of supervision signals is to balance the supervision signals, *i.e.*, sample a balanced number of different forms of supervision signals. Uniformly sampling in DPSH [14] leads to an unbalanced number of positive and negative image pairs. One may propose to devise sampling methods such that an equal number of positive and negative image pairs are sampled. However, in some case, there exists a very limited number of positive image pairs. For example, UKBench [23] has thousands of object instances but each instance only has 4 images. For UKBench [23], if we enforce the number of positive and negative image

pairs to be the same, we will miss the opportunity to feed the model with a large number of negative image pairs, which are also valuable for the model to learn about the similarity between images. To avoid this drawback of pairwise labels, we propose an alternative method.

We propose to employ triplet labels as supervision. Different from pairwise labels, triplet labels only have one single form: (query, positive, negative). This means the problem of unbalance is naturally resolved. Using triplet labels eliminates the need to worry about the balance of supervision signals. Triplet labels have additional advantages over pairwise labels. Triplet labels inherently contain richer information than pairwise labels. A triplet label ensures that in the learned hash code space, the query image is close to the positive image and far from the negative image simultaneously. However, a pairwise label can only ensure that one constraint is observed. Triplet labels explicitly provide a notion of relative similarities between images while pairwise labels can only encode that implicitly.

Given a set of triplet labels $\mathcal{T}$, the triplet label likelihood is formulated as:

$$p(\mathcal{T} \mid \mathcal{B}) = \prod_{m=1}^{M} p((q_m, p_m, n_m) \mid \mathcal{B}), \tag{3.9}$$

with

$$p((q_m, p_m, n_m) \mid \mathcal{B}) = \sigma(\Theta_{q_m p_m} - \Theta_{q_m n_m} - \alpha), \tag{3.10}$$

where $\sigma(x)$ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, α is the margin, a positive hyper-parameter and $\mathcal{B}$ is the set of all hash codes.

We now explain how maximizing the triplet label likelihood fits our goal to preserve the relative similarity between the query, positive and negative images. According to Eq. 3.4, we can have

$$\begin{aligned} &\text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{p_m}) - \text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{n_m}) \\ &= -(\Theta_{q_m p_m} - \Theta_{q_m n_m}). \end{aligned} \tag{3.11}$$

According to Eq. 3.10, we know that the larger $p((q_m, p_m, n_m) \mid \mathcal{B})$ is, the larger $(\Theta_{q_m p_m} - \Theta_{q_m n_m} - \alpha)$ will be. Since α is a constant number here, the larger $(\Theta_{q_m p_m} - \Theta_{q_m n_m} - \alpha)$ is, the smaller $(\text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{p_m}) - \text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{n_m}))$ will be. Thus, by maximizing the triplet label likelihood $p(\mathcal{T} \mid \mathcal{B})$, we can enforce the Hamming distance between the query and the positive image to be smaller than that between the query and the negative image. The margin α here can regularize the distance gap between $\text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{p_m})$ and $\text{dist}_H(\mathbf{b}_{q_m}, \mathbf{b}_{n_m})$. The margin α can also help speed up training our model as explained later in this section and verified in our experiments in Section 4.4.

Now we define our objective function as the negative log triplet label likelihood as follows,

$$\begin{aligned} L_t &= -\log p(\mathcal{T} \mid \mathcal{B}) \\ &= -\sum_{m=1}^{M} \log p((q_m, p_m, n_m) \mid \mathcal{B}), \end{aligned} \tag{3.12}$$

where the subscript t denotes that L_t is derived from the triplet label likelihood. Plug Eq. 3.10

into the above equation, we can derive that

$$L_t = -\sum_{m=1}^{M}(\Theta_{q_m p_m} - \Theta_{q_m n_m} - \alpha \\ - \log(1 + e^{\Theta_{q_m p_m} - \Theta_{q_m n_m} - \alpha})). \tag{3.13}$$

Impact of Margin α: We argue that a positive margin α can help speed up the training process. We now analyze this theoretically by looking at the derivative of the objective function. In particular, for n^{th} image, we compute the derivative of the objective function L_t with respect to $\mathbf{b}_n$ as follows:

$$\begin{aligned} \frac{\partial L_t}{\partial \mathbf{b}_n} = &-\frac{1}{2}\sum_{m:(n,p_m,n_m)\in\mathcal{T}} d_m \cdot (\mathbf{b}_{p_m} - \mathbf{b}_{n_m}) \\ &-\frac{1}{2}\sum_{m:(q_m,n,n_m)\in\mathcal{T}} d_m \cdot \mathbf{b}_{q_m} \\ &+\frac{1}{2}\sum_{m:(q_m,p_m,n)\in\mathcal{T}} d_m \cdot \mathbf{b}_{q_m} \end{aligned} \tag{3.14}$$

where $d_m = (1 - \sigma(\Theta_{q_m p_m} - \Theta_{q_m n_m} - \alpha))$, $\sigma(x)$ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and $\mathcal{T}$ is the set of triplet labels. In the derivative shown above, we observe the term d_m. We know that $\sigma(x)$ saturates very quickly, *i.e.*, being very close to 1, as x increases. If $\alpha = 0$, when $(\Theta_{q_m p_m} - \Theta_{q_m n_m})$ becomes positive, the term d_m will be very close to 0. This will make the magnitude of the derivative very small and further make the model hard to train. A positive margin α adds a negative offset on $(\Theta_{q_m p_m} - \Theta_{q_m n_m})$ and can prevent d_m from being very small. Further this makes the model easier to train and helps speed up the training process. We give experiment results to verify this in Section 4.4.

3.4 Model Learning

One common problem that our proposed methods and previous deep hashing methods [13, 14, 18, 34, 35] all face is how to train the network whose output is binary codes. As shown in Fig. 3.1, we adopt the sign function as the activation function of the last layer of the network to obtain binary codes. Note that the gradient of the sign function always equals to zero (ignoring the point discontinuity at $x = 0$). This means when back-propagating gradients from the last layer to previous layers, all the gradients will be zero. In this case, back-propagation fails to train the network. We would like to emphasize that it is not the discontinuity and non-differentiability of the sign function at $x = 0$ that makes the network hard to train. What makes the network hard to train is that the the gradient of sign function is always zero. We now describe two typical methods used in deep hashing to solve this problem and compare the two methods with comprehensive experimental results in Section 4.4.

The reason why back-propagation fails is that the gradient of the sign function always equals to zero. So the first method is to approximate the sign function with a sigmoid function such that the gradient is non-zero. Similar methods are used in previous deep hashing methods [13, 34, 35].

To be more concrete, in our paper, we employ a tanh-like function to approximate the sign function, which is defined as follows:

$$t(x) = \frac{1 - e^{-\beta x}}{1 + e^{-\beta x}}, \tag{3.15}$$

where β is a parameter that controls the shape of the tanh-like function $t(x)$. The larger the value of β is, the better $t(x)$ will approximate the sign function is but the smaller gradident $t(x)$ will have. So we employs a curriculum learning strategy that progressively increases the value of β during training. For example, we set the initial value of β to 2 and increase β by 2 every epoch. This strategy is also utilized in [34]. $t(x)$ is continuous and has non-zero gradients, thus now we can use the usual back-propagation to train the network. During the test time, we still adopt the sign function to obtain binary codes.

The second method is to relax the binary codes and add a quantization error term in the objective function during training. This method is also utilized in [14, 18]. To formally describe this method, we use $\mathbf{u}_n$ to denote the continuous output of the last layer before the sign function for image I_n. Thus, as defined above, $\mathbf{b}_n$, the hash code of image I_n, can be obtained by $\mathbf{b}_n = sgn(\mathbf{u}_n)$, where $sgn(\cdot)$ is the element-wise sign function and $sgn(\mathbf{u}_n^{(k)})$ equals to 1 if $\mathbf{u}_n^{(k)} > 0$ and -1, otherwise. Specifically, we relax binary codes $\{\mathbf{b}_n\}$ to real vectors $\{\mathbf{u}_n\}$ and re-define Θ_{ij} as

$$\Theta_{ij} = \frac{1}{2}\mathbf{u}_i^T\mathbf{u}_j. \tag{3.16}$$

We then rewrite our objective function for DPSH-Weighted and DTSH as

$$L_{wpq} = L_{wp} + \eta \sum_{n=1}^{N} ||\mathbf{b}_n - \mathbf{u}_n||_2^2, \tag{3.17}$$

and

$$L_{tq} = L_t + \eta \sum_{n=1}^{N} ||\mathbf{b}_n - \mathbf{u}_n||_2^2, \tag{3.18}$$

respectively. L_{wp} and L_t are defined in Eq. 3.8 and Eq. 3.13 respectively but Θ_{ij} has been redefined in Eq. 3.16. $\sum_{n=1}^{N} ||\mathbf{b}_n - \mathbf{u}_n||_2^2$ represents the quantization error term. η is a hyper-parameter to balance the original objective function and the quantization error. The new objective functions L_{wpq} and L_{tq} are differentiable with respect to $\mathbf{u}_n$. For example, regarding L_{tq}, L_t is differentiable with respect to $\mathbf{u}_n$ and the quantization error term $\sum_{n=1}^{N} ||\mathbf{b}_n - \mathbf{u}_n||_2^2$ is also differentiable with respect to $\mathbf{u}_n$ (in practice we can ignore that $\mathbf{b}_n = sgn(\mathbf{u}_n)$ is non-differentiable at 0), therefore L_{tq} is differentiable with respect to $\mathbf{u}_n$. During the training stage, we can ignore the sign function in the last layer and optimize the network by back-propagating the new objective function's gradient with respect to $\mathbf{u}_n$ to previous layer.

To sum up, we have described two typical methods to train the network. Once training is completed, we can apply our network to generate hash codes for new images. For a new image I, we pass it into the trained network and take the continuous output of the last layer $\mathbf{u}$. Then the element-wise sign function is adopted to obtain the hash code $\mathbf{b}$ of image I, *i.e.*, $\mathbf{b} = sgn(\mathbf{u})$.

Chapter 4

Experiment

4.1 Datasets and Evaluation Protocol

We conduct experiments on four publicly benchmark datasets, including CIFAR-10 [10], NUS-WIDE [3], Standford Cars [9] and UKBench [23]. The four datasets represent three different types of image similarity. In particular, CIFAR-10 [10] and NUS-WIDE [3] represent the object category level similarity between images, Stanford Cars [9] represents the fine-grained category level similarity and UKBench [23] represents the object instance level similarity. The four datasets also have different characteristics in terms of the number of classes and the number of images per class. We care about the number of classes and the number of images per class because they determine the number of positive image pairs that can be sampled from the dataset. For example, the CIFAR-10 [10] dataset contains 10 categories and each category has 6,000 images. For CIFAR-10, it is easy to sample a large number of positive image pairs. But for UKBench, which contains 2,550 different object instances and each instance has 4 images, the number of positive image pairs is inherently limited. The information of the four datasets is summarized in Table 4.1. We believe that evaluating on the four datasets can provide a comprehensive understanding of the performance of our methods.

We split each dataset into two subsets: query image set and database image set. For Standford Cars [9] and UKBench [23], we also have validation images. During the training stage, we train the model with all the database images or a subset of the database images, which depends on the dataset and settings. More detailed settings will be described in the following text. During the test stage, we apply the learned model to generate hash codes for all the query and database images. Then we perform image retrieval with the generated hash codes and evaluate the search results. Similar to most previous work [13, 14, 32, 34], we employ mean average precision (MAP) as the main evaluation metric. For the UKBench [23] dataset, we also employ Precision@4 as the metric. Precision@4 measures the fraction of top-4 retrieved database images that contain the same object instance with the query image. We select Precision@4 as the metric since in UKBench [23], each object instance has exactly 4 images.

Table 4.1: Overview of the four datasets: CIFAR-10 [10], NUS-WIDE [3], Standford Cars [9] and UKBench [23].

Dataset	#Class	#Images per Class	Label Type	Similarity Type
CIFAR-10 [10]	10	6000	Single Label	Object Category
NUS-WIDE [3]	21	$\sim$7600	Multi Label	Object Category
Stanford Cars [9]	196	$\sim$80	Single Label	Fine Grained Category
UKBench [23]	2550	4	Single Label	Object Instance

4.2 Baselines and Experimental Settings

Following [14], we consider the following baselines:

1. Traditional unsupervised hashing methods using hand-crafted features, including SH [31] and ITQ [5].
2. Traditional supervised hashing methods using hand-crafted features, including SPLH [29], KSH [20], FastH [16], LFH [33] and SDH [28].
3. The above traditional hashing methods using features extracted by CNN-F network [2] pre-trained on ImageNet.
4. Pairwise label based deep hashing methods: CNNH [32] and DPSH [14].
5. Triplet label based deep hashing methods: NINH [13], DSRH [35], DSCH [34] and DRSCH [34].

During the training stage, we initialize the first seven layers of our network with the CNN-F network [2] pre-trained on ImageNet following DPSH [14]. For our method DTSH, the margin parameter α is set to half of the length of hash codes, *e.g.*, 16 for 32-bit hash codes. We have described two methods above to learn the model. In practice we find that the second one that add a quantization term in the objective function performs much better, so in the following experiments, we employ the second method to learn the model unless otherwise stated. We will compare the two methods to learn the model with comprehensive results in Section 4.4. We now describe the detailed experimental settings and other parameter settings for each dataset:

CIFAR-10 [10]:

The CIFAR-10 dataset contains 60,000 color images of size 32×32, which can be divided into 10 categories. Each category has 6,000 images and each image is only associated with one category. Two images in CIFAR-10 are considered similar if they belong to the same category.

We compare our methods to most baselines under the following experimental setting, which is also used in previous work [13, 14, 32]. In CIFAR-10, 100 images per category, *i.e.*, in total 1,000 images, are randomly sampled as query images. The remaining images are used as database images. For unsupervised hashing methods, all the database images are used as training images. For supervised hashing methods, 500 database images per category, *i.e.*, in total 5,000 images, are randomly sampled as training images.

We also compare our method to DSRH [35], DSCH [34], DRSCH [34] and DPSH [14] under a different experimental setting. In this setting, 1,000 images per category, *i.e.*, in total 10,000 images, are randomly sampled as query images. The remaining images are used as database

images and all the database images are used as training images. When running experiments on CIFAR-10, we set η to 100 for DTSH. For DPSH-Weighted, we set η to 10 and λ to 20.

NUS-WIDE [3]:

The NUS-WIDE dataset contains nearly 27,000 color images from the web. Different from CIFAR-10 [10], NUS-WIDE is a multi-label dataset. Each image is annotated with one or multiple labels in 81 semantic concepts. Following the setting in [13, 14, 32, 34], we only consider images annotated with 21 most frequent labels. For each of the 21 labels, at least 5,000 images are annotated with the label. In addition, NUS-WIDE provides links to images for downloading and some links are now invalid. This causes some differences between the image set used by previous work and our work. In total, we use 161,463 images from the NUS-WIDE dataset. Two images in NUS-WIDE are considered similar if they share at least one label.

Similar to CIFAR-10 [10], we also consider two experimental settings. We compare our methods to most baselines under the following setting. In NUS-WIDE, 100 images per label, *i.e.*, in total 2,100 images, are randomly sampled as query images. Likewise, the remaining images are used as database images. For unsupervised hashing methods, all the database images are used as training images. For supervised hashing methods, 500 database images per label, *i.e.*, in total 10,500 images, are randomly sampled as training images. Since NUS-WIDE contains a huge number of images, when computing MAP for NUS-WIDE, only the top 5,000 returned neighbors are considered.

When comparing our method to DSRH [35], DSCH [34], DRSCH [34] and DPSH [14], we use a different experimental setting. In NUS-WIDE, 100 images per label, *i.e.*, in total 2,100 images, are randomly sampled as query images. The remaining images are used as database images and still, all the database images are used as training images. Under this setting, when computing MAP for NUS-WIDE, we only consider the top 50,000 returned neighbors. In the experiments on NUS-WIDE, we set η to 100 for DTSH. For DPSH-Weighted, we set η to 100 and λ to 100.

Stanford Cars [9]:

The Stanford Cars dataset consists of 16,185 images of 196 classes of cars. Different from CIFAR-10 [10] and NUS-WIDE [3], Stanford Cars is usually used for evaluating fine-grained object categorization instead of object category classification. We select 15 images from each class, *i.e.*, in total 2,940 images, as query images. We also select another 15 images from each class as validation images. The remaining 10,305 images are used as database images. During the training stage, All the database images are used for training. In the following experiments on Stanford Cars, for DTSH, we set η to 100. For DPSH-Weighted, we set η to 1 and λ to 500.

UKBench [23]:

The UKBench dataset contains 2,550 object instances. Each object instance has exactly 4 images, *i.e.*, the dataset has 10,200 images in total. UKBench is a widely-used dataset for object instance retrieval. We consider two different experimental settings for UKBench. The main difference between the two settings is whether the model has seen images of query object instances during the training stage, *i.e.*, whether the training images contain the object instances that appear in the query images.

More specifically, we select 1 image from 500 randomly selected 500 object instances as query images and select 1 image from 500 different object instances as validation images. This gives us 500 query images and 500 validation images respectively. We use all the 10,200 images

in the dataset as database images. The query and validation images are contained in the database images but they will not be a part of the training images. In the first setting, we use all the database images except the query and validation images as training images. This gives us 9,200 training images, which contain the object instances what appear in the query images. We call this setting UKBench-Seen. In the second setting, we select database images that do not contain object instances appear in the query and validation images. Since query and validation images contain 1,000 different object instances, we have 6,200 (=(2550-1000)*4) database images left for training. We call this setting UKBench-Unseen. In the setting of UKBench-Seen, we set η to 1000 for DTSH. For DPSH-Weighted η is set to 100 and λ is set to 10000. In the setting of UKBench-Unseen, we set η to 100 for DTSH. For DPSH-Weighted, η is set to 100. We set λ to 2000 for 12-bit and 24-bit hash codes, and 10000 for 32-bit and 48-bit hash codes. We set different values for different lengths of hash codes because we find that large λ may lead to divergence of the loss during training for short hash codes.

4.3 Performance Evaluation

4.3.1 Comparison with Traditional Hashing Methods using Hand-crafted Features

We first compare our methods DPSH-Weighted and DTSH with traditional hashing methods using hand-crafted features on CIFAR-10 and NUS-WIDE. In terms of hand-crafted features, we use a 512-dimensional GIST descriptor [25] to represent CIFAR-10 images. For NUS-WIDE images, we represent them by a 1134-dimensional feature vector, which is the concatenation of a 64-D color histogram, a 144-D color correlogram, a 73-D edge direction histogram,a 128-D wavelet texture, a 225-D block-wise color moments and a 500-D BoW representation based on SIFT descriptors.

As shown in Table 4.2, we can see that on both CIFAR-10 and NUS-WIDE, our methods DPSH-Weighted and DTSH significantly outperform traditional hashing methods using hand-crafted features. In Table 4.2, the results of NINH, CNNH, KSH and ITQ are from [13, 32] and the results of other methods except our methods are from [14]. This is reasonable as we use the same experimental setting and evaluation protocol.

4.3.2 Comparison with Traditional Hashing Methods using Deep Features

When we train our model, we initialize the first 7 layers of our network with CNN-F network [2] pre-trained on ImageNet. Thus one may argue that the boost of the performance comes from the pre-trained network instead of our methods. To further validate our methods, we compare our methods with traditional hashing methods using deep features extracted by CNN-F network on CIFAR-10 and NUS-WIDE. As shown in Table 4.3, we can see that our methods can significantly outperform traditional methods on CIFAR-10 and obtain comparable performance with the best performing traditional methods on NUS-WIDE. The results in Table 4.3 are copied from [14], which is reasonable as we used the same experimental setting and evaluation protocol.

Table 4.2: Mean Average Precision (MAP) under the first experimental setting on CIFAR-10 and NUS-WIDE. The MAP for NUS-WIDE is computed based on the top 5,000 returned neighbors. The best performance is shown in boldface. DPSH* denotes the performance we obtain by running the code provided by the authors of DPSH in our experiments. DPSHW and DTSH refer to our methods DPSH-Weighted and DTSH respectively.

Method	CIFAR-10				Method	NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits		12 bits	24 bits	32 bits	48 bits
DTSH	0.710	**0.750**	**0.765**	**0.774**	DTSH	**0.773**	**0.808**	0.812	0.824
DPSHW	0.708	0.747	0.764	0.773	DPSHW	0.769	0.800	**0.813**	**0.833**
DPSH	**0.713**	0.727	0.744	0.757	DPSH*	0.752	0.790	0.794	0.812
NINH	0.552	0.566	0.558	0.581	NINH	0.674	0.697	0.713	0.715
CNNH	0.439	0.511	0.509	0.522	CNNH	0.611	0.618	0.625	0.608
FastH	0.305	0.349	0.369	0.384	FastH	0.621	0.650	0.665	0.687
SDH	0.285	0.329	0.341	0.356	SDH	0.568	0.600	0.608	0.637
KSH	0.303	0.337	0.346	0.356	KSH	0.556	0.572	0.581	0.588
LFH	0.176	0.231	0.211	0.253	LFH	0.571	0.568	0.568	0.585
SPLH	0.171	0.173	0.178	0.184	SPLH	0.568	0.589	0.597	0.601
ITQ	0.162	0.169	0.172	0.175	ITQ	0.452	0.468	0.472	0.477
SH	0.127	0.128	0.126	0.129	SH	0.454	0.406	0.405	0.400

Table 4.3: Mean Average Precision (MAP) under the first experimental setting on CIFAR-10 and NUS-WIDE. The MAP for NUS-WIDE is computed based on the top 5,000 returned neighbors. The best performance is shown in boldface. DPSHW and DTSH refer to our methods DPSH-Weighted and DTSH respectively.

Method	CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
DTSH	**0.710**	**0.750**	**0.765**	**0.774**	0.773	**0.808**	0.812	0.824
DPSHW	0.708	0.747	0.764	0.773	0.769	0.800	0.813	**0.833**
FastH + CNN	0.553	0.607	0.619	0.636	0.779	0.807	**0.816**	0.825
SDH + CNN	0.478	0.557	0.584	0.592	**0.780**	0.804	0.815	0.824
KSH + CNN	0.488	0.539	0.548	0.563	0.768	0.786	0.79	0.799
LFH + CNN	0.208	0.242	0.266	0.339	0.695	0.734	0.739	0.759
SPLH + CNN	0.299	0.33	0.335	0.33	0.753	0.775	0.783	0.786
ITQ + CNN	0.237	0.246	0.255	0.261	0.719	0.739	0.747	0.756
SH + CNN	0.183	0.164	0.161	0.161	0.621	0.616	0.615	0.612

4.3.3 Comparison with Deep Hashing Methods

Now we compare our methods with previous deep hashing methods. We first evaluate our methods and all the previous deep hashing methods on CIFAR-10 and NUS-WIDE. In particular, we compare our methods with CNNH, NINH and DPSH under the first experimental setting of CIFAR-10 and NUS-WIDE in Table 4.2 and DSRH, DSCH, DRSCH and DPSH under the

Table 4.4: Mean Average Precision (MAP) under the second experimental setting on CIFAR-10 and NUS-WIDE. The MAP for NUS-WIDE is computed based on the top 50,000 returned neighbors. The best performance is shown in boldface. DPSH* denotes the performance we obtain by running the code provided by the authors of DPSH in our experiments. DPSHW and DTSH refer to our methods DPSH-Weighted and DTSH respectively.

Method	CIFAR-10				NUS-WIDE			
	16 bits	24 bits	32 bits	48 bits	16 bits	24 bits	32 bits	48 bits
DTSH	**0.915**	**0.923**	**0.925**	**0.926**	**0.756**	**0.776**	**0.785**	**0.799**
DPSHW	0.908	0.911	0.913	0.915	0.749	0.767	0.773	0.786
DPSH	0.763	0.781	0.795	0.807	0.715	0.722	0.736	0.741
DRSCH	0.615	0.622	0.629	0.631	0.618	0.622	0.623	0.628
DSCH	0.609	0.613	0.617	0.62	0.592	0.597	0.611	0.609
DSRH	0.608	0.611	0.617	0.618	0.609	0.618	0.621	0.631
DPSH*	0.903	0.885	0.915	0.911	N/A			

Table 4.5: Mean Average Precision (MAP) on Stanford Cars. The best performance is shown in boldface. DPSHW and DTSH refer to our methods DPSH-Weighted and DTSH respectively.

Method	Stanford Cars			
	96 bits	128 bits	192 bits	256 bits
DTSH	**0.281**	**0.316**	**0.352**	**0.368**
DPSHW	0.262	0.280	0.300	0.304
DPSH	0.013	0.014	0.014	0.016

Table 4.6: Mean Average Precision (MAP) on UKBench. The best performance is shown in boldface. DPSHW and DTSH refer to our methods DPSH-Weighted and DTSH respectively.

Method	UKBench-Seen				UKBench-Unseen			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
DTSH	0.355	**0.737**	**0.811**	**0.895**	**0.256**	**0.519**	**0.594**	**0.709**
DPSHW	**0.359**	0.657	0.715	0.781	0.226	0.428	0.531	0.605
DPSH	0.211	0.417	0.487	0.593	0.187	0.372	0.433	0.521

Table 4.7: Precision@4 on UKBench. The best performance is shown in boldface. DPSHW and DTSH refer to our methods DPSH-Weighted and DTSH respectively.

Method	UKBench-Seen				UKBench-Unseen			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
DTSH	**0.338**	**0.664**	**0.745**	**0.849**	**0.282**	**0.456**	**0.520**	**0.642**
DPSHW	0.334	0.587	0.641	0.717	0.248	0.385	0.468	0.549
DPSH	0.267	0.381	0.447	0.556	0.253	0.347	0.399	0.479

Table 4.8: Mean Average Precision (MAP) on the four datasets. The best performance is shown in boldface. DTSH-tanh refers to learn the model by approximating the sign function with a tanh-line function. DTSH refers to learn the model by adding a quantization error term in the objective function.

Method	CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
DTSH	**0.710**	**0.750**	**0.765**	**0.774**	**0.773**	**0.808**	**0.812**	**0.824**
DTSH-tanh	0.557	0.591	0.607	0.600	0.638	0.646	0.655	0.664
Method	Stanford Cars				UKBench-Seen			
	96 bits	128 bits	192 bits	256 bits	12 bits	24 bits	32 bits	48 bits
DTSH	**0.281**	**0.316**	**0.352**	**0.368**	**0.355**	**0.737**	**0.811**	**0.895**
DTSH-tanh	0.102	0.103	0.100	0.092	0.313	0.669	0.758	0.831

second experimental setting of CIFAR-10 and NUS-WIDE in Table 4.4. The results of DSRH, DSCH and DRSCH are directly from [34]. As shown in Table 4.2 and Table 4.4, we can see that our methods DPSH-Weighted and DTSH significantly outperforms all the previous deep hashing methods.

We now take a closer look at the current state-of-the-art method DPSH. As shown in Table 4.2, our methods DPSH-Weighted and DTSH outperform DPSH by about 2% on both CIFAR-10 and NUS-WIDE datasets under the first experimental setting. Note that on NUS-WIDE, we are comparing with DPSH* instead of DPSH. DPSH represents the performance reported in [14] and DPSH* represents the performance we obtain by running the code of DPSH provided by the authors of [14] on NUS-WIDE. We re-run their code on NUS-WIDE because the NUS-WIDE dataset does not provide the original images to download instead of the links to image, which results in some differences between the images used by them [14] and us.

As shown in Table 4.4, our methods DTSH and DPSH-Weighted outperform DPSH by more than 10% on CIFAR-10 and about 5% on NUS-WIDE under the second experimental setting. We also re-run the code of DPSH on CIFAR-10 under the same setting and we can obtain much higher the performance then what was reported in [14].[1] The performance we obtain is denoted by DPSH* in Table 4.4 and we can see DTSH still outperforms DPSH* by about 1% on CIFAR-10.

We also compare our methods with DPSH on Stanford Cars and UKBench. Stanford Cars is usually used for evaluating fine-grained object classification, which is a harder task than object classification. On Stanford Cars, we need longer bits to achieve satisfactory performance. As shown in Table 4.5, DPSH-Weighted and DTSH significantly outperform DPSH on Stanford Cars. We observe that the performance of DPSH is very bad on Stanford Cars. Stanford Cars have 196 classes of cars, which are much more than the number of classes contained in CIFAR-10 and UKBench. This results in that the sampled image pairs are highly unbalanced, in which case DPSH-Weighted is expected to perform much better than DPSH. Moreover, we observe

[1]We communicated with the authors of DPSH. The main difference between our experiments and their experiments is the step size and decay factor for learning rate change. They say that with our parameters, they can also get better results than what is reported in their paper [14].

that DTSH performs even better than DPSH-Weighted, which proves our idea that triplet labels inherently contain richer information than pairwise labels.

UKBench has two experimental settings: UKBench-Seen and UKBench-Unseen. As shown in Table 4.6 and Tabel 4.7, DPSH-Weighted and DTSH significantly outperform DPSH in terms of both MAP and Precision@4. As expected, the performance under UKBench-Seen is higher than the performance under UKBench-Unseen. Similar to Stanford Cars, we observe that the performance gap between DPSH and our method DPSH-Weighted is huge. Also, DTSH achieves the highest performance.

In summary, our methods DPSH-Weighted and DTSH have similar performance on CIFAR-10 and NUS-WIDE, and consistently outperform all the baseline methods. On Stanford Cars and UKBench, DPSH-Weighted and DTSH achieve better performance than the current state-of-the-art method DPSH. It is worthwhile to mention that DTSH achieves the best performance among all the deep hashing methods on all the four datasets.

4.4 Ablation Studies

4.4.1 Impact of the method to learn the model

In Section 3.4, we describe two methods to learn the model. In practice, we find that the second method that adds a quantization error term in the objective function consistently outperform the first method that approximates the sign function with a tanh-like function. We given comprehensive experimental results in Tabel 4.8 to verify this. In all the experiments, when using the first model learning method, we set the initial value of β to 2 and increase it by 2 every training epoch. As shown in Table 4.8, we find that learning the model by approximating the sign function with a tanh-like function results in much worse performance than the second model learning method. We think the reason is tanh-like functions are saturating nonlinearities, which usually make the network hard to train.

4.4.2 Impact of the hyper-parameter η

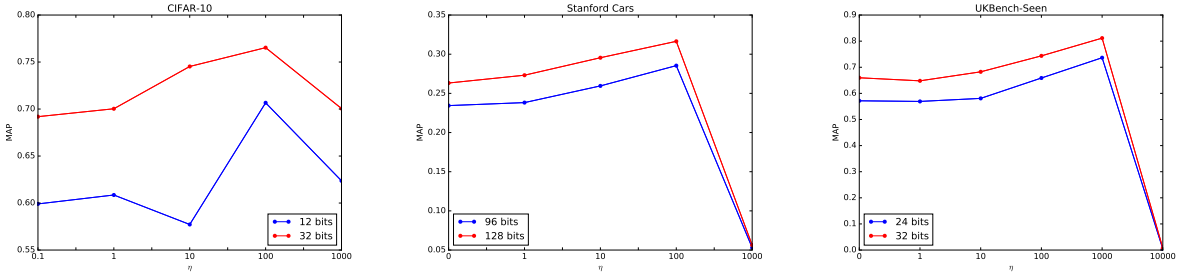
When use use the second model learning method, there is a hyper-parameter η to balance the likelihood term and the quantization error. We choose DTSH as an example and study the impact of η on its performance. Figure 4.1 shows results. Generally speaking, the performance first increases and then drops as η increases. This is reasonable since η is designed to balance the negative log triplet likelihood and the quantization error. Setting eta too small will result in a big quantization error. Setting eta to a large value will result in a small quantization error but will make the model pay very few attention to satisfying the given training labels. For example, when we set η to 1000 on Stanford Cars and set η to 10,000 on UKBench, the performance is almost zero.

4.4.3 Impact of the hyper-parameter α

We mention that a positive margin α can help speed up training in the above text. Figure 4.2 shows the effect of the margin α. We can see that within the same number of training epochs, we can obtain better performance with a larger margin. This verifies our previous analysis in Section 3.3.3.

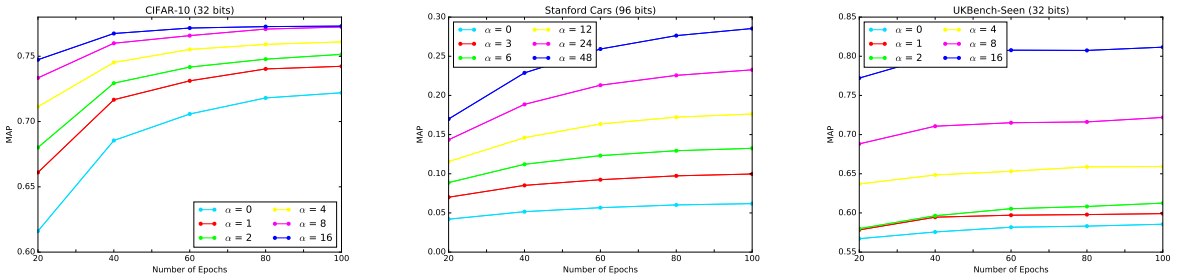
4.4.4 Impact of the hyper-parameter λ

In DPSH-Weighted, we put a weight λ on positive image pairs. We now study the impact of λ on the performance of DPSH-Weighted. Figure 4.3 shows the results. We observe that there is a general trend that the performance improves as λ increases. But when λ is too large, the performance becomes zero in Figure 4.3. In our experiments, we fix all other settings including the learning rate and only change the value of λ. When λ is too large, the training will diverge so we show zero performance for some value of λ in Figure 4.3. It is totally possible that setting a smaller learning rate for large λ can make network converge instead of diverging. Here we fix the learning rate for all the value of λ on purpose to show the impact of λ. We observe that, on Stanford Cars, $\lambda = 500$ does not lead to divergence but $\lambda = 1000$ leads to divergence. This reflects that, at least to some extent, DPSH-Weighted is sensitive to the learning rate when λ is large.



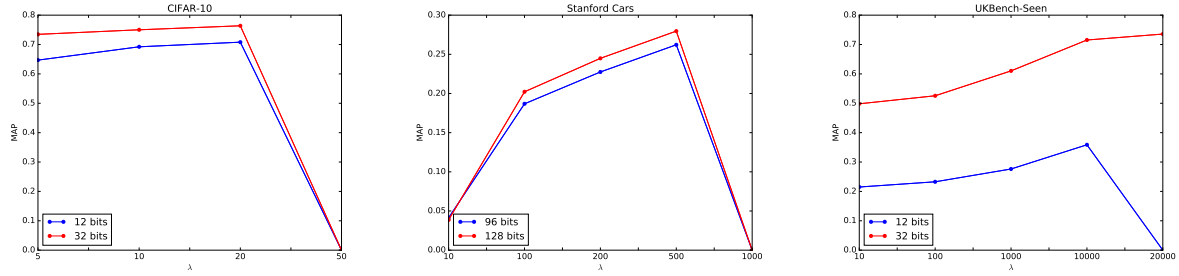
(a) CIFAR-10 (b) Stanford Cars (c) UKBench

Figure 4.1: Impact of η.



(a) CIFAR-10 (b) Stanford Cars (c) UKBench

Figure 4.2: Impact of α.



(a) CIFAR-10 (b) Stanford Cars (c) UKBench

Figure 4.3: Impact of λ.

Chapter 5

Conclusion

In this thesis we present two novel deep hashing methods DPSH-Weighted and DTSH. Both DPSH-Weighted and DTSH perform image feature learning and hash code learning simultaneously and are end-to-end trainable. Extensive experiments on standard benchmark datasets show that our methods outperform all the baselines, including the state-of-the-art deep hashing method DPSH and other deep hashing methods. In particular, DTSH performs the best among all the deep hashing methods. Our experiments also show that adding a quantization error term in the objective function is an effective way to train a network to output binary codes

Bibliography

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006. 2

[2] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014. 3.2.1, 3, 4.2, 4.3.2

[3] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, page 48. ACM, 2009. (document), 1, 4.1, 4.1, 4.2

[4] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2475–2483, 2015. 2

[5] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 817–824. IEEE, 2011. 1, 2, 1

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 3.2.1

[7] Q.-Y. Jiang and W.-J. Li. Scalable graph hashing with feature transformation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2015. 2

[8] W. Kong and W.-J. Li. Isotropic hashing. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2012. 2

[9] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013. (document), 1, 4.1, 4.1, 4.2

[10] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009. (document), 1, 3.3.1, 4.1, 4.1, 4.2

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3.2.1

[12] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*, pages 1042–1050, 2009. 2

[13] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3270–3278, 2015. 1, 2, 3.2, 3.2.2, 3.4, 4.1, 5, 4.2, 4.3.1

[14] W.-J. Li, S. Wang, and W.-C. Kang. Feature learning based deep supervised hashing with pairwise labels. *arXiv preprint arXiv:1511.03855*, 2015. (document), 1, 2, 3.2, 3.2.1, 3.3, 3.3.1, 3.3.1, 3.3.2, 3.3.3, 3.4, 3.4, 4.1, 4.2, 4, 4.2, 4.3.1, 4.3.2, 4.3.3, 1

[15] X. Li, G. Lin, C. Shen, A. Van den Hengel, and A. Dick. Learning hash functions using column generation. In *Proceedings of The 30th International Conference on Machine Learning*, pages 142–150, 2013. 2

[16] G. Lin, C. Shen, Q. Shi, A. Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1963–1970, 2014. 2, 2

[17] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen. Deep learning of binary hash codes for fast image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 27–35, 2015. 2

[18] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2064–2072, 2016. 1, 2, 3.2, 3.4, 3.4

[19] W. Liu, C. Mu, S. Kumar, and S.-F. Chang. Discrete graph hashing. In *Advances in Neural Information Processing Systems*, pages 3419–3427, 2014. 2

[20] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2074–2081. IEEE, 2012. 2, 2

[21] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1–8, 2011. 2

[22] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 1

[23] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 2161–2168. Ieee, 2006. (document), 1, 3.3.1, 3.3.3, 4.1, 4.1, 4.2

[24] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. *mij*, 1:2, 2011. 2

[25] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001. 1, 4.3.1

[26] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in neural information processing systems*, pages 1509–1517, 2009. 2

[27] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009. 2

[28] F. Shen, C. Shen, W. Liu, and H. Tao Shen. Supervised discrete hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 37–45, 2015. 2

[29] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 1127–1134, 2010. 2

[30] J. Wang, W. Liu, A. Sun, and Y.-G. Jiang. Learning hash codes with listwise supervision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3032–3039, 2013. 2

[31] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009. 2, 1

[32] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, volume 1, page 2, 2014. 2, 4.1, 4, 4.2, 4.3.1

[33] P. Zhang, W. Zhang, W.-J. Li, and M. Guo. Supervised hashing with latent factor models. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 173–182. ACM, 2014. 2, 2

[34] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *Image Processing, IEEE Transactions on*, 24(12):4766–4779, 2015. 1, 2, 3.2, 3.4, 3.4, 4.1, 5, 4.2, 4.3.3

[35] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1556–1564, 2015. 1, 2, 3.2, 3.4, 5, 4.2