World Wide Web (2019) 22:447–497
https://doi.org/10.1007/s11280-018-0540-y

Class consistent hashing for fast Web data searching

Xin Luo[1] · Ye Wu[1] · Wan-Jin Yu[1] · Xin-Shun Xu[1]

Received: 10 August 2017 / Revised: 18 January 2018 / Accepted: 27 February 2018 /
Published online: 17 March 2018

Abstract Hashing based ANN search has drawn lots of attention due to its low storage and time cost. Supervised hashing methods can leverage label information to generate compact and accurate hash codes and have achieved promising results. However, when dealing with the learning problem, most of existing supervised hashing methods are time-consuming and unscalable. To overcome these limitations, we propose a novel supervised hashing method named Class Consistent Hashing (CCH). In particular, CCH avoids using instance pairwise semantic similarity matrix which is widely used in existing methods. Instead, it uses class-pairwise semantic similarity whose size is far less than the former one, and generates hash codes for every class by optimizing the least-squares style objective function. Then, instances in the same class share the same class hash codes. Finally, we adopt a two-step hashing design strategy to learn the hash functions for out-of-sample instances. Experimental results on several widely used datasets illustrate that CCH can outperform several state-of-the-art shallow methods with the fastest training speed among supervised hashing methods.

Keywords Hashing · Similarity preserving · Large-scale data · Web data search · Approximate nearest neighbor search

This article belongs to the Topical Collection: *Special Issue on Deep vs. Shallow: Learning for Emerging Web-scale Data Computing and Applications*
Guest Editors: Jingkuan Song, Shuqiang Jiang, Elisa Ricci, and Zi Huang

✉ Xin-Shun Xu
xuxinshun@sdu.edu.cn

Xin Luo
luoxin.lxin@gmail.com

Ye Wu
kimiwadewu@gmail.com

Wan-Jin Yu
rcwanjinyu@gmail.com

[1] Shandong University, Jinan, China

1 Introduction

Last decade has witnessed the tremendous explosion of data in different fields on the Internet [6, 25, 29, 46, 52, 53]. The ever increasing demand of efficiently searching such large-scale Web data is noticeable. Recently, lots of promising hashing based approximate nearest neighbor (ANN) methods have been proposed for fast search [2, 20, 23, 24, 44]. Generally, hashing methods first transform data into binary codes, making the Hamming distances on similar examples minimized and simultaneously maximized on dissimilar examples; then, pairwise comparisons can be carried out extremely efficiently in the Hamming space, using XOR operations. Moreover, by using binary hash codes to represent original data, the storage cost can be dramatically reduced. Due to such advantages, hashing methods have become more and more popular for ANN search, especially for large-scale data search [19, 27, 28, 42, 43].

Generally, there are two kinds of hashing methods: data-independent hashing methods and data-dependent ones. Data-independent hashing methods generate hash functions by random projections. Among them, Locality-Sensitive Hashing (LSH) [3] is one of the most well-known data-independent hashing methods, and it has been generalized to accommodate other distance and similarity measures such as p-norm distance [1], Mahalanobis metric [15], and kernel similarity [14, 32]. Learning-based data-dependent hashing methods have recently drawn lots of attention because they take data into consideration and learn compact binary codes which can effectively and highly efficiently index and organize large-scale data.

Data-dependent hashing methods can be further divided into two categories: unsupervised ones [5, 26, 36, 37, 47] and supervised ones [38, 39, 50]. Unsupervised hashing methods focus on exploiting the relations of training data to learn hash codes without usage of label/semantic information. Spectral Hashing (SH) [48], Self-Taught Hashing (STH) [54], Iterative Quantization (ITQ) [4], Isotropic Hashing (IsoHash) [12], Inductive Hashing on Manifolds (IMH) [34], and Scalable Graph Hashing (SGH) [9] are some of the representative unsupervised hashing methods. When label/semantic information is available, supervised hashing methods can leverage them for hash function and hash code learning. Some representative supervised hashing methods include Minimal Loss Hashing (MLH) [30], Supervised Hashing with Kernels (KSH) [21], Two-Step Hashing (TSH) [17], Supervised Hashing with Latent Factor (LFH) [56], Fast Supervised Hashing (FastH) [18], Supervised Discrete Hashing (SDH) [35], and Column Sampling Based Discrete Supervised Hashing (COSDISH) [11]. More recently, some deep hashing models have been proposed, such as Deep Self-Taught Hashing (DSTH) [57], Binary Generative Adversarial Networks (BGAN) [41] and Deep Cross-Modal Hashing (DCMH) [10]. Such deep hashing methods have shown great improvement in performance since deep networks can learn much more complex nonlinear functions and obtain better representations [7, 40].

By leveraging label/semantic information, supervised hashing has demonstrated better accuracy than unsupervised hashing in many real applications. Therefore, supervised hashing has attracted more and more attention in recent years. However, most of them have two drawbacks: (1) they are typically time-consuming; (2) their space cost is unscalable. In the training procedure, most existing supervised methods adopt an instance-pairwise similarity matrix of labeled training data to guide the learning of hash codes. The space cost for generating this matrix is quadratic to the size of the labeled training data, and is unscalable when data amount is very large. Therefore, sampling techniques are usually adopted to avoid these two problems [11, 17, 21, 56]. However, sampling may lead to information loss as some useful information may not be sampled, and then may result in unsatisfactory performance in real applications.

To address the problems mentioned above, in this paper, we propose a novel supervised hashing method, named Class Consistent Hashing (CCH), which can learn effective hash codes in an extremely fast way. Particularly, CCH learns class consistent hash codes which means that instances in one class have the same hash codes; and the learning of hash codes is dependent on the number of classes instead of the number of training instances. To summarize, the main contributions of this paper are as follows:

- The training of CCH only depends on the number of classes rather than that of instances. Usually, the number of classes is far less than the number of instances. Thus, CCH ensures a fast training speed even on large-scale data.
- CCH avoids using sampling strategy which means that it can take advantages of full class label information. The full use of label information can better guide the learning procedure, and lead to more accurate hash codes which well preserve the semantic similarity.
- Extensive experiments are conducted on datasets. The results demonstrate that CCH can outperform the state-of-the-art hashing methods for retrieval task, e.g., image retrieval. Especially, the results on large-scale dataset further illustrate that CCH can be easily applied to large-scale data search.

The rest of this paper is organized as follows. The related work is discussed in Section 2. Section 3 introduces the proposed CCH model including the framework, optimization algorithm and its extensions to out-of-sample data. Section 4 presents the experimental results and some analysis on several benchmark datasets. Finally, Section 5 concludes this paper.

2 Related work

2.1 Two-step hashing

There is one kind of hashing methods which is called two-step hashing methods [11, 17, 18, 21]. Generally, a two-step hashing method has two steps: learning binary hash codes in the first step, and learning hash functions which transform original features into binary hash codes in the second step. And these two steps are usually called hash code learning step and hash function learning step, respectively. Actually, as [17] revealed, for any bit of the hash code, learning the corresponding hash function to project features into it can be modeled as a binary classification problem. Thus, the learning of hash functions is open for any effective predictive model, like linear regression, SVM, and boosted decision trees, etc.

Our proposed CCH is also a two-step hashing method.

2.2 Class-wise supervised hashing

Most supervised hashing methods use a similarity matrix which is instance-pairwise to supervise the learning procedure. Class-wise Supervised Hashing (CSH) [8] is a newly proposed supervised hashing method. Instead of using the instance-pairwise matrix, it uses a class-pairwise similarity matrix, whose size is much smaller than the instance-pairwise one in many applications. This is similar with our CCH, as CCH also uses class-pairwise similarity matrix.

However, there is a lot of differences between CSH and our CCH. CSH learns a code-prototype for each class, whose length is the same as the number of hash functions to be learned. Strictly speaking, the code-prototype is not a hash code, as it has three possible values (-1, 0 and 1, where 0 means this bit is useless for the code-prototype); but, one hash

code only has two possible values. Furthermore, CSH uses the code-prototype to supervise the learning of hash codes for all instances, and there is a great possibility that instances from one class have different hash codes. Our CCH directly learns hash codes for every class, and instances from one class have the same hash code. Another main difference is that CSH simultaneously learns hash codes and hash functions, while our CCH is a two-step hashing method.

3 Our method

3.1 Notations

Assume we are given n labeled instances, i.e., $\{(\mathbf{x}_1, \mathbf{l}_1), (\mathbf{x}_2, \mathbf{l}_2), \cdots, (\mathbf{x}_n, \mathbf{l}_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector of the i-th instance and $\mathbf{l}_i \in \{1, 2, \cdots, c\}$ is its corresponding class label. Without loss of generality, we assume the input instances to be zero centered, i.e., $\sum_{i=1}^{n} \mathbf{x}_i = 0$. A hashing method aims to learn a set of hash functions which map one input instance to a r-bit binary hash code $\mathbf{b}_i$:

$$\mathbf{b}_i = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \cdots, h_r(\mathbf{x}_i)]^\top = sgn(\mathbf{W}^\top \mathbf{x}_i), \tag{1}$$

where each hash function $h_j(x)$ is associated with a vector $\mathbf{w}_j \in \mathbb{R}^d$, and $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_r] \in \mathbb{R}^{d \times r}$ is the projection matrix. The $sgn(\cdot)$ function is an element-wise sign function. $\mathbf{b}_i \in \{-1, 1\}^r$ is the r-bit hash code of instance $\mathbf{x}_i$, and $\mathbf{B} \in \{-1, 1\}^{n \times r}$ is the binary hash code matrix for all instances with each row $\mathbf{B}_{i*} = \mathbf{b}_i$.

Kernelization is widely used in supervised hashing methods [21, 22, 35, 51], which can better capture the nonlinear structure underlying the original features. To use it, in this paper, we randomly sampled m ($m < n$) anchor points, i.e., $\mathbf{o}_1, \mathbf{o}_2, \cdots, \mathbf{o}_m$, from training data. Then, with the RBF kernel, kernel features can be represented as $\phi(\mathbf{x}) = \left[exp\left(\frac{-\|\mathbf{x}-\mathbf{o}_1\|_2^2}{2\sigma^2}\right), \cdots, exp\left(\frac{-\|\mathbf{x}-\mathbf{o}_m\|_2^2}{2\sigma^2}\right)\right]^\top$, where σ is the kernel width. In this paper, we calculate it with $\sigma = \frac{1}{mn}\sum_{i=1}^{n}\sum_{t=1}^{m} \| \mathbf{x}_i - \mathbf{o}_t \|_2$, $m = 500$. Then hash functions can be represented as:

$$\mathbf{b}_i = [h_1(\phi(\mathbf{x}_i)), h_2(\phi(\mathbf{x}_i)), \cdots, h_r(\phi(\mathbf{x}_i))]^T = sgn(\mathbf{W}_K^\top \phi(\mathbf{x}_i)), \tag{2}$$

where $\mathbf{W}_K \in \mathbb{R}^{m \times r}$ is the projection matrix.

3.2 Motivation

In general, supervised hashing methods aim to enforce the binary codes to be similar (or different) if the corresponding instances belong to the same class (or different classes). Ideally, the best situation is that the hash codes of similar instances, which are in one class, are exactly the same while dissimilar instances have different hash codes.

Thus, we design our hashing method following the idea that instances from the same class have the same hash codes while the hash codes of different classes should be different. As described above, the proposed method tries to make instances from the same class have consistent hash codes; we thus name it as Class Consistent Hashing (CCH). Following this design strategy, CCH has the following remarkable advantages:

– It can be extremely fast in the hash code learning procedure. The learning procedure has no relation to the size of data set, but only to the number of different classes. This

property can dramatically reduce the time cost and make it possible that our hashing method can do fast training on large-scale data.
- Full label information can be leveraged. Some supervised hashing methods suffer from high time complexity problem and cannot afford to learn hash codes using all data samples. They thus adopt some sampling strategies in learning procedure without leveraging all training data, which may lead to the loss of some useful label information.
- CCH is more scalable than most supervised hashing methods. Most existing supervised methods adopt an instance-pairwise similarity matrix of labeled training data whose size is $n \times n$ (n is the number of training samples). This kind of similarity matrix is unscalable when n is very large. But CCH only needs to generate a $c \times c$ class-pairwise similarity matrix. Usually, c is much smaller than n.
- The storage cost can also be reduced. Most of existing hashing methods need to store a matrix $\mathbf{B}$ whose size is $n \times r$; while CCH only needs to store $\mathbf{Y} \in \{-1, 1\}^{c \times r}$ and label vector $\mathbf{L}$ whose size is $n \times 1$, where $\mathbf{Y}$ is the matrix containing the hash codes of c different classes. When n is large, the storage cost of exiting methods is r times larger than that of CCH.

3.3 Objective function

As supervised hashing methods aim to force the learned hash codes to preserve the similarity of label information, i.e, the semantic similarity, we first define the class-wise similarity $\mathbf{S}$. Here, $\mathbf{S} \in \{-1, 1\}^{c \times c}$ is the class pairwise semantic similarity matrix, where $\mathbf{S}_{ij} = -1$ means that class i and class j are semantically dissimilar, $\mathbf{S}_{ij} = 1$ means class i and class j are semantically similar. In fact, it is clear that $\mathbf{S}_{ii} = 1$, where $i = 1, 2, \cdots, c$ and $\mathbf{S}_{ij} = -1$, where $i \neq j$. Besides, as there are c classes, there are only c different kinds of $\mathbf{l}_i$ for all training instances; thus, if we can design an appropriate hashing scheme which can well preserve the similarity matrix S, then, all label information can be well leveraged without missing useful one.

We use $\mathbf{y}_i \in \{-1, 1\}^r$ to denote the hash code for the i-th class and $\mathbf{Y} \in \{-1, 1\}^{c \times r}$ to represent the hash code matrix for c classes. From [21], the correlation between the hash code inner product and the Hamming distance is as follows:

$$\mathbf{y}_i \circ \mathbf{y}_j = r - 2\mathcal{D}_h(\mathbf{y}_i, \mathbf{y}_j), \tag{3}$$

where the operation $\circ$ is the inner product; $\mathcal{D}_h(\mathbf{y}_i, \mathbf{y}_j)$ is the Hamming distance between $\mathbf{y}_i$ and $\mathbf{y}_j$. Moreover, if $\mathbf{y}_i$ and $\mathbf{y}_j$ are more similar, $\mathbf{y}_i \circ \mathbf{y}_j$ will be closer to r; and if $\mathbf{y}_i$ and $\mathbf{y}_j$ are totally different, $\mathbf{y}_i \circ \mathbf{y}_j$ will be $-r$. Thus, we can construct a relation between the similarity matrix $\mathbf{S}$ and the inner product of hash codes by multiplying $\mathbf{S}$ with the code length r, i.e., $\parallel r\mathbf{S} - \mathbf{Y}^\top\mathbf{Y} \parallel$. Therefore, $\parallel r\mathbf{S} - \mathbf{Y}^\top\mathbf{Y} \parallel$ also measures the relationship between the hamming distance $\mathcal{D}_h(\mathbf{y}_i, \mathbf{y}_j)$ and semantic similarity between class i and class j. This scheme is consistent with the definition of supervised hashing which maps similar (dissimilar) instances into hash codes with small (large) hamming distances.

In addition, we adopt the least-squares style, and give the objective function:

$$\begin{aligned} &\min_{\mathbf{Y}} \parallel r \cdot \mathbf{S} - \mathbf{Y}\mathbf{Y}^\top \parallel_F^2, \\ &s.t. \quad \mathbf{Y} \in \{-1, 1\}^{c \times r}, \end{aligned} \tag{4}$$

where $\parallel \cdot \parallel_F$ represents the Frobenius norm. As $\mathbf{S} \in \{-1, 1\}^{c \times c}$, $\mathbf{Y} \in \{-1, 1\}^{c \times r}$, and $c \ll n$, r is usually range from 8 to 96 in supervised hashing literature, the time cost for

solving (4) is extremely low which will lead to a very fast learning procedure. And as (4) does not depend on the training data size n, CCH can be naturally, easily and efficiently used to deal with large-scale data.

By solving (4), we can learn the hash codes for all c classes. Then, the hash codes for every instance can be easily assigned according to its label information $\mathbf{l}_i$:

$$\mathbf{b}_i = \mathbf{y}_j, \quad \mathbf{l}_i = j. \tag{5}$$

Hence, when we store the binary codes for all n instances, there is no need to store the large matrix $\mathbf{B}$. We can instead store class hash codes matrix $\mathbf{Y}$ and label vector $\mathbf{L}$. As analyzed above, the storage cost can be reduced almost r times in CCH compared with other existing methods.

3.4 Optimization

We can solve the optimization problem of (4) in an incremental mode. First, we rewrite it as:

$$\begin{aligned} &\min_{\mathbf{Y}} \parallel r \cdot \mathbf{S} - \sum_{k=1}^{c} \mathbf{y}_{(k)} \mathbf{y}_{(k)}^{\top} \parallel_F^2, \\ &s.t. \quad \mathbf{Y} \in \{-1, 1\}^{c \times r}, \end{aligned} \tag{6}$$

where $\mathbf{y}_{(k)} \in \{-1, 1\}^c$ contains the binary codes of the k-th bit. Considering that block coordinate decent (BCD) is a technique that iteratively optimizes a subset of variables at a time, we can solve $\mathbf{y}_{(k)}$ sequentially. At a time, we only solve $\mathbf{y}_{(k)}$ provided with the previously solved vector $\mathbf{y}_{(1)}^*, \cdots, \mathbf{y}_{(k-1)}^*$. Let us define a residue matrix $\mathbf{R}_{k-1} = r\mathbf{S} - \sum_{t=1}^{k-1} \mathbf{y}_{(t)}^* (\mathbf{y}_{(t)}^*)^{\top}$. Then, we have the problem of solving $\mathbf{y}_{(k)}$:

$$\begin{aligned} &\min_{\mathbf{y}_{(k)}} \parallel \mathbf{R}_{k-1} - \mathbf{y}_{(k)} \mathbf{y}_{(k)}^{\top} \parallel_F^2, \\ &s.t. \quad \mathbf{y}_{(k)} \in \{-1, 1\}^c, \end{aligned} \tag{7}$$

Further, we rewrite (7):

$$\parallel \mathbf{R}_{k-1} - \mathbf{y}_{(k)} \mathbf{y}_{(k)}^{\top} \parallel_F^2 = tr(\mathbf{R}_{k-1}^2) - 2(\mathbf{y}_{(k)}^{\top} \mathbf{R}_{k-1} \mathbf{y}_{(k)}) + (\mathbf{y}_{(k)}^{\top} \mathbf{y}_{(k)})^2, \tag{8}$$

where $(\mathbf{y}_{(k)}^{\top} \mathbf{y}_{(k)})^2 = c^2$, and $tr(\mathbf{R}_{k-1}^2) = const$. By discarding the constant term, we obtain the equivalent form of (7) as follows.

$$\begin{aligned} &\min_{\mathbf{y}_{(k)}} -\mathbf{y}_{(k)}^{\top} \mathbf{R}_{k-1} \mathbf{y}_{(k)}, \\ &s.t. \quad \mathbf{y}_{(k)} \in \{-1, 1\}^c. \end{aligned} \tag{9}$$

Then, the optimization is equivalently reformulated as a binary quadratic problem (BQP) in (9). However, minimizing (9) is not easy because it is neither convex nor smooth. Motivated by previous works [17, 21, 48], we relax the binary constraint. Here, we adopt spectral relaxation. Then, we rewrite (9) as follows.

$$\begin{aligned} &\max_{\mathbf{y}_{(k)}} \mathbf{y}_{(k)}^{\top} \mathbf{R}_{k-1} \mathbf{y}_{(k)}, \\ &s.t. \quad \mathbf{y}_{(k)} \in [-1, 1]^c, \end{aligned} \tag{10}$$

So far, we can find that (10) is a standard generalized eigenvalue problem which can be efficiently solved.

3.5 Out-of-sample extension

As stated above, CCH is a two-step method: learning hash codes in the first step, and then training r hash functions based on the feature matrix $\mathbf{X}$ and the learned code matrix $\mathbf{B}$ in the second step. As discussed in Section 2.1, if the hash codes have already been obtained, the learning problem of hash functions can be viewed as a binary classification problem for any bit of the code. Thus, an arbitrary classifier, such as Support Vector Machines (SVM), boosting, and neural networks, may be adopted to train the hash functions. For example, [11, 45, 55] use linear classifiers; some other methods use more powerful nonlinear classifiers, such as SVM with RBF kernel [17], deep convolutional network [49] and boosted decision trees [11, 18] and so on. Generally, the more powerful classifiers we use as hash functions, the better accuracy we can achieve and also the more training time will be consumed.

Usually, by using linear regression, we can quickly learn the hash functions and the performance is also good enough. Thus, considering the tradeoff between the time cost and performance, we finally choose linear regression to train hash functions W_K for CCH, which is also adopted by [11, 56]. Consequently, the squared loss with regularization term is:

$$\| \mathbf{B} - \phi(\mathbf{X})\mathbf{W}_K \|_F^2 + \lambda_e \| \mathbf{W}_K \|_F^2, \tag{11}$$

And we can get the optimal W as:

$$W = (\phi(\mathbf{X})^\top \phi(\mathbf{X}) + \lambda_e \mathbf{I})^{-1} \phi(\mathbf{X})^\top \mathbf{B}. \tag{12}$$

Note that CCH can also use other kinds of classifiers. To show this, we give the results of CCH with SVM and boosted decision trees as classifiers in Section 4.8.

3.6 Analysis

Compared with conventional two-step supervised hashing methods, the memory space cost of CCH in the first step is less than that of others. For example, the size of the similarity matrix used in CCH is $c \times c$, where c is the number of classes. The size of the similarity matrix used in Two-Step Hashing (TSH) [17] is $n \times n$ where n is the number of training instances. For Supervised Hashing with Latent Factor (LFH) [56] and Column Sampling Based Discrete Supervised Hashing (COSDISH) [11], they use sampling strategy to randomly select a subset from the large similarity matrix and the size of such subset is $n \times r$, where r is the hash code length. Obviously, since n is always much lager than c, the space cost of CCH in the first step is much smaller than that of TSH, LFH and COSDISH. Therefore, CCH can learn the binary codes more efficiently than other two-step methods. In the second step of two-step hashing methods, the only difference is the choice of classifiers. Thus, the space cost of CCH and other two-step methods, e.g., TSH, LFH and COSDISH, is the same.

Compared with other conventional supervised hashing methods, the space cost of CCH is also less than that of them. For example, Supervised Hashing with Kernels (KSH) [21] uses an $n \times n$ similarity matrix and an $n \times d$ feature matrix. Supervised Discrete Hashing (SDH) [35] needs to load an $n \times c$ semantic label matrix and an $n \times d$ feature matrix into the

memory in the learning stage. It is worth noting that Class-wise Supervised Hashing (CSH) [8] also uses class-pairwise similarity matrix in the training stage. However, CSH learns not only the hash codes of all instances, but also the code-prototype for all classes. Therefore, other conventional supervised hashing methods also need more memory space than CCH.

4 Experimental results

We conduct extensive experiments on several widely adopted datasets, i.e., MNIST [16], CIFAR-10 [13] and CIFAR-100 [13] to evaluate the effectiveness of CCH. In order to show the effectiveness on large-scale data, we also conduct experiments on the training set of ILSVRC2010 [33]. In addition, we compare CCH with several state-of-the-art hashing methods for retrieval task.

4.1 Datasets

Following the literature of hashing, we adopt four datasets, i.e., MNIST, CIFAR-10, CIFAR-100 and the training set of ILSVRC2010 which are widely used in former works [8, 11, 35, 54].

The **MNIST** dataset consists of 70,000 images of handwritten digits from '0' to '9', and it thus has 10 classes corresponding to numbers ranging from '0' to '9'. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. And each image in this dataset is represented by 784-dimension feature vectors. If two samples have the same class label, they are considered to be semantically similar; otherwise, they are considered as semantically dissimilar. For MNIST, we randomly sample 1,000 images as query set leaving the remaining 69,000 images as training set.

The **CIFAR-10** has a number of 60,000 images which are manually labeled. This dataset has 10 classes with 6000 images for each class. Each image is represented by a 512-dimension GIST [31] feature vector extracted from the original color image of size 32×32. If two images have the same class label, they are considered to be semantically similar; otherwise, they are considered as semantically dissimilar. For CIFAR-10, the whole dataset is split into a query set with 1,000 images and a training set with all remaining images.

CIFAR-100 contains 60,000 images in total with each image belonging to one class. There are 100 classes and each class consists of 600 color images of size 32×32. Each image is represented by a 512-dimensional GIST feature vector. For CIFAR-100, following the settings in [8], 58,000 images are randomly selected from the whole set to comprise a training set while the remained 2000 images are used as queries.

For easy notation, we will call the training set of ILSVRC2010 as **ILSVRC2010** in our paper. This dataset is the subset of ImageNet and contains 1.2 million images. These images belong to 1000 categories, with each category having at least 600 samples. If two images belong to the same category, they are viewed to be semantically similar. We use the released 1,000-dimension BoW features[1] which are computed from SIFT features of original images. And we construct a query set by randomly sampling 5% images, and leave the other ones as training set (1,198,336 images).

[1] http://image-net.org/download-features

4.2 Baseline methods

We compare CCH with eight recently proposed state-of-the-art hashing algorithms, including both unsupervised and supervised ones. The unsupervised ones include Locality-Sensitive Hashing (LSH) [3], and Iterative Quantization (ITQ) [4]. The supervised ones are Supervised Hashing with Kernels (KSH) [21], Two-Step Hashing (TSH) [17], Supervised Hashing with Latent Factor (LFH) [56], Supervised Discrete Hashing (SDH) [35], Column Sampling Based Discrete Supervised Hashing (COSDISH) [11], and Class-wise Supervised Hashing (CSH) [8]. LSH is the most popular data-independent hashing and its performance is guaranteed by probability theory. ITQ is an unsupervised data-dependent hashing method which is widely adopted as one baseline method in hashing literature. TSH, LFH and COSDISH are two-step hashing methods, which have promising performance, thus are chosen as comparison methods. SDH is a recently published supervised hashing which is often in the list of baseline methods in supervised hashing papers. CSH trains a model based on a class-pairwise similarity matrix, whose supervision information is similar to ours; thus we choose it as one baseline method as well. In this paper, we do not include deep hashing baselines because our CCH is a shallow method.

For most baselines we use the code kindly provided by the respective authors and we carefully tune their parameters according to the scheme suggested by the authors. As the CSH's codes are not publicly available, we conduct comparison experiments exactly following the settings of CSH on CIFAR-100; and we compare against the results of CSH reported by the authors in their paper.

It is notable that the baselines KSH and TSH have high time complexity, which require much computational time to learn hash functions on large datasets. Following [11, 17, 21], we randomly sample 2000 images as training set for them.

4.3 Evaluation metrics

To fully evaluate the proposed method and all of the baselines, we adopt three widely used performance measures for hashing, i.e., mean average precision (MAP), top-N precision and precision-recall curves. For all metrics, a larger value indicates better retrieval performance.

Given a query, the average precision (AP) is defined as:

$$AP = \frac{1}{R}\sum_{r=1}^{n} Precision(r)\delta(r) \tag{13}$$

where R is the number of ground-truth neighbors of the query in database (training set), n is the number of entities in the database, $Precision(r)$ denotes the precision of the top r retrieved entities, and $\delta(r) = 1$ if the r-th retrieved entity is a ground-truth neighbor and $\delta(r) = 0$ otherwise. Ground-truth neighbors are defined as those instances sharing at least one semantic label. Given a query set of size Q, the MAP is defined as the mean of the average precision scores for all the queries in the query set:

$$MAP = \frac{1}{Q}\sum_{i=1}^{Q} AP_i, \tag{14}$$

Top-N precision reflects the change of precision with respect to the number of top-ranked N instances presented to the users, which is expressive for data retrieval.

Precision-recall curves can be obtained by varying the Hamming radius of the retrieved points and evaluating the precision, recall and the number of retrieved points.

4.4 Results on MNIST

The MAP results on MNIST with code length varying from 8 to 96 are shown in Figure 1. From this figure, we have the following observations:

- Supervised hashing methods can always perform better than unsupervised ones, which proves that label information has an important role in hash codes learning.
- Most of these methods can achieve good performance on MNIST, which is also reported in [17]. The main reason is that MNIST is an 'easy' dataset which is not as challenging as others.
- CCH achieves the highest search accuracy in all situations, which proves the effectiveness of CCH. This confirms that CCH can obtain better hash codes by making full use of all label information.

The curves of top-N retrieved samples and precision-recall curves are plotted in Figures 2 and 3, respectively. From these figures, we can also find that CCH obtains the best results in most cases. The only exception is the top-N precision with code length 16, i.e., Figure 2b. However, CCH performs better than TSH at the beginning, i.e., N is small. This means that CCH returns highly related samples when N is small, which is very important in retrieval task.

The training time of CCH and all baseline methods is listed in Table 1. From this table, we have the following observations:

- Unsupervised hashing methods are faster than supervised ones, which is at the price of accuracy as unsupervised methods cannot get satisfactory results.

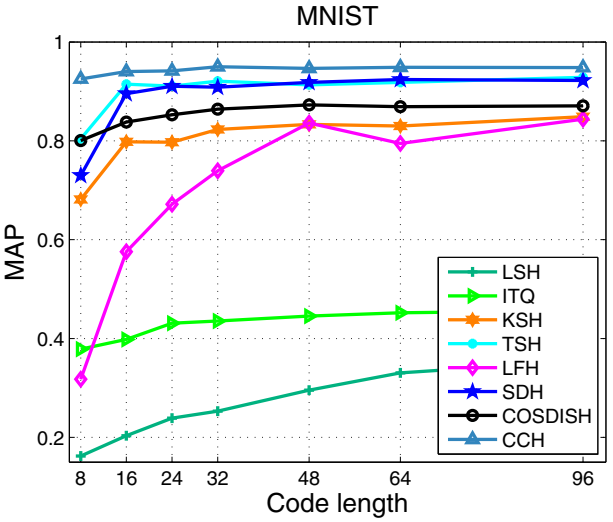

Figure 1 MAP results on MNIST with code length varying from 8 to 96

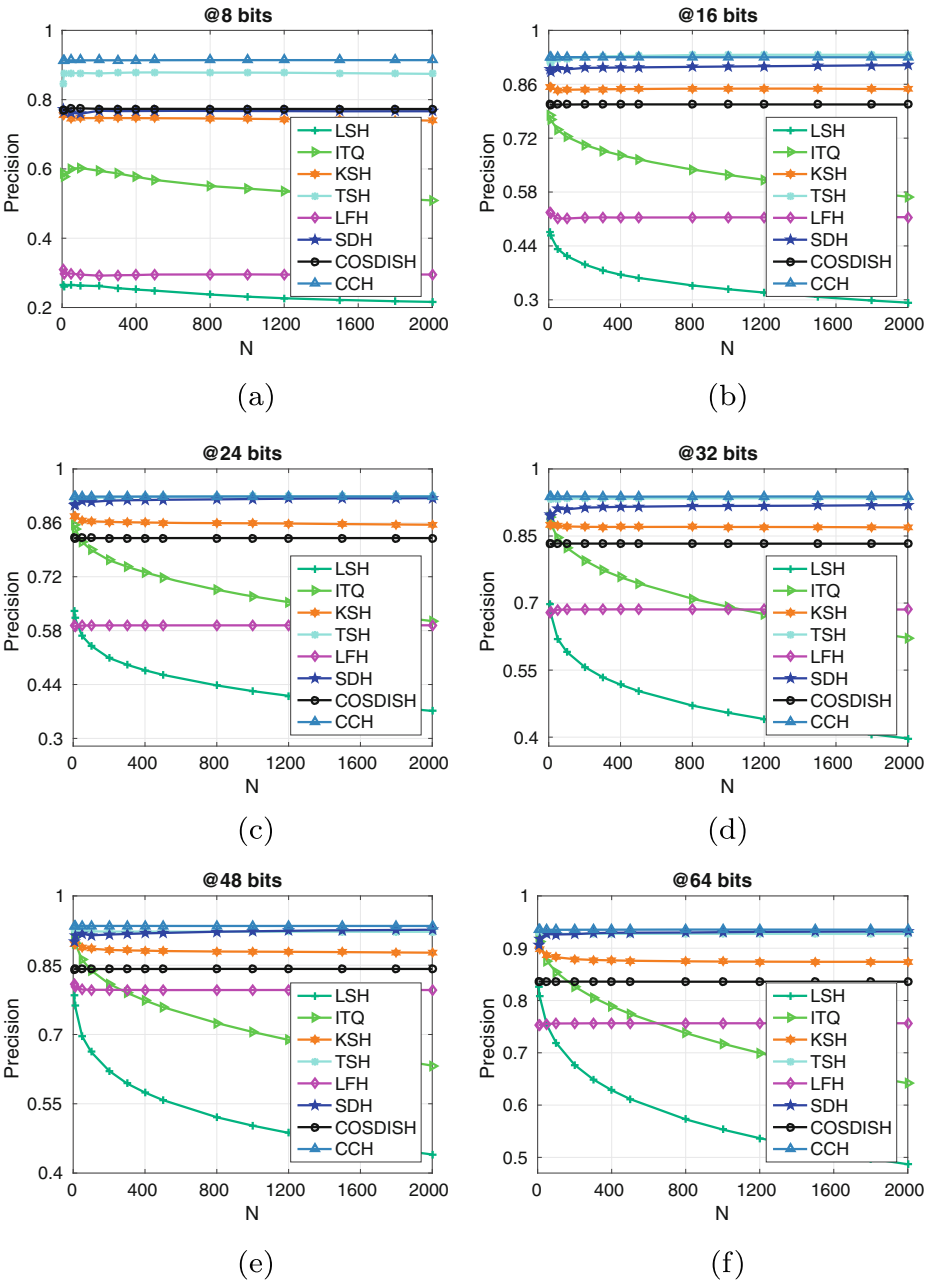

Figure 2 Top-N precision curves of all methods on MNIST with code length varying from 8 to 64

– Both KSH and TSH sample a subset of 2000 images from the overall training set to train their models. Although they only use a small set of images, their time cost is still large.
– LFH and COSDISH are all sampling based methods, which sample several columns of the pairwise similarity in one iteration. They thus have small time cost.

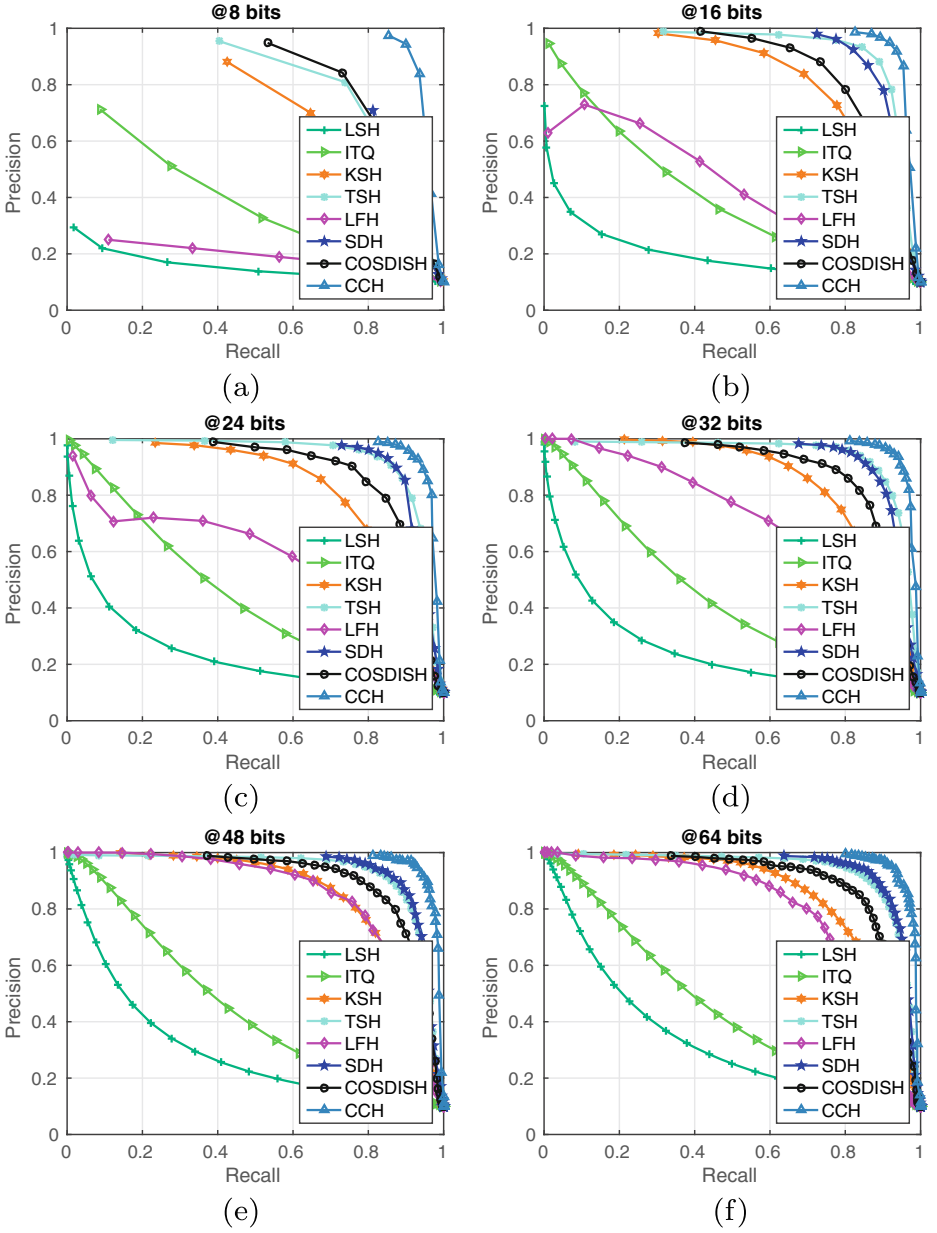

Figure 3 Precision-Recall curves of all methods on MNIST with code length varying from 8 to 64

– CCH is the fastest supervised hashing method. This is because the hash codes learning procedure of CCH is dependent on the amount of classes; and on MNIST, the amount of classes is 10, which is far less than the training data size, i.e., 69,000. Thus, CCH is much faster than other state-of-the-art supervised hashing methods.

Table 1 Training time (in second) of all methods on MNIST with code length varying from 8 to 96

Method	8 bits	16 bits	24 bits	32 bits	48 bits	64 bits	96 bits	Size of training set
LSH	0.03	0.04	0.05	0.05	0.06	0.08	0.12	69000
ITQ	0.81	1.14	1.45	2.16	3.02	4.09	5.87	69000
KSH	28.77	54.32	77.51	104.99	170.70	239.27	317.44	2000
TSH	29.05	53.35	82.10	106.88	200.34	277.51	327.85	2000
LFH	3.39	3.80	4.62	5.27	7.86	10.56	15.98	69000
SDH	11.84	13.16	15.90	17.11	26.59	48.71	116.18	69000
COSDISH	3.60	6.52	9.00	17.36	48.68	82.36	190.21	69000
CCH	1.61	1.68	1.72	1.77	1.78	1.87	1.99	69000

From the results on MNIST, we can conclude that CCH is extraordinarily efficient and effective.

4.5 Results on CIFAR-10

The MAP results on CIFAR-10 with code length varying from 8 to 96 are shown in Figure 4. From this figure, we can observe that:

- Similar to the results on MNIST, supervised hashing methods can always perform better than unsupervised ones.
- Compared with the results on MNIST, some methods cannot obtain satisfying performance on it due to the fact that CIFAR-10 is a challenging dataset as.
- CCH significantly outperforms all baseline methods in all situations, which further proves its effectiveness.

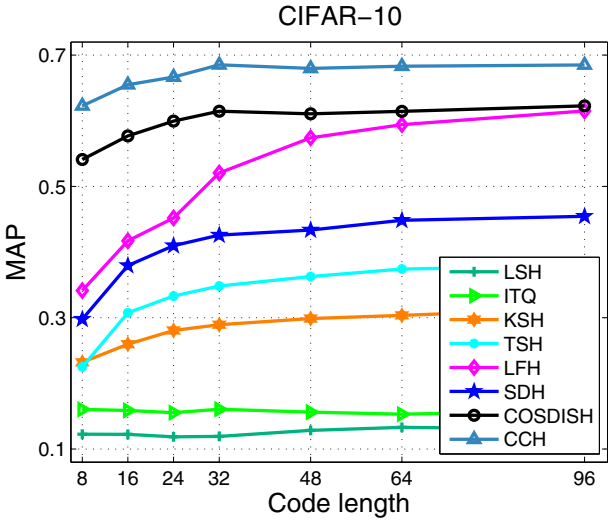

Figure 4 MAP results of all methods on CIFAR-10 with code length varying from 8 to 96

The Top-N precision curves are plotted in Figure 5. From (a)–(f) in Figure 5, we can find CCH can always obtain the highest precision. Moreover, there are large performance gains provided by CCH over the best baseline in all cases.

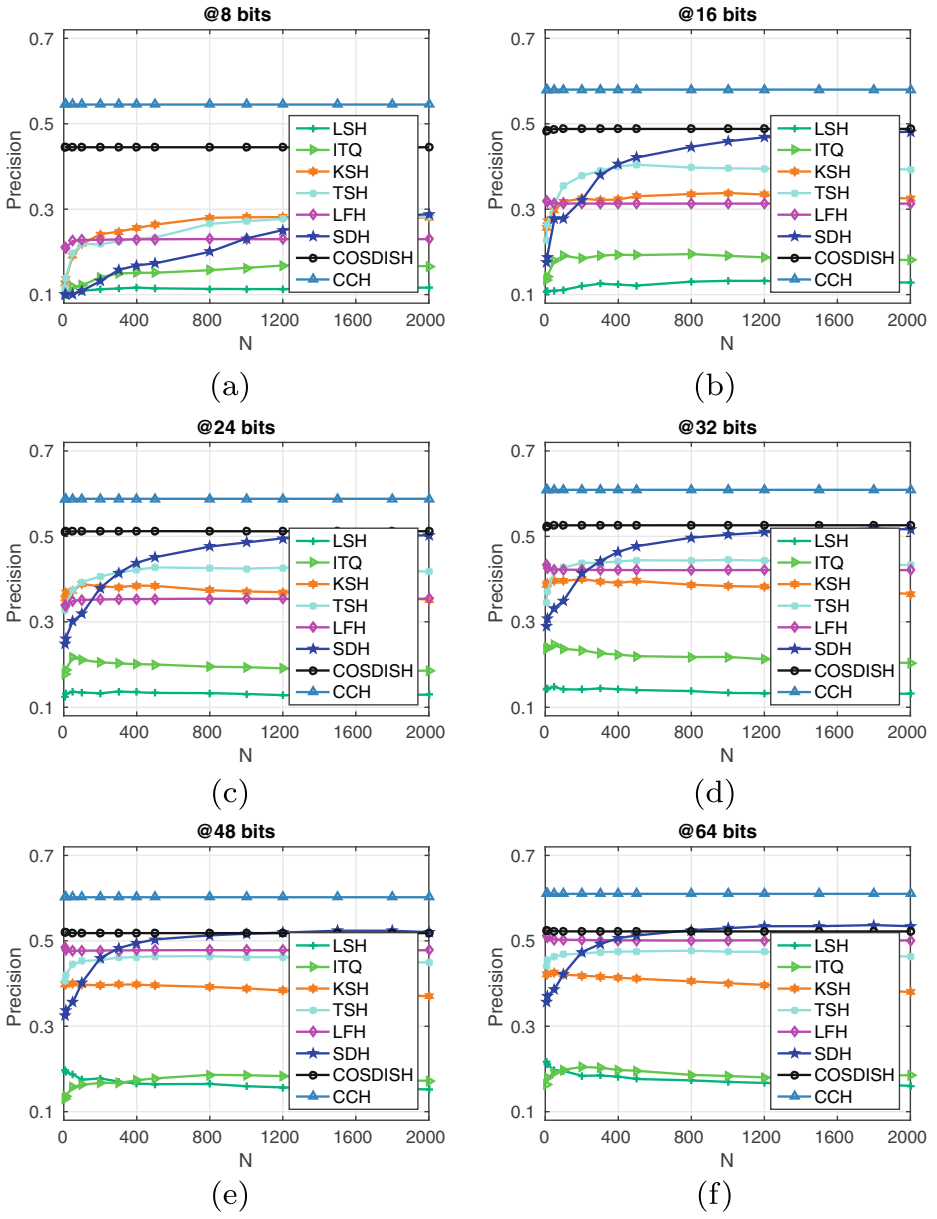

Figure 5 Top-N precision curves of the compared methods on CIFAR-10 with code length from 8 to 64

Figure 6 demonstrates the precision-recall curves on CIFAR-10 with code length varying from 8 to 64. It is obvious that CCH performs much better than these compared methods in all cases.

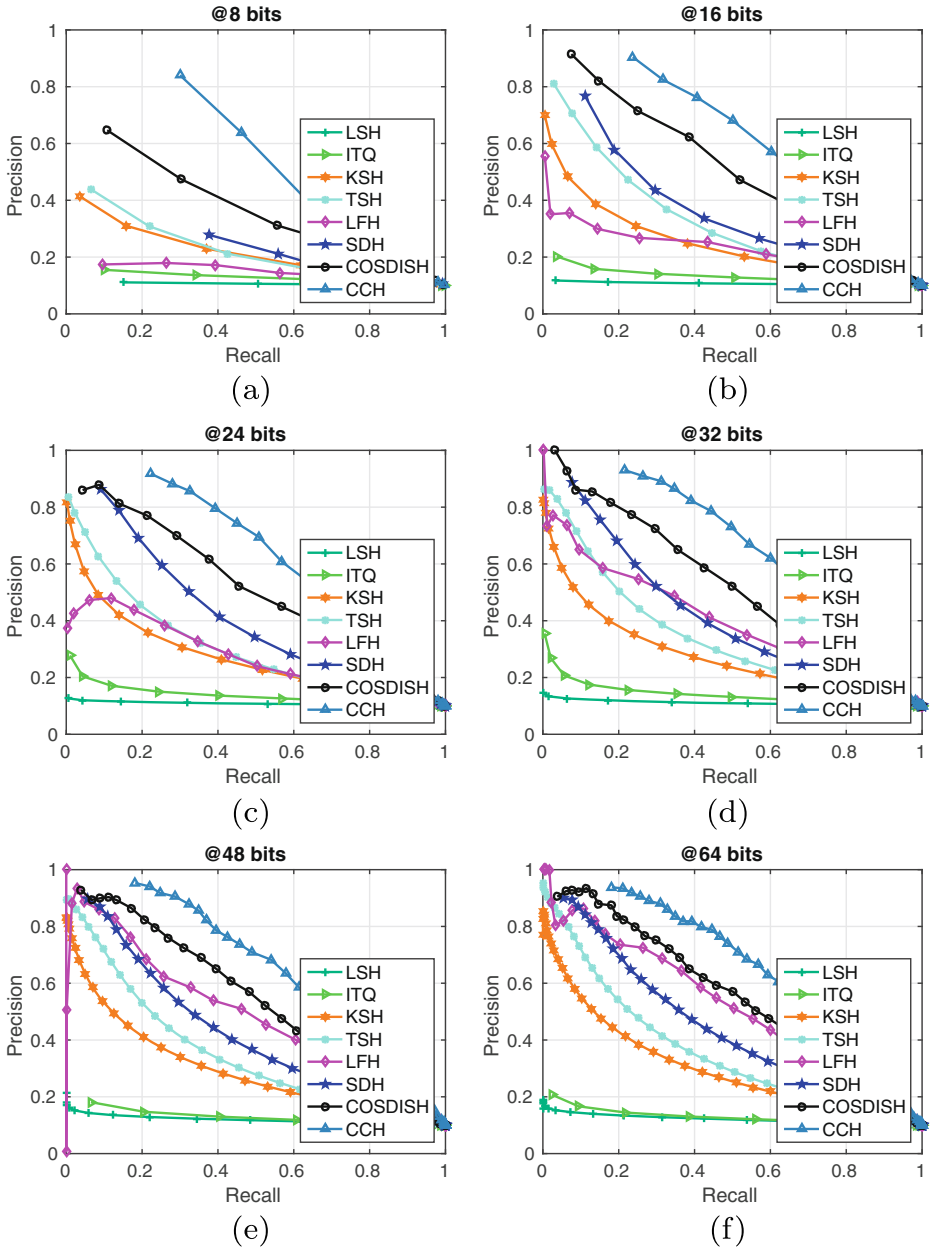

Figure 6 Precision v.s. Recall curves of the compared methods on CIFAR-10 with code length from 8 to 64

Table 2 Training time (in second) of all methods on CIFAR-10 with code length varying from 8 to 96

Method	8 bits	16 bits	24 bits	32 bits	48 bits	64 bits	96 bits	Size of training set
LSH	0.03	0.04	0.04	0.04	0.06	0.07	0.08	59000
ITQ	0.68	0.91	1.10	1.46	2.00	2.66	3.73	59000
KSH	32.85	64.75	103.25	128.01	193.72	256.10	410.80	2000
TSH	34.90	65.14	96.98	124.99	199.72	249.16	379.81	2000
LFH	2.09	2.77	3.47	4.41	6.44	9.03	13.82	59000
SDH	10.00	11.18	14.52	17.13	23.15	36.76	102.55	59000
COSDISH	2.04	4.96	9.63	19.13	49.86	90.22	183.31	59000
CCH	1.48	1.49	1.52	1.53	1.56	1.67	1.83	59000

Table 2 lists training time of CCH and all compared baseline methods on CIFAR-10. From this table, we have the following observations, which are very similar to those on MNIST:

– As the code length becomes longer, all methods need more time to train the models.
– Unsupervised hashing methods are faster than supervised ones on this dataset.
– KSH and TSH still need much time to train their models even with only 2,000 samples.
– CCH is the fastest supervised hashing method, which again proves that CCH is more scalable than other state-of-the-art supervised hashing methods.

From all of the results on CIFAR-10, we can conclude that: (1) CCH outperforms all state-of-the-art hashing methods on this dataset. (2) The time cost of CCH is very low.

4.6 Results on CIFAR-100

The MAP results on CIFAR-100 are summarized in Table 3. The symbol '-' means that we cannot obtain the results from the original paper directly; in addition, we do not have the

Table 3 MAP results of all methods on CIFAR-100 with code length varying from 8 to 96. The best MAPs for each category are shown in boldface

Method	8 bits	16 bits	24 bits	32 bits	48 bits	64 bits	96 bits
LSH	0.0114	0.0133	0.0132	0.0135	0.0150	0.0148	0.0162
ITQ	0.0126	0.0155	0.0157	0.0163	0.0180	0.0192	0.0200
KSH	0.0119	0.0121	0.0124	0.0125	0.0137	0.0144	0.0152
TSH	0.0116	0.0130	0.0143	0.0185	0.0200	0.0202	0.0256
LFH	0.0233	0.0239	0.0321	0.0366	0.0363	0.0437	0.0413
SDH	0.0270	0.0439	0.0528	0.0614	0.0743	0.0745	0.0530
COSDISH	0.0162	0.0231	0.0283	0.0310	0.0431	0.0453	0.0451
CSH	–	**0.0980**	–	0.0847	–	0.0800	0.0820
CCH	**0.0427**	0.0891	**0.1069**	**0.1269**	**0.1546**	**0.1843**	**0.2123**

Table 4 MAP results of all methods on ILSVRC2010 with code length varying from 16 to 96. The best MAPs for each category are shown in boldface

Method	16 bits	32 bits	64 bits	96 bits
LSH	0.0015	0.0018	0.0021	0.0023
ITQ	0.0027	0.0028	0.0031	0.0032
KSH	0.0022	0.0023	0.0023	0.0024
TSH	0.0012	0.0013	0.0014	0.0016
LFH	0.0038	0.0040	0.0045	0.0049
SDH	0.0026	0.0038	0.0051	0.0059
COSDISH	0.0041	0.0141	0.0212	0.0319
CCH	**0.0136**	**0.0204**	**0.0329**	**0.0396**

codes of CSH. And the MAP results of CSH with code length 16, 32, 64 and 96 bits are all obtained from [8], where the experimental setting and the evaluation metric are exactly the same with ours on CIFAR-100.

From this table, we can observe that:

- As CIFAR-100 has 100 different classes in all, it is more challenging than both MNIST and CIFAR-10. Therefore, most baselines cannot get good MAP results on CIFAR-100.
- CSH and CCH get much better accuracy than others, which can be easily explained. Both of them use class-pairwise similarity matrix to supervise the hash codes learning. More specifically, CSH and CCH do not use the sampling method; thus they can embed all useful label information into the learned binary codes. However, other methods may omit some useful information in sampling.
- In most situations, CCH can get the best results. Note that, in the case of 16 bits, CSH performs better than CCH. However, in this case, CCH can get better results than all other baselines.

To summarize, from the results on CIFAR-100, we can conclude that CCH can outperform or is comparable to these state-of-the-art hashing methods used for comparison.

Table 5 Training time (in second) of all methods on ILSVRC2010 with code length varying from 16 to 96

Method	16 bits	32 bits	64 bits	96 bits	Size of training set
LSH	0.46	0.51	0.81	1.17	1198336
ITQ	20.53	33.81	57.32	86.37	1198336
KSH	56.68	116.89	236.64	379.25	2000
TSH	365.65	920.55	1574.47	2001.50	2000
LFH	58.80	61.02	65.06	71.42	1198336
SDH	570.46	826.73	2116.91	3702.92	1198336
COSDISH	105.52	316.37	1197.52	2575.17	1198336
CCH	28.58	29.57	36.48	40.45	1198336

Table 6 MAP results of different hash functions on MNIST with code length from 8 to 96. The best MAPs for each category are shown in boldface

Method	8 bits	16 bits	24 bits	32 bits	48 bits	64 bits	96 bits
CCH	0.9320	0.9450	0.9398	0.9488	0.9456	0.9509	0.9487
CCH_SVM	0.9572	0.9653	0.9673	0.9657	0.9732	0.9675	0.9707
CCH_BT	**0.9584**	**0.9708**	**0.9748**	**0.9713**	**0.9784**	**0.9765**	**0.9790**

4.7 Results on ILSVRC2010

To further evaluate the scalability of CCH and all baselines, we conduct experiments on ILSVRC2010. It is a large-scale dataset with as many as 1000 classes and 1.2 million images.

The MAP results on ILSVRC2010 are shown in Table 4. From this table, we can find that CCH obtains the best results in all cases. With up to 1000 classes, other baselines fail to maintain good retrieval performance. As CCH utilizes the class-pairwise similarity matrix, it can thus not only handle well with datasets which consist of a small number of classes, e.g., MNIST and CIFAR-10, but also deal well with datasets whose amount of classes is large, e.g., CIFAR-100 and ILSVRC2010.

The efficiency of handling large-scale data is one of the core problems, we also give the training time of all methods on ILSVRC2010, which is listed in Table 5. Among all supervised hashing methods, CCH is the most efficient one.

From both MAP results and training time, we can conclude that CCH can be easily applied for large-scale data search.

4.8 CCH with different hash functions

To demonstrate that CCH can also leverage other classifiers, we further give the results of CCH with SVM and boosted decision trees as hash functions. In order to evaluate their performance, we conduct experiments on both CIFAR-10 and MNIST with code length from 8 to 96. And the MAP results are listed in Tables 6 and 7, respectively. In both tables, CCH means that our method uses linear regression as hash functions; CCH_SVM and CCH_BT means that CCH uses SVM and boosted decision trees as hash functions, respectively.

From these tables, we can find that CCH_SVM and CCH_BT outperform CCH in all situations, and this phenomenon is consist with the previous two-step hashing [11, 17, 56]. Generally, the more powerful classifiers we use as hash functions, the better accuracy we can achieve.

Table 7 MAP results of different hash functions on CIFAR-10 with code length varying from 8 to 96. The best MAPs for each category are shown in boldface

Method	8 bits	16 bits	24 bits	32 bits	48 bits	64 bits	96 bits
CCH	0.6207	0.6563	0.6646	0.6716	0.6742	0.6766	0.6816
CCH_SVM	**0.6413**	**0.6724**	**0.6866**	**0.6966**	**0.6972**	**0.6936**	**0.6930**
CCH_BT	0.6217	0.6721	0.6724	0.6758	0.6758	0.6838	0.6882

5 Conclusion and future work

In this paper, a novel two-step supervised hashing method for data retrieval is presented, named Class Consistent Hashing (CCH). By leveraging class pairwise similarity in hash codes learning step, CCH can use all useful semantic information and avoid large space cost. The design strategy of CCH is that instances belonging to the same class have the same hash codes. By doing this, the storage cost can be further reduced. In the second step, CCH adopts linear regression to fast train hash functions. The results on four datasets demonstrate that CCH outperforms several state-of-the-art shallow hashing methods with low time cost.

It is worth noting that, in this work, we present CCH as a shallow model because we want to concentrate on the criterion of hash codes, i.e., design of loss functions. However, we believe that our approach can easily accommodate deep networks and we leave this for future pursuit.

Acknowledgments This work was partially supported by National Natural Science Foundation of China (61573212), Key Research and Development Program of Shandong Province (2016GGX101044).

References

1. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the 20th Annual Symposium on Computational Geometry (SoCG 2004), pp. 253–262 (2004)
2. Ding, G., Guo, Y., Zhou, J.: Collective matrix factorization hashing for multimodal data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014), pp. 2083–2090 (2014)
3. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of 25th International Conference on Very Large Data Bases (VLDB 1999), pp. 518–529 (1999)
4. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE Trans. Pattern Anal. Mach. Intell. **35**(12), 2916–2929 (2013)
5. Hao, Y., Mu, T., Goulermas, J.Y., Jiang, J., Hong, R., Wang, M.: Unsupervised t-distributed video hashing and its deep hashing extension. IEEE Trans. Image Process. **26**(11), 5531–5544 (2017)
6. He, X., Zhang, H., Kan, Y.M., Chua, T.S.: Fast matrix factorization for online recommendation with implicit feedback. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2016), pp. 549–558 (2016)
7. Herranz, L., Jiang, S., Li, X.: Scene recognition with CNNs: objects, scales and dataset bias. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), pp. 571–579 (2016)
8. Huang, L.K., Pan, S.J.: Class-wise supervised hashing with label embedding and active bits. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), pp. 1585–1591 (2016)
9. Jiang, Q.Y., Li, W.J.: Scalable graph hashing with feature transformation. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 2248–2254 (2015)
10. Jiang, Q.Y., Li, W.J.: Deep cross-modal hashing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017), pp. 3270–3278 (2017)
11. Kang, W.C., Li, W.J., Zhou, Z.H.: Column sampling based discrete supervised hashing. In: Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016), pp. 1230–1236 (2016)
12. Kong, W., Li, W.J.: Isotropic hashing. In: Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS 2012), pp. 1655–1663 (2012)
13. Krizhevsky, A.: Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Toronto (2009)
14. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: Proceedings of the 12th International Conference on Computer Vision (ICCV 2009), pp. 2130–2137 (2009)
15. Kulis, B., Jain, P., Grauman, K.: Fast similarity search for learned metrics. IEEE Trans. Pattern Anal. Mach. Intell. **31**(12), 2143–2157 (2009)

16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
17. Lin, G., Shen, C., Suter, D., Hengel, A.V.D.: A general two-step approach to learning-based hashing. In: Proceedings of the 16th International Conference on Computer Vision (ICCV 2013), pp. 2552–2559 (2013)
18. Lin, G., Shen, C., Shi, Q., Hengel, A.V.D., Suter, D.: Fast supervised hashing with decision trees for high-dimensional data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014), pp. 1971–1978 (2014)
19. Lin, Z., Ding, G., Hu, M., Wang, J.: Semantics-preserving hashing for cross-view retrieval. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), pp. 3864–3872 (2015)
20. Liu, W., Wang, J., Kumar, S., Chang, S.F.: Hashing with graphs. In: Proceedings of the 28th International conference on machine learning (ICML 2011), pp. 1–8 (2011)
21. Liu, W., Wang, J., Ji, R., Jiang, Y.G., Chang, S.F.: Supervised hashing with kernels. In: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR 2012), pp. 2074–2081 (2012)
22. Liu, X., He, J., Lang, B.: Multiple feature kernel hashing for large-scale visual search. Pattern Recog. **47**(2), 748–757 (2014)
23. Liu, X., Mu, Y., Zhang, D., Lang, B., Li, X.: Large-scale unsupervised hashing with shared structure learning. IEEE Trans. Image Cybern. **45**(9), 1811–1822 (2015)
24. Liu, X., Deng, C., Lang, B., Tao, D., Li, X.: Query-adaptive reciprocal hash tables for nearest neighbor search. IEEE Trans. Image Process. **25**(2), 907–919 (2016)
25. Liu, X., Huang, L., Deng, C., Lang, B., Tao, D.: Query-adaptive hash code ranking for large-scale multi-view visual search. IEEE Trans. Image Process. **25**(10), 4514–4524 (2016)
26. Liu, X., Du, B., Deng, C., Liu, M., Lang, B.: Structure sensitive hashing with adaptive product quantization. IEEE Trans. Cybern. **46**(10), 2252–2264 (2016)
27. Liu, X., He, J., Chang, S.F.: Hash bit selection for nearest neighbor search. IEEE Trans. Image Process. **26**(11), 5367–5380 (2017)
28. Liu, X., Li, Z., Deng, C., Tao, D.: Distributed adaptive binary quantization for fast nearest neighbor search. IEEE Trans. Image Process. **26**(11), 5324–5336 (2017)
29. Nie, L., Wang, M., Zha, Z.J., Chua, T.S.: Oracle in image search: A content-based approach to performance prediction. ACM Trans. Inf. Syst. **30**(2), 13:1–13:23 (2012)
30. Norouzi, M., Fleet, D.J.: Minimal loss hashing for compact binary codes. In: Proceedings of the 28th International Conference on Machine Learning (ICML 2011), pp. 353–360 (2011)
31. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. Int. J. Comput. Vis. **42**(3), 145–175 (2001)
32. Raginsky, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels (2009)
33. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Fei-Fei, L.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)
34. Shen, F., Shen, C., Shi, Q., Hengel, A.V.D., Tang, Z.: Inductive hashing on manifolds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2013), pp. 1562–1569 (2013)
35. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), pp. 37–45 (2015)
36. Song, J., Yang, Y., Huang, Z., Shen, H.T., Luo, J.: Effective multiple feature hashing for large-scale near-duplicate Video Retrieval. IEEE Trans. Multimed. **15**(8), 1997–2008 (2013)
37. Song, J., Yang, Y., Li, X., Huang, Z., Yang, Y.: Robust hashing with local models for approximate similarity search. IEEE Trans. Cybern. **44**(7), 1225–1236 (2014)
38. Song, J., Gao, L., Yan, Y., Zhang, D., Sebe, N.: Supervised hashing with pseudo labels for scalable multimedia retrieval. In: Proceedings of the 23rd ACM international conference on Multimedia (MM 2015), pp. 827–830 (2015)
39. Song, J., Gao, L., Zou, F., Yan, Y., Sebe, N.: Deep and fast: Deep learning hashing with semi-supervised graph construction. Image Vision Comput. **55**, 101–108 (2016)
40. Song, X., Jiang, S., Gao, Y., Herranz, L.: Multi-scale multi-feature context modeling for scene recognition in the semantic manifold. IEEE Trans. Image Process. **26**(6), 2721–2735 (2017)
41. Song, J.: Binary generative adversarial networks for image retrieval. In: Proceedings of the 32nd Conference on Artificial Intelligence (AAAI 2018) (2018)
42. Song, J., Gao, L., Liu, L., Zhu, X., Sebe, N.: Quantization-based hashing: a general framework for scalable image and video retrieval. Pattern Recog. **75**, 175–187 (2018)

43. Song, J., He, T., Gao, L., Xu, X., Shen, H.T.: Deep region hashing for efficient large-scale instance search from images. In: Proceedings of the 32nd Conference on Artificial Intelligence (AAAI 2018) (2018)
44. Tang, J., Li, Z., Wang, M., Zhao, R.: Neighborhood discriminant hashing for large-scale image retrieval. IEEE Trans. Image Process. **24**(9), 2827–2840 (2015)
45. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for scalable image retrieval. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2010), pp. 3424–3431 (2010)
46. Wang, S., Jiang, S.: INSTRE: a new benchmark for instance-level object retrieval and recognition. ACM Trans. Multimedia Comput. Commun. Appl. **11**(3), 37:1–37:21 (2015)
47. Wang, J., Xu, X.S., Guo, S., Cui, L., Wang, X.: Linear unsupervised hashing for ANN search in Euclidean space. Neurocomputing **171**, 283–292 (2016)
48. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS 2008), pp. 1753–1760 (2008)
49. Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S.: Supervised hashing for image retrieval via image representation learning. In: Proceedings of the 28th Conference on Artificial Intelligence (AAAI 2014), pp. 2156–2162 (2014)
50. Xu, X.S.: Dictionary learning based hashing for cross-modal retrieval. In: Proceedings of the 24th ACM International Conference on Multimedia (MM 2016), pp. 177–181 (2016)
51. Yan, T.K., Xu, X.S., Guo, S., Huang, Z., Wang, X.: Supervised robust discrete multimodal hashing for cross-media retrieval. In: Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM 2016), pp. 1271–1280 (2016)
52. Yang, Y., Zha, Z.J., Gao, Y., Zhu, X., Chua, T.S.: Exploiting Web images for semantic video indexing via robust sample-specific loss. IEEE Trans. Multimed. **16**(6), 1677–1689 (2014)
53. Yang, Y., Ma, Z., Yang, Y., Nie, F., Shen, H.T.: Multitask spectral clustering by exploring intertask correlation. IEEE Trans. Cybern. **45**(5), 1069–1080 (2015)
54. Zhang, D., Wang, J., Cai, D., Lu, J.: Self-taught hashing for fast similarity search. In: Proceedings of the 33rd international ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2010), pp. 18–25 (2010)
55. Zhang, D., Li, W.J.: Large-scale supervised multimodal hashing with semantic correlation maximization. In: Proceedings of the 28th Conference on Artificial Intelligence (AAAI 2014), pp. 2177–2183 (2014)
56. Zhang, P., Zhang, W., Li, W.J., Guo, M.: Supervised hashing with latent factor models. In: Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2014), pp. 173–182 (2014)
57. Zhou, K., Liu, Y., Song, J., Yan, L., Zou, F., Shen, F.: Deep self-taught hashing for image retrieval. In: Proceedings of the 23rd ACM International Conference on Multimedia (MM 2015), pp. 1215–1218 (2015)