Graph Cuts for Supervised Binary Coding

Tiezheng Ge[1,⋆], Kaiming He[2], and Jian Sun[2]

[1] University of Science and Technology of China
[2] Microsoft Research

Abstract. Learning short binary codes is challenged by the inherent discrete nature of the problem. The graph cuts algorithm is a well-studied discrete label assignment solution in computer vision, but has not yet been applied to solve the binary coding problems. This is partially because it was unclear how to use it to learn the encoding (hashing) functions for out-of-sample generalization. In this paper, we formulate supervised binary coding as a single optimization problem that involves both the encoding functions and the binary label assignment. Then we apply the graph cuts algorithm to address the discrete optimization problem involved, with no continuous relaxation. This method, named as Graph Cuts Coding (GCC), shows competitive results in various datasets.

1 Introduction

Learning binary compact codes [32] has been attracting growing attention in computer vision. In the application aspect, binary encoding makes it feasible to store and search large-scale data [32]; in the theory aspect, the studies on binary encoding have been advancing the investigation on the nontrivial discrete-valued problems, *e.g.*, [14,36,19,18,35,10,25,23,21,28,13]. Binary coding solutions (*e.g.*, [10]) can also facilitate the research on non-binary coding problems (*e.g.*, [8,9,24]).

Recent studies [36,18,35,10,25,23,21,28,13] mostly formulate binary encoding as optimization problems with several concerns. The Hamming distance [14] between the codes should preserve the similarity among the data, whereas the bits of the codes should be informative for better data compression. Besides, the encoding functions (also known as hashing functions) are expected to be simple, *e.g.*, to be linear functions or simple kernel maps [19,21]. If available, supervised/semi-supervised information [18,35,25,21,28] should also be respected. The formulations of these concerns lead to nontrivial optimization problems.

A main challenge in the optimization comes from the binarization of the encoding functions f, *e.g.*, given by $\text{sign}(f)$ or its equivalence. Various optimization techniques have been adopted, including spectral relaxation [36,35], coordinate descent [18], procrustean quantization [10], concave-convex optimization [25], sigmoid approximation [21], and stochastic gradient descent [28]. Despite the different strategies, these methods mainly focus on the optimization *w.r.t.* the

⋆ This work was done when Tiezheng Ge was an intern at Microsoft Research.

D. Fleet et al. (Eds.): ECCV 2014, Part VII, LNCS 8695, pp. 250–264, 2014.

continuous parameters of the encoding function f. However, the discrete nature of the problem is often overlooked.

Nevertheless, discrete label assignment problems [5,4] are widely involved in computer vision, *e.g.*, in image segmentation [5], stereo matching [29], and many other applications [20,4,1,31,27,12]. The assignment problems are often formulated as energy minimization on a graph structure. The Graph Cuts algorithm [5,4] is a well investigated solution to this problem.

Though the graph cuts algorithm is an effective solution to discrete label assignment problems, it has not been applied to binary encoding. This is partially because the graph cuts algorithm can only give solutions to a finite set of samples, but make no prediction for the unseen ones (known as "out-of-sample" generalization). To take advantage of graph cuts, we also need to include the encoding functions in the optimization.

In this paper, we propose a binary coding method driven by graph cuts. We mainly focus on the supervised scenario as in [18,35,25,21]. We formulate supervised binary coding as an optimization problem. Unlike most existing methods that only involves the parameterized encoding functions f, we further incorporate the binary codes as auxiliary variables in the optimization. Our objective function fits the binary codes to the supervision, and also controls the loss between the binary codes and $\text{sign}(f)$. Then we can separate the binary label assignment as a sub-problem in the optimization and solve it via the graph cuts algorithm [5,4]. In experiments, this Graph Cuts Coding (GCC) method gives competitive results in various datasets.

Our method provides a novel way of addressing the inherent issue of discreteness in binary encoding. While most existing methods (*e.g.*, [36,35,25,21]) address this issue by kinds of continuous relaxation or gradient descent, our graph cuts solution focuses more closely on the binary nature. We are not the first to consider binary coding via a "*graph*" structure, but to the best of our knowledge, ours is the first method that uses the classical "*graph cuts*" algorithm [5,4] to solve this problem. In the work of Spectral Hashing (SH) [36], it has been pointed out that the SH problem is equivalent to "graph partitioning". However, the presented solution to SH in [36] is based on continuous spectral relaxation. The work of Anchor Graph Hashing (AGH) [23] has also considered a graph structure, but has pointed out that the usage of a graph is challenged by the "out-of-sample" generalization. AGH addresses this issue via continuous relaxation, rather than the discrete graph cuts.

2 Related Work

Our work is related to two seemingly unrelated areas in computer vision: binary coding and graph cuts.

Learning Binary Codes. Earlier methods for binary encoding are randomized solutions such as Locality-Sensitive Hashing (LSH) [14,19]. Recent studies on binary encoding resort to optimization. In several supervised methods like

Binary Reconstructive Embedding (BRE) [18], Semi-Supervised Hashing (SSH) [35], Minimal Loss Hashing [25], and Kernel-based Supervised Hashing [21], the energy function minimizes the discrepancy between the data similarities and the Hamming distances of sign(f). These methods optimize the energy function *w.r.t.* the continuous parameters of the encoding function f. This is often addressed by gradient descent and/or continuous relaxation.

Graph Cuts. In computer vision, the graph cuts algorithm [5,4] is a fast and effective solution to binary/multi-label assignment problems. We refer to [3] for a comprehensive introduction. The graph cuts algorithm has been applied in image segmentation [5], image restoration [5], stereo matching [29], texture synthesis [20], image stitching [1], image enhancement [31], image retargeting [27], image inpainting [12], and so on. The graph cuts algorithm is usually used to minimize an energy in a form as [5]:

$$\mathcal{E}(\mathcal{I}) = \sum_i \mathcal{E}_{\mathrm{u}}(\mathcal{I}_i) + \sum_{(i,j)} \mathcal{E}_{\mathrm{p}}(\mathcal{I}_i, \mathcal{I}_j). \tag{1}$$

Here $\mathcal{I}$ are the labeling of the image pixels, *e.g.*, 0/1 in binary segmentation. $\mathcal{E}_{\mathrm{u}}$ is a unary term (also called the data term) that depends on a single pixel, and $\mathcal{E}_{\mathrm{p}}$ is a pairwise term (also called binary/smoothness term) that depends on two pixels. The graph cuts solver is based on the max-flow/min-cut [4], in which no continuous relaxation is needed. Graph cuts can also be used to solve higher-order energies [3].

3 Graph Cuts for Supervised Binary Coding

3.1 Formulations

We denote the data sets as $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ that contains n training samples in $\mathbb{R}^d$. We first discuss the case of the linear encoding function $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} - b$, where $\mathbf{w}$ is a d-by-1 vector and b is a scalar. We will discuss the kernelized cases later.

Existing binary coding methods [14,36,19,18,35,10,25,23,21,28,30] mostly compute a single bit $y \in \{-1, 1\}$ by taking the sign of $f(\mathbf{x})$. However, in our *training procedure*, we allow y to be different from sign($f(\mathbf{x})$). We treat the binary code y as an auxiliary variable, and control its deviation from sign($f(\mathbf{x})$) by a loss function. This makes the energy function more flexible. As an overview, we minimize an energy function in the form of:

$$\min_{\mathbf{W},\mathbf{b},\mathbf{Y}} E_0(\mathbf{W}, \mathbf{b}, \mathbf{Y}) + \lambda E_1(\mathbf{Y}) \tag{2}$$
$$\text{s.t.} \quad y = 1 \text{ or } -1.$$

Here $\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_B]^{\mathrm{T}}$ and $\mathbf{b} = [b_1, ..., b_B]^{\mathrm{T}}$ are the parameters of the encoding functions $\{f_1, ..., f_B\}$ if B bits are given. $\mathbf{Y}$ is an n-by-B matrix with each row

being the B bits of a sample vector $\mathbf{x} \in \mathbf{X}$. The values of $\mathbf{Y}$ are in either -1 or 1. λ is a weight.

In our optimization, y need not be the same as $\text{sign}(f(\mathbf{x}))$, and the term $E_0(\mathbf{W}, \mathbf{b}, \mathbf{Y})$ is used to measure the loss between each y and $f(\mathbf{x})$. The term $E_1(\mathbf{Y})$ is used to fit the codes to the supervision. Note the auxiliary variable $\mathbf{Y}$ is only used in the training procedure. After training, all the data and queries are still encoded using $\text{sign}(f(\mathbf{x}))$ with the optimized $\{\mathbf{W}, \mathbf{b}\}$.

We design the energy based on the following concerns: (i) each encoding function should maximize the margin of each bit; (ii) the encoded bits should respect the supervision; and (iii) the bits should be as independent as possible. We incorporate all concerns in a single energy function.

Encoding Functions with Maximal Margins

We regard each encoding function f_k $(k = 1, ..., B)$ as a classifier trained for the n samples in $\mathbf{X}$ and their n labels $\mathbf{y}_k$. Here the n-dimensional vector $\mathbf{y}_k$ is the k-th column of $\mathbf{Y}$ (the k-th bits of all samples). In this paper, we expect each classifier to maximize the margin between the positive/negative samples as in SVM [6][1]. A binary SVM classifier can be formulated as minimizing this energy function [11]:

$$\frac{1}{2c}\|\mathbf{w}_k\|^2 + \mathcal{L}(\mathbf{y}_k, f_k(\mathbf{X})), \tag{3}$$

where $\mathcal{L}(\mathbf{y}_k, f_k(\mathbf{X})) = \sum_i \max(0, 1 - y_{k,i} f_k(\mathbf{x}_i))$ represents the hinge loss, and c is a parameter in SVM controlling the soft margin.

We put all B encoding functions together and aggregate their costs as:

$$\begin{aligned} E_0(\mathbf{W}, \mathbf{b}, \mathbf{Y}) &= \sum_k^B \frac{1}{2c}\|\mathbf{w}_k\|^2 + \mathcal{L}(\mathbf{y}_k, f_k(\mathbf{X})) \\ &= \frac{1}{2c}\|\mathbf{W}\|_{\text{F}}^2 + \mathcal{L}(\mathbf{Y}, \mathbf{f}(\mathbf{X})), \end{aligned} \tag{4}$$

where $\|\cdot\|_{\text{F}}$ is the Frobenius norm.

In the viewpoint of classification, this energy maximizes the margin between the positive/negative samples for each bit. But in the viewpoint of binary encoding, this energy measures the loss $\mathcal{L}$ between $\mathbf{Y}$ and the encoded values $\mathbf{f}(\mathbf{X})$.

Respecting the Supervision

We suppose the supervision is provided as an n-by-n affinity matrix S as in [18,35,25,21]. For example, in KSH [21] it uses $S_{ij} = 1$ if the pair $(\mathbf{x}_i, \mathbf{x}_j)$ are

[1] However, as we will introduce, the graph cuts solution does not require a specific form in this term. If only this term is an unary term (*i.e.*, it does not involve any pair-wise relation between samples), the graph cuts solution should apply.

denoted as similar, $S_{ij} = -1$ if dissimilar, and $S_{ij} = 0$ if unknown. To respect the supervision, we consider to minimize an energy as:

$$\begin{aligned} &-\sum_{k}^{B}\sum_{i}^{n}\sum_{j}^{n} S_{ij} y_{k,i} y_{k,j} \\ =& -tr(\mathbf{Y}^{\mathrm{T}}\mathbf{S}\mathbf{Y}). \end{aligned} \tag{5}$$

where $tr(\cdot)$ is the trace. Intuitively, if $S_{ij} = 1$, then this energy favors $y_{k,i}$=$y_{k,j}$ (note y is either -1 or 1); if $S_{ij} = -1$, then it favors $y_{k,i} \neq y_{k,j}$.

If our optimization problem only has the terms in Eqn.(4) and (5), it will trivially produce B identical encoding functions, because both Eqn.(4) and (5) just simply aggregate B energy functions that share the same form. Next we introduce a term to avoid this trivial case.

Bits Independence

We expect all the encoding functions to be independent to each other so as to avoid the trivial case. Ideally, we would like to have a constraint as: $\frac{1}{n}\mathbf{Y}^{\mathrm{T}}\mathbf{Y} = I$ where I is a B-by-B unit matrix. This constraint was first considered in Spectral Hashing (SH) [36]. But this leads to a challenging constrained discrete optimization problem, which was continuous-relaxed in [36]. Here we instead consider a regularization of minimizing $\|\mathbf{Y}^{\mathrm{T}}\mathbf{Y} - nI\|_{\mathrm{F}}^2$. We expand it and omit the constant terms[2], and show it is equivalent to minimizing:

$$\|\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\|_{\mathrm{F}}^2, \tag{6}$$

We put Eqn.(5) and (6) together:

$$E_1(\mathbf{Y}) = -tr(\mathbf{Y}^{\mathrm{T}}\mathbf{S}\mathbf{Y}) + \gamma\|\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\|_{\mathrm{F}}^2, \tag{7}$$

where γ is a weight. E_1 is the energy that involves the variable $\mathbf{Y}$ only.

The Energy Function

Considering Eqn.(4) and (7), we minimize this problem:

$$\begin{aligned} \min_{\mathbf{W},\mathbf{b},\mathbf{Y}} \frac{1}{2c}\|\mathbf{W}\|_{\mathrm{F}}^2 + \mathcal{L}(\mathbf{Y}, \mathbf{f}(\mathbf{X})) - tr(\mathbf{Y}^{\mathrm{T}}\mathbf{S}\mathbf{Y}) + \gamma\|\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\|_{\mathrm{F}}^2, \\ \text{s.t.} \quad y_{k,i} = 1 \text{ or } -1, \quad \forall k, i. \end{aligned} \tag{8}$$

where we have empirically set the parameter λ in Eqn.(2) as 1. The variables to be optimized are $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{Y}$. Here we explicitly treat $\mathbf{Y}$ as variables to be optimized, whereas many previous works (*e.g.*, [18,35,25,21]) only involve $\mathbf{W}$ and $\mathbf{b}$. As such, our energy function allows to directly assign binary values y to the data during the training procedure.

[2] $\|\mathbf{Y}^{\mathrm{T}}\mathbf{Y} - nI\|_{\mathrm{F}}^2 = \|\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\|_{\mathrm{F}}^2 - 2n\ tr(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}I) + n^2\|I\|_F^2 = \|\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\|_{\mathrm{F}}^2 - 2n\|\mathbf{Y}\|_F^2 + n^2\|I\|_F^2 = \|\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\|_{\mathrm{F}}^2 + const.$

3.2 Graph Cuts for One Bit

We optimize the energy (3.1) by iteratively solving two sub-problems: (i) fix $\mathbf{Y}$, update $\{\mathbf{W}, \mathbf{b}\}$; and (ii) fix $\{\mathbf{W}, \mathbf{b}\}$, update $\mathbf{Y}$. The first sub-problem is equivalent to solving B independent binary SVM classifiers as in Eqn.(3). The second sub-problem is a binary assignment problem involving $\mathbf{Y}$. We sequentially solve each bit with the remaining bits fixed. Formally, at each time we update the n-by-1 vector $\mathbf{y}_k$ with the rest $\{\mathbf{y}_{k'}; k' \neq k\}$ fixed. Then we update each bit iteratively.

We show that the problem involving $\mathbf{y}_k$ can be presented as a graph cuts problem: it only involves unary terms and pairwise terms. For the ease of presentation we denote $\mathbf{z} \triangleq \mathbf{y}_k$. With $\{\mathbf{W}, \mathbf{b}\}$ and the rest $\{\mathbf{y}_{k'} \mid k' \neq k\}$ fixed, we will show that optimizing (3.1) *w.r.t.* $\mathbf{z}$ is equivalent to minimizing:

$$\begin{aligned} E(\mathbf{z}) &= \sum_i \mathcal{E}_{\mathrm{u}}(z_i) + \sum_{(i,j)} \mathcal{E}_{\mathrm{p}}(z_i, z_j) \\ \text{s.t.} \quad & z_i = 1 \text{ or } -1, \quad i = 1, ...n, \end{aligned} \tag{9}$$

where "$\sum_{(i,j)}$" sums all possible pairs of (i, j) and $i \neq j$. Here $\mathcal{E}_{\mathrm{u}}$ represents the unary term, and $\mathcal{E}_{\mathrm{p}}$ represents the pairwise term as in Eqn.(1).

It is easy to show the unary term in Eqn.(9) is:

$$\mathcal{E}_{\mathrm{u}}(z_i) = \begin{cases} \max(0, 1 - f_k(\mathbf{x}_i)) & \text{if } z_i \text{ is 1} \\ \max(0, 1 + f_k(\mathbf{x}_i)) & \text{if } z_i \text{ is -1} \end{cases}. \tag{10}$$

To compute the pairwise term, we need to express the contribution of $\mathbf{z}$ to E. Denote $\mathbf{Y}'$ as the concatenation of $\{\mathbf{y}_{k'} \mid k' \neq k\}$, which is an $n \times (B\text{-}1)$ matrix. Note only E_1 contributes to the pairwise term[3]. With some algebraic operations[4] we can rewrite Eqn.(7) as:

$$E_1 = \mathbf{z}^{\mathrm{T}}(2\gamma \mathbf{Y}'\mathbf{Y}'^{\mathrm{T}} - \mathbf{S})\mathbf{z} + const, \tag{11}$$

where the constant is independent of $\mathbf{z}$. Denote $\mathbf{Q} = 2\gamma \mathbf{Y}'\mathbf{Y}'^{\mathrm{T}} - \mathbf{S}$, then the contribution of $\mathbf{z}$ to E_1 is $\sum_{(i,j)} 2Q_{ij} z_i z_j$. As such, the pairwise term is:

$$\mathcal{E}_{\mathrm{p}}(z_i, z_j) = \begin{cases} 2Q_{ij} & \text{if } (z_i, z_j) = (1,1) \text{ or } (-1,-1) \\ -2Q_{ij} & \text{if } (z_i, z_j) = (1,-1) \text{ or } (-1,1) \end{cases}. \tag{12}$$

Given these definitions, Eqn.(9) is a standard energy minimization problem with unary/pairwise terms and two labels (+1/-1) as in Eqn.(1).

The problem in (9) can be represented as a graph. There are n vertexes corresponding to $z_i, i = 1, ..., n$. An edge in the graph linking z_i and z_j represents

[3] E_1 should also contain a term in the form of an unary term. But this term is a constant due to the fact that $z^2 = 1$.

[4] $tr(\mathbf{Y}^{\mathrm{T}}\mathbf{S}\mathbf{Y}) = tr(\mathbf{S}\mathbf{Y}\mathbf{Y}^{\mathrm{T}}) = tr(\mathbf{S}(\mathbf{Y}'\mathbf{Y}'^{\mathrm{T}} + \mathbf{z}\mathbf{z}^{\mathrm{T}})) = const + \mathbf{z}^{\mathrm{T}}\mathbf{S}\mathbf{z}$, and $\|\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\|_{\mathrm{F}}^2 = tr((\mathbf{Y}\mathbf{Y}^{\mathrm{T}})(\mathbf{Y}\mathbf{Y}^{\mathrm{T}})^{\mathrm{T}}) = tr((\mathbf{Y}'\mathbf{Y}'^{\mathrm{T}} + \mathbf{z}\mathbf{z}^{\mathrm{T}})(\mathbf{Y}'\mathbf{Y}'^{\mathrm{T}} + \mathbf{z}\mathbf{z}^{\mathrm{T}})^{\mathrm{T}}) = const + 2\mathbf{z}^{\mathrm{T}}\mathbf{Y}'\mathbf{Y}'^{\mathrm{T}}\mathbf{z}$.

Algorithm 1. Graph Cuts Coding: Training

Input: $\mathbf{X}$ and $\mathbf{S}$.
Output: $\mathbf{W}$ and $\mathbf{b}$.

```
1: Initialize Y using PCA hashing on X.
2: repeat
3:    for k = 1 to B do
4:       Train SVM using y_k as labels and update w_k, b_k.
5:    end for
6:    for t = 1 to t_max do
7:       for k = 1 to B do
8:          Update y_k by graph cuts as in Eqn.(9).
9:       end for
10:   end for
11: until convergence or max iterations reached
```

a pairwise term $\mathcal{E}_{\mathrm{p}}(z_i, z_j)$. There are two extra vertexes representing the two labels, usually called the *source* and the *sink* [4]. These two vertexes are linked to each z_i, representing the unary terms.

Minimizing the energy in (9) is equivalent to finding a "cut" [5] that separates the graph into two disconnected parts. The cost of this cut is given by the sum of the costs of the disconnected edges. The graph cuts algorithm [5,4] is a solution to finding such a cut.

Theoretically, the graph cut algorithm requires the pairwise term to be "submodular" [15], that is, $\mathcal{E}_{\mathrm{p}}(-1,-1) + \mathcal{E}_{\mathrm{p}}(1,1) \leq \mathcal{E}_{\mathrm{p}}(-1,1) + \mathcal{E}_{\mathrm{p}}(1,-1)$. Based on Eqn.(12), the submodular condition is $Q_{ij} \leq 0$ in our case, which in fact does not always hold. However, in various applications [20,1,27,3,12] it has been empirically observed that the deviations from this condition can still produce practically good results. Furthermore, we also empirically the graph cuts algorithm works well for effectively reducing our objective function. Another choice is to apply solvers developed for nonsubmodular cases, such as the QPBO [15]. We will investigate this alternative in the future.

3.3 Algorithm Summary and Discussions

Our solution to (3.1) is described in Algorithm 1. In lines 3-5 we update $\{\mathbf{W}, \mathbf{b}\}$, and lines 6-10 we update $\mathbf{Y}$. We set the iteration number $t_{\max}$ for updating $\mathbf{Y}$ as 2 (more iterations could still decrease the energy but training is slower). The update of $\mathbf{Y}$ is analogous to the α-expansion in multi-label graph cuts [4]. The SVM in line 4 uses the `LIBLINEAR` package [7], and the SVM parameter c is tuned by cross validation. From the definition of Q we empirically set $\gamma = \frac{1}{2B}$, such that S_{ij} and $\frac{1}{\gamma}(\mathbf{Y}'\mathbf{Y}'^{\mathrm{T}})_{ij}$ have similar magnitudes. We adopt the `GCO`[5] as the graph cuts solver to Eqn.(9). Some discussions are as follows.

[5] `http://vision.csd.uwo.ca/code/`

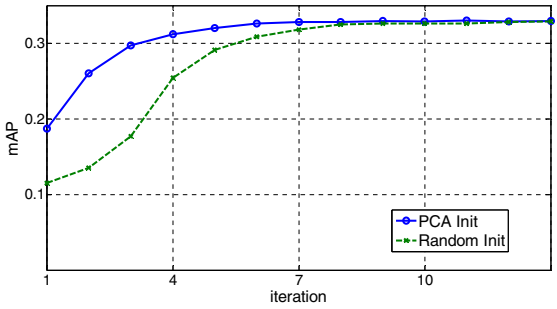

Fig. 1. The impact of initialization. The y-axis is the mean Average Precision (mAP) in the CIFAR10 dataset. See Sec. 4.1 for the experiment settings on CIFAR10. The bit number is B=32.

Table 1. The mAP of GCC using four kernels on CIFAR10

kernel	linear	inters.	χ^2	κ in Eqn.(13)
B=16	26.6	26.2	26.8	30.3
B=32	28.6	29.6	28.8	33.3

Initialization

To initialize $\mathbf{Y}$, we take the sign of the PCA projections of $\mathbf{X}$ (known as PCA hashing (PCAH) [35]). This works well in our experiments. But we also empirically observe that the final accuracy of our algorithm is *insensitive* to the initialization, and the initialization mainly impacts the convergence speed. To show this, we have tried to initialize each entries in $\mathbf{Y}$ fully in random. Fig. 1 shows the accuracy of using PCAH/random initializations in CIFAR10 (see the details in Sec. 4.1). We see that in both cases the final accuracy is very comparable. The random initialization also demonstrates the effectiveness of our optimization - though extremely incorrect labels are given at the beginning, our algorithm is able to correct them in the remaining iterations.

Kernelization

Our algorithm can be easily kernelized. This is achieved by a mapping function on $\mathbf{x} : \mathbb{R}^d \mapsto \mathbb{R}^D$ where D can be different from d. The mapped set is used in place of $\mathbf{X}$. We have tried the Explicit Feature Mapping [34] to approximate the intersection kernel and the Chi-squared kernel.[6]. We have also tried the kernel map used in [19,21]:

$$\kappa(\mathbf{x}) = [g(\mathbf{x}, \mathbf{x}'_\mathbf{1}), ..., g(\mathbf{x}, \mathbf{x}'_\mathbf{D})]^{\mathrm{T}} \tag{13}$$

[6] This can be computed via `vl_homkermap` in the VLFeat library [33].

Table 2. The training time and mAP on CIFAR10 with/without removal. The bit number is 32. The iteration number is 10.

	training time (s)	mAP
no removal	2540	33.5
with removal	605	33.3

where g is a Gaussian function, and $\{\mathbf{x'_1}, ..., \mathbf{x'_D}\}$ are random samples from the data (known as *anchors* [23,21]). This can be considered as an approximate Explicit Feature Mapping of RBF kernels [34]. In this paper, we use 1,000 anchors ($D = 1,000$). In Table. 1 we compare the performance of our algorithm using four kernels. It shows the kernel κ in (13) performs the best. In the rest of this paper we use this kernel.

Reduced Graph Cuts

Even though the graph cuts solver is very efficient, it can still be time-consuming because during the training stage it is run repeatedly. We propose a simplification to reduce the training time. In each time applying graph cuts to optimize one bit, we randomly set a portion of the pairwise terms in Eqn.(9) as zero. This can effectively reduce the running time because the number of edges in the graph are reduced. The removed terms are different for each bit and for each iteration, so although less information is exposed to each bit at each time, the entire optimizer is little degraded. We randomly remove 90% of pairwise terms (each sample is still connected to 10% of all the training samples). The accuracy and training speed with/without removal is in Table 2. We see that it trains faster and performs comparably. The remaining results are given with the reduced version.

4 Experiments

We compare our Graph Cuts Coding (GCC) with several state-of-the-art supervised binary coding (hashing) methods: Binary Reconstructive Embedding (BRE) [18], Semi-Supervised Hashing (SSH) [35], Minimal Loss Hashing (MLH) [25], Iterative Quantization with CCA projection (CCA+ITQ) [10], Kernel-based Supervised Hashing (KSH) [21], and Discriminative Binary Codes (DBC) [28]. We also evaluate several unsupervised binary coding methods: Locality-Sensitive Hashing (LSH) [14,2], Spectral Hashing (SH) [36], ITQ [10], Anchor Graph Hashing (AGH) [23], and Inductive Manifold Hashing (IMH) [30]. All these methods have publicly available code. Our method is implemented in Matlab with the graph cuts solver in mex. All experiments are run on a server using an Intel Xeon 2.67GHz CPU and 96 GB memory. We evaluate on three popular datasets: CIFAR10 [16], MNIST[7], and LabelMe [25].

[7] `http://yann.lecun.com/exdb/mnist/`

Table 3. The training time (single-core) on CIFAR10. All methods are using 32 bits. The GCC runs 10 iterations.

method	seconds	method	seconds
GCC	605	MLH [25]	3920
KSH [21]	483	BRE [18]	1037
DBC [28]	35	SSH [35]	3.0
KDBC [28]	56	CCA+ITQ [10]	5.3

4.1 Experiments on CIFAR10

CIFAR10 [16] contains 60K images in 10 object classes. As in previous studies of binary coding, we represent these images as 512-D GIST features [26][8]. We follow the experiment setting (and their evaluation implementation) in the KSH paper [21] and its public code. 1K images (100 per class) are randomly sampled as queries and the rest as the database. 2K images are randomly sampled from the database (200 per class) to build the pairwise supervision matrix $\mathbf{S}$: $S_{ij} = 1$ if the pair are in the same class and otherwise $S_{ij} = -1$ (0 for BRE/MLH). Our GCC and BRE/SSH/MLH/KSH accept pairwise supervision. DBC and CCA+ITQ needs explicit class labels for training. Table 3 shows the training time of several supervised methods.

Table 4 shows the results evaluated by two popular metrics: Hamming ranking and Hamming look-up [32]. The results of Hamming ranking is evaluated via the mean Average Precision (mAP), *i.e.*, the mean area under the precision-recall curve. We see that KSH [21] is very competitive and outperforms other previous methods. The GCC improves substantially upon KSH: it outperforms KSH by **3.0%** in 16 bits, **3.3%** in 32 bits, and **2.6%** in 64 bits (relative **8%-10%** improvement). Fig. 2 further shows the Hamming ranking results evaluated by the recall at the top N ranked data.

Table 4 also shows the results using Hamming look-up [32], *i.e.*, the accuracy when the Hamming distance is $\leq r$. Here we show $r = 2$. We see GCC is also superior in this evaluation setting. GCC outperforms KSH by 2.3% in 16 bits, 6.1% in 32 bits (and outperforms DBC by 4%).

Comparisons with SVM-Based Encoding Methods. Our method is partially based on SVMs (more precisely, Support Vector Classifiers or SVCs). In our SVM sub-problem, the labels $\mathbf{Y}$ directly come from the discrete graph cuts sub-problem. Consequently, throughout our optimization, the auxiliary variables $\mathbf{Y}$ are always treated as discrete in both sub-problems. There are previous solutions [22,28] that also partially rely on SVMs. However, the labels $\mathbf{Y}$ in those solutions are continuous-relaxed at some stage.

In Table 5 we compare with a method call SVM Hashing (SVMH), which was discussed in the thesis [22] of the first author of KSH [21]. If class labels are

[8] Actually, it is not necessary to represent them as GIST. Advanced representations such as CNN (convolutional neural networks) features [17] may significantly improve the overall accuracy of all encoding methods.

Table 4. The results on CIFAR10. On the top section are the supervised methods, and on the bottom section are the unsupervised ones. The middle column shows the Hamming ranking results evaluated by mAP. The right column shows the Hamming look-up results when the Hamming radius r=2. (Hamming look-up of $B = 64$ is ignored because this is impractical for longer codes [32]).

	Hamming ranking (mAP, %)			precision (%) @ $r = 2$	
B	16	32	64	16	32
GCC	**30.3**	**33.3**	**34.6**	**38.0**	**39.6**
KSH [21]	27.3	30.0	32.0	35.7	33.5
KDBC [28]	25.1	26.2	27.0	25.5	31.0
DBC [28]	23.8	26.3	28.6	29.3	35.6
CCA+ITQ [10]	21.4	21.7	23.1	23.6	27.6
MLH [25]	21.3	22.3	25.7	26.3	30.0
BRE [18]	18.7	19.5	20.1	24.2	20.7
SSH [35]	16.3	16.7	18.0	14.6	17.3
IMH [30]	18.4	19.4	20.1	21.9	25.3
ITQ [10]	16.9	17.3	17.7	24.2	18.1
AGH [23]	14.6	14.1	13.7	20.2	25.6
LSH [14]	13.4	14.2	14.7	17.6	8.5
SH [36]	13.2	13.0	13.1	19.2	21.8

Table 5. Comparisons on CIFAR10 with SVM Hashing [22] and its kernelized variant. All methods are using 10 bits. The kernel of KSVMH is the same as KSH.

method	SVMH [22]	KSVMH	KSH [21]	**GCC**
mAP	21.5	23.3	25.0	**28.2**

available, SVMH trains 10 one-vs-rest SVM classifiers, and uses the prediction functions as the encoding functions. SVMH is limited to 10 bits in CIFAR10. One can train the classifier using linear kernel or the kernel map κ. Table 5 shows the results of linear SVMH, Kernelized SVMH, KSH, and GCC (all using 10 bits for fair comparison). We see that GCC is still superior, even though the class labels are unknown to GCC. Actually, the one-vs-rest SVMs operate in a winner-take-all manner; but for binary coding or hashing, it is not sufficient to make two similar samples to be similar in just one bit. In our objective function, the term in Eqn.(5) is introduced to address this issue - it encourages as many as possible bits to be similar if two samples are similar. In our formulation, GCC is also able to produce >10 bits and shows increased performance.

More closely related to our method, DBC [28] is another method that adopts SVMs to train a classifier for each bit. However, DBC has a different objective function and applies a subgradient descent technique to solve for the labels that will be provided for SVMs. Table 4 shows that our method performs better

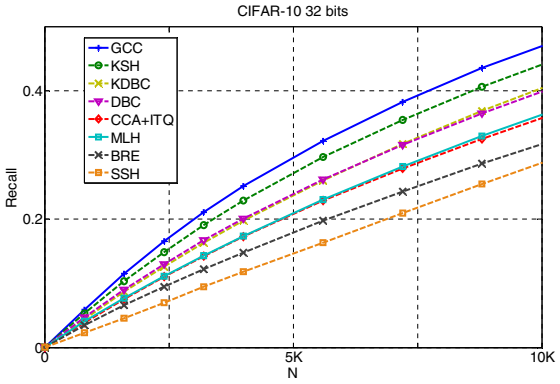

Fig. 2. The recall@N results on CIFAR10 using 32 bits. The x-axis is the number of top ranked data in Hamming ranking. The y-axis is the recall.

Table 6. The results on MNIST

	Hamming ranking (mAP, %)			precision (%) @ $r = 2$	
B	16	32	64	16	32
GCC	**86.3**	**88.1**	**88.9**	**87.1**	**87.5**
KSH [21]	78.9	82.4	83.7	84.1	85.8
MLH [25]	69.9	75.2	79.5	78.1	85.3
BRE [18]	52.2	59.9	62.4	65.4	79.2
DBC [28]	53.9	57.1	60.4	64.7	66.9
CCA+ITQ [10]	54.9	56.4	57.9	54.9	63.5
SSH [35]	43.2	48.6	48.7	64.8	74.3

than DBC. The original DBC in [28] uses linear encoding functions, so for fair comparison, we have also tested its kernelized version using the same kernel map κ as we use. We term this as kernelized DBC (KDBC) in Table 4. We see that our method also outperforms KDBC using the same kernel map. These experiments indicate the performance of GCC is not simply due to the SVMs.

4.2 Experiments on MNIST

The MNIST dataset has 70K images of handwritten digits in 10 classes. We represent each image as a 784-D vector concatenating all raw pixels. We randomly sample 1K vectors (100 per class) as queries and use the rest as the database. 2K vectors (200 per class) are sampled from the database as the training data.

We compare with the supervised methods in Table 6 (the unsupervised methods perform worse, *e.g.*, than KSH, and so are ignored). We find that KSH still outperforms other previous methods substantially, and GCC improves on KSH by considerable margins.

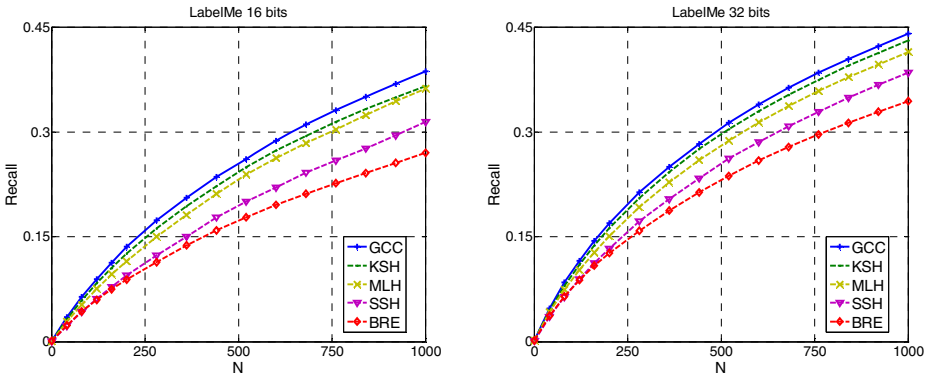

Fig. 3. The results on LabelMe using 16 and 32 bits. The x-axis is the number of top ranked data in Hamming ranking. The y-axis is the recall among these data.

4.3 Experiments on LabelMe

The LabelMe dataset [25] contains 22K images represented as 512-D GIST, where each image has 50 semantic neighbors marked as the ground truth. Only pairwise similarity labels are available in this dataset. We follow the evaluation protocol as in [25]. The data are ranked by their Hamming distances to the query, and the recall at the top N ranked data is evaluated ($R@N$). In this dataset, the reduced graph cuts step in our algorithm does not remove the pairwise terms with positive labels ($S_{ij} = 1$) because they are in a small number. All the methods are trained using 2K randomly sampled images and their pairwise labels.

Fig. 3 shows the performance of the supervised methods. Because only pairwise labels are available, the CCA+ITQ and DBC methods which need class-wise labels are not directly applicable. This also indicates an advantage of GCC that it does not require class-wise labels. We see that GCC is competitive. The measure $R@1000$ of GCC outperforms KSH by 2.1% when B=16, and 1.0% when B=32.

5 Discussion and Conclusion

We have presented a graph cuts algorithm for learning binary encoding functions. This is a beginning attempt to use discrete label assignment solvers in the binary encoding problems. In the formulations in this paper, a term has been introduced to measure the loss $\mathcal{L}$ between $\mathbf{y}$ and $\mathbf{f}(\mathbf{x})$. We note the loss function $\mathcal{L}$ need not be limited to the form (hinge loss) used in this paper. Our graph cuts solution is applicable for other forms of $\mathcal{L}$, and only the unary term needs to be modified. The development of a better $\mathcal{L}$ can be an open question, and we will study it as future work.

References

1. Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D., Cohen, M.: Interactive digital photomontage. In: SIGGRAPH (2004)
2. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: FOCS, pp. 459–468 (2006)
3. Blake, A., Kohli, P., Rother, C.: Markov random fields for vision and image processing. The MIT Press (2011)
4. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. TPAMI (2004)
5. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. In: ICCV (1999)
6. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning, 273–297 (1995)
7. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. JMLR, 1871–1874 (2008)
8. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: CVPR (2013)
9. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization. TPAMI (2014)
10. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. In: CVPR (2011)
11. Hastie, T.J., Tibshirani, R.J., Friedman, J.H.: The Elements of Statistical Learning. Springer, New York (2009)
12. He, K., Sun, J.: Statistics of patch offsets for image completion. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part II. LNCS, vol. 7573, pp. 16–29. Springer, Heidelberg (2012)
13. He, K., Wen, F., Sun, J.: K-means Hashing: an Affinity-Preserving Quantization Method for Learning Binary Compact Codes. In: CVPR (2013)
14. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC, pp. 604–613 (1998)
15. Kolmogorov, V., Rother, C.: Minimizing nonsubmodular functions with graph cuts-a review. TPAMI (2007)
16. Krizhevsky, A.: Cifar-10, `http://www.cs.toronto.edu/~kriz/cifar.html`
17. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks (2012)
18. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. In: NIPS, pp. 1042–1050 (2009)
19. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: ICCV (2009)
20. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: image and video synthesis using graph cuts. In: SIGGRAPH, pp. 277–286 (2003)
21. Liu, W., Wang, J., Ji, R., Jiang, Y.-G., Chang, S.-F.: Supervised hashing with kernels. In: CVPR (2012)
22. Liu, W.: Large-Scale Machine Learning for Classification and Search. Ph.D. thesis, Columbia University (2012)
23. Liu, W., Wang, J., Kumar, S., Chang, S.-F.: Hashing with graphs. In: ICML (2011)
24. Norouzi, M., Fleet, D.: Cartesian k-means. In: CVPR (2013)
25. Norouzi, M.E., Fleet, D.J.: Minimal loss hashing for compact binary codes. In: ICML, pp. 353–360 (2011)

26. Oliva, A., Torralba, A.: Modeling the shape of the scene: a holistic representation of the spatial envelope. IJCV (2001)
27. Pritch, Y., Kav-Venaki, E., Peleg, S.: Shift-map image editing. In: ICCV (2009)
28. Rastegari, M., Farhadi, A., Forsyth, D.: Attribute discovery via predictable discriminative binary codes. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part VI. LNCS, vol. 7577, pp. 876–889. Springer, Heidelberg (2012)
29. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. IJCV, 7–42 (2002)
30. Shen, F., Shen, C., Shi, Q., van den Hengel, A., Tang, Z.: Inductive hashing on manifolds. In: CVPR (2013)
31. Tan, R.T.: Visibility in bad weather from a single image. In: CVPR, pp. 1–8 (2008)
32. Torralba, A.B., Fergus, R., Weiss, Y.: Small codes and large image databases for recognition. In: CVPR (2008)
33. Vedaldi, A., Fulkerson, B.: Vlfeat: An open and portable library of computer vision algorithms. In: Proceedings of the International Conference on Multimedia, pp. 1469–1472. ACM (2010)
34. Vedaldi, A., Zisserman, A.: Efficient additive kernels via explicit feature maps. TPAMI, 480–492 (2012)
35. Wang, J., Kumar, S., Chang, S.-F.: Semi-supervised hashing for scalable image retrieval. In: CVPR (2010)
36. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS, pp. 1753–1760 (2008)