A Sparse Embedding and Least Variance Encoding Approach to Hashing

Xiaofeng Zhu, Lei Zhang, *Member, IEEE,* Zi Huang

***Abstract*—Hashing is becoming increasingly important in large-scale image retrieval for fast approximate similarity search and efficient data storage. Many popular hashing methods aim to preserve the kNN graph of high dimensional data points in the low dimensional manifold space, which is however difficult to achieve when the number of samples is big. In this paper, we propose an effective and efficient hashing approach by sparsely embedding a sample in the training sample space and encoding the sparse embedding vector over a learned dictionary. To this end, we partition the sample space into clusters via a linear spectral clustering method, and then represent each sample as a sparse vector of normalized probabilities that it falls into its several closest clusters. This actually embeds each sample sparsely in the sample space. The sparse embedding vector is employed as the feature of each sample for hashing. We then propose a least variance encoding model, which learns a dictionary to encode the sparse embedding feature, and consequently binarize the coding coefficients as the hash codes. The dictionary and the binarization threshold are jointly optimized in our model. Experimental results on benchmark datasets demonstrated the effectiveness of the proposed approach in comparison with state-of-the-art methods.**

***Index Terms*—hashing, manifold learning, image retrieval, dictionary learning.**

I. Introduction

Nearest neighbor search in large-scale datasets is an indispensable process in many machine learning algorithms [1], [2], [3], [4], [5], such as spectral clustering [6], kernel density estimation [7], semi-supervised learning [8], and sparse subspace clustering [9], etc. Exhaustively comparing the query sample with each training sample is prohibitive in large-scale visual data retrieval due to its linear query time to the training size. Besides, in many real applications (e.g., [10], [11]) the dimensionality of visual data is very high, leading to the problem of curse of dimensionality. The high dimensionality and the big sample size make the data storage and matching very challenging, limiting the use of nearest neighbor search in practical applications.

Instead of conducting the exact nearest neighbor search with linear scan, approximate nearest neighbors (ANN) search can be used to retrieve the query sample with sub-linear, logarithmic, or even constant query time. For example, tree-based approaches (including KD tree [12], ball-tree [13], metric tree [14], and vantage point tree [15]) can achieve not only logarithmic query time but also efficient data structure to store the large-scale data. However, the retrieval performance of tree-based methods will drastically degrade for high-dimensional data [16].

Recently, hashing methods for fast ANN search have shown good performance to retrieve high dimensional data [17], [18], [19] and have been applied to image retrieval [20], document analysis [21], near-duplicate detection [22], etc. In order for fast ANN in large-scale datasets, hashing methods first transform the high dimensional data into a low dimensional subspace and output the hash functions (hash function learning in short), followed by binarizing the low dimensional data into binary codes (binarization in short). The hashing methods aim to preserve the original sample similarities in the binary codes as much as possible. As a result, retrieving the original data is replaced by matching the binary codes in memory of a PC, which makes the retrieval process very fast.

One type of well-known hashing methods is the LSH-like hashing methods, including LSH [21], [23], [24], Kernelized LSH (KLSH) [25], and Shift-Invariant KLSH (SKLSH) [26]. Although having some nice properties such as constant query time, LSH-like hashing methods need long binary codes to achieve reasonable performance. Unfortunately, representing each sample with long binary codes will easily lead to a low recall rate so that multiple hash tables have to be considered [27], [28]. Consequently, LSH-like hashing methods (with both the long binary codes and multiple hash tables) suffer from long query time and high storage cost.

A good hashing method should have the following properties [29], [30]. First of all, the codes should be short to enable the easy storage of large amounts of images in memory. For example, to store 1 billion images in a normal server with 64GB of RAM, we should represent each image with tens of bits. Second, the binary codes should preserve the similarity structures of the original data as much as possible. Finally, the training stage should be applicable to large-scale applications (e.g., at most linear time complexity to the training size for hash function learning) and the query stage should be efficient and scalable, e.g., constant query time to the training size.

To meet the above requirements, various learning techniques have been developed and applied to hashing. For example, different from the data-independent hashing methods (such as LSH-like hashing) which learn hash functions without considering the statistical properties of the data, data-dependent hashing methods (such as PCA-like hashing [29], [31], [32], [33], [34] and manifold-like hashing [20], [30], [35], [36], [37]) generate hash functions by making use of the correlation among the data. PCA-like hashing methods learn hash functions via preserving the maximal variance of original

Xiaofeng Zhu is with the College of Computer Science & Information Technology, Guangxi Normal University, China. Email: seanzhuxf@gmail.com.
Lei Zhang (Corresponding author) is with Department of Computing, The Hong Kong Polytechnic University, Hong Kong. Email: cslzhang@comp.polyu.edu.hk.
Zi Huang is with School of Information Technology and Electrical Engineering, The University of Queensland, Australia. Email: huang@itee.uq.edu.au.

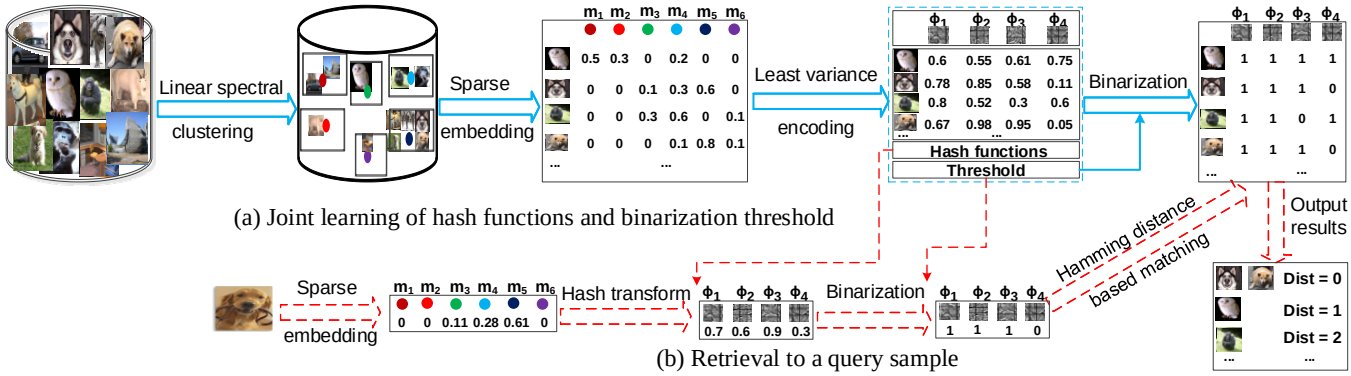


Fig. 1. Flowchart of the proposed SELVE approach to hashing.

data; however, they assign the same weight to each bit and ignore the fact that different directions have different variances. To address this problem, Spectral Hashing (SH) [33] uses a separable Laplacian eigenfunction to prohibit assigning more bits to the directions along which the data have a greater range. However, this approach is a heuristic method and it makes an impractical assumption that the data should uniformly distribute in the high dimensional space. Isotropic Hashing (IsoHash) [38] learns the hash functions to produce isotropic variances (equal variances) along different directions. Iterative Quantization (ITQ) [32] uses non-orthogonal relaxation or sequential projections to alleviate the problem of unbalanced variances.

Manifold-like hashing methods aims to preserve the neighborhood structures of samples, i.e., similar samples should have similar binary codes. Self-taught hashing (STH) [20] preserves the neighborhood structures of single-modality data, while composite hashing with multiple information sources (CHMIS) [37]. Anchor graph hashing (AGH) [39] preserves approximate neighborhood structures by building an anchor graph. The main drawbacks of most manifold-like hashing methods are the high complexity and the out-of-sample extension problem (i.e., no explicit hash functions, such as AGH). Many manifold-like hashing methods preserve the neighborhood structures of training samples by building a sparse kNN graph, which has at least quadratic time complexity. Such a high complexity is prohibitive for large-scale image retrieval. AGH reduces the high time complexity to linear by preserving approximate neighborhood structures. However, it needs to use long binary codes to achieve good hashing performance. In particular, AGH achieves its best performance with 48-bit (or above) binary codes, but its performance can be worse than other representative methods (e.g., ITQ) when shorter binary codes are used.

In this paper, we propose a new hashing approach from a different point of view to previous ones. Instead of building a sparse kNN graph to preserve the neighborhood structures of training samples, we preserve and encode the spatial embedding of each sample in the space spanned by k clustering centroids of the training samples, aiming to achieve good hashing performance with short binary codes and linear time complexity. Fig.1 illustrates the flowchart of the proposed approach. In the training stage (refer to Fig.1(a)), we first partition the training samples into k clusters by a linear clustering method such as linear spectral clustering [40]. The obtained k centroids are used to measure the distance between a sample and each cluster, and then each training sample can be mapped into the space spanned by the k centroids to obtain its spatial embedding. We sparsely represent each sample by its several nearest centroids, and generate a sparse vector of normalized probabilities that it falls into the several closest clusters. This sparse embedding process has linear time complexity and it converts the original high dimensional data into a low dimensional space with approximate neighborhood structure. The resulting low dimensional sparse embedding vectors are used to learn the hash functions.

We consequently propose to learn a dictionary to encode the sparse embedding features with a least variance encoding model, which jointly outputs the hash functions and the binarization threshold. In the test stage (please refer to Fig.1(b)), the query samples are first converted into binary codes by the learnt hash functions, and fast retrieval can then be conducted via calculating the Hamming distance between the binary codes of query sample and training samples. To further improve the effectiveness of the proposed sparse embedding and least variance encoding (SELVE) approach, we transform the samples into a new space where each cluster is more concentrated, resulting in a sparser embedding vector and better hashing performance.

The proposed SELVE method could learn effective hash functions with short binary codes. Our extensive experimental results on four real-world datasets show that SELVE outperforms many state-of-the-art hashing algorithms ([21], [41], [33], [32], [39], [34], [42]) in terms of various kinds of evaluation indices, such as precision-recall, mean precision of Hamming radius 2 (HAM2) and mean average precision (MAP). The success of SELVE mainly comes from the following two aspects. First, the sparse embedding vector is a distinctive indicator of the spatial location of a sample in the whole sample space. Second, the least variance encoding model couples the learning of hash functions with the learning of binarization threshold. As a result, it makes the hash code generation more effective.

The remainder of the paper is organized as follows. The

proposed approach and its analysis are presented in Section II. Section III reports the experimental results and Section IV concludes the paper.

II. The methodology

In this section, we describe in detail the proposed sparse embedding and least variance encoding (SELVE) approach. In general, given a dataset $\mathbf{X} \in \mathbb{R}^{d\times n}$ where each sample is a d-dimensional column vector and there are n training samples, our goal is to represent $\mathbf{X}$ by $\mathbf{B} \in \{0,1\}^{c\times n}$ (where $c \ll d$) such that the neighborhood structure of each training sample $\mathbf{x}_i$ is still preserved in the Hamming space. To this end, we first represent each training sample as a k-dimensional sparse embedding vector (see Section II-A), which is then encoded over a dictionary (see Section II-B). Furthermore, we can learn a transformation matrix to make the training samples more concentrated, i.e., achieving a sparser embedding (see Section II-C). Finally, we summarize the proposed approach (See Section II-D) and analyze its complexity (see Section II-E).

A. Sample representation by spatial embedding

Many hashing methods (e.g., STH, MFH, etc.) build a sparse kNN graph such that each training sample $\mathbf{x}_i$ in $\mathbf{X}$ is represented as an n-dimensional sparse vector $\bar{\mathbf{p}}_i \in \mathbb{R}^n$, which stands for the relationship between $\mathbf{x}_i$ and other training samples in $\mathbf{X}$. To preserve the neighborhood of each training sample, $\bar{\mathbf{p}}_i$ is usually constructed by the k nearest neighbors to $\mathbf{x}_i$; that is, only k elements (or coordinates) in $\bar{\mathbf{p}}_i$ are non-zeros. However, building such a sparse kNN graph needs at least quadratic time complexity, which is impractical in large-scale search.

Instead of building a sparse kNN graph to preserve the neighborhood structures of training samples, we propose to represent each sample as its spare spatial embedding in a low dimensional space. The rationale and motivation of such a strategy are as follows. First of all, if two samples are neighbors, they will have similar spatial location and thus similar spatial embedding. Second, the spatial embedding vector tends to be sparser than the kNN vector because one sample can have many neighbors but it can only be close to several clusters. Third, the spatial embedding has much less complexity than kNN graph building, and the resulting representation vector has much lower dimensionality.

To compute the spatial embedding of the training samples, we first partition the training sample space into k ($k << n$ and $k < d$) cells. This can be simply done by a linear clustering method such as the spectral clustering method in [40]. As a result, the clustering will output k clusters with centroids $\mathbf{m}_i$, $i = 1, 2, ..., k$. We denote the Euclidean distance between a sample $\mathbf{x}_i$ and the centroid $\mathbf{m}_j$ as

$$r_{i,j} = \|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \tag{1}$$

Like in [43], the Euclidean distance $r_{i,j}$ can be converted into a probability that $\mathbf{x}_i$ belongs to cluster j via the following formula:

$$p_{i,j} = \frac{exp(-r_{i,j}/\sigma)}{\sum_{l=1}^{k} exp(-r_{i,l}/\sigma)} \tag{2}$$

where σ is a parameter to control the decay rate of $p_{i,j}$ with respect to distance $r_{i,j}$. For simplicity, we set σ=1 in this paper.

Let $\mathbf{p}_i = [p_{i,1}, ..., p_{i,j}, ..., p_{i,k}]^T$, and $\mathbf{p}_i$ forms a representation of $\mathbf{x}_i$, characterizing its spatial location in the sample space. Since in general a sample can only belong to one of its several closest clusters, we represent each training sample using its s ($s << k$) nearest centroids, and hence vector $\mathbf{p}_i$ will be a very sparse vector. Denote by p_s the s^{th} largest probability in $\mathbf{p}_i$, and let

$$p'_{i,j} = \begin{cases} p_{i,j} & \text{if } \ p_{i,j} \geq p_s \\ 0 & \text{if } \ p_{i,j} < p_s \end{cases} \tag{3}$$

We then normalize $p'_{i,j}$ as:

$$\bar{p}_{i,j} = \frac{p'_{i,j}}{\sum_{l=1}^{k} p'_{i,l}}. \tag{4}$$

Finally we regard

$$\bar{\mathbf{p}}_i = [\bar{p}_{i,1}, ..., \bar{p}_{i,j}, ..., \bar{p}_{i,k}]^T \tag{5}$$

as the sparse embedding feature vector of $\mathbf{x}_i$.

B. Least variance encoding via dictionary learning

Denote by $\mathbf{P} = [\bar{\mathbf{p}}_1, \bar{\mathbf{p}}_2, ..., \bar{\mathbf{p}}_n] \in \mathbb{R}^{k\times n}$ the sparse embedding feature set of the training samples. The next step is to learn the hash functions from $\mathbf{P}$ to convert $\bar{\mathbf{p}}_i$ into binary codes. To this end, we propose to encode each $\bar{\mathbf{p}}_i$ over a dictionary $\mathbf{\Phi}$ and binarize its coding vector as the hash codes. We propose the following model to learn the desired dictionary:

$$\min_{\mathbf{\Phi},\mathbf{\Lambda}} \ \|\mathbf{P} - \mathbf{\Phi}\mathbf{\Lambda}\|_F^2 + \lambda \sum_{i=1}^{n} \|\boldsymbol{\alpha}_i - \boldsymbol{\mu}\|_2^2, \quad s.t. \ \ \|\boldsymbol{\phi}_j\|_2^2 = 1 \tag{6}$$

where $\mathbf{\Phi} \in \mathbb{R}^{k\times c}$ is called the encoding dictionary and each atom $\boldsymbol{\phi}_j \in \mathbb{R}^k$ of $\mathbf{\Phi}$ has unit ℓ_2-norm; $\mathbf{\Lambda} = [\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_n] \in \mathbb{R}^{c\times n}$ is the coding matrix of $\mathbf{P}$ over $\mathbf{\Phi}$ and $\boldsymbol{\alpha}_i$ is the coding vector of $\bar{\mathbf{p}}_i$; $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\alpha}_i$ is the mean of all coding vectors; the rationale of the regularization term $\|\boldsymbol{\alpha}_i - \boldsymbol{\mu}\|_2^2$ is to reduce the variance of coding vectors $\boldsymbol{\alpha}_i$ so that similar $\bar{\mathbf{p}}_i$ can more likely have similar codes; and λ is a constant.

The proposed least variance encoding (LVE) model in Eq.6 jointly optimizes the dictionary $\mathbf{\Phi}$ and the coding matrix $\mathbf{\Lambda}$, while the mean vector $\boldsymbol{\mu}$ is simultaneously obtained with $\mathbf{\Lambda}$. Via the LVE model, the learned dictionary $\mathbf{\Phi}$ can be directly converted into the desired *hash functions* (see Eq.13 and Eq.14), while the mean vector $\boldsymbol{\mu}$ can be directly taken as the *binarization threshold* to binarize $\boldsymbol{\alpha}_i$ (see Eq.14). Since the hash functions and the binarization threshold are jointly optimized, the loss caused by the hashing can be more effectively reduced. There are interactions between the threshold and the hash functions in the optimization process. In contrast, many existing hashing methods [20], [44] learn the hash functions and the threshold separately.

The minimization in Eq.6 is not jointly convex but it is convex w.r.t. either $\mathbf{\Phi}$ or $\mathbf{\Lambda}$ when another is fixed. Therefore, we optimize $\mathbf{\Phi}$ and $\mathbf{\Lambda}$ alternatively. We first randomly initialize $\mathbf{\Phi}$ and optimize $\mathbf{\Lambda}$ by fixing $\mathbf{\Phi}$. Eq.6 becomes:

$$\min_{\mathbf{\Lambda}} \ \|\mathbf{P} - \mathbf{\Phi}\mathbf{\Lambda}\|_F^2 + \lambda \sum_{i=1}^{n} \|\boldsymbol{\alpha}_i - \boldsymbol{\mu}\|_2^2 \tag{7}$$

After some mathematical derivation, we have the following analytical solution to each $\boldsymbol{\alpha}_i$ in $\mathbf{\Lambda}$:

$$\begin{aligned}\hat{\boldsymbol{\alpha}}_i &= \arg\min_{\boldsymbol{\Lambda}} \|\mathbf{P}-\boldsymbol{\Phi}\boldsymbol{\Lambda}\|_F^2 + \lambda\sum_{i=1}^{n}\langle\boldsymbol{\alpha}_i-\boldsymbol{\mu},\boldsymbol{\alpha}_i-\boldsymbol{\mu}\rangle \\ &= \arg\min_{\boldsymbol{\Lambda}} \|\mathbf{P}-\boldsymbol{\Phi}\boldsymbol{\Lambda}\|_F^2 \\ &\quad + \lambda\sum_{i=1}^{n}(\langle\boldsymbol{\alpha}_i,\boldsymbol{\alpha}_i\rangle - 2\langle\boldsymbol{\alpha}_i,\boldsymbol{\mu}\rangle + \langle\boldsymbol{\mu},\boldsymbol{\mu}\rangle) \\ &= \arg\min_{\boldsymbol{\Lambda}} \|\mathbf{P}-\boldsymbol{\Phi}\boldsymbol{\Lambda}\|_F^2 + \lambda\sum_{i=1}^{n}\|\boldsymbol{\alpha}_i\|_2^2 - \lambda n\|\boldsymbol{\mu}\|_2^2\end{aligned} \tag{8}$$

Let the partial derivative of Eq.8 with respect to $\boldsymbol{\alpha}_i$ equal 0, we have:

$$(\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\lambda\mathbf{I})\hat{\boldsymbol{\alpha}}_i - \boldsymbol{\Phi}^T\bar{\mathbf{p}}_i = \lambda(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}(\boldsymbol{\Phi}^T\frac{1}{n}\sum_{i=1}^{n}\bar{\mathbf{p}}_i) \tag{9}$$

where $\mathbf{I}$ is an identity matrix. Finally, we obtain the following analytical solution to each $\boldsymbol{\alpha}_i$ in $\boldsymbol{\Lambda}$:

$$\hat{\boldsymbol{\alpha}}_i = (\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\lambda\mathbf{I})^{-1}(\boldsymbol{\Phi}^T\bar{\mathbf{p}}_i + \lambda(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{z}) \tag{10}$$

where $\mathbf{z} = \frac{1}{n}\sum_{i=1}^{n}\bar{\mathbf{p}}_i$ is the mean of all vectors $\bar{\mathbf{p}}_i$.

Fixing $\boldsymbol{\Lambda}$, the minimization problem in Eq.6 is reduced to

$$\min_{\boldsymbol{\Phi}} \|\mathbf{P}-\boldsymbol{\Phi}\boldsymbol{\Lambda}\|_F^2, \quad s.t. \quad \|\boldsymbol{\phi}_j\|_2^2 = 1 \tag{11}$$

via which the dictionary $\boldsymbol{\Phi}$ can be updated by methods such as [45].

The coding matrix $\boldsymbol{\Lambda}$ and dictionary $\boldsymbol{\Phi}$ are alternatively updated until convergence or the maximum iteration number is reached. Once $\boldsymbol{\Lambda}$ and dictionary $\boldsymbol{\Phi}$ are learnt, we let

$$\begin{cases}\mathbf{T} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\lambda\mathbf{I})^{-1}\boldsymbol{\Phi}^T \\ \boldsymbol{\beta} = \lambda(\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\lambda\mathbf{I})^{-1}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{z}\end{cases} \tag{12}$$

According to Eq.10, we have

$$\boldsymbol{\alpha}_i = \mathbf{T}\bar{\mathbf{p}}_i + \boldsymbol{\beta}. \tag{13}$$

Eq.13 gives our hash function; that is, the real-valued code vector $\boldsymbol{\alpha}_i$ is obtained by projecting the feature $\bar{\mathbf{p}}_i$ onto $\mathbf{T}$ and then shifting with $\boldsymbol{\beta}$.

With all $\boldsymbol{\alpha}_i$ available, we obtain the mean vector via $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\alpha}_i$. We then binarize $\boldsymbol{\alpha}_i$, $i = 1, ..., n$, as follows:

$$\begin{cases}\mathbf{b}_i(j) = 1 & if \quad \boldsymbol{\alpha}_i(j) \geq \boldsymbol{\mu}(j) \\ \mathbf{b}_i(j) = 0 & if \quad \boldsymbol{\alpha}_i(j) < \boldsymbol{\mu}(j)\end{cases} \tag{14}$$

where $j = 1, ..., c$. That is, the mean value is used as the threshold for binarization. Finally, we obtain the binary codes $\mathbf{B} = [\mathbf{b}_1, ..., \mathbf{b}_n]$ of the original training samples in $\mathbf{X}$.

The rationale of the least variance regularization $\sum_{i=1}^{n}\|\boldsymbol{\alpha}_i-\boldsymbol{\mu}\|_2^2$ in Eq.7 is twofold. On one hand, the least variance regularization is used to penalize the deviation of each hash value (i.e., the real value of hash code) from the mean (i.e., the threshold). This reduces the impact of outliers in the process of hash function learning. On the other hand, it makes the learning of threshold (i.e., $\boldsymbol{\mu}$) and the learning of hash codes unified. In contrast, traditional methods usually set the threshold separately after the real hash values are learned. In each iteration of SELVE training, $\boldsymbol{\mu}$ is adjusted by the real valued hash codes (i.e., $\boldsymbol{\alpha}_i$). Those $\boldsymbol{\alpha}_i$ are used to update the dictionary $\boldsymbol{\Phi}$, while $\boldsymbol{\Phi}$ will be used to update $\boldsymbol{\alpha}_i$ in return. This iterative process makes the final threshold fits much better the hash codes and the given data.

In the test stage, given a new data point $\mathbf{y}$, we first calculate its sparse embedding feature vector $\bar{\mathbf{p}}_y$ using Eqs.(1-5), and then compute its real-valued code vector $\boldsymbol{\alpha}_y$ using Eq.13, and obtain its binary codes $\mathbf{b}_y$ by Eq.14. Finally, the Hamming distance between $\mathbf{b}_y$ and $\mathbf{B}$ is computed to find the nearest neighbours of $\mathbf{y}$.

C. Enhanced sparse embedding

From Section II-A and Section II-B, we see that the hash function learning is to encode the spatial embedding vector $\bar{\mathbf{p}}_i$ of the training sample $\mathbf{x}_i$. Intuitively, if we can have a sparser embedding representation of $\mathbf{x}_i$, i.e., only a couple of elements in $\bar{\mathbf{p}}_i$ are significant, the sample $\mathbf{x}_i$ can be better identified in the space spanned by the training samples. This can consequently make the binary codes more informative. To this end, we propose an enhanced sparse embedding (E-SE) of the training samples in $\mathbf{X}$. Our goal is to learn a transformation matrix, denote by $\mathbf{W} \in \mathbb{R}^{l\times d}$, so that in the new space spanned by $\mathbf{X}' = \mathbf{W}\mathbf{X} \in \mathbb{R}^{l\times n}$, each cluster can be more concentrated. That is, after the transformation each sample in cluster k will be closer to its centroid and farther to other centroids, resulting in a sparser embedding vector of each sample.

The learning of such a $\mathbf{W}$ can be performed based on the k clustering results of $\mathbf{X}$. Denote by $\mathbf{X}_j$, $j = 1, ..., k$, the set of training samples in the j^{th} cluster with the centroid $\mathbf{m}_j$. Let $\mathbf{M}_j$ be a matrix which has the same size as $\mathbf{X}_j$ but each column of $\mathbf{M}_j$ is $\mathbf{m}_j$. Let $\boldsymbol{\Gamma}_j = \mathbf{X}_j - \mathbf{M}_j$. We can see that $\|\boldsymbol{\Gamma}_j\|_F^2$ reflects the variance of the training samples in cluster $\mathbf{X}_j$. The goal of E-SE is to find a transformation matrix $\mathbf{W} \in \mathbb{R}^{l\times d}$ ($l \leq d$) such that $\mathbf{W}\mathbf{X}$ can preserve the energy of $\mathbf{X}$, while the variance of each cluster $\mathbf{X}_j$ can be reduced (i.e., the cluster is more concentrated). This leads to the following objective function [46]:

$$\begin{aligned}\min_{\mathbf{W}} \|\mathbf{X}-\mathbf{W}^T\mathbf{W}\mathbf{X}\|_F^2 + \gamma\sum_{j=1}^{k}\|\mathbf{W}\boldsymbol{\Gamma}_j\|_F^2, \\ \text{s.t.}, \mathbf{W}\mathbf{W}^T = \mathbf{I}\end{aligned} \tag{15}$$

where γ is a tuning parameter.

In Eq.15, the term $\|\mathbf{X}-\mathbf{W}^T\mathbf{W}\mathbf{X}\|_F^2$ is a data fidelity term, ensuring that $\mathbf{X}$ can be well reconstructed from its transformed data $\mathbf{W}\mathbf{X}$. The second term $\|\mathbf{W}\boldsymbol{\Gamma}_j\|_F^2$ is a regularization term, enforcing that each cluster has less variance after transformation. One can see that if we set $\gamma = 0$, Eq.15 will be reduced to PCA. By setting a suitable value of γ, the learned transformation matrix $\mathbf{W}$ will make the sample distribution more concentrated while preserving the energy of original training samples.

Let $\boldsymbol{\Gamma} = [\boldsymbol{\Gamma}_1, ..., \boldsymbol{\Gamma}_k]$, the minimization of Eq.15 is equivalent to:

$$\begin{aligned}&\min_{\mathbf{W}} tr\{(\mathbf{X}-\mathbf{W}^T\mathbf{W}\mathbf{X})^T(\mathbf{X}-\mathbf{W}^T\mathbf{W}\mathbf{X}) + \gamma W\boldsymbol{\Gamma}\boldsymbol{\Gamma}^T\mathbf{W}^T\} \\ &= \min_{\mathbf{W}} tr\{\mathbf{X}^T\mathbf{X} - 2\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X} + \mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{X} \\ &\qquad + \gamma\mathbf{W}\boldsymbol{\Gamma}\boldsymbol{\Gamma}^T\mathbf{W}^T\} \\ &= \min_{\mathbf{W}} tr\{\mathbf{X}^T\mathbf{X} - \mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X} + \gamma\mathbf{W}\boldsymbol{\Gamma}\boldsymbol{\Gamma}^T\mathbf{W}^T\} \\ &= \min_{\mathbf{W}} tr\{\mathbf{W}(\gamma\boldsymbol{\Gamma}\boldsymbol{\Gamma}^T - \mathbf{X}^T\mathbf{X})\mathbf{W}^T + \mathbf{X}^T\mathbf{X}\} \\ &= \max_{\mathbf{W}} tr\{\mathbf{W}(\mathbf{X}\mathbf{X}^T - \gamma\boldsymbol{\Gamma}\boldsymbol{\Gamma}^T)\mathbf{W}^T\}\end{aligned} \tag{16}$$

Clearly, the desired $\mathbf{W}$ can be solved by applying Singular Value Decomposition (SVD) to the matrix $(\mathbf{X}\mathbf{X}^T - \gamma\boldsymbol{\Gamma}\boldsymbol{\Gamma}^T)$; that is, $\mathbf{W}$ is composed by the l eigenvectors associated with the first l largest eigenvalues of $(\mathbf{X}\mathbf{X}^T - \gamma\boldsymbol{\Gamma}\boldsymbol{\Gamma}^T)$.

With the learned $\mathbf{W}$, we transform $\mathbf{X}$ into $\mathbf{X}' = \mathbf{W}\mathbf{X}$, and the sparse embedding is applied to $\mathbf{X}'$ using the method described in Section II-A. The geometric illustration of the role of the transform matrix $\mathbf{W}$ is shown in Fig.2.

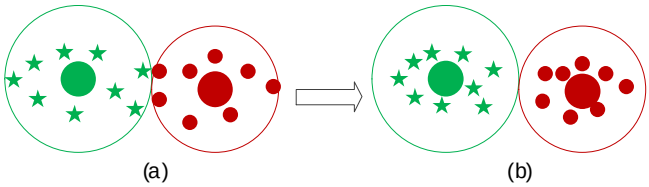


Fig. 2. Illustration of the role of transform $\mathbf{W}$. (a) Two clusters before transform; (b) the two clusters after transform. One can see that the data distribution is more concentrated after transform. Some points originally locate in the boundary of the two clusters are moved inward so that they will have a more concentrated and sparser spatial embedding.

D. Summary of the algorithm

The proposed approach in the original space spanned by $\mathbf{X}$ is abbreviated as SELVE, while the proposed method in the transformed space spanned by $\mathbf{X}'$ is abbreviated as E-SELVE. Since SELVE can be viewed as a special case of E-SELVE, we summarize the E-SELVE algorithm in Algorithm 1 (training stage) and Algorithm 2 (test stage).

Algorithm 1: The Algorithm of E-SELVE: Training Stage

Input: training data: $\mathbf{X} \in \mathbb{R}^{d\times n}$; c (the number of bits); l (the dimensionality of transformed space); k (the number of centroids).

Output: $\mathbf{B} \in \mathbb{R}^{c\times n}$, $\mathbf{W} \in \mathbb{R}^{d\times l}$, $\mathbf{T} \in \mathbb{R}^{c\times k}$, $\boldsymbol{\beta} \in \mathbb{R}^{k}$, $\boldsymbol{\mu} \in \mathbb{R}^{c}$.

Perform a clustering method on $\mathbf{X}$ to obtain k centroids $\mathbf{m}_j$;
Compute $\mathbf{W}$ by Eq.16;
$\mathbf{X} = \mathbf{W}\mathbf{X}$, $\mathbf{m}_j = \mathbf{W}\mathbf{m}_j$;
Compute $\mathbf{P}$ by Eq.4;
$\boldsymbol{\Phi} = \text{rand}(l, c)$;
repeat
  Update $\boldsymbol{\Phi}$ by Eq.11;
  Update $\mathbf{T}$ and $\boldsymbol{\beta}$ by Eq.12;
  Update $\boldsymbol{\alpha}$ by Eq.13;
  $\mathbf{u} = \text{mean}(\boldsymbol{\alpha})$;
until *Convergence*;
Compute $\mathbf{B}$ by Eq.14;

Algorithm 2: The Algorithm of E-SELVE: Test Stage

Input: $\mathbf{y} \in \mathbb{R}^{d}$, $\mathbf{B} \in \mathbb{R}^{c\times n}$, $\mathbf{W} \in \mathbb{R}^{d\times l}$, $\mathbf{T} \in \mathbb{R}^{c\times k}$, $\boldsymbol{\beta} \in \mathbb{R}^{k}$, $\boldsymbol{\mu} \in \mathbb{R}^{c}$;

Output: $\mathbf{b}_y \in \mathbb{R}^{c}$.

$\mathbf{y} = \mathbf{W}\mathbf{y}$;
Compute $\bar{\mathbf{p}}_y$ by Eq.4;
Compute $\boldsymbol{\alpha}_y$ by Eq.13;
Compute $\mathbf{b}_y$ by Eq.14;
Match $\mathbf{b}_y$ to $\mathbf{B}$;

The transform used in the proposed E-SE step is a PCA-like subspace learning method. Different from PCA which preserves the maximal variance (a.k.a., the energy) after the transformation, the proposed transform preserves the energy while making the clusters more concentrated, aiming at obtaining a sparser embedding vector of each sample. Traditional PCA is a special case of our transformation, i.e., setting $\gamma = 0$ in Eq.15. By setting $\gamma > 0$, the proposed E-SELVE method can achieve better hashing accuracy than setting $\gamma = 0$ (i.e., using PCA for transform). For example, in the experiment on the MNIST dataset (Please refer to Section III-D for details), the HAM2 results of the proposed hashing method are 0.8805 and 0.8712, respectively, for $\gamma = 0.1$ and $\gamma = 0$. In addition, our E-SELVE algorithm is not sensitive to the selection of γ.

On the other hand, the SE step in our method embeds the sample sparsely in the manifold subspace. It aims to build an approximate kNN graph with linear time complexity, which is favorable for large-scale hashing. The conventional manifold learning methods (e.g., [47], [48], [49]) build an exact kNN graph, which have at least quadratic time complexity and are prohibitive for large-scale hashing applications.

As in most dictionary learning methods [45], [50], the LVE step in the proposed hashing algorithm is not jointly convex to $\boldsymbol{\Phi}$ and $\boldsymbol{\Lambda}$, but it is convex to each of them when another one is fixed. In the alternative optimization, the energy will decrease step by step. In Fig. 3 we plot the convergence curves of the proposed E-SELVE hashing method on the datasets used in our experiments (please refer to Section III for details). One can see that the value of objective function decreases rapidly in the first several iterations and then becomes stable after about ten iterations.

E. Time and space complexity

In E-SELVE, the time complexities of clustering method [40], sparse embedding, hash function learning, binarization, and E-SE transform learning are $O(dkn)$ [40], $O(lkn + scn)$, $O(t(c^2d + cdn + c^3 + c^2n))$, $O(cn)$ and $O(d^3)$, respectively, in the training stage, where t refers to the number of iterations in learning $\boldsymbol{\Phi}$. In the test stage, it will cost $O(dk + kl + sc)$ for E-SE representation, hashing transform and binarization, plus $O(1)$ for performing inverse lookup in the hash table. Considering that l is much smaller than d, and s and c are usually small numbers, the time complexity of E-SELVE is basically $O(dkn + d^3)$ in the training stage and $O(dk)$ in the test stage. Without the E-SE transform, the time complexity of SELVE is $O(dkn)$ in the training stage and $O(dk)$ in the test stage. Table 1 compares the time complexity of E-SELVE with those of representative algorithms such as LSH [21], LSI [41], SH [33], ITQ [32] AGH[39], MDSH [34] and SpH [42].

The space complexity of E-SELVE is about $O(d(l + k + n))$ in the training stage and $O(dl + ck)$ plus $O(cn)$ (binary bits) in the test stage, while for SELVE the space complexity is $O(d(k + n))$ in the training stage and $O(ck)$ plus $O(cn)$ (binary bits) in the test stage.

III. Experimental results

We compare our proposed methods with two baseline approaches, LSH [21] and LSI [41], and five state-of-the-arts, SH [33], ITQ [32], AGH [39], SPH [42], and MDSH [34], on four widely used datasets, including MNIST (70K)[1], CIFAR(60K) [51], NUS-WIDE (193K) [52] and GIST (500K) [53]. Note that AGH has two variants and we use the one with better results (i.e., 2-AGH in the original paper).

[1]http://yann.lecun.com/exdb/mnist/

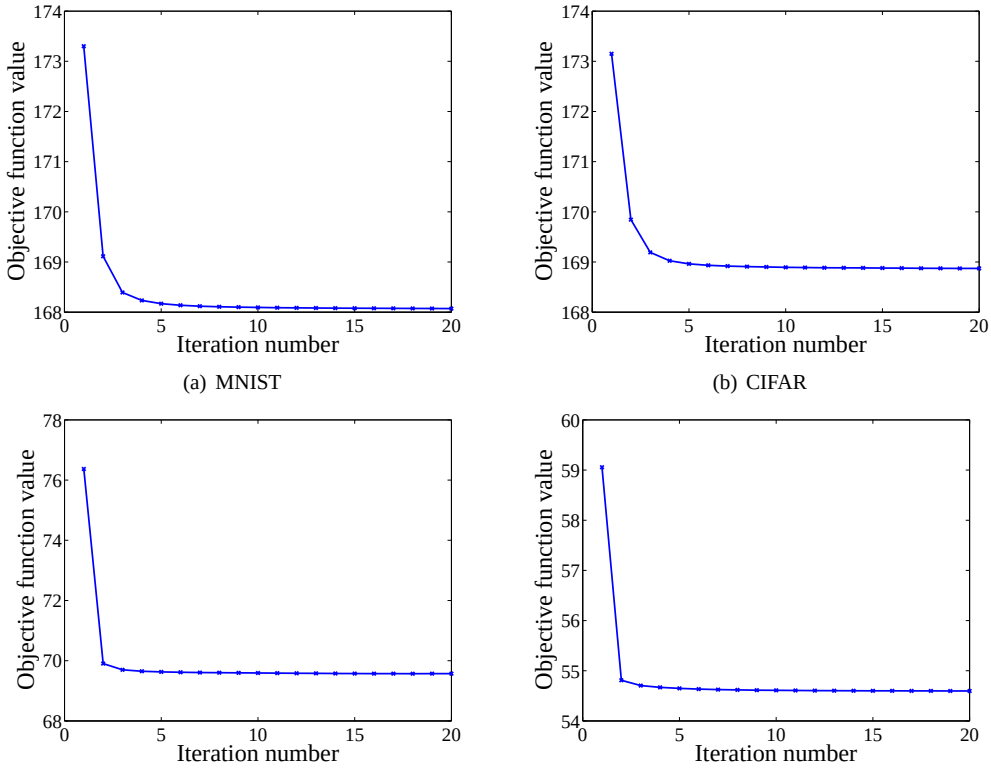


Fig. 3. The convergence curves of E-SLVE model ($\lambda = 1$, $\gamma = 1$, $c = 28$, and $k = 300$).

TABLE I
TIME COMPLEXITY COMPARISON BETWEEN COMPETING ALGORITHMS.

	Training stage	Test stage
LSH	$O(1)$	$O(1)$
LSI	$O(dn^2)$	$O(dc)$
SH	$O(d^2n)$	$O(dc)$
ITQ	$O(d^2n + c^3)$	$O(dc)$
AGH	$O(dknT + k^2n + scn)$	$O(dk + sc)$
MDSH	$O(d^2n)$	$O(dc)$
SpH	$O(c^2 + cdn)$	$O(dc^2)$
LVE	$O(dn)$	$O(sc)$
SELVE	$O(dkn)$	$O(dk)$
E-SELVE	$O(dkn + d^3)$	$O(dk)$

Note that c, d, k, n, s, and T are the length of binary codes, the dimensionality of the samples, the number of centroids, the training size, the number of nearest neighbors, and the iteration of k-means algorithm, respectively.

A. Datasets

The MNIST dataset consists of 70,000 digit images of size 28×28. We use the Matlab format of MNIST dataset provided by [39], [8], including 69,000 training samples and 1,000 test samples. Following Liu et al. [39], we split the MNIST dataset into two parts: a training set containing 69,000 data points and a test set of 1,000 data points. Because MNIST is fully annotated, we regard true nearest neighbors as semantic nearest neighbors based on the associated digit labels.

The CIFAR datasets consists of 60,000 colour images (size: 32×32) from 10 classes, while each class has 6,000 samples. 50,000 images are used for training and the remaining 10,000 images are used for testing. For each image, a 512-dimensional GIST feature vector is extracted [54]. The CIFAR dataset is also fully annotated, and we regard their true nearest neighbors as semantic nearest neighbors based on the associated class labels.

The NUS-WIDE dataset originally contains 269,648 images associated with 81 ground truth concept tags. We pruned the original NUS-WIDE in our experiments based on the description in [18, 29]. First, a subset of 195,969 images was extracted, in which each image is annotated by at least one of the 21 most frequent labels. Second, we uniformly sampled 100 images from each of the 21 tags to form a query set of 2,100 images. The other 193,869 images serve as the training set. Third, each image is represented by a 500-dimensional SIFT feature vector. Finally, the ground truth is defined according to whether two images share at least one common tag.

In the GIST dataset [53], we selected all the 500K learning vectors and 1,000 query vectors as our training dataset and query dataset, respectively. In GIST, each image is represented by a 960-dimensional GIST descriptor. The original ground truths in GIST were not designed for the training dataset. Similar to the setting in [26], we use a nominal threshold of the average distance to the 50^{th} nearest neighbours to determine whether a training sample returned for a given query sample is considered as a ground truth.

B. Evaluation criteria and comparison approaches

In order to evaluate the proposed hashing approach, some quantitative criteria should be used. We follow the literatures (such as [8], [39]) to conduct two popular search procedures, i.e., hash lookup and Hamming ranking.

Hash lookup first constructs a lookup table for the binary codes of all data samples, and then returns the percentage of the data samples falling into a small Hamming radius centered at the query sample. For example, HAM2 (i.e., the Hamming radius is 2) means that the Hamming distance between the training samples and the query sample is smaller than 2. In our experiments, we employed HAM2 to evaluate the results of hash lookup. Although the complexity of hash lookup is constant, it may fail in the case of long hash codes because the Hamming codes space is very sparse so that only few samples fall into the same hash bucket as the query.

Hamming ranking ranks all the samples in the database based on their Hamming distances to the query sample and then returns the top ones. The complexity of Hamming ranking is linear. In our experiments, we employ the mean average precision (MAP) (i.e., the mean of all query samples' average precision) to evaluate the performance of Hamming ranking under different hash bits. Since computing MAP is slow on the two large datasets NUS-WIDE and GIST, following the literature [8], [39], we report the results of Mean Precision (MP) of the top-50K returned neighbors.

These two distinct criteria (i.e., Hash lookup and Hamming ranking) are designed to evaluate different characteristics of hashing techniques. The larger the HAM2 or MAP or MP, the better the hashing performance.

Besides, we use the precision-recall curve to measure the hashing performance. The definitions of precision and recall are as follows [37]:

$$precision = \frac{Number\ of\ retrieved\ relevant\ images}{Total\ number\ of\ all\ retrieved\ images} \quad (17)$$

$$recall = \frac{Number\ of\ retrieved\ relevant\ images}{Total\ number\ of\ all\ relevant\ images} \quad (18)$$

In order to more comprehensively evaluate the proposed hashing method, we evaluate three variants of it:

- LVE, i.e., using LVE to perform hashing in the original data space;
- SELVE, i.e., LVE with sparse embedding (SE);
- E-SELVE, i.e., LVE with enhanced SE (E-SE).

We compare LVE, SELVE and E-SELVE with the following representative and state-of-the-art hashing approaches.

- Locality Sensitive Hashing (LSH) [21]: LSH generates random linear functions as hash functions. Following the literature [32], we generate a Gaussian random matrix as the projection matrix (i.e., the hash functions) in our experiments.
- Latent Semantic Indexing (LSI) [41], [55], [56]: Binarized Latent Semantic Indexing (LSI) employs truncated SVD [57], [58] to find the best low-rank description of original data $\mathbf{X}$, i.e., the low-rank matrix $\mathbf{D}$ with minimum error $\|\mathbf{X}-\mathbf{D}\|_F^2$. Then the low-rank matrix $\mathbf{D}$ revealing the underlying semantic structure of the original data matrix $\mathbf{X}$ is used to learn the hash functions.
- Spectral Hashing (SH) [33]: SH first conducts PCA on the original data, and then uses the analytical Laplacian eigenfunctions computed along the principal directions of the data for projection to generate the hash codes. It belongs to a PCA-like hashing method.
- Iterative Quantization (ITQ) [32]: ITQ first conducts PCA on the original data, and then learns an orthogonal matrix to solve the unbalanced variance problem on different PCA directions. ITQ, as a PCA-like hashing, has shown better performance than other PCA-like hashing methods (such as SH [33], SKLSH [26], and PCA-Nonorth [59]).
- Anchor Graph Hashing (AGH) [39]: AGH is a manifold-like hashing method. It first generates the new representation for each sample, and then uses eigenfunction generalization to encode test samples to solve the out-of-sample problem. AGH has shown superior performance to many state-of-the-art hashing methods, such as SH [33], PCA-Nonorth [59], KLSH [25], SKLSH [26], and PCA Hashing.
- Multidimensional Spectral Hashing (MDSH) [34]: Based on the observation that the relative performance of hashing can change dramatically with the definition of ground-truth neighbors, MDSH first reconstructs the affinity matrix of the data points, and then solves a binary matrix factorization problem of the affinity matrix, i.e., finding the top eigenvectors of the affinity matrix.
- Spherical Hashing (SpH) [42]: SpH first uses a hypersphere based hashing function to map spatially coherent data points into a binary code, and then tailors the hypersphere based binary coding scheme by designing a binary code distance function, namely, spherical Hamming distance.

Among the competing methods, LSH is the only data-dependent hashing method. All the other methods, including the proposed LVE, SELVE and E-SELVE, belong to data-dependent hashing methods.

C. Parameter setting

In the proposed SELVE and E-SELVE, we set the parameters $k = 300$, $l = 300$ and $s = 10$ on the GIST dataset, and set $k = 300$, $l = 300$ and $s = 4$ on other three datasets. The parameter λ in Eq.6 is set via performing line search over $\lambda = [0.01, 0.1, 1, 10, 100]$. For the ESE step in E-SELVE, the parameter γ is also set by line search over $\gamma = [0.01, 0.1, 1, 10, 100]$. In the experiments, we vary the length of hash codes (i.e., number of hash bits) as $[12, 16, 24, 28, 32, 48, 64]$. The sensitivity of the proposed method to these parameters will be discussed in Section III-E.

For all the other comparison approaches, the MATLAB codes are provided by the authors. We tune the parameters according to the corresponding papers.

D. Results

The hashing results of all competing approaches are shown in Figs.4-6 and Table II. From Figs.4-6, we can have the

TABLE II
CPU TIME (IN SECONDS) FOR ALL HASHING APPROACHES BY FIXING CODE LENGTH AS 28.

Methods	MNIST		CIFAR		NUS-WIDE		GIST	
	training	test	training	test	training	test	training	test
LSH	**1.871**	2.7×10^{-5}	**1.056**	6.7×10^{-7}	**4.321**	2.5×10^{-5}	**25.37**	5.1×10^{-4}
LSI	43.74	$\mathbf{8.0\times10^{-6}}$	90.41	$\mathbf{4.2\times10^{-7}}$	120.4	$\mathbf{5.7\times10^{-6}}$	801.6	$\mathbf{1.0\times10^{-5}}$
SH	9.012	7.9×10^{-5}	5.306	6.6×10^{-6}	19.57	9.9×10^{-5}	97.56	8.7×10^{-4}
ITQ	17.37	2.5×10^{-5}	16.89	1.3×10^{-6}	37.80	2.4×10^{-5}	228.1	4.5×10^{-4}
AGH	24.75	5.8×10^{-5}	11.61	1.8×10^{-6}	31.51	7.1×10^{-5}	167.9	2.1×10^{-4}
MDSH	10.77	1.5×10^{-5}	4.311	6.2×10^{-6}	13.72	1.3×10^{-5}	88.67	1.1×10^{-4}
SpH	16.84	8.1×10^{-5}	12.09	8.0×10^{-6}	17.22	7.3×10^{-5}	91.50	7.2×10^{-4}
LVE	23.99	3.6×10^{-5}	8.172	1.0×10^{-6}	25.85	6.7×10^{-5}	143.5	2.0×10^{-4}
SELVE	24.87	5.4×10^{-5}	10.71	1.1×10^{-6}	30.20	9.1×10^{-5}	145.7	2.1×10^{-4}
E-SELVE	30.23	6.7×10^{-5}	15.93	2.6×10^{-6}	35.73	9.6×10^{-5}	207.9	7.3×10^{-4}

following findings. First, the proposed E-SELVE performs the best, while the proposed SELVE is very competitive with other approaches. More specifically, E-SELVE outperforms the other approaches, followed by SELVE, SpH, AGH, MDSH, ITQ, SH, LVE, LSI and LSH. It is not surprising that LSH performs the worst since it is a data-independent hashing method by randomly generating hash functions. Both SH and ITQ are PCA-like hashing methods but they solve the problem of unbalanced variances along different directions in different ways. ITQ employs an orthogonal transformation, while SH employs a separable Laplacian eigenfunction with uniform distribution, which is a too strong assumption for real data. As a result, ITQ outperforms SH in general, as shown in Fig. 6(d) on the large dataset GIST.

The proposed SELVE method is basically a manifold-like hashing method, while E-SELVE further employs a PCA-like transformation before performing sparse embedding. Compared with the state-of-the-art manifold-like hashing method AGH, in most cases SELVE performs better, though AGH employs a more complex binarization process, i.e., hierarchical method. In addition, SELVE achieves its best performance with shorter binary code than AGH. These observations demonstrate that the proposed LVE process is very useful. From the experimental results, we can see that using LVE alone, the hashing performance is better than LSI and LSH but worse than other state-of-the-art methods such as SH, SpH, MDSH and AGH. With the SE step, the SELVE method outperforms all the competing methods. This indicates that the SE process is very important in our hashing framework. Compared with SELVE, E-SELVE performs consistently better. This demonstrates that the learned transform can make the SE process more effective. According to our experimental results, we can conclude that the SE process is the most important component in our E-SELVE framework.

From Fig.4, we see that the precision decreases when the recall increases, and vice versa. This is because the value of precision is sensitive to true positive rate while that of recall is sensitive to false positive rate. E-SELVE has much better precision-recall curves than other methods (i.e., its precision decreases much more slowly with the increase of recall than others).

From Fig.5, we see that all hashing methods first achieve their best HAM2 performance (i.e., the peak), and then drop. This is mainly because a longer binary code may lead to less retrieved results given the fixed Hamming distance threshold. Such a phenomenon has also been discussed in [39], [60]. We can also see from Fig.5 that all competing approaches, except for AGH, achieve their best HAM2 performance when the number of hash bits is between 24 and 32. For example, the proposed SELVE and E-SELVE obtain their best HAM2 performance on all the four datasets with hash code of 28 bits. SpH, MDSH, ITQ, SH, LSI and LSH obtain their best HAM2 performance on all the four datasets with code length between 24 and 32. However, AGH obtains its best HAM2 performance when the code length is between 48 and 64. This shows that although AGH achieves the third best performance among all competing hashing algorithms, it needs to represent each sample with longer binary codes, which leads to more data storage cost and longer query time. Overall, from Fig.5 one can conclude that the proposed hashing framework achieves the best HAM2 results with short binary codes.

Finally, from Fig.6 we see that the MAP or MP results of all methods will increase with the increase of hash bits, while the proposed E-SELVE achieves the best MAP or MP results in almost all cases on the four databases. The results of SELVE are also very competitive.

Table II lists the training time and test time of all methods. In terms of training time, LSI is the most time consuming one since it needs to conduct SVD on all the training samples. LSH is the fastest one, while ITQ, SpH, MDSH, AGH, SH, SELVE and E-SELVE have similar training time. In terms of test time on each test sample, LSI is the fastest one, while all the other methods are in the same order. This is consistent to the complexity analysis in Table I. Overall, the proposed E-SELVE has comparable running time to state-of-the-arts but with better accuracy.

E. Sensitivity to parameters

We then test the proposed methods with different parameter settings. More specifically, we first report the results of SELVE and E-SELVE in terms of HAM2 by varying the number of

centroids (i.e., setting k = [100, 200, 300, 400, 500, 600]). Then we test the HAM2 performance of E-SELVE via setting different dimensionality (i.e., setting l = [25, 50, 100, 200, 300, 600]) in the E-SE process. Finally, we test the HAM2 performance of E-SELVE via setting different values on λ and γ (i.e., $\lambda = [0.01, 0.1, 1, 10, 100]$ and $\gamma = [0.01, 0.1, 1, 10, 100]$). The results are reported in Fig.7, Fig.8, and Fig.9, respectively.

From Fig.7, we see that increasing k can lead to better results of SELVE and E-SELVE. This is reasonable because using more clusters can more accurately describe the data distribution of the training samples. Nonetheless, using too many clusters will need much more computational cost. According to our experience, setting $k \leq \sqrt{n}$ can make a good trade-off between the hashing accuracy and cost. From Fig.8, we see that even with a small l, E-SELVE still obtains good performance. This validates the effectiveness and advantage of the E-SE step in the E-SELVE approach. From Fig. 9, one can clearly see that the proposed method is not sensitive to the setting λ in Eq.6 and γ in Eq.15. In addition, we also found the proposed method is not also sensitive to the setting of the s. By varying s from 2 to 12, the hashing performance of E-SELVE does not vary much. We do not report the detailed results here to save space.

F. The role of the centralized term in Eq.6

Finally, we validate the role of the centralized term $\lambda \sum_{i=1}^{n} \|\boldsymbol{\alpha}_i - \boldsymbol{\mu}\|_2^2$ in Eq.6. This term is important in the proposed framework because it is designed to simultaneously generate hash functions and the binarization threshold. It can also reduce the impact of outliers. In Fig.10, we test E-SELVE with (i.e., let $\lambda = 1$ in Eq.6) and without (i.e., let $\lambda = 0$ in Eq.6) the centralized term. It can be seen that E-SELVE with the centralized term always outperforms E-SELVE without the centralized term.

IV. Conclusion

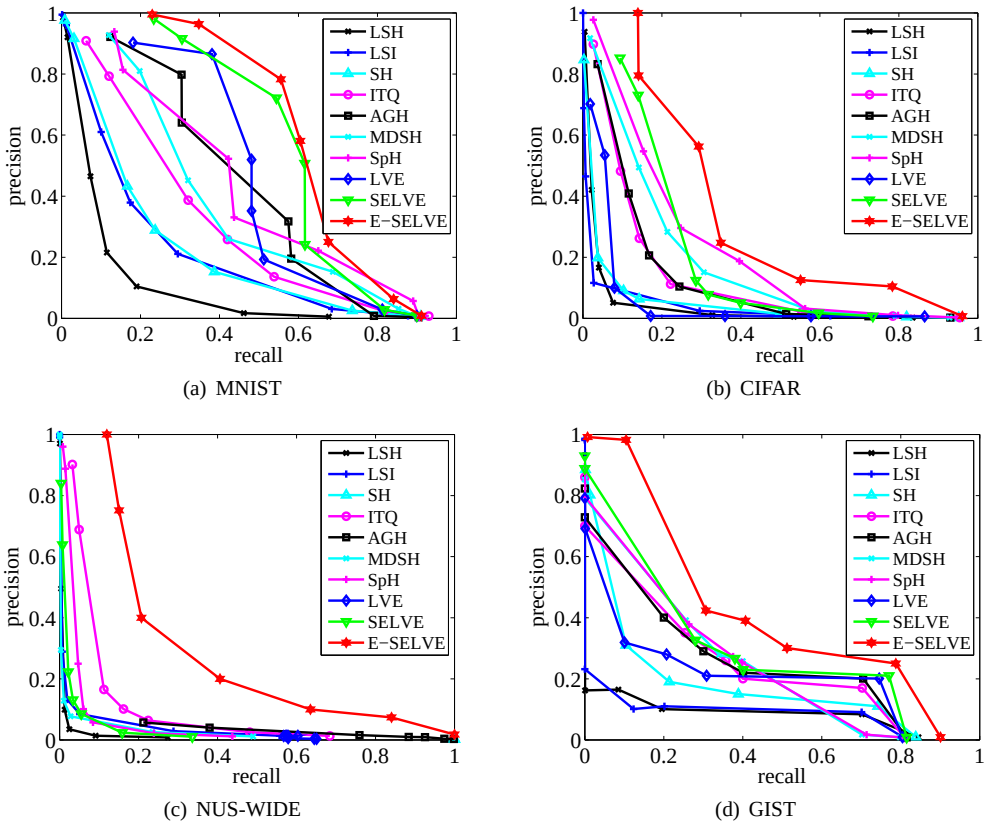
We proposed an effective sparse embedding and least variance encoding (SELVE) approach to hashing, aiming at high hashing accuracy with short binary codes. We first partitioned the whole training dataset into clusters and then represented each training sample by the normalized probabilities that it falls into the several closest clusters. Such a spatially sparse embedding process leads to advantages such as sparse representation, similarity preservation and linear time complexity. We then encoded the sparse embedding features of training samples over a dictionary to learn the explicit hash functions and the binarization threshold jointly, which makes the whole hashing process more accurate. In addition, by learning a transformation matrix to make the sample space more concentrated, an enhanced sparse embedding was developed to further improve the performance of SELVE. The experimental results on four benchmark datasets validated that SELVE and its enhanced counterpart, i.e., E-SELVE, achieve very promising hashing performance.

V. Acknowledgement

This research is supported by the HK PolyU research fund under grant No. G-YK79. Zi Huang is supported by the Australia Research Council (ARC) under research Grant ARC FT130101530. Xiaofeng Zhu is supported by the National Natural Science Foundation of China under grant 61263035 and the funding of Guangxi 100 Plan.

References

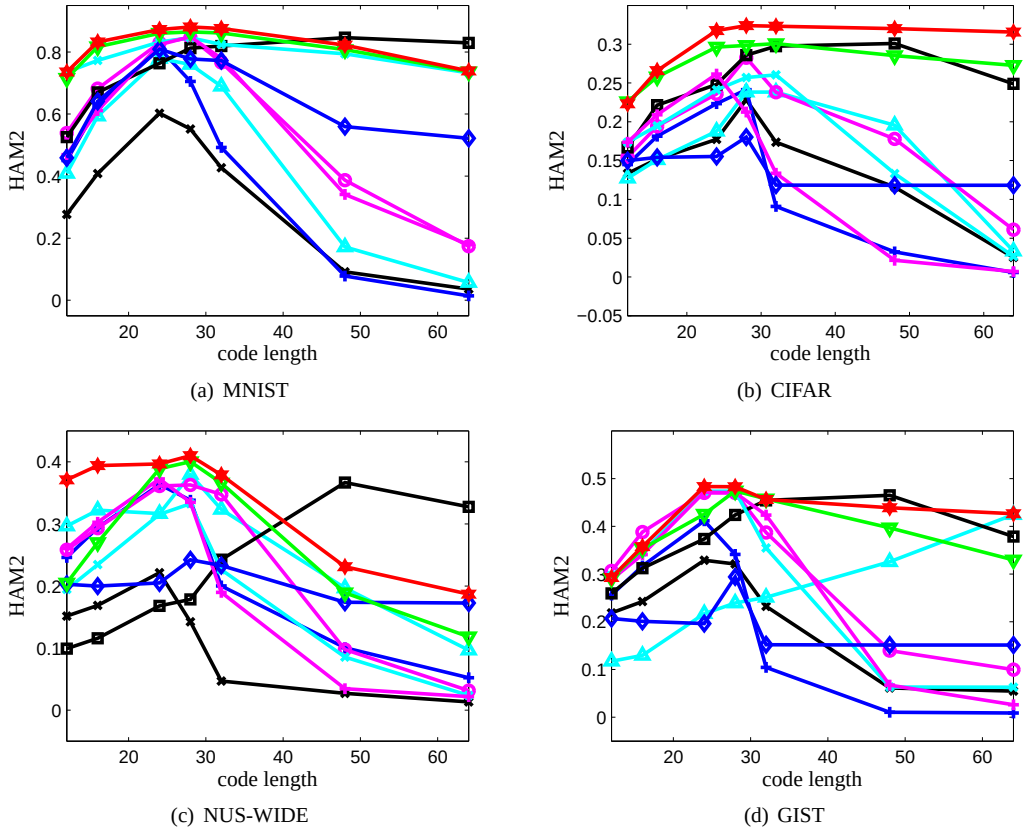
[1] W. Bian and D. Tao, "Biased discriminant euclidean embedding for content-based image retrieval," *IEEE Trans. Image Process*, vol. 19, no. 2, pp. 545–554, 2010.
[2] S. Bondugula, "Survey of hashing techniques for compact bit representations of images."
[3] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Comput. Surv.*, vol. 40, no. 2, 2008.
[4] J. He, S. Kumar, and S.-F. Chang, "On the difficulty of nearest neighbor search," in *ICML*, 2012.
[5] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-neighbor methods in learning and vision: theory and practice*. MIT press Cambridge, MA, USA:, 2005, vol. 3.
[6] L. Hong, Y. Qingwei, and Z. Tingkai, "Spectral clustering algorithm based on k-nearest neighbor measure," in *Information Science and Engineering (ICISE), 2010 2nd International Conference on*, 2010, pp. 5399–5402.
[7] J. Orava, "K-nearest neighbour kernel density estimation, the choice of optimal k," *Tatra Mount. Math. Pub.*, vol. 50, no. 1, p. 3950, 2012.
[8] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2393–2406, 2012.
[9] E. Elhamifar and R. Vidal, "Sparse manifold clustering and embedding," in *NIPS*, 2011, pp. 55–63.
[10] J. Li, N. M. Allinson, D. Tao, and X. Li, "Multitraining support vector machine for image retrieval," *IEEE Trans. Image Process*, vol. 15, no. 11, pp. 3597–3601, 2006.
[11] D. Xu, S. Yan, D. Tao, S. Lin, and H.-J. Zhang, "Marginal fisher analysis and its variants for human gait recognition and content- based image retrieval," *IEEE Trans. Image Process*, vol. 16, no. 11, pp. 2811–2821, 2007.
[12] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *CVPR*, 2008, pp. 1–8.
[13] S. M. Omohundro, "Efficient algorithms with neural network behavior," *Comp. Sys.*, vol. 1, no. 2, pp. 273–347, 1987.
[14] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Inf. Process. Lett.*, vol. 40, no. 4, pp. 175–179, 1991.
[15] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *SODA*, 1993, pp. 311–321.
[16] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISAPP*, 2009, pp. 331–340.
[17] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 9, pp. 1704–1716, 2012.
[18] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang, "Compact hyperplane hashing with bilinear functions," in *ICML*, 2012.
[19] J. He, S.-F. Chang, R. Radhakrishnan, and C. Bauer, "Compact hashing with joint optimization of search accuracy and time," in *CVPR*, 2011, pp. 753–760.
[20] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *SIGIR*, 2010, pp. 18–25.
[21] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.
[22] X. Zhu, Z. Huang, H. T. Shen, and X. Zhao, "Linear cross-modal hashing for efficient multimedia search," in *ACM MM*, 2013, pp. 143–152.
[23] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *SOCG*, 2004, pp. 253–262.
[24] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB*, 1999, pp. 518–529.
[25] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *ICCV*, 2009, pp. 2130–2137.



(a) MNIST (b) CIFAR

(c) NUS-WIDE (d) GIST

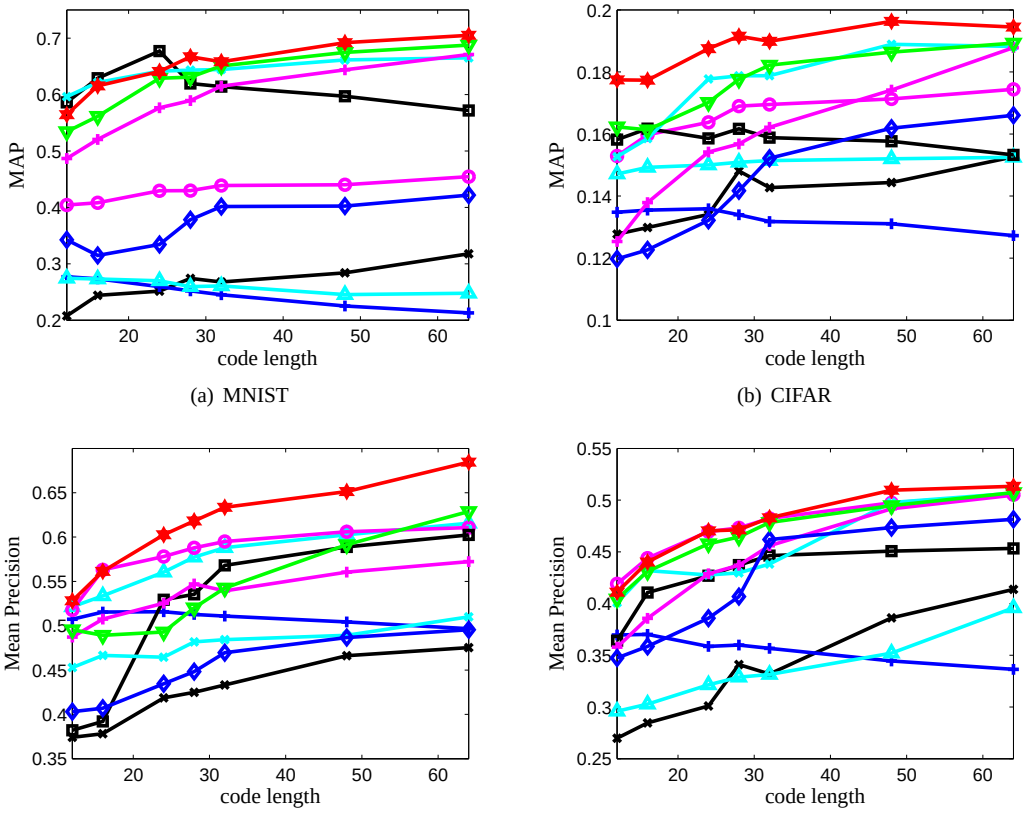
Fig. 4. Precision-Recall curves of all approaches.



(a) MNIST (b) CIFAR

(c) NUS-WIDE (d) GIST

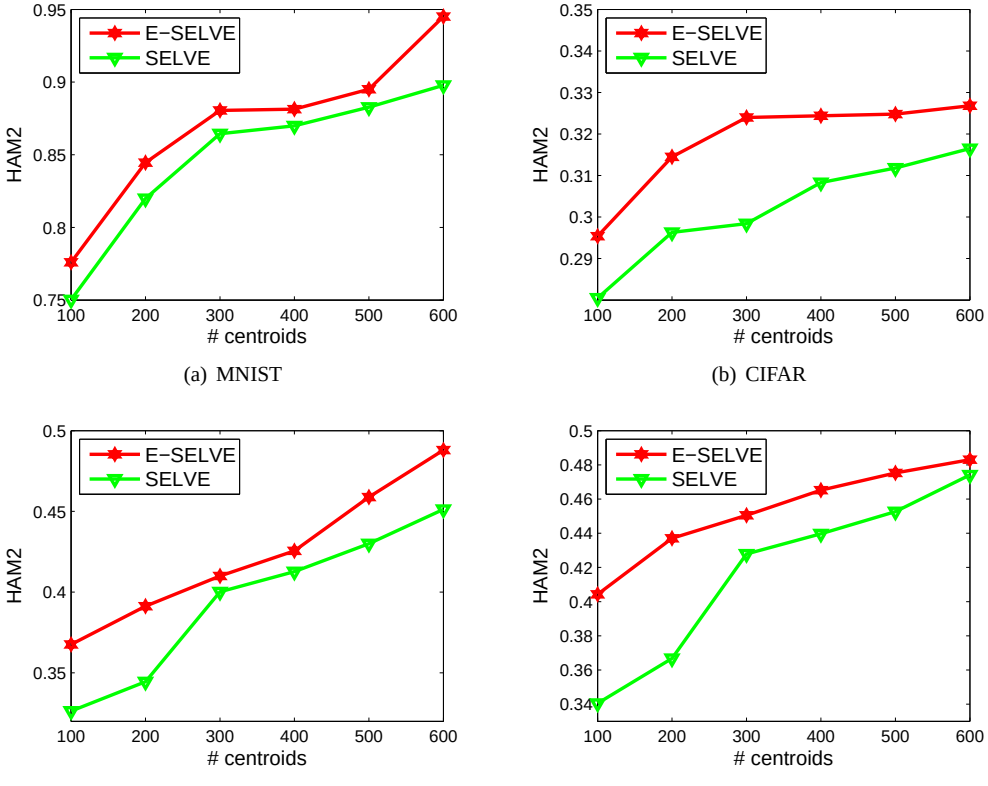
Fig. 5. HAM2 of all approaches. The legends are the same as Fig.4.



(a) MNIST (b) CIFAR

(c) NUS-WIDE (d) GIST

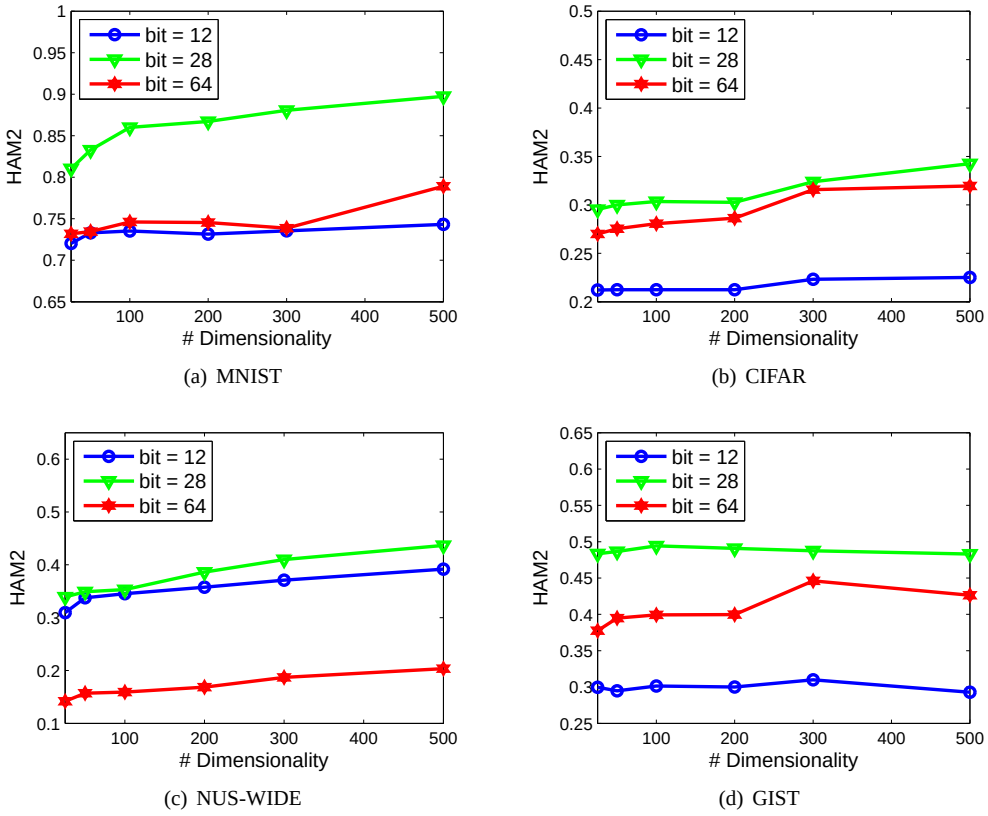
Fig. 6. MAP of all approaches. The legends are the same as Fig.4.



(a) MNIST (b) CIFAR

(c) NUS-WIDE (d) GIST

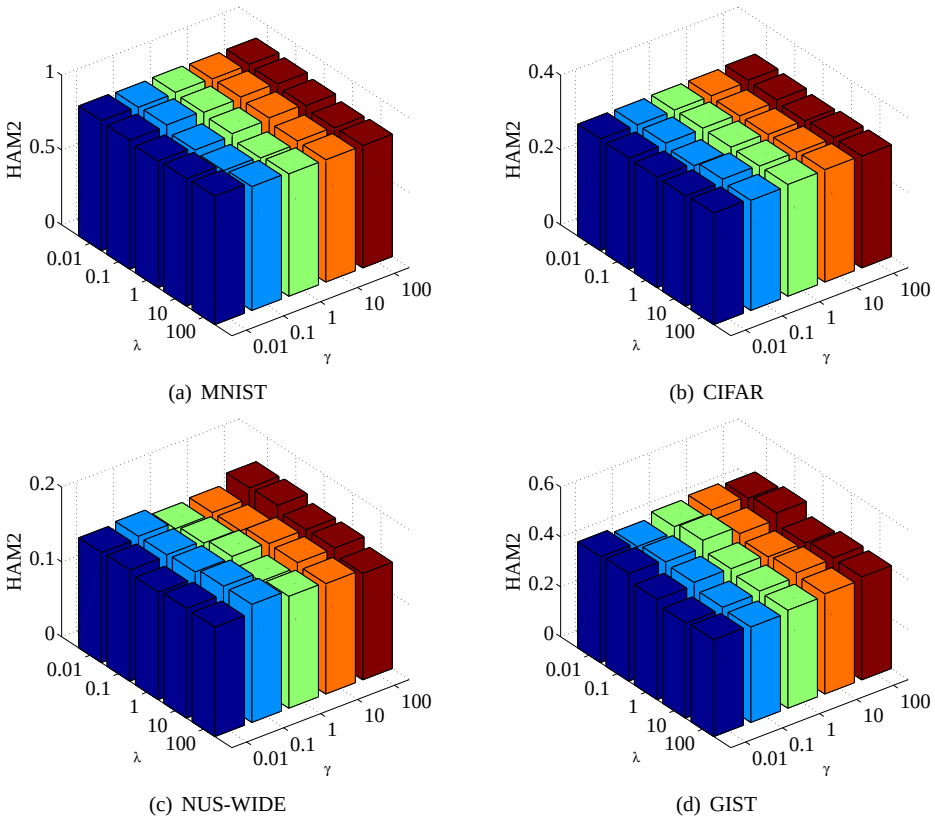
Fig. 7. HAM2 of the proposed E-SELVE and SELVE under different number of centroids by fixing $l = 300$ and $c = 28$.



(a) MNIST (b) CIFAR

(c) NUS-WIDE (d) GIST

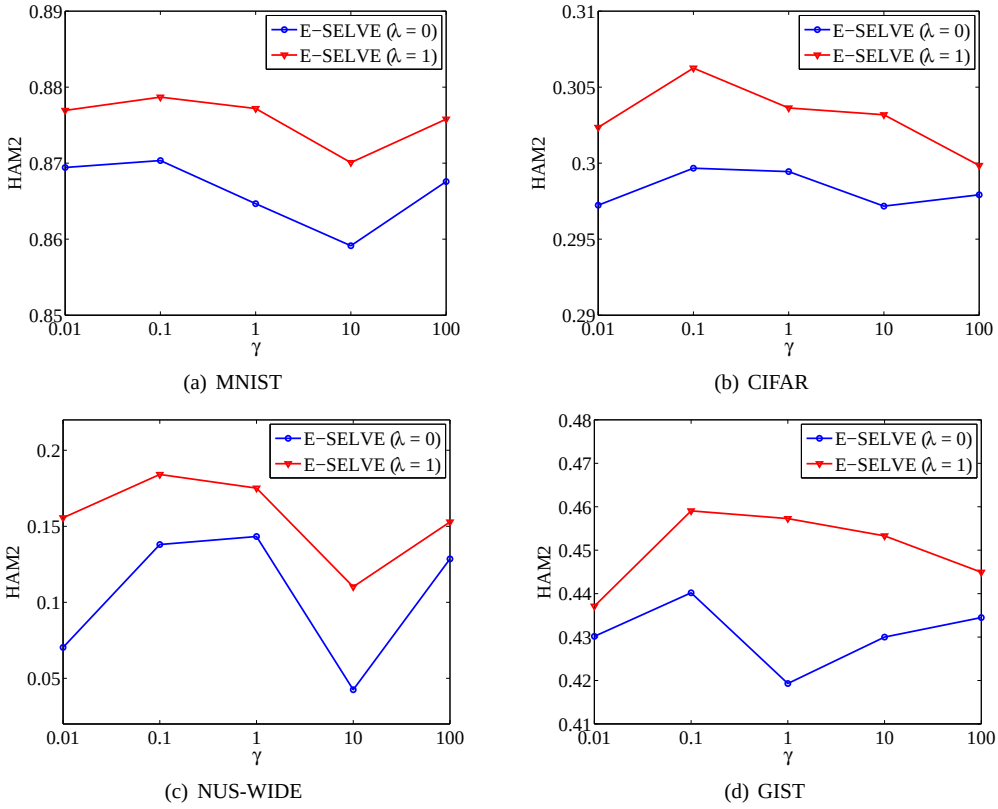
Fig. 8. HAM2 of the proposed E-SELVE approach under different dimensions of the transformed space by $\mathbf{W}$ (c = 28 and k = 300).



(a) MNIST (b) CIFAR

(c) NUS-WIDE (d) GIST

Fig. 9. HAM2 of the proposed E-SELVE with different values of λ and γ (c = 28 and k = 300).



(a) MNIST (b) CIFAR

(c) NUS-WIDE (d) GIST

Fig. 10. The HAM2 performance of the proposed E-SELVE with (i.e, $\lambda = 1$) and without (i.e, $\lambda = 0$) the centralized term in Eq.6. We vary γ from 0.01 to 100 and fix $c = 28$ and $k = 300$.

[26] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *NIPS*, 2009, pp. 1509–1517.

[27] M. Norouzi and D. J. Fleet, "Minimal loss hashing for compact binary codes," in *ICML*, 2011, pp. 353–360.

[28] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *CVPR*, 2012, pp. 3108–3115.

[29] J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *ICML*, 2010, pp. 1127–1134.

[30] X. Zhu, Z. Huang, H. Cheng, J. Cui, and H. T. Shen, "Sparse hashing for fast multimedia search," *ACM Trans. Inf. Syst.*, vol. 31, no. 2, p. 9, 2013.

[31] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *CVPR*, 2011, pp. 817–824.

[32] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," in *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012, p. accepted.

[33] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*, 2008, pp. 1753–1760.

[34] Y. Weiss, R. Fergus, and A. Torralba, "Multidimensional spectral hashing," in *ECCV*, 2012, pp. 340–353.

[35] J. He, W. Liu, and S.-F. Chang, "Scalable similarity search with optimized kernel hashing," in *KDD*, 2010, pp. 1129–1138.

[36] Y. Liu, F. Wu, Y. Yang, Y. Zhuang, and A. G. Hauptmann, "Spline regression hashing for fast image search," *IEEE Trans. Image Process*, vol. 21, no. 10, pp. 4480–4491, 2012.

[37] D. Zhang, F. Wang, and L. Si, "Composite hashing with multiple information sources," in *SIGIR*, 2011, pp. 225–234.

[38] W. Kong and W.-J. Li, "Isotropic hashing," in *NIPS*, 2012, pp. 1655–1663.

[39] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *ICML*, 2011, pp. 1–8.

[40] X. Chen and D. Cai, "Large scale spectral clustering with landmark-based representation," in *AAAI*, 2011, pp. 313–318.

[41] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," *Int. J. Approx. Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.

[42] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *CVPR*, 2012, pp. 2957–2964.

[43] J. Goldberger, S. T. Roweis, G. E. Hinton, and R. Salakhutdinov, "Neighbourhood components analysis," in *NIPS*, 2004.

[44] C. Wu, J. Zhu, D. Cai, C. Chen, and J. Bu, "Semi-supervised nonlinear hashing using bootstrap sequential projection learning," *IEEE Trans. Knowl. Data Eng.*, vol. PP, no. 99, p. 1, 2012.

[45] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *J. Mach. Learn. RES.*, vol. 11, pp. 19–60, 2010.

[46] M. Yang, L. Zhang, J. Yang, and D. Zhang, "Metaface learning for sparse representation based face recognition," in *ICIP*, 2010, pp. 1601–1604.

[47] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *NIPS*, 2005, pp. 1–8.

[48] S.-J. Wang, H.-L. Chen, X.-J. Peng, and C.-G. Zhou, "Exponential locality preserving projections for small sample size problem," *Neurocomputing*, vol. 74, no. 17, pp. 3654–3662, 2011.

[49] S.-J. Wang, S. Yan, J. Yang, C.-G. Zhou, and X. Fu, "A general exponential framework for dimensionality reduction," 2014.

[50] X. Zhu, Z. Huang, Y. Yang, H. T. Shen, C. Xu, and J. Luo, "Self-taught dimensionality reduction on the high-dimensional small-sized data," *Pattern Recognition*, vol. 46, no. 1, pp. 215–229, 2013.

[51] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, 2008.

[52] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "Nus-wide: a real-world web image database from national university of singapore," in *CIVR*, 2009, pp. 48–56.

[53] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *CVPR*, vol. 33, no. 1, pp. 117–128, 2011.

[54] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. T. J. Comput. Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[55] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *JASIS*, vol. 41, no. 6, pp. 391–407, 1990.

[56] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent

semantic indexing: A probabilistic analysis," *J. Comput. Syst. Sci.*, vol. 61, no. 2, pp. 217–235, 2000.
[57] G. H. Golub and C. F. van Loan, *Matrix computations (3. ed.)*. Johns Hopkins University Press, 1996.
[58] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge University Press, 1990.
[59] J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *ICML*, 2010, pp. 1127–1134.
[60] Y. Zhen and D.-Y. Yeung, "Co-regularized hashing for multimodal data," in *NIPS*, 2012, pp. 2559–2567.

Xiaofeng Zhu received the B.S. degree in Mathematics rom Guangxi Normal University, Guangxi, P.R. China, the M.Sc. degree in Computer Science from The University of Queensland, Australia. His research interests include data mining, machine learning, and pattern recognition, especially focusing on feature selection, medical image classification, hashing in large scale datasets and sparse learning.

Lei Zhang (M' 04) received the B.Sc. degree in 1995 from Shenyang Institute of Aeronautical Engineering, Shenyang, P.R. China, the M.Sc. and Ph.D degrees in Control Theory and Engineering from Northwestern Polytechnical University, Xi'an, P.R. China, respectively in 1998 and 2001. From 2001 to 2002, he was a research associate in the Dept. of Computing, The Hong Kong Polytechnic University. From Jan. 2003 to Jan. 2006 he worked as a Postdoctoral Fellow in the Dept. of Electrical and Computer Engineering, McMaster University, Canada. In 2006, he joined the Dept. of Computing, The Hong Kong Polytechnic University, as an Assistant Professor. Since Sept. 2010, he has been an Associate Professor in the same department. His research interests include Image and Video Processing, Computer Vision, Pattern Recognition and Biometrics, etc. Dr. Zhang has published about 200 papers in those areas. Dr. Zhang is currently an Associate Editor of IEEE Trans. on CSVT and Image and Vision Computing. He was awarded the 2012-13 Faculty Award in Research and Scholarly Activities. More information can be found in his homepage http://www4.comp.polyu.edu.hk/~cslzhang/.

Dr Zi Huang is an ARC Future Fellow in the School of Information Technology & Electrical Engineering at The University of Queensland. She received her BSc degree from Tsinghua University, China, and her PhD in Computer Science from The University of Queensland. Dr Huang's research interests include multimedia search, social media analysis, database and information retrieval. Most of her publications have been published in leading conferences and journals including ACM Multimedia, ACM SIGMOD, IEEE ICDE, IEEE TMM, IEEE TKDE, ACM TOIS, ACM Computing Surveys, etc.