Cross-view Retrieval via Probability-based Semantics-Preserving Hashing

Zijia Lin, *Student Member, IEEE,* Guiguang Ding, *Member, IEEE,* Jungong Han, Jianmin Wang

***Abstract*—For efficiently retrieving nearest neighbours from large-scale multi-view data, recently hashing methods are widely investigated, which can substantially improve query speeds. In this paper, we propose an effective probability-based Semantics-Preserving Hashing method to tackle the problem of cross-view retrieval, termed SePH. Considering the semantic consistency between views, SePH generates one unified hash code for all observed views of any instance. For training, SePH firstly transforms the given semantic affinities of training data into a probability distribution, and aims to approximate it with another one in Hamming space, via minimizing their Kullback-Leibler divergence. Specifically, the latter probability distribution is derived from all pair-wise Hamming distances between to-be-learnt hash codes of the training data. Then with learnt hash codes, any kind of predictive models like linear ridge regression, logistic regression or kernel logistic regression, can be learnt as hash functions in each view for projecting the corresponding view-specific features into hash codes. As for out-of-sample extension, given any unseen instance, the learnt hash functions in its observed views can predict view-specific hash codes. Then by deriving or estimating the corresponding output probabilities *w.r.t* the predicted view-specific hash codes, a novel probabilistic approach is further proposed to utilize them for determining a unified hash code. To evaluate the proposed SePH, we conduct extensive experiments on diverse benchmark datasets, and the experimental results demonstrate that SePH is reasonable and effective.**

***Index Terms*—Semantics-Preserving Hashing, SePH, Cross-view retrieval, Approximate nearest neighbour retrieval**

I. Introduction

FOR numerous algorithms in the fields of cybernetics, computer vision and machine learning, *etc.*, retrieving nearest neighbours for an instance plays a fundamental role, as also revealed in [1] and [2]. However, with the explosion of data in recent years, efficient nearest neighbour retrieval from large-scale and rapidly-increasing databases becomes quite challenging. For tackling that, various tree-based indexing methods [3]–[6] and hashing methods [1], [2], [7]–[39] are proposed to perform exact or approximate nearest neighbour (ANN) retrieval with much higher speeds. As tree-based indexing methods can suffer from the so-called "curse of dimensionality" for high-dimensional data, recently hashing methods are becoming preferred and widely researched for handling such data. Generally, for hashing methods, by generating a k-bit binary (*i.e.* 0 or 1) hash code for each instance, we can store the data compactly in hardware bits. Meanwhile, to perform ANN retrieval, the Hamming distances between the query hash code and those in the retrieval set can be efficiently calculated using fast bit-wise XOR and bit-count operations [1] with a sub-linear time complexity. And with all Hamming distances calculated, generally only the small ones are kept and then ranked in an ascending order to select instances with smallest Hamming distances as the approximate nearest neighbours. Therefore, if the binary hash codes can well preserve the affinities between instances, hashing methods can perform ANN retrieval with much lower storage costs and higher query speeds [18], while the quality loss of the retrieved neighbours would be acceptable.

Generally speaking, we can roughly classify existing hashing methods into single-view hashing [1], [2], [7]–[19] and multi-view hashing [20]–[39]. The former focuses on data with a single view, while the latter focuses on that with multiple views, like an object with pictures from different cameras or a news report with texts and images. Our work in this paper is about cross-view retrieval for multi-view data. Specifically, cross-view retrieval can utilize just one view of a query to retrieval its nearest neighbours in other different views, like using a query picture from one camera to retrieve relevant ones from other cameras, or using a textual query to retrieve semantically relevant images. Since cross-view retrieval can be utilized in many applications, it is becoming more and more popular, as also revealed in [35].

In recent years, researchers have proposed many effective hashing methods for cross-view retrieval, ranging from unsupervised ones [20]–[25] to supervised ones [26]–[39]. The former ones generally utilize only the features of training data in different views to exploit intra-view and inter-view correlations for learning hash functions, which project features into binary hash codes. Meanwhile, the latter ones can further exploit other available supervised information like semantic affinities of training data, to better learn the projections and yield superior performance. Actually, for supervised ones, well preserving the semantic affinities between instances is the key to reducing the quality loss of retrieved nearest neighbours, which is also the focus of our research.

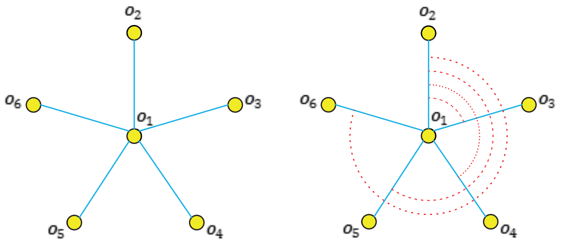
In this paper, we propose a probability-based **Se**mantics-**P**reserving **H**ashing method for cross-view retrieval, termed SePH. The proposed SePH belongs to supervised hashing.

Zijia Lin is with Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: linzijia07@tsinghua.org.cn).

Guiguang Ding and Jianmin Wang are with School of Software, Tsinghua University, Beijing 100084, China (e-mail: {dinggg, jimwang}@tsinghua.edu.cn). Guiguang Ding is the corresponding author.

Jungong Han is with Civolution Technology, Eindhoven 5656AE, The Netherlands (e-mail: jungonghan77@gmail.com).

[1] Both bit-wise XOR and bit-count operations are generally supported or implemented by hardware.



(a) Previous Work (b) Proposed SePH

Fig. 1. Illustration of the differences between previous work (left) and the proposed SePH (right). Here $o_i (i = 1, 2, \ldots, 6)$ are to-be-learnt hash codes, blue solid edges between them denote their pairwise distances (similarities), and red dotted lines between edges denote the correlations between distances (similarities). Note that for clarity, some edges/lines are omitted.

Moreover, considering the semantic consistency between observed views, SePH generates one unified hash code for all observed views of any instance, like [23] and [24]. For training, SePH firstly transforms the given semantic affinities of training instances into a probability distribution $\mathcal{P}$ and aims to approximate it in Hamming space. Specifically, SePH transforms all pairwise Hamming distances between to-be-learnt hash codes of the training instances into another probability distribution $\mathcal{Q}$, and then minimizes its Kullback-Leibler divergence (KL-divergence) from $\mathcal{P}$. In previous work [27], [34], [35], the supervised information, *i.e.* the semantic affinities of training instances, is generally utilized to *independently* weight each pairwise distance (similarity) between hash codes. Differently, SePH standardizes all pairwise Hamming distances into a global probability distribution by transforming each into a probability and thus makes them *dependent* on each others. In that way, apart from weighting pair-wise distances (similarities) between hash codes as previous work, SePH can further incorporate the correlations between distances (similarities) to force the to-be-learnt hash codes of training instances to better preserve the semantic affinities, as illustrated in Fig. 1, which shows the differences between previous work and SePH in a vivid way. After learning the hash codes of training instances, SePH further learns hash functions independently in each view for projecting the corresponding features into binary hash codes, which can be open for any kind of effective predictive models. Specifically, in this paper, we respectively utilize linear ridge regression, logistic regression and kernel logistic regression as hash functions. As for out-of-sample extension, given an unseen instance, the learnt hash functions in each of its observed views can predict view-specific hash codes. Then by deriving or estimating the corresponding output probabilities *w.r.t* the predicted view-specific hash codes, a novel probabilistic approach is further proposed to utilize them for determining a unified hash code. Similar to [9], here SePH employs a two-step hashing framework. The reason why SePH adopts a two-step framework is two-fold. First and most important, utilizing a two-step framework can make SePH more flexible and enable it to use any kind of effective predictive models as hash functions. Second, utilizing a two-step framework can simplify the optimization process, since directly learning hash functions in a one-step manner can probably make the objective function quite complex and even unable to be optimized. The reasonableness and effectiveness of SePH is well demonstrated by comprehensive experiments on diverse benchmark datasets.

We summarize the contributions of this paper as follows.

- We propose a probability-based **Se**mantics-**P**reserving **H**ashing method for cross-view retrieval, which approximates a probability distribution derived from given semantic affinities of training data with another one derived from the to-be-learnt hash codes in Hamming space via minimizing their KL-divergence.
- We propose a novel probabilistic approach to determine a unified hash code for any given unseen instance, utilizing its predicted view-specific hash codes from different observed views and the corresponding derived or estimated output probabilities.

This paper is based on our previous work presented in [40], but it substantially extends that work. Specifically, apart from non-linear kernel logistic regression, here we also utilize linear ridge regression and logistic regression as hash functions, so as to show that the learning of hash functions in SePH can be open for different predictive models. Actually, the experiments with linear ridge regression and logistic regression also well demonstrate the effectiveness of SePH. Particularly, for hash functions like linear ridge regression that cannot naturally provide output probabilities with the predicted view-specific hash codes, here we further propose an effective and general method to estimate the output probabilities, which are required for determining unified hash codes. Moreover, experiments are conducted on all benchmark datasets to validate the effectiveness of the proposed probabilistic approach for determining the unified hash code of an unseen instance. We also analyse the convergence of the optimization process for SePH with experiments, and report its off-line training costs and on-line hashing costs on all datasets. Additionally, more details of the experimental results, like standard errors, are also presented here. Detailed derivations for the gradient of the objective function of SePH are also provided in the supplementary material due to the limited space.

We organize the remainder of this paper as follows. Section II gives an overview of previous researches on cross-view hashing. Section III presents formula details of the proposed SePH, including off-line training and on-line hashing. Then experiments are described in Section IV, including settings, results and analyses. And finally we come to conclusions in Section V.

II. Related Work

As mentioned previously, researchers have proposed many effective unsupervised and supervised cross-view hashing methods in recent years.

Unsupervised cross-view hashing methods [20]–[25] generally utilize only the features of training data in different views to exploit intra-view and inter-view correlations for learning hash functions to project features into binary hash codes. Song *et al.* [21] proposed inter-media hashing (IMH), which learns linear hash functions with intra-view and inter-view consistencies to map view-specific features into a common Hamming space. Zhen *et al.* [22] proposed Spectral Multimodal Hashing (SMH) based on spectral analysis of the correlation matrix of different views and developed an efficient algorithm to learn parameters from the data distribution so as to obtain binary hash codes. Ding *et al.* [23] proposed Collective Matrix Factorization Hashing (CMFH) that performs collective matrix factorization in different views with latent factor model to learn unified hash codes for training instances. Zhou *et al.* [24] proposed Latent Semantic Sparse Hashing (LSSH), which respectively utilizes sparse coding for images and matrix factorization for texts to learn their latent semantic features and eventually maps the learnt features to a joint abstraction space to generate unified hash codes. Xie *et al.* [25] proposed Online Cross-modal Hashing (OCMH), which performs efficient updating of hash codes and analysis of cross-modal correlations for online hashing by learning shared latent codes.

Differently, supervised cross-view hashing methods [26]–[39] can further exploit available supervised information like semantic labels or semantic affinities of training data for gaining further performance improvements. Bronstein *et al.* [26] proposed CMSSH that models the projections from features in each view to hash codes as binary classification problems with positive and negative examples, and utilizes boosting methods to efficiently learn them. Kumar and Udupa [27] proposed a principled cross-view hashing method termed CVH, which is an extension of the single-view spectral hashing [8] in multi-view cases. Specifically, CVH learns hash functions to map semantically similar instances to similar hash codes across different views, via minimizing the similarity-weighted pairwise Hamming distances between the hash codes of training instances. Zhen and Yeung [28] proposed Co-Regularized Hashing (CRH) to learn hash functions for multi-view data based on a boosted co-regularization framework. In CRH, hash functions for each bit of the hash codes are learnt by solving DC (difference of convex functions) programs, while the learning for multiple bits is performed via a boosting procedure. Yu *et al.* [32] proposed Discriminative Coupled Dictionary Hashing (DCDH). Specifically, DCDH firstly learns a coupled dictionary for each view with side information like category labels to represent data from different views as the sparse codes in a shared dictionary space, and then learns unified hash functions for mapping them into binary hash codes. Zhou *et al.* [34] proposed a spectral-based hashing method termed KSH-CV, which removes the orthogonality constraints on hash code bits and learns kernel hash functions under an Adaboost framework to preserve inter-view similarities. Zhang and Li [35] proposed SCM to take semantic labels into consideration for the hash learning procedure for large-scale datasets via maximizing semantic correlations. SCM can learn orthogonal hash functions via eigenvalue decomposition (SCM-Orth) or non-orthogonal ones via sequential learning (SCM-Seq). Moreover, Jiang and Li [37] integrated feature learning and hash-code learning into an end-to-end learning framework with deep neural networks (one for each view) for cross-view hashing.

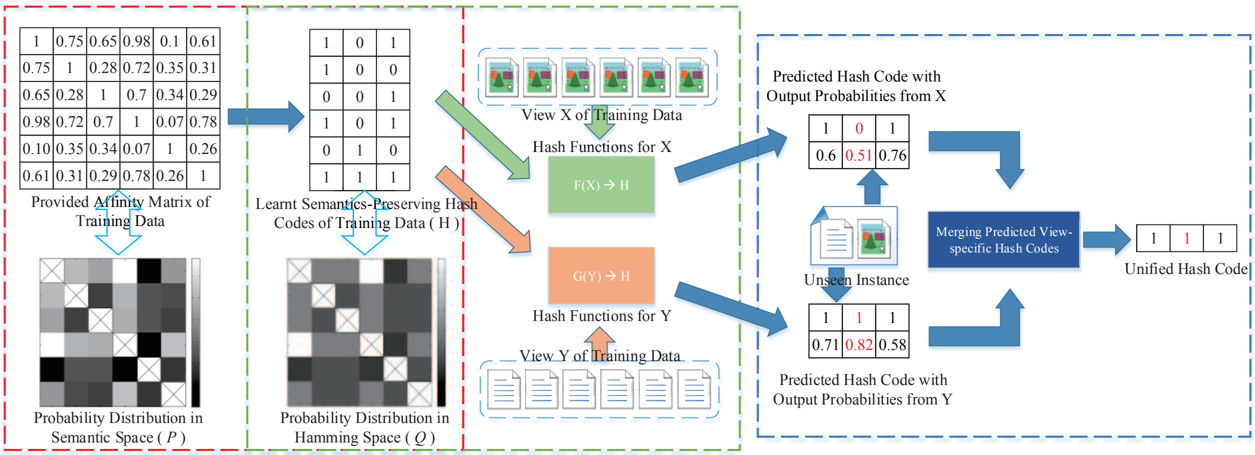
After reviewing the previous cross-view hashing methods, especially the supervised ones, we realize that well preserving the semantic affinities between instances is the key to reducing the quality loss of retrieved neighbours and achieving better performance. Generally, in supervised cases, given semantic affinities of training data, previous methods like [27], [34], [35] utilize them to independently weight each pairwise distance (similarity) between to-be-learnt hash codes. Differently, in this paper the proposed SePH further incorporates the correlations between pairwise Hamming distances to force the to-be-learnt hash codes to better preserve the semantic affinities. As will be demonstrated by our experiments, SePH is reasonable and yields superior performance.

III. Proposed SePH

Fig. 2 illustrates the framework of the proposed SePH. Like [23] and [24], considering the semantic consistency between views, SePH generates one unified hash code for each instance, rather than respectively generate one different hash code for each observed view as other previous researches [26], [27], [34], [35]. That also allows SePH to store data with less space costs. As shown in Fig. 2, for hash learning SePH requires the view-specific features of training instances in each view and an affinity matrix indicating their semantic affinities. Specifically, SePH firstly transforms the given affinity matrix into a probability distribution $\mathcal{P}$ in semantic space, and learns the semantics-preserving hash codes of training instances via utilizing their Hamming distances for deriving another probability distribution $\mathcal{Q}$ in Hamming space to approximate $\mathcal{P}$ (red dotted rectangle). Then with learnt hash codes and view-specific features of training instances, SePH learns hash functions in each view independently for projecting features into hash codes (green dotted rectangle). As for out-of-sample extension, given any unseen instance, learnt hash functions in observed views firstly predict view-specific hash codes. Then by deriving or estimating the corresponding output probabilities *w.r.t* the predicted view-specific hash codes, SePH utilizes a novel probabilistic approach to merge them and determine a unified hash code (blue dotted rectangle). For ease of presentation, here we firstly describe SePH in the case with only two views, and then extend it to cases with more views.

A. Problem Formulation

Suppose that the training data is made up of n training instances, denoted as $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ with o_i being the ith one, and we can observe two views, *i.e.* $\mathcal{X}$ and $\mathcal{Y}$, of the training instances. Moreover, SePH requires the view-specific feature matrices $X \in \mathbb{R}^{n\times d_x}$ and $Y \in \mathbb{R}^{n\times d_y}$ of the training data, which are respectively built with the d_x-dimensional feature vectors in $\mathcal{X}$ and the d_y-dimensional feature vectors in $\mathcal{Y}$ row by row. Specifically, the ith row of X, denoted as $X_{i,\cdot} \in \mathbb{R}^{d_x}$, is the feature vector of o_i in the view $\mathcal{X}$, and likewise the ith row in Y, denoted as $Y_{i,\cdot} \in \mathbb{R}^{d_y}$, is the feature vector of o_i in the view $\mathcal{Y}$. The affinity matrix of the training data, denoted as $A \in \mathbb{R}^{n\times n}$, is also required by SePH to provide supervised information. Here A is supposed to be symmetric, *i.e.* $\forall 1 \leq i,j \leq n, A_{i,j} = A_{j,i}$, where $A_{i,j} \in [0,1]$



Training of SePH: 1) Learning semantics-preserving hash codes of the training data (red dotted rectangle), via approximating the probability distribution in semantic space (P) with that in Hamming space (Q) , 2) Learning hash functions for each view (green dotted rectangle)

Out-of-sample Extension: 1) Predicting hash codes from observed views, 2) Determining the unified hash code with a proposed probabilistic approach

Fig. 2. Framework of the proposed SePH, illustrated with two-view toy data. For training, SePH firstly learns semantics-preserving hash codes of the training data and then learns hash functions for each view. For out-of-sample extension, SePH firstly predicts view-specific hash codes and derive or estimate their corresponding output probabilities, and then merges them into a unified one.

TABLE I
IMPORTANT SYMBOLS IN SEPH.

n	the number of training instances
o_i	the ith training instance
X, Y	view-specific feature matrices of training instances
d_x, d_y	view-specific feature dimensions
A	the given semantic affinity matrix of training instances
H	to-be-learnt binary hash code matrix of training instances
$\hat{H}$	relaxed H, real-valued hash code matrix
d_c	hash code length
$p_{i,j}$	probability of observing the similarity between o_i and o_j in semantic space
$q_{i,j}$	probability of observing the similarity between o_i and o_j in Hamming space
$X_{i,\cdot}$, $Y_{i,\cdot}$, $H_{i,\cdot}$, $\hat{H}_{i,\cdot}$	the ith row of each matrix, corresponding to o_i
$\mathbf{h}^{(k)}$	the kth column of H, corresponding to the kth bit
$\mathbf{x}$, $\mathbf{y}$	view-specific feature vectors of an unseen instance
$\mathbf{c}^{\mathcal{X}}$, $\mathbf{c}^{\mathcal{Y}}$	predicted view-specific hash codes of an unseen instance
$\mathbf{c}$	unified hash code of an unseen instance

indicates the semantic affinity between o_i and o_j. Generally, we can derive A from manual scoring, or estimate it from correlations between semantic labels of the training instances, like cosine similarities. With A, semantics-preserving hash codes of the training instances can be learnt by SePH, which form a hash code matrix $H \in \{-1, 1\}^{n \times d_c}$ row by row. Specifically, the ith row in H, denoted as $H_{i,\cdot} \in \{-1, 1\}^{d_c}$, is the d_c-bit hash code of o_i. Note that for model simplicity, here we utilize $\{-1, 1\}$ to represent binary hash codes, and they can be directly mapped into $\{0, 1\}$. Table I summarizes the important symbols in this paper, which will be frequently used in the later description of SePH.

B. Semantics-Preserving Hashing

For preserving semantic affinities, if o_i and o_j are semantically similar, their corresponding hash codes should also be similar, and vice versa. As mentioned before, unlike previous related researches that utilize the given semantic affinities for *independently* weighting each pairwise distance (similarity) between hash codes, SePH can further incorporate the correlations between distances (similarities) to make the semantic affinities of training instances be better preserved by their to-be-learnt hash codes. Specifically, as illustrated in Fig. 2, in SePH the given semantic affinities are firstly transformed into a probability distribution $\mathcal{P}$, and then another probability distribution $\mathcal{Q}$ is derived from all the pairwise Hamming distances between to-be-learnt hash codes to approximate $\mathcal{P}$ in Hamming space. In that way, by transforming each pairwise Hamming distance into a probability, SePH standardizes them and makes them *dependent* on each other, and thus correlations between Hamming distances are incorporated.

To derive the probability distribution $\mathcal{P}$ in semantic space, we define $p_{i,j}$ as the probability of observing the semantic similarity between o_i and o_j among all pairs of training instances. Assuming that $p_{i,j}$ is proportional to $A_{i,j}$, *i.e.* the corresponding semantic affinity, we can derive $p_{i,j}$ as the following formula, which guarantees that $\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{i,j} = 1$.

$$p_{i,j} = \frac{A_{i,j}}{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} A_{i,j}} \tag{1}$$

To derive the probability distribution $\mathcal{Q}$ in Hamming space, we define $q_{i,j}$ as the probability of observing the similarity between o_i and o_j in Hamming space. Following t-SNE [41], a Student t-distribution with one degree of freedom is utilized for transforming each pairwise Hamming distance into a probability, as formulated as follows.

$$q_{i,j} = \frac{(1 + h(H_{i,\cdot}, H_{j,\cdot}))^{-1}}{\sum_{k=1}^{n}\sum_{m=1,m\neq k}^{n}(1 + h(H_{k,\cdot}, H_{m,\cdot}))^{-1}} \tag{2}$$

where $h(\cdot,\cdot)$ denotes the Hamming distance between two hash codes. Considering that $\forall 1 \leq i \leq n, H_{i,\cdot} \in \{-1, 1\}^{d_c}$, for any two binary hash codes we can derive their Hamming distance from their corresponding squared Euclidean distance, as shown in formula (3).

$$h(H_{i,\cdot}, H_{j,\cdot}) = \frac{1}{4}\|H_{i,\cdot} - H_{j,\cdot}\|_2^2 \tag{3}$$

By substituting formula (3) into formula (2), we can rewrite $q_{i,j}$ as follows to make it more tractable for optimization.

$$q_{i,j} = \frac{(1+\frac{1}{4}\|H_{i,\cdot} - H_{j,\cdot}\|_2^2)^{-1}}{\sum_{k=1}^{n}\sum_{m=1,m\neq k}^{n}(1+\frac{1}{4}\|H_{k,\cdot} - H_{m,\cdot}\|_2^2)^{-1}} \tag{4}$$

As mentioned previously, SePH aims to learn an optimal binary H that can enable $\mathcal{Q}$ to well approximate $\mathcal{P}$, so as to preserve the semantic affinities modelled by $\mathcal{P}$. Here we take the Kullback-Leibler divergence to measure the difference between $\mathcal{Q}$ and $\mathcal{P}$, as defined as follows.

$$D_{KL}(P\|Q) = \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{i,j}\log\frac{p_{i,j}}{q_{i,j}} \tag{5}$$

Then by minimizing $D_{KL}(P\|Q)$, SePH can learn the optimal binary hash code matrix H of the training data. And thus the objective function of SePH is formulated as follows.

$$\Psi_0 = \min_{H\in\{-1,1\}^{n\times d_c}} \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{i,j}\log\frac{p_{i,j}}{q_{i,j}} \tag{6}$$

where $p_{i,j}$ is defined as formula (1) and $q_{i,j}$ as formula (4). The objective function above, however, is NP-hard for directly deriving the optimal binary H. To make it more tractable, like previous work, here H is relaxed to be a real-valued matrix $\hat{H}$. Moreover, as shown in the following formula, to make the learnt $\hat{H}$ near to the optimal binary H, we further introduce a quantization loss term in the objective function to lead the entries of $\hat{H}$ to be near to -1 or 1.

$$\begin{aligned}\Psi &= \min_{\hat{H}\in\mathbb{R}^{n\times d_c}} \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{i,j}\log\frac{p_{i,j}}{q_{i,j}} + \frac{\alpha}{C}\||\hat{H}| - \mathbf{I}\|_2^2 \\ s.t.\quad q_{i,j} &= \frac{(1+\frac{1}{4}\|\hat{H}_{i,\cdot} - \hat{H}_{j,\cdot}\|_2^2)^{-1}}{\sum_{k=1}^{n}\sum_{m=1,m\neq k}^{n}(1+\frac{1}{4}\|\hat{H}_{k,\cdot} - \hat{H}_{m,\cdot}\|_2^2)^{-1}}\end{aligned} \tag{7}$$

where $\mathbf{I}$ is a matrix with each entry being 1, and $\||\hat{H}| - \mathbf{I}\|_2^2$ measures the quantization loss from real-valued $\hat{H}$ to binary H. Additionally, α is a model parameter for weighting the quantization loss term, and $C = n \times d_c$ is a normalizing factor to make the parameter tuning for α less affected by the hash code length and the training set size.

C. Solution and Implementation Issues

The objective function Ψ of SePH is an unconstrained non-convex optimization problem. Actually, its non-convexity comes from both the KL-divergence term and the quantization loss term. And thus for optimizing Ψ, we can only derive a locally optimal $\hat{H}$. Compared to other hashing methods that utilize a convex objective function, it may seem to be a weakness of the proposed SePH. However, as our experiments will demonstrate, the performance of SePH is fortunately not sensitive to the local optimality of its objective function. For optimizing $\hat{H}$, various effective gradient descent methods can be utilized. Specifically, for the ith row of $\hat{H}$, *i.e.* $\hat{H}_{i,\cdot}$, we can derive its corresponding gradient as follows.

$$\begin{aligned}\frac{\partial\Psi}{\partial\hat{H}_{i,\cdot}} = &\sum_{j=1,j\neq i}^{n}(p_{i,j} - q_{i,j})(1+\frac{1}{4}\|\hat{H}_{i,\cdot} - \hat{H}_{j,\cdot}\|_2^2)^{-1}(\hat{H}_{i,\cdot} - \hat{H}_{j,\cdot}) \\ &+ \frac{2\alpha}{C}(|\hat{H}_{i,\cdot}| - \mathbf{1}^T)\odot\sigma(\hat{H}_{i,\cdot})\end{aligned} \tag{8}$$

where $\mathbf{1}$ is a d_c-dimensional column vector with each entry being 1, $\odot$ denotes entry-wise multiplication between vectors, and $\sigma(\hat{H}_{i,\cdot})$ is a d_c-dimensional row vector made up of the signs of entries in $\hat{H}_{i,\cdot}$. Actually, here $\sigma(\hat{H}_{i,\cdot}) = \frac{\partial|\hat{H}_{i,\cdot}|}{\partial\hat{H}_{i,\cdot}}$, and the gradients *w.r.t* non-differentiable zero entries are simply set as 0. For detailed derivations, one can refer to the supplementary material.

By calculating $\frac{\partial\Psi}{\partial\hat{H}_{i,\cdot}}$ for all $1 \leq i \leq n$, effective gradient descent methods can be applied to derive an optimal $\hat{H}$. Then by getting the signs of entries in $\hat{H}$, we can derive an optimized binary hash code matrix H, *i.e.* $H = sign(\hat{H})$, with the signs of zero entries in $\hat{H}$ set as 1. For gradient descent methods, the time complexity of deriving H is $\mathcal{O}(Tn^2d_c)$, where T is the number of needed iterations.

D. Learning Hash Functions

With the learnt hash codes of training instances, *i.e.* H, SePH will independently learn hash functions for each view to perform out-of-sample extension. Actually, for SePH, any effective predictive models can be utilized as hash functions. Hence linear ridge regression, support vector machine (SVM) or its variants like bagging-based SVM [42], logistic regression, kernel logistic regression, and many other models can be utilized.

In this paper, we respectively utilize linear ridge regression, logistic regression and kernel logistic regression, to learn the projections from features to hash codes for each view. Linear ridge regression is widely-used in many previous researches on hashing, while for logistic regression and kernel logistic regression, they are employed because both can naturally provide output probabilities *w.r.t* the predicted hashing results, which, as will be explained later, are required for determining the unified hash code of an unseen instance. Note that here hash functions are learnt independently in different views. And thus for ease of presentation, in the following, only the hash function learning process in the view $\mathcal{X}$ is described, which can be directly extended to other views.

Like [26] and [23], here we learn hash functions bit by bit. Actually, considering that bits in the hash codes may not be independent of each other in cases, more sophisticated learning methods that incorporate the correlations between bits can also be investigated to obtain performance improvements, which is left to our future work. Denote the column corresponding to the kth bit in the learnt hash code matrix H as $\mathbf{h}^{(k)} \in \{-1,1\}^n$, *i.e.* the kth column of H. For linear ridge regression, its objective function to project features, *i.e.* X, into $\mathbf{h}^{(k)}$, is given as follows.

$$\mathcal{F}^{(k)} = \min_{\mathbf{u}^{(k)}}\|\mathbf{h}^{(k)} - X\mathbf{u}^{(k)}\|_2^2 + \mu\|\mathbf{u}^{(k)}\|_2^2 \tag{9}$$

where $\mathbf{u}^{(k)} \in \mathbb{R}^{d_x}$ is the to-be-learnt weighting vector, and μ is a weighting parameter for the regularizer. By setting $\frac{\partial \mathcal{F}^{(k)}}{\partial \mathbf{u}^{(k)}} = \mathbf{0}$, the optimal $\mathbf{u}^{(k)}$ can be directly derived as $\mathbf{u}^{(k)} = \left(X^T X + \mu \mathbf{E}\right)^{-1} X^T \mathbf{h}^{(k)}$, where $\mathbf{E} \in \mathbb{R}^{d_x \times d_x}$ is an identity matrix. Here the time complexity for deriving $\mathbf{u}^{(k)}$ is $\mathcal{O}(2nd_x^2 + d_x^3)$. Then by learning $\mathbf{u}^{(k)}$ for all $1 \leq k \leq d_c$, we can derive $\{\mathbf{u}^{(k)}\}_{k=1}^{d_c}$ as the hash function set based on linear ridge regression for the view $\mathcal{X}$. Here SePH with linear ridge regression as hash functions is denoted as SePH$_{linear}$.

Regarding logistic regression, its objective function is formulated as follows.

$$\mathcal{G}^{(k)} = \min_{\mathbf{w}^{(k)}} \sum_{i=1}^{n} \log\left(1 + e^{-\mathbf{h}_i^{(k)} X_{i,\cdot} \mathbf{w}^{(k)}}\right) + \eta \|\mathbf{w}^{(k)}\|_2^2 \quad (10)$$

where $\mathbf{h}_i^{(k)} \in \{-1, 1\}$ is the ith entry in $\mathbf{h}^{(k)}$, $\mathbf{w}^{(k)} \in \mathbb{R}^{d_x}$ is the to-be-learnt weighting vector, and η is a parameter for weighting the regularizer. Here $\mathcal{G}^{(k)}$ can be optimized with gradient descent methods, and the corresponding time complexity will be $\mathcal{O}(T_1^{(k)} n d_x)$ with $T_1^{(k)}$ being the number of needed iterations. By optimizing $\mathcal{G}^{(k)}$ for all $1 \leq k \leq d_c$, the derived $\{\mathbf{w}^{(k)}\}_{k=1}^{d_c}$ will work as the hash function set based on logistic regression for the view $\mathcal{X}$. Here SePH with logistic regression as hash functions is denoted as SePH$_{lr}$.

Furthermore, we introduce kernel logistic regression as hash functions, expecting to utilize kernel tricks to better handle non-linear projections from features to hash codes. Here we map each feature vector $X_{i,\cdot}$ to the Reproducing Kernel Hilbert Space (RKHS) as $\phi(X_{i,\cdot})$, and utilize them to build a kernel feature matrix Φ row by row. In RKHS, for kernel features $\phi(X_{i,\cdot})$ and $\phi(X_{j,\cdot})$, we can efficiently calculate their inner product $\phi(X_{i,\cdot})\phi^T(X_{j,\cdot})$ as $\kappa(X_{i,\cdot}, X_{j,\cdot})$ with kernel tricks, where $\kappa(\cdot, \cdot)$ denotes a kernel function. Then similarly, with kernel features, the objective function of kernel logistic regression corresponding to the kth bit can be formulated as follows.

$$\mathcal{H}^{(k)} = \min_{\mathbf{w}^{(k)}} \sum_{i=1}^{n} \log\left(1 + e^{-\mathbf{h}_i^{(k)} \phi(X_{i,\cdot}) \mathbf{w}^{(k)}}\right) + \lambda \|\mathbf{w}^{(k)}\|_2^2 \quad (11)$$

where λ is a parameter for weighting the regularizer. Following kernel CCA [43], here $\mathbf{w}^{(k)}$ is required to be in the span of the training kernel features, *i.e.* $\mathbf{w}^{(k)} = \Phi^T \mathbf{v}^{(k)}$ where $\mathbf{v}^{(k)}$ is the to-be-learnt spanning weights. Then $\phi(X_{i,\cdot})\mathbf{w}^{(k)}$ in formula (11) is rewritten as $(\phi(X_{i,\cdot})\Phi^T)\mathbf{v}^{(k)}$, where we can calculate $\phi(X_{i,\cdot})\Phi^T$ as $\kappa(X_{i,\cdot}, X)$. It can be observed that, for kernel logistic regression, its costs for training and predicting will be proportional to n, *i.e.* the training set size, which is unsuitable for large training sets. As pointed out by Hu *et al.* [44], generally the training kernel features would be redundant for spanning $\mathbf{w}^{(k)}$. And thus here we propose to sample kernel features from Φ via random sampling or other alternative methods like k-means to build a much smaller one for spanning $\mathbf{w}^{(k)}$, which is denoted as $\hat{\Phi}$. Suppose that the sampling size is $s (s \ll n)$. Then we need to learn a s-dimensional weighting vector $\hat{\mathbf{v}}^{(k)}$ for spanning $\mathbf{w}^{(k)}$ with $\hat{\Phi}$, *i.e.* $\mathbf{w}^{(k)} = \hat{\Phi}^T \hat{\mathbf{v}}^{(k)}$. And formula (11) can be rewritten as follows.

$$\begin{aligned}\mathcal{H}^{(k)} = \min_{\hat{\mathbf{v}}^{(k)}} \sum_{i=1}^{n} \log\left(1 + e^{-\mathbf{h}_i^{(k)} \left(\phi(X_{i,\cdot})\hat{\Phi}^T\right) \hat{\mathbf{v}}^{(k)}}\right) \\ + \lambda \|\hat{\Phi}^T \hat{\mathbf{v}}^{(k)}\|_2^2\end{aligned} \quad (12)$$

In this case, for kernel logistic regression, its costs for training and predicting will be proportional to the sampling size s rather than the training set size n. Then its training can become more scalable and its predicting can be more efficient. Here $\mathcal{H}^{(k)}$ can also be optimized with gradient descent methods, and the corresponding time complexity will be $\mathcal{O}(K + T_2^{(k)} ns)$, where $T_2^{(k)}$ is the number of needed iterations and K is the costs of calculating $\Phi\hat{\Phi}^T$ and $\hat{\Phi}\hat{\Phi}^T$. By optimizing $\mathcal{H}^{(k)}$ for all $1 \leq k \leq d_c$, we can derive $\{\hat{\Phi}, \hat{\mathbf{v}}^{(1)}, \hat{\mathbf{v}}^{(2)}, \ldots, \hat{\mathbf{v}}^{(d_c)}\}$ as the non-linear hash function set based on kernel logistic regression for the view $\mathcal{X}$. It should be noticed that here all $\hat{\mathbf{v}}^{(k)} (1 \leq k \leq d_c)$ share an identical $\hat{\Phi}$, which can further reduce the training and predicting costs for all d_c kernel logistic regressions. For example, the total training costs for the d_c kernel logistic regressions will be $\mathcal{O}(K + \sum_{k=1}^{d_c} T_2^{(k)} ns)$ rather than $\mathcal{O}(Kd_c + \sum_{k=1}^{d_c} T_2^{(k)} ns)$. Here SePH with kernel logistic regression as hash functions is denoted as SePH$_{klr}$.

E. Generating Hash Codes

With learnt hash functions, the view-specific hash codes of any unseen instance o_u can be predicted. Taking the view $\mathcal{X}$ as an example, assume that the feature vector of o_u is $\mathbf{x}$, and its predicted view-specific hash code is denoted as $\mathbf{c}^{\mathcal{X}}$, with the kth bit denoted as $\mathbf{c}_k^{\mathcal{X}}$. Then we can derive that $\mathbf{c}_k^{\mathcal{X}} = sign(\mathbf{x}\mathbf{u}^{(k)})$ for linear ridge regression, $\mathbf{c}_k^{\mathcal{X}} = sign\left(\mathbf{x}\mathbf{w}^{(k)}\right)$ for logistic regression, and $\mathbf{c}_k^{\mathcal{X}} = sign\left((\phi(\mathbf{x})\hat{\Phi}^T)\hat{\mathbf{v}}^{(k)}\right)$ for kernel logistic regression, with $sign(\cdot)$ denoting the sign of an expression. By predicting $\mathbf{c}_k^{\mathcal{X}}$ for $1 \leq k \leq d_c$, we can get the predicted view-specific hash code $\mathbf{c}^{\mathcal{X}}$. The time complexity for linear ridge regression, logistic regression and kernel logistic regression to predict $\mathbf{c}^{\mathcal{X}}$ are respectively $\mathcal{O}(d_x d_c)$, $\mathcal{O}(d_x d_c)$ and $\mathcal{O}(K' + sd_c)$, where K' denotes the costs of calculating $\phi(\mathbf{x})\hat{\Phi}$ in kernel logistic regression.

Given an unseen instance o_u, if only one view is observed, its predicted view-specific hash code can be directly utilized as its unified hash code. Meanwhile, if both views are observed, we need to determine its unified hash code by merging predicted view-specific hash codes from both views, especially in cases where the predicted view-specific hash codes conflict, as illustrated in Fig. 2. To tackle that, we propose a novel probabilistic approach in this paper for determining the value of each bit in the unified hash code of o_u. As mentioned previously, the proposed combining approach requires the output probabilities *w.r.t* each bit of the predicted view-specific hash codes, *i.e.* $p(\mathbf{c}_k^{\mathcal{Z}} = b|\mathbf{z})$ where $\mathcal{Z} \in \{\mathcal{X}, \mathcal{Y}\}$, $1 \leq k \leq d_c$, $b \in \{-1, 1\}$ and $\mathbf{z} \in \{\mathbf{x}, \mathbf{y}\}$.

Taking the view $\mathcal{X}$ as an example, here we introduce an effective method to estimate $p(\mathbf{c}_k^{\mathcal{Z}} = b|\mathbf{Z})$ for hash functions, especially for linear ridge regression and similar methods that cannot naturally provide output probabilities with predicted results. Inspired by Gaussian Mixture Model (GMM) [45], for linear ridge regression and similar methods, to estimate $p(\mathbf{c}_k^{\mathcal{X}} = -1|\mathbf{x})$ and $p(\mathbf{c}_k^{\mathcal{X}} = 1|\mathbf{x})$, we assume that the corresponding original predicted result (*i.e.* $\mathbf{x}\mathbf{u}^{(k)}$ for linear ridge regression) comes from either of two Gaussian distributions that respectively correspond to -1 and 1. The two Gaussian distributions are modelled as follows. During training, given $\mathbf{h}^{(k)} \in \{-1, 1\}^n$, the training instances are separated into two sets, one consisting of training instances with the kth bit of their corresponding hash codes being -1 and the other consisting of those with the kth bit being 1. Suppose that features of training instances in the first set forms a feature matrix X_n, and those in the second set forms another feature matrix X_p. With the learnt weighting vector $\mathbf{u}^{(k)}$, we can derive $X_n\mathbf{u}^{(k)}$ and $X_p\mathbf{u}^{(k)}$, and both are assumed to be respectively sampled from the two to-be-modelled Gaussian distributions corresponding to -1 and 1. Then we take the mean value μ_n and the standard deviation σ_n of $X_n\mathbf{u}^{(k)}$ to model the Gaussian distribution corresponding to -1, and similarly take the mean value μ_p and the standard deviation σ_p of $X_p\mathbf{u}^{(k)}$ to model the Gaussian distribution corresponding to 1. Similar to GMM, with both Gaussian distributions, the output probabilities $p(\mathbf{c}_k^{\mathcal{X}} = -1|\mathbf{x})$ and $p(\mathbf{c}_k^{\mathcal{X}} = 1|\mathbf{x})$ for any $\mathbf{x}$ can be estimated as follows.

$$\begin{aligned}
g_n &= \frac{1}{\sigma_n\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{x}\mathbf{u}^{(k)}-\mu_n)^2}{2\sigma_n^2}\right) \\
g_p &= \frac{1}{\sigma_p\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{x}\mathbf{u}^{(k)}-\mu_p)^2}{2\sigma_p^2}\right) \\
p(\mathbf{c}_k^{\mathcal{X}} = -1|\mathbf{x}) &= \frac{g_n}{g_n+g_p} \\
p(\mathbf{c}_k^{\mathcal{X}} = 1|\mathbf{x}) &= \frac{g_p}{g_n+g_p}
\end{aligned} \tag{13}$$

As for logistic regression and kernel logistic regression, the required output probabilities are naturally provided, and can be respectively derived as the following formulas, with $b \in \{-1, 1\}$.

$$p(\mathbf{c}_k^{\mathcal{X}} = b|\mathbf{x}) = \left(1 + e^{-b\mathbf{x}\mathbf{w}^{(k)}}\right)^{-1} \tag{14}$$

$$p(\mathbf{c}_k^{\mathcal{X}} = b|\mathbf{x}) = \left(1 + e^{-b\left(\phi(\mathbf{x})\hat{\Phi}^T\right)\hat{\mathbf{v}}^{(k)}}\right)^{-1} \tag{15}$$

With output probabilities derived or estimated, the predicted view-specific hash codes can be merged into a unified one. Suppose that for an unseen instance o_u, its feature vectors in $\mathcal{X}$ and $\mathcal{Y}$ are respectively denoted as $\mathbf{x}$ and $\mathbf{y}$, and $\mathbf{c} \in \{-1, 1\}^{d_c}$ is its to-be-determined unified hash code, with the kth bit denoted as $\mathbf{c}_k$. Then bit by bit, $\mathbf{c}_k$ is determined as the following formula.

$$\mathbf{c}_k = sign\Big(p(\mathbf{c}_k = 1|\mathbf{x}, \mathbf{y}) - p(\mathbf{c}_k = -1|\mathbf{x}, \mathbf{y})\Big) \tag{16}$$

Assuming that $\mathcal{X}$ and $\mathcal{Y}$ are conditionally independent on $\mathbf{c}_k$, we can derive the following formula with Bayes' theorem.

$$\begin{aligned}
\mathbf{c}_k = sign\Big(&p(\mathbf{x}|\mathbf{c}_k = 1)p(\mathbf{y}|\mathbf{c}_k = 1)p(\mathbf{c}_k = 1) \\
&-p(\mathbf{x}|\mathbf{c}_k = -1)p(\mathbf{y}|\mathbf{c}_k = -1)p(\mathbf{c}_k = -1)\Big)
\end{aligned} \tag{17}$$

Moreover, with the Bayes' theorem, we can further transform the formula above into the following one.

$$\begin{aligned}
\mathbf{c}_k = sign\Big(&\frac{p(\mathbf{c}_k = 1|\mathbf{x})p(\mathbf{c}_k = 1|\mathbf{y})}{p(\mathbf{c}_k = 1)} \\
&-\frac{p(\mathbf{c}_k = -1|\mathbf{x})p(\mathbf{c}_k = -1|\mathbf{y})}{p(\mathbf{c}_k = -1)}\Big)
\end{aligned} \tag{18}$$

where $p(\mathbf{c}_k = b|\mathbf{z}) = p(\mathbf{c}_k^{\mathcal{Z}} = b|\mathbf{z})$ with $b \in \{-1, 1\}, \mathbf{z} \in \{\mathbf{x}, \mathbf{y}\}, \mathcal{Z} \in \{\mathcal{X}, \mathcal{Y}\}$, and all these probabilities can be derived with formula (13) / (14) / (15) or using other more sophisticated estimation methods. Here $p(\mathbf{c}_k = -1)$ and $p(\mathbf{c}_k = 1)$ are the priori probabilities for the kth bit being -1 or 1. In our previous work [40], both priori probabilities are simply set to be equal, *i.e.* $p(\mathbf{c}_k = 1) = p(\mathbf{c}_k = -1)$. However, the assumption about the balance between -1 and 1 in [40] can sometimes be unreasonable, especially in some imbalanced datasets. Therefore, here we propose that $p(\mathbf{c}_k = -1)$ and $p(\mathbf{c}_k = 1)$ should be dataset-dependent, and statistics-based or learning-based methods are expected to be utilized for estimating them. Specifically, in this paper, $p(\mathbf{c}_k = -1)$ and $p(\mathbf{c}_k = 1)$ are respectively estimated as the relative frequencies of -1 and 1 in $\mathbf{h}^{(k)}$, *i.e.* $p(\mathbf{c}_k = -1) = \frac{\sum_{i=1}^{n} Cond(\mathbf{h}_i^{(k)}=-1)}{n}$ and $p(\mathbf{c}_k = 1) = \frac{\sum_{i=1}^{n} Cond(\mathbf{h}_i^{(k)}=1)}{n}$ where $Cond(\cdot)$ is a condition function returning 1 if the condition holds and 0 otherwise. Actually, our experiments show that such a dataset-dependent estimation method can help SePH to obtain further performance improvements, compared to simply setting $p(\mathbf{c}_k = 1) = p(\mathbf{c}_k = -1)$. We will further investigate other more sophisticated estimation methods in our future work.

For the unseen instance o_u, with all $\mathbf{c}_k (1 \leq k \leq d_c)$ determined, SePH will generate its unified hash code $\mathbf{c}$. Note that alternatively one can utilize multi-view learning methods like [46], [47] to learn hash functions for each combination of views and then directly generate unified hash codes without combining, but that can probably lead to much higher learning costs due to the "exponential explosion" of view combinations.

F. Extensions

Actually, for cases with more than two views, we can perform training for SePH in nearly the same way, except that we need to learn hash functions for more views. Meanwhile, for out-of-sample extension, after predicting view-specific hash codes in the same manner, it is slightly different to merge them into a unified one. Specifically, we extend formula (18) as follows for cases of more views with similar derivations.

$$\mathbf{c}_k = sign\left(\frac{\prod_{i=1}^{m} p(\mathbf{c}_k = 1|\mathbf{z}^i)}{(p(\mathbf{c}_k = 1))^{m-1}} - \frac{\prod_{i=1}^{m} p(\mathbf{c}_k = -1|\mathbf{z}^i)}{(p(\mathbf{c}_k = -1))^{m-1}}\right) \tag{19}$$

where $m \geq 1$ indicates how many views are observed, and $\mathbf{z}^i$ denotes the feature vector in the ith view. Here all needed probabilities can be derived or estimated in the same way as those in formula (18).

IV. Experiments

A. Experimental Settings

In this paper, we conduct experiments on three benchmark datasets to evaluate the proposed SePH. Specifically, the benchmark datasets include Wiki [48], MIRFlickr [49] and NUS-WIDE [50], and they are all with an image view and a text view. Table II gives some statistics of them.

Wiki is made up of 2,866 instances collected from Wikipedia. For each instance, a 128-D Bag-of-Visual-Words

TABLE II
STATISTICS OF THREE BENCHMARK DATASETS.

	Wiki	MIRFlickr	NUS-WIDE
Dataset Size	2,866	16,738	186,577
Retrieval Set	2,173	15,902	184,711
Training Set	2,173	5,000	5,000
Query Set	693	836	1,866
Nr. of Labels	10	24	10

SIFT feature vector is provided to describe its image view and a 10-D topic vector is given to describe its text view. Each instance is mannually annotated with one semantic label from 10 candidates. Following [23], [24], we take 25% of Wiki to form the query set, and the rest works as the retrieval set.

MIRFlickr originally contains 25,000 instances collected from Flickr. Each instance consists of an image and its associated textual tags, and is manually annotated with one or more of 24 provided semantic labels. To avoid noises, here we remove textual tags that appear less than 20 times in the dataset, and then delete instances without textual tags or semantic labels. After pretreatment, we get 16,738 instances left. For each instance, a 150-D edge histogram is provided to describe its image view, while its text view is represented as a 500-D feature vector derived from PCA [51] on its binary tagging vector *w.r.t* the remaining textual tags. We take 5% of MIRFlickr to form the query set, and the rest works as the retrieval set.

NUS-WIDE is a large dataset originally containing 269,648 instances. Like MIRFlickr, each instance in NUS-WIDE consists of an image and its associated textual tags, and is manually annotated with one or more semantic labels from 81 candidates. Following [23], [24], here we only keep the top 10 most frequent labels and the corresponding 186,577 instances annotated with them. For each instance, a 500-D Bag-of-Visual-Words SIFT feature vector is provided to describe its image view, while its text view is represented as a binary tagging vector *w.r.t* the top 1,000 most frequent tags. We also take 1% of NUS-WIDE to form the query set, and the rest works as the retrieval set.

Considering the small size of Wiki, we follow [23] and take its retrieval set as the training set. As for the large MIRFlickr and NUS-WIDE, to simulate real-world cases where only the supervised information of a small fraction of the data is provided, for either dataset we just sample 5,000 instances from the corresponding retrieval set to form the training set. It should be noticed that, the learnt hash codes of training instances in the training process of SePH will be discarded after hash functions are learnt, and then SePH generates hash codes for all instances in the dataset with the learnt hash functions. Moreover, although each bit in the hash codes generated by SePH is in $\{-1, 1\}$, in our experiments we map them into $\{0, 1\}$ and compactly store them bit by bit. Like most previous hashing methods, to perform ANN retrieval for any query hash code H_q, its Hamming distance to any ith hash code H_i in the retrieval set, denoted as $h(H_q, H_i)$, is calculated as $h(H_q, H_i) = bit\_count(H_q \oplus H_i)$, where $\oplus$ denotes XOR operation between the bits of H_q and H_i, and bit_count counts the number of 1 in the binary XOR result. Then we rank all instances in the retrieval set based on their corresponding Hamming distances in an ascending order and take the top ones as the ANNs for the query instance.

In our experiments, the annotated semantic labels of any training instance are represented as a binary labelling vector. Then we derive the affinity matrix of each dataset, *i.e.* A in formula (1), as the cosine similarities between labelling vectors of training instances. The only model parameter α in the objective function of SePH (*i.e.* formula (7)) is empirically set as 0.01 for all datasets. As for the learning of hash functions in each view, μ in formula (9) for linear ridge regression in SePH$_{linear}$, η in formula (10) for logistic regression in SePH$_{lr}$, and λ in formula (12) for kernel logistic regression in SePH$_{klr}$, are automatically set via 5-fold cross-validation on the corresponding features and learnt hash codes of training instances. Particularly, for kernel logistic regression in SePH$_{klr}$, a RBF kernel is utilized, with its parameter σ^2 set as the mean squared Euclidean distance between feature vectors of training instances. Additionally, on all datasets the sampling size for $\hat{\Phi}$ in formula (12) is empirically set as 500. We perform both random sampling and k-means sampling for SePH$_{klr}$, which are denoted as SePH$_{klr+rnd}$ and SePH$_{klr+km}$ respectively. To encourage further developments, the codes of SePH will be published in a near future.

We employ the supervised CMSSH [26], CVH [27], KSH-CV [34], SCM-Orth and SCM-Seq [35], and the unsupervised IMH [21], LSSH [24], CMFH [23] as baselines to compare with the proposed SePH. Note that for IMH, we calculate its required affinity matrices with the provided semantic labels of training instances, and thus it is actually supervised here. To make fair comparisons, we carefully perform parameter tuning for baselines, and report their best performance in this paper. We perform 10 runs for SePH and any compared baseline with a non-convex objective function with different initial values, and report the average performance.

Following previous researches, we utilize *mean average precision* (mAP) to measure the retrieval performance of all cross-view hashing methods. A higher mAP value means better retrieval performance. Here the definition of mAP is given as follows.

$$mAP = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{m_i} \sum_{j=1}^{m_i} precision(R_{i,j}) \tag{20}$$

where Q is the query set with its size being $|Q|$, and for the ith query, $\frac{1}{m_i} \sum_{j=1}^{m_i} precision(R_{i,j})$ denotes its *average precision* (AP), m_i denotes the number of its ground-truth relevant instances in the retrieval set, $R_{i,j}$ is a subset of its ranked retrieval result consisting of instances from the top one to the jth ground-truth relevant one, and $precision(R_{i,j})$ measures the precision value in $R_{i,j}$. Like [23], [24], an instance is ground-truth relevant to a query if they share at least one semantic label.

B. Experimental Results

The cross-view retrieval performance of the proposed SePH and the compared baselines on all datasets is reported in

TABLE III
CROSS-VIEW RETRIEVAL PERFORMANCE OF THE PROPOSED SEPH (*i.e.* SEPH$_{linear}$, SEPH$_{lr}$, SEPH$_{klr+rnd}$ AND SEPH$_{klr+km}$) AND COMPARED BASELINES ON ALL BENCHMARK DATASETS WITH DIFFERENT HASH CODE LENGTHS, IN TERMS OF mAP. FOR SEPH, THE STANDARD ERRORS OF mAP OVER 10 RUNS ARE ALSO REPORTED.

		Wiki				MIRFlickr				NUS-WIDE			
		16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits
Image Query *v.s.* Text Database	CMSSH [26]	0.1877	0.1771	0.1646	0.1552	0.5728	0.5743	0.5706	0.5706	0.4063	0.3927	0.3939	0.3739
	CVH [27]	0.1257	0.1212	0.1215	0.1171	0.6067	0.6177	0.6157	0.6074	0.3687	0.4182	0.4602	0.4466
	IMH [21]	0.1573	0.1575	0.1568	0.1651	0.6016	0.6120	0.6070	0.5982	0.4187	0.3975	0.3778	0.3668
	LSSH [24]	0.2141	0.2216	0.2218	0.2211	0.5784	0.5804	0.5797	0.5816	0.3900	0.3924	0.3962	0.3966
	CMFH [23]	0.2132	0.2259	0.2362	0.2419	0.5861	0.5835	0.5844	0.5849	0.4267	0.4229	0.4207	0.4182
	KSH-CV [34]	0.1965	0.1839	0.1701	0.1662	0.5793	0.5767	0.5732	0.5744	0.4229	0.4162	0.4026	0.3877
	SCM-Orth [35]	0.1598	0.1460	0.1383	0.1131	0.5854	0.5751	0.5704	0.5649	0.3787	0.3668	0.3593	0.3520
	SCM-Seq [35]	0.2210	0.2337	0.2442	0.2596	0.6237	0.6343	0.6448	0.6489	0.4842	0.4941	0.4947	0.4965
	SePH$_{linear}$	0.2479 ±0.0023	0.2589 ±0.0027	0.2788 ±0.0017	0.2833 ±0.0016	0.6672 ±0.0009	0.6724 ±0.0005	0.6757 ±0.0006	0.6782 ±0.0003	0.5465 ±0.0023	**0.5603** **±0.0009**	**0.5660** **±0.0007**	**0.5694** **±0.0005**
	SePH$_{lr}$	0.2375 ±0.0030	0.2531 ±0.0024	0.2619 ±0.0016	0.2686 ±0.0013	0.6640 ±0.0011	0.6699 ±0.0004	0.6736 ±0.0003	0.6757 ±0.0004	0.5393 ±0.0013	0.5503 ±0.0016	0.5562 ±0.0006	0.5601 ±0.0005
	SePH$_{klr+rnd}$	0.2835 ±0.0028	0.3003 ±0.0022	**0.3099** **±0.0024**	0.3204 ±0.0014	0.6727 ±0.0007	0.6792 ±0.0005	**0.6833** **±0.0007**	0.6860 ±0.0006	0.5450 ±0.0011	0.5532 ±0.0012	0.5605 ±0.0006	0.5650 ±0.0008
	SePH$_{klr+km}$	**0.2838** **±0.0026**	**0.3009** **±0.0023**	0.3074 ±0.0019	**0.3207** **±0.0015**	**0.6733** **±0.0008**	**0.6793** **±0.0007**	0.6829 ±0.0005	**0.6864** **±0.0005**	**0.5477** **±0.0014**	0.5568 ±0.0010	0.5640 ±0.0008	0.5666 ±0.0009
Text Query *v.s.* Image Database	CMSSH [26]	0.1630	0.1617	0.1539	0.1517	0.5715	0.5732	0.5699	0.5697	0.3874	0.3849	0.3704	0.3699
	CVH [27]	0.1185	0.1034	0.1024	0.0990	0.6026	0.6041	0.6017	0.5972	0.3646	0.4024	0.4339	0.4255
	IMH [21]	0.1463	0.1311	0.1290	0.1301	0.5895	0.6031	0.6010	0.5930	0.4053	0.3892	0.3758	0.3627
	LSSH [24]	0.5031	0.5224	0.5293	0.5346	0.5898	0.5927	0.5932	0.5932	0.4286	0.4248	0.4248	0.4175
	CMFH [23]	0.4884	0.5132	0.5269	0.5375	0.5937	0.5919	0.5931	0.5919	0.4627	0.4556	0.4518	0.4478
	KSH-CV [34]	0.1710	0.1665	0.1696	0.1576	0.5786	0.5763	0.5728	0.5715	0.4088	0.3906	0.3869	0.3834
	SCM-Orth [35]	0.1553	0.1389	0.1262	0.1096	0.5857	0.5747	0.5672	0.5604	0.3756	0.3641	0.3565	0.3523
	SCM-Seq [35]	0.2134	0.2366	0.2479	0.2573	0.6133	0.6209	0.6295	0.6340	0.4536	0.4620	0.4630	0.4644
	SePH$_{linear}$	0.5431 ±0.0042	0.5619 ±0.0017	0.5809 ±0.0015	0.5872 ±0.0014	0.7188 ±0.0013	0.7285 ±0.0006	0.7356 ±0.0004	0.7385 ±0.0003	0.6375 ±0.0022	**0.6532** **±0.0012**	**0.6633** **±0.0005**	**0.6674** **±0.0004**
	SePH$_{lr}$	0.5531 ±0.0050	0.5724 ±0.0025	0.5888 ±0.0016	0.5966 ±0.0008	0.7176 ±0.0009	0.7283 ±0.0006	0.7347 ±0.0004	0.7385 ±0.0004	0.6291 ±0.0013	0.6455 ±0.0007	0.6545 ±0.0006	0.6597 ±0.0004
	SePH$_{klr+rnd}$	**0.6310** **±0.0031**	0.6512 ±0.0015	0.6633 ±0.0015	0.6692 ±0.0015	0.7216 ±0.0005	0.7296 ±0.0007	0.7372 ±0.0006	0.7408 ±0.0006	0.6283 ±0.0013	0.6415 ±0.0015	0.6530 ±0.0008	0.6584 ±0.0005
	SePH$_{klr+km}$	**0.6310** **±0.0024**	**0.6516** **±0.0018**	**0.6652** **±0.0018**	**0.6701** **±0.0013**	**0.7247** **±0.0011**	**0.7328** **±0.0011**	**0.7410** **±0.0010**	**0.7437** **±0.0008**	**0.6378** **±0.0011**	0.6513 ±0.0012	0.6612 ±0.0007	**0.6674** **±0.0011**

Table III, including both the performance of retrieving text with image (*i.e.* "Image Query *v.s.* Text Database") and that of retrieving image with text (*i.e.* "Text Query *v.s.* Image Database"). For the former task, the image view of instances in the query set is utilized to generate their corresponding query hash codes, while for the latter one, the text view is utilized. As for any instance in the retrieval set, like CMFH and LSSH, SePH generates one unified hash code for both views. Moreover, considering that the objective function of SePH is non-convex, here we also report the standard errors *w.r.t* the performance of SePH$_{linear}$, SePH$_{lr}$, SePH$_{klr+rnd}$ and SePH$_{klr+km}$ over the ten runs on each dataset, so as to investigate how different initial values of $\hat{H}$ can affect the performance of SePH

From Table III, we can get the following observations. 1) Even with varying hash code lengths, the proposed SePH, including SePH$_{linear}$, SePH$_{lr}$, SePH$_{klr+rnd}$ and SePH$_{klr+km}$, significantly outperforms all compared baselines on all the three benchmark datasets, which well demonstrates its effectiveness. The superiority of SePH is attributed to both its capability of better preserving semantic affinities in Hamming space and the effectiveness of the learnt hash functions. 2) On all datasets, the performance of SePH keeps increasing as the hash code length increases, meaning that it can well utilize longer hash codes for better preserving the semantic affinities. Meanwhile, as also observed in [23], [34], [35], the performance of CMSSH, KSH-CV and SCM-Orth decreases, which may be caused by the imbalance between bits in the hash codes learnt by singular value decomposition or eigenvalue decomposition. 3) The standard errors *w.r.t* the performance of SePH$_{linear}$, SePH$_{lr}$, SePH$_{klr+rnd}$ and SePH$_{klr+km}$ are quite small on all datasets (less than 2% of the corresponding mAP value), meaning that the performance of SePH is not sensitive to the local optimality of its objective function. 4) Generally, SePH$_{linear}$ and SePH$_{lr}$ are inferior to SePH$_{klr+rnd}$ / SePH$_{klr+km}$, while on the large MIRFlickr and NUS-WIDE, the performance of SePH$_{linear}$ and that of SePH$_{lr}$ are quite comparable to that of SePH$_{klr+rnd}$ / SePH$_{klr+km}$. That, on one hand, shows the superiority of kernel logistic regression in modelling the non-linear projections from features to binary hash codes, and on the other hand, also reflects the effectiveness of utilizing linear ridge regression or logistic regression as hash functions. 5) On all datasets, it can be seen that SePH$_{klr+km}$ is generally superior to SePH$_{klr+rnd}$, but the superiority is insignificant (less than 2%). Therefore, the performance of SePH$_{klr}$ is not sensitive to the sampling strategy for the learning of kernel logistic regression.

Furthermore, we perform *paired-sample t-test* [52] for evaluating the significance of the improvements achieved by the proposed SePH over the compared baselines in both cross-view retrieval tasks on all datasets with different hash code lengths. For each algorithm, we take the corresponding AP (*average precision*) values of the query set as samples from its AP distribution, and compare them between algorithms for significance tests. The significance level is set as a typical value 0.01 here. And we find that the maximal *P-value* in

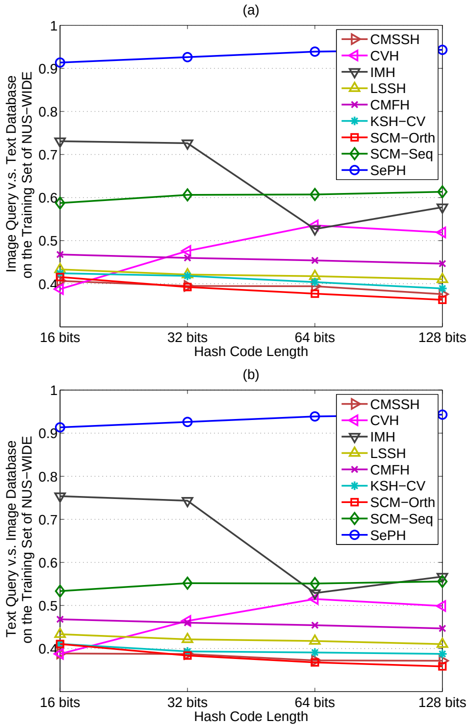


Fig. 3. Cross-view retrieval performance of all algorithms on the training set of NUS-WIDE with different hash code lengths, which reflects the quality of the hash codes learnt by each algorithm for a training set.

all significance tests between variants of SePH and compared baselines is around 10^{-7}, which is far less than the significance level 0.01, meaning that the improvements gained by SePH over the compared baselines are statistically significant.

To get more inside details about the superiority of SePH, we further analyse the quality of the learnt hash codes of training instances. Specifically, on the training set of each dataset, we utilize the corresponding learnt hash codes to perform cross-view retrieval, repeatedly using one as a query to retrieve nearest neighbours from the rest, and then measure the corresponding mAP value. Since we utilize the semantic labels of instances to define their ground-truth relevance for calculating mAP, the derived mAP values can quantitatively reflect how well the learnt hash codes can preserve the given semantic affinities of training instances. Fig. 3 illustrates the performance of learnt hash codes by SePH in the two cross-view retrieval tasks on the training set of the largest NUS-WIDE, with the hash code length varying from 16 to 128. Fig. 3 also presents the performance of baselines for comparison. We can observe that SePH significantly outperforms the baselines, with the corresponding mAP being above 0.9. Actually, similar results can also be observed on Wiki and MIRFlickr, with the corresponding mAP value of SePH being 1.0 on Wiki and above 0.9 on MIRFlickr. For more details, one can refer to the supplementary material. Therefore, it can be seen that the hash codes learnt by SePH can well preserve the semantic affinities of training instances. Additionally, by comparing Fig. 3 and Table III, one can observe that the retrieval performance of the learnt hash codes of training instances is significantly better than that of the hash codes generated by learnt hash functions. We attribute this to: 1) the view-specific features of the three datasets are somewhat weak and may not well describe the instance in the corresponding view, 2) the employed predictive models, *i.e.* linear ridge regression, logistic regress and kernel logistic regression, may not be capable enough. Therefore, stronger features and more powerful predictive models need to be further investigated.

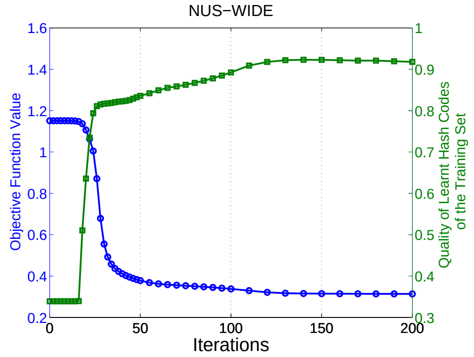


Fig. 4. Variances of the objective function value and the corresponding quality of learnt hash codes of training instances in SePH as the number of iterations increases, on NUS-WIDE with the hash code length fixed as 16 bits.

In our experiments, we utilize the method of gradient descent with a momentum of 0.5 to optimize the objective function Ψ (*i.e.* formula (7)) of SePH. Here we further conduct experiments to analyse the convergence of the optimization process and see how the quality of the learnt hash codes of training instances varies with iterations. Specifically, by fixing the hash code length as 16 bits, we perform 200 iterations of gradient descent on Wiki, MIRFlickr and NUS-WIDE to optimize Ψ. Then for each iteration, we calculate the value of Ψ. Meanwhile, we take the corresponding value of $\hat{H}$ to derive hash codes of the training instances, and analyse their quality by measuring their retrieval performance on the corresponding training set. Note that since SePH learns one unified hash code for each training instance, the retrieval performance of learnt hash codes in "Image Query *v.s.* Text Database" will be identical to that in "Text Query *v.s.* Image Database" on the training set, and thus we just report one. The experimental results on the largest NUS-WIDE is illustrated in Fig. 4, and those on Wiki, MIRFlickr are provided in the supplementary material due to the limited space. Then we can obtain the following observations. 1) The optimization process for SePH can generally converge in around 100 iterations, and for Wiki and MIRFlickr it can even converge faster. 2) As the number of iterations increases, the quality of the learnt hash codes of training instances quickly increases and then converges.

C. Experimental Validations of the Proposed Probabilistic Approach for Determining Unified Hash Codes

To validate the proposed probabilistic approach for determining the unified hash code of an unseen instance, *i.e.* formula (18) and (19), we further conduct experiments on all datasets to see whether it can help to improve the cross-view retrieval performance. As all datasets contain only two views,

TABLE IV
COMPARISONS BETWEEN THE PROPOSED PROBABILISTIC APPROACH (*i.e.* SEPH$_{linear}$, SEPH$_{lr}$, SEPH$_{klr+rnd}$ AND SEPH$_{klr+km}$) AND OTHER STRATEGIES FOR DETERMINING THE UNIFIED HASH CODES OF UNSEEN INSTANCES.

		Wiki				MIRFlickr				NUS-WIDE			
		16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits
Image Query *v.s.* Text Database	SePH$_{linear}$[Img]	0.1443	0.1503	0.1557	0.1574	0.6128	0.6153	0.6169	0.6184	0.4608	0.4674	0.4707	0.4743
	SePH$_{linear}$[Txt]	0.2281	0.2334	0.2491	0.2518	0.6585	0.6637	0.6668	0.6688	0.5211	0.5309	0.5354	0.5408
	SePH$_{linear}$[Rand]	0.1901	0.2054	0.2255	0.2336	0.6528	0.6602	0.6647	0.6679	0.5226	0.5389	0.5479	0.5552
	SePH$_{linear}$[Equal]	0.2407	0.2477	0.2677	0.2710	**0.6673**	**0.6724**	0.6756	0.6779	0.5431	0.5554	0.5608	0.5660
	SePH$_{linear}$	**0.2479**	**0.2589**	**0.2788**	**0.2833**	0.6672	**0.6724**	**0.6757**	**0.6782**	**0.5465**	**0.5603**	**0.5660**	**0.5694**
	SePH$_{lr}$[Img]	0.1463	0.1527	0.1574	0.1596	0.6110	0.6143	0.6163	0.6172	0.4612	0.4668	0.4707	0.4725
	SePH$_{lr}$[Txt]	0.2333	0.2480	0.2556	0.2618	0.6550	0.6611	0.6649	0.6671	0.5226	0.5329	0.5384	0.5428
	SePH$_{lr}$[Rand]	0.1959	0.2163	0.2320	0.2422	0.6484	0.6569	0.6619	0.6652	0.5254	0.5403	0.5494	0.5558
	SePH$_{lr}$[Equal]	0.2311	0.2454	0.2545	0.2608	0.6633	0.6693	0.6729	0.6750	0.5366	0.5463	0.5516	0.5558
	SePH$_{lr}$	**0.2375**	**0.2531**	**0.2619**	**0.2686**	**0.6640**	**0.6699**	**0.6736**	**0.6757**	**0.5393**	**0.5503**	**0.5562**	**0.5601**
	SePH$_{klr+rnd}$[Img]	0.1882	0.2018	0.2099	0.2163	0.6263	0.6311	0.6332	0.6352	0.4640	0.4705	0.4756	0.4784
	SePH$_{klr+rnd}$[Txt]	0.2689	0.2815	0.2900	0.2994	0.6624	0.6686	0.6725	0.6752	0.5222	0.5310	0.5375	0.5417
	SePH$_{klr+rnd}$[Rand]	0.2415	0.2644	0.2815	0.2953	0.6597	0.6691	0.6745	0.6782	0.5256	0.5399	0.5496	0.5560
	SePH$_{klr+rnd}$[Equal]	0.2801	0.2954	0.3052	0.3161	0.6723	0.6787	0.6829	0.6856	0.5428	0.5509	0.5581	0.5626
	SePH$_{klr+rnd}$	**0.2835**	**0.3003**	**0.3099**	**0.3204**	**0.6727**	**0.6792**	**0.6833**	**0.6860**	**0.5450**	**0.5532**	**0.5605**	**0.5650**
	SePH$_{klr+km}$[Img]	0.1867	0.1989	0.2070	0.2149	0.6257	0.6298	0.6323	0.6346	0.4653	0.4726	0.4780	0.4795
	SePH$_{klr+km}$[Txt]	0.2698	0.2825	0.2871	0.2992	0.6630	0.6689	0.6721	0.6755	0.5267	0.5364	0.5432	0.5464
	SePH$_{klr+km}$[Rand]	0.2406	0.2640	0.2785	0.2950	0.6600	0.6685	0.6737	0.6783	0.5287	0.5443	0.5541	0.5585
	SePH$_{klr+km}$[Equal]	0.2805	0.2956	0.3027	0.3160	0.6729	0.6788	0.6824	0.6859	0.5455	0.5547	0.5620	0.5644
	SePH$_{klr+km}$	**0.2838**	**0.3009**	**0.3074**	**0.3207**	**0.6733**	**0.6793**	**0.6829**	**0.6864**	**0.5477**	**0.5568**	**0.5640**	**0.5666**
Text Query *v.s.* Image Database	SePH$_{linear}$[Img]	0.2158	0.2350	0.2481	0.2568	0.6335	0.6387	0.6415	0.6435	0.4999	0.5116	0.5169	0.5199
	SePH$_{linear}$[Txt]	0.5320	0.5489	0.5615	0.5667	0.7121	0.7213	0.7279	0.7304	0.6170	0.6301	0.6393	0.6457
	SePH$_{linear}$[Rand]	0.4561	0.5078	0.5377	0.5553	0.6961	0.7088	0.7173	0.7216	0.6065	0.6297	0.6448	0.6531
	SePH$_{linear}$[Equal]	0.5419	0.5595	0.5768	0.5828	**0.7198**	**0.7293**	**0.7361**	**0.7390**	0.6371	0.6511	0.6611	0.6669
	SePH$_{linear}$	**0.5431**	**0.5619**	**0.5809**	**0.5872**	0.7188	0.7285	0.7356	0.7385	**0.6375**	**0.6532**	**0.6633**	**0.6674**
	SePH$_{lr}$[Img]	0.2251	0.2444	0.2572	0.2645	0.6316	0.6368	0.6401	0.6421	0.4987	0.5085	0.5141	0.5173
	SePH$_{lr}$[Txt]	0.5415	0.5573	0.5742	0.5808	0.7075	0.7184	0.7250	0.7286	0.6111	0.6282	0.6371	0.6426
	SePH$_{lr}$[Rand]	0.4604	0.5131	0.5475	0.5643	0.6907	0.7044	0.7130	0.7184	0.6034	0.6269	0.6415	0.6494
	SePH$_{lr}$[Equal]	**0.5553**	**0.5731**	**0.5916**	**0.5996**	0.7173	0.7282	0.7345	0.7384	0.6287	0.6452	0.6543	**0.6597**
	SePH$_{lr}$	0.5531	0.5724	0.5888	0.5966	**0.7176**	**0.7283**	**0.7347**	**0.7385**	**0.6291**	**0.6455**	**0.6545**	**0.6597**
	SePH$_{klr+rnd}$[Img]	0.3916	0.4325	0.4520	0.4625	0.6521	0.6582	0.6621	0.6648	0.4966	0.5066	0.5134	0.5169
	SePH$_{klr+rnd}$[Txt]	0.5761	0.5884	0.5989	0.6035	0.7089	0.7167	0.7237	0.7271	0.6019	0.6159	0.6260	0.6303
	SePH$_{klr+rnd}$[Rand]	0.5675	0.6129	0.6397	0.6505	0.7021	0.7141	0.7231	0.7281	0.5960	0.6176	0.6324	0.6404
	SePH$_{klr+rnd}$[Equal]	0.6302	**0.6514**	**0.6643**	**0.6702**	0.7215	0.7294	0.7371	0.7405	0.6267	0.6398	0.6522	0.6575
	SePH$_{klr+rnd}$	**0.6310**	0.6512	0.6633	0.6692	**0.7216**	**0.7296**	**0.7372**	**0.7408**	**0.6283**	**0.6415**	**0.6530**	**0.6584**
	SePH$_{klr+km}$[Img]	0.3813	0.4194	0.4422	0.4522	0.6528	0.6583	0.6633	0.6655	0.5018	0.5118	0.5178	0.5216
	SePH$_{klr+km}$[Txt]	0.5762	0.5882	0.5991	0.6036	0.7118	0.7199	0.7273	0.7301	0.6148	0.6287	0.6377	0.6440
	SePH$_{klr+km}$[Rand]	0.5637	0.6107	0.6381	0.6485	0.7044	0.7162	0.7260	0.7304	0.6060	0.6282	0.6420	0.6508
	SePH$_{klr+km}$[Equal]	0.6304	0.6510	**0.6653**	**0.6709**	0.7246	0.7326	0.7408	0.7435	0.6359	0.6494	0.6601	0.6661
	SePH$_{klr+km}$	**0.6310**	**0.6516**	0.6652	0.6701	**0.7247**	**0.7328**	**0.7410**	**0.7437**	**0.6378**	**0.6513**	**0.6612**	**0.6674**

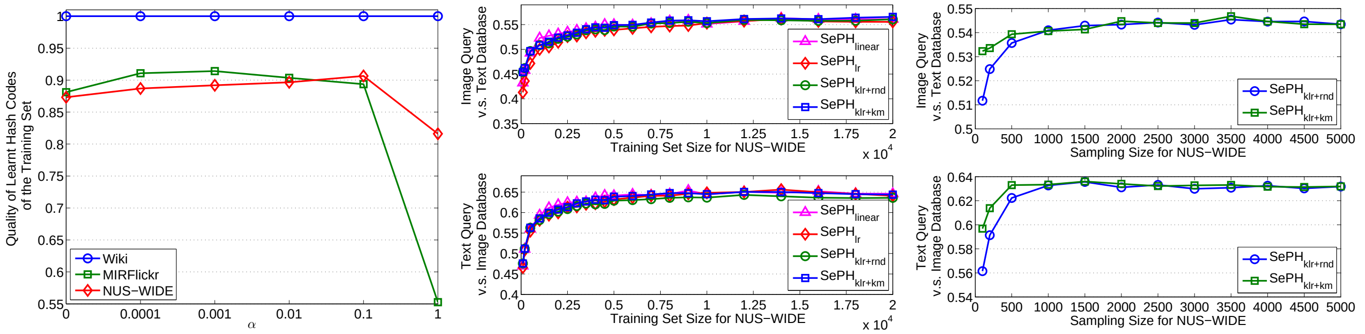
i.e. image and text, for comparison, we introduce the following baselines with other strategies. 1) SePH$_\bullet$[Img]: using the predicted hash code from the image view as the unified one, 2) SePH$_\bullet$[Txt]: using the predicted hash code from the text view as the unified one, 3) SePH$_\bullet$[Rand]: randomly taking -1 or 1 for a bit when predicted values from different views conflict, 4) SePH$_\bullet$[Equal]: using the proposed approach but setting $p(\mathbf{c}_k = 1) = p(\mathbf{c}_k = -1)$ for all bits in formula (18) and (19), which is used in our previous work [40]. Here SePH$_\bullet$ stands for SePH$_{linear}$, SePH$_{lr}$, SePH$_{klr+rnd}$ or SePH$_{klr+km}$. And different combining strategies will result in different unified hash codes for instances in the retrieval sets. The experimental results are shown in Table IV. And we can observe that on all datasets with different hash code lengths, 1) SePH$_\bullet$ and SePH$_\bullet$[Equal] generally outperform SePH$_\bullet$[Img], SePH$_\bullet$[Txt] and SePH$_\bullet$[Rand], which well demonstrates the superiority of the proposed probabilistic approach for determining the unified hash codes of unseen instances, and 2) SePH$_\bullet$ generally outperforms SePH$_\bullet$[Equal], which demonstrates the reasonableness of making $p(\mathbf{c}_k = 1)$ and $p(\mathbf{c}_k = -1)$ dataset-dependent and the effectiveness of estimating them with relative frequencies of -1 and 1 in the corresponding bit of the learnt hash codes of training instances.

TABLE V
OFF-LINE TRAINING COSTS AND ON-LINE HASHING COSTS FOR COMPARED HASHING METHODS THAT USE LINEAR RIDGE REGRESSION AS HASH FUNCTIONS, IN TERMS OF SECOND.

		Wiki	MIRFlickr	NUS-WIDE
Off-line Training Costs	CMSSH [26]	367.506	1017.434	2040.297
	CVH [27]	0.982	5.366	18.657
	IMH [21]	8.564	92.149	95.987
	LSSH [24]	387.848	878.047	889.888
	CMFH [23]	3.510	35.115	26.956
	SCM-Orth [35]	0.015	0.078	1.264
	SCM-Seq [35]	3.588	9.500	187.824
	SePH$_{linear}$	47.066	198.511	202.801
On-line Hashing Costs	CMSSH [26]	0.015	0.109	2.262
	CVH [27]	0.015	0.140	2.278
	IMH [21]	0.031	0.251	3.869
	LSSH [24]	18.627	112.367	1252.604
	CMFH [23]	0.031	0.203	4.478
	SCM-Orth [35]	0.016	0.141	2.168
	SCM-Seq [35]	0.016	0.125	2.215
	SePH$_{linear}$	0.031	0.265	3.432

D. Comparison of Training and Hashing Costs

Apart from theoretical analyses, here we also conduct experiments to compare the off-line training costs and the on-line



(a) Effects of Model Parameter α (b) Effects of Training Set Size (c) Effects of Sampling Size

Fig. 5. Analyses on affecting factors. Sub-figure 5a illustrates the effects of the only model parameter α on the quality of the learnt hash codes of the training set in each dataset. Sub-figure 5b and 5c respectively illustrate the effects of the training set size on the performance of SePH and the effects of the sampling size for learning kernel logistic regression (*i.e.* hash functions) on the performance of SePH$_{klr}$. Both are conducted on the largest NUS-WIDE with the hash code length fixed as 16 bits.

hashing costs of the proposed SePH with those of baselines. Considering that most baselines utilize linear ridge regression as hash functions, here we only take SePH$_{linear}$, CMSSH [26], CVH [27], IMH [21], LSSH [24], CMFH [23], SCM-Orth and SCM-Seq [35] for comparison. Specifically, by fixing the hash code length as 128 bits to make the comparisons more significant, we perform each compared hashing method on Wiki, MIRFlickr and NUS-WIDE, and then measure its time costs for training and generating hash codes for all instances in each dataset. The experiments are conducted on a server with 2 Intel Xeon E5645 CPUs and 48GB RAM, with all compared methods run on Matlab 2014a. For simplicity, here we perform 100 iterations for optimizing the objective function of SePH on each dataset, which can well guarantee convergence. Experimental results are reported in Table V. Note that for SePH$_{linear}$, the training costs include those of learning the hash codes of training instances and those of learning view-specific hash functions. It can be seen that for off-line training, SePH$_{linear}$ generally costs more time than most baselines, but still costs significantly less than the boosting based CMSSH and the sparse coding based LSSH. As for on-line hashing, SePH$_{linear}$ costs slightly more time than most baselines, as it needs extra time to estimate the output probabilities. Meanwhile, its on-line hashing costs are still much lower than those of LSSH, which generally needs to perform sparse coding for view-specific features. Actually, on average SePH$_{linear}$ costs less than 0.1 millisecond for generating the hash code of an instance, which would generally be acceptable in real-world applications.

E. Effects of Model Parameters

In previous experiments, for training SePH, the only model parameter α in its objective function (*i.e.* formula (7)) is empirically set as 0.01. Here we further conduct experiments to analyse its effects. Actually, the effects of α on SePH come from its effects on the quality of the learnt hash codes of the training set. And thus in our experiments, by fixing the hash code length as 16 bits on each dataset and using identical initial values for $\hat{H}$, we vary α in $\{0, 10^{-4}, 10^{-3}, \ldots, 1\}$ to learn the hash codes of the corresponding training set. For each setting of α, the quality of learnt hash codes is measured with their cross-view retrieval performance on the training set. Like the experiments of converge analyses, considering that the retrieval performance of learnt hash codes on a training set in the two cross-view retrieval tasks would be equal, here we only report one, as illustrated in Fig. 5a. It can be observed that as α increases from 0 to 1, the quality of the learnt hash codes of the training sets in MIRFlickr and NUS-WIDE firstly increases and then decreases, while that *w.r.t* Wiki keeps unchanged with an optimal mAP value of 1.0. The reasonable experimental results show that an appropriate positive α can make the learnt real-valued hash code matrix $\hat{H}$ close to the optimal binary one H via reducing the quantization loss, while a large α can lead the KL-divergence term to be less optimized and thus disable the learnt hash codes to well preserve the semantic affinities. It should also be noticed that the empirical value 0.01 is near to the optimal settings for α on all datasets and it consistently yields superior performance than $\alpha = 0$.

F. Effects of Training Set Size

To analyse how the training set size affects the performance of SePH, by fixing the hash code length as 16 bits, we increase the training set size of each dataset from 100 to $20,000$ ($2,000$ for Wiki and $14,000$ for MIRFlickr), and measure the corresponding cross-view retrieval performance of SePH on the query set for each size. The experimental results on the largest NUS-WIDE are illustrated in Fig. 5b. It can be seen that as the training set size increases, the performance of SePH, *i.e.* SePH$_{linear}$, SePH$_{lr}$, SePH$_{klr+rnd}$ and SePH$_{klr+km}$, keeps increasing and finally tends to converge. Actually, on NUS-WIDE, when the training set size increases to around $3,000$, the performance of SePH begins to converge. Considering that a training set size of 3000 is less than 2% of the retrieval set size, the experimental results well demonstrate that SePH is capable of exploiting the limited supervised information of a dataset. And thus it can be applicable for large-scale datasets, since SePH can be well trained with only the supervised information of a small fraction. Similar experimental results can also be observed on Wiki and MIRFlickr, as provided in the supplementary material.

G. Effects of Sampling Size for Kernel Logistic Regression

In previous experiments *w.r.t* SePH$_{klr}$, we empirically utilize a sampling size of 500 to learn kernel logistic regressions on all datasets. Here we further conduct experiments to investigate its effects. Similarly, we fix the hash code length as 16 bits. And for each dataset, with learnt hash codes of training instances, we increase the sampling size from 100 to $5,000$ ($2,000$ for Wiki), and respectively utilize random sampling and k-means sampling for each size to learn the corresponding kernel logistic regressions as hash functions. Moreover, for each sampling size, we measure the cross-view retrieval performance on the query set with the hash codes generated by the corresponding learnt hash functions. Fig. 5c shows the experimental results on the largest NUS-WIDE, and we can see that the performance of SePH$_{klr}$, *i.e.* SePH$_{klr+rnd}$ and SePH$_{klr+km}$, firstly increases and then converges quickly as the sampling size increases. Actually, on NUS-WIDE, when the sampling size increases to around $1,000$, the performance of SePH$_{klr}$ begins to converge. Moreover, the empirical setting of sampling size in our experiments (*i.e.* 500) achieves more than 98% of the performance achieved by the largest sampling size (*i.e.* $5,000$), while its training and predicting costs, as theoretically analysed before, would be much lower. And thus it is reasonable to perform sampling for learning kernel logistic regression in SePH$_{klr}$. Additionally, we can observe that at small sampling sizes (*e.g.* 100), k-means sampling shows more significant superiority over random sampling. It is because that in those cases the sampled kernel feature vectors are not sufficient enough for spanning the to-be-learnt weighting vector and k-means sampling can probably select better ones.

V. Conclusions

In this paper, we propose a supervised cross-view hashing method termed SePH. For training, given the semantic affinities of training data, SePH firstly transforms them into a probability distribution and aims to approximate it with another one derived from to-be-learnt binary hash codes of training instances in Hamming space. Then with the hash codes learnt, any kind of effective predictive models can be learnt as hash functions in each view to project the corresponding features into binary hash codes, such as linear ridge regression, logistic regression and kernel logistic regression, *etc.* To perform out-of-sample extension, given an unseen instance, the learnt hash functions firstly predict view-specific hash codes and derive or estimate the corresponding output probabilities in each of its observed views, and then a novel probabilistic approach is utilized to determine a unified hash code. Experiments on three benchmark datasets show that SePH yields state-of-the-art performance for cross-view retrieval.

Acknowledgment

This research was supported by the National Natural Science Foundation of China (Grant No. 61271394 and 61571269). The authors would like to thank the anonymous reviewers for their valuable comments.

References

[1] Z. Jin, C. Li, Y. Lin, and D. Cai, “Density sensitive hashing,” *IEEE Transactions on Cybernetics*, vol. 44, no. 8, pp. 1362–1371, 2014. 1

[2] J. Song, Y. Yang, X. Li, Z. Huang, and Y. Yang, “Robust hashing with local models for approximate similarity search,” *IEEE Transactions on Cybernetics*, vol. 44, no. 7, pp. 1225–1236, 2014. 1

[3] J. H. Friedman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, Sep. 1977. 1

[4] A. W. Moore, “The anchors hierarchy: Using the triangle inequality to survive high dimensional data,” in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000. 1

[5] T. Liu, A. W. Moore, K. Yang, and A. G. Gray, “An investigation of practical approximate nearest neighbor algorithms,” in *Advances in Neural Information Processing Systems*, 2004. 1

[6] T. Liu, C. Rosenberg, and H. Rowley, “Clustering billions of images with large scale nearest neighbor search,” in *IEEE Workshop on Applications of Computer Vision*, 2007. 1

[7] A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” in *Proceedings of the 25th International Conference on Very Large Data Bases*, 1999. 1

[8] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in *Advances in Neural Information Processing Systems*, 2009. 1, 3

[9] D. Zhang, J. Wang, D. Cai, and J. Lu, “Self-taught hashing for fast similarity search,” in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010. 1, 2

[10] Y. Gong and S. Lazebnik, “Iterative quantization: A procrustean approach to learning binary codes,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2011. 1

[11] F. Shen, C. Shen, Q. Shi, A. Hengel, and Z. Tang, “Inductive hashing on manifolds,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013. 1

[12] G. Irie, Z. Li, X. Wu, and S. Chang, “Locally linear hashing for extracting non-linear manifolds,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 1

[13] P. Zhang, W. Zhang, W. Li, and M. Guo, “Supervised hashing with latent factor models,” in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2014. 1

[14] G. Lin, C. Shen, Q. Shi, A. Hengel, and D. Suter, “Fast supervised hashing with decision trees for high-dimensional data,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 1

[15] L. Chen, D. Xu, I.-H. Tsang, and X. Li, “Spectral embedded hashing for scalable image retrieval,” *IEEE Transactions on Cybernetics*, vol. 44, no. 7, pp. 1180–1190, 2014. 1

[16] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, and X. He, “Fast and accurate hashing via iterative nearest neighbors expansion,” *IEEE Transactions on Cybernetics*, vol. 44, no. 11, pp. 2167–2177, 2014. 1

[17] X. Liu, Y. Mu, D. Zhang, B. Lang, and X. Li, “Large-scale unsupervised hashing with shared structure learning,” *IEEE Transactions on Cybernetics*, vol. 45, no. 9, pp. 1811–1822, 2015. 1

[18] R. Ye and X. Li, “Compact structure hashing via sparse and similarity preserving embedding,” *IEEE Transactions on Cybernetics*, vol. PP, no. 99, pp. 1–1, 2015. 1

[19] W. W. Y. Ng, X. Tian, Y. Lv, D. S. Yeung, and W. Pedrycz, “Incremental hashing for semantic image retrieval in nonstationary environments,” *IEEE Transactions on Cybernetics*, vol. PP, no. 99, pp. 1–13, 2016. 1

[20] X. Zhu, Z. Huang, H. Shen, and X. Zhao, “Linear cross-modal hashing for efficient multimedia search,” in *Proceedings of the 21st ACM International Conference on Multimedia*, 2013. 1, 2

[21] J. Song, Y. Yang, Y. Yang, Z. Huang, and H. Shen, “Inter-media hashing for large-scale retrieval from heterogeneous data sources,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013. 1, 2, 8, 9, 11, 12

[22] Y. Zhen, Y. Gao, D.-Y. Yeung, H. Zha, and X. Li, “Spectral multimodal hashing and its application to multimedia retrieval,” *IEEE Transactions on Cybernetics*, vol. 46, no. 1, pp. 27–38, 2016. 1, 2, 3

[23] G. Ding, Y. Guo, and J. Zhou, “Collective matrix factorization hashing for multimodal data,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 1, 2, 3, 5, 8, 9, 11, 12

[24] J. Zhou, G. Ding, and Y. Guo, “Latent semantic sparse hashing for cross-modal similarity search,” in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2014. 1, 2, 3, 8, 9, 11, 12

[25] L. Xie, J. Shen, and L. Zhu, "Online cross-modal hashing for web image retrieval," in *AAAI Conference on Artificial Intelligence*, 2016. 1, 2, 3

[26] M. Bronstein, A. Bronstein, F. Michel, and N. Paragios, "Data fusion through cross-modality metric learning using similarity-sensitive hashing," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010. 1, 3, 5, 8, 9, 11, 12

[27] S. Kumar and R. Udupa, "Learning hash functions for cross-view similarity search," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2011. 1, 2, 3, 8, 9, 11, 12

[28] Y. Zhen and D. Yeung, "Co-regularized hashing for multimodal data," in *Advances in Neural Information Processing Systems*, 2012. 1, 3

[29] ——, "A probabilistic model for multimodal hash function learning," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012. 1, 3

[30] D. Zhai, H. Chang, Y. Zhen, X. Liu, X. Chen, and W. Gao, "Parametric local multimodal hashing for cross-view similarity search," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2013. 1, 3

[31] J. Masci, M. Bronstein, A. Bronstein, and J. Schmidhuber, "Multimodal similarity-preserving hashing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 824–830, 2014. 1, 3

[32] Z. Yu, F. Wu, Y. Yang, Q. Tian, J. Luo, and Y. Zhuang, "Discriminative coupled dictionary hashing for fast cross-media retrieval," in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2014. 1, 3

[33] Z. Yu, Y. Zhang, S. Tang, Y. Yang, Q. Tian, and J. Luo, "Cross-media hashing with kernel regression," in *IEEE International Conference on Multimedia and Expo*, 2014. 1, 3

[34] J. Zhou, G. Ding, Y. Guo, Q. Liu, and X. Dong, "Kernel-based supervised hashing for cross-view similarity search," in *IEEE International Conference on Multimedia and Expo*, 2014. 1, 2, 3, 8, 9

[35] D. Zhang and W. Li, "Large-scale supervised multimodal hashing with semantic correlation maximization," in *AAAI Conference on Artificial Intelligence*, 2014. 1, 2, 3, 8, 9, 11, 12

[36] B. Wu, Q. Yang, W. Zheng, Y. Wang, and J. Wang, "Quantized correlation hashing for fast cross-modal search," 2015. 1, 3

[37] Q. Jiang and W. Li, "Deep cross-modal hashing," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1602.02255 1, 3

[38] T. Zhang and J. Wang, "Collaborative quantization for cross-modal similarity search," in *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 1, 3

[39] J. Tang, K. Wang, and L. Shao, "Supervised matrix factorization hashing for cross-modal retrieval," *IEEE Transactions on Image Processing*, vol. 25, no. 7, pp. 3157–3166, 2016. 1, 3

[40] Z. Lin, G. Ding, M. Hu, and J. Wang, "Semantics-preserving hashing for cross-view retrieval," in *2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015. 2, 7, 11

[41] L. V. and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 2579-2605, p. 85, 2008. 4

[42] D. Tao, X. Tang, X. Li, and X. Wu, "Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1088–1099, 2006. 5

[43] D. Hardoon, S. Szedmak, and J. Shawe-taylor, "Canonical correlation analysis: An overview with application to learning methods," *Neural Computation*, vol. 16, no. 12, pp. 2639–2664, 2004. 6

[44] M. Hu, Y. Chen, and J. Kwok, "Building sparse multiple-kernel svm classifiers," *IEEE Transactions on Neural Networks*, vol. 20, no. 5, pp. 827–839, 2009. 6

[45] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006. 7

[46] C. Xu, D. Tao, and C. Xu, "Large-margin multi-view information bottleneck," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1559–1572, 2014. 7

[47] ——, "Multi-view intact space learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 12, pp. 2531–2544, 2015. 7

[48] J. Pereira, E. Coviello, G. Doyle, N. Rasiwasia, G. Lanckriet, R. Levy, and N. Vasconcelos, "On the role of correlation and abstraction in cross-modal multimedia retrieval," *Transactions of Pattern Analysis and Machine Intelligence*, vol. 36, no. 3, pp. 521–535, 2014. 7

[49] M. Huiskes and M. Lew, "The mir flickr retrieval evaluation," in *Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*, 2008. 7

[50] T. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "Nus-wide: A real-world web image database from national university of singapore," in *Proceedings of the ACM International Conference on Image and Video Retrieval*, 2009. 7

[51] I. Jolliffe, *Principal Component Analysis*. Springer-Verlag New York, 1986, vol. 487. 8

[52] J. McDonald, *Handbook of Biological Statistics (3rd ed.)*. Sparky House Publishing, 2014. 9

Zijia Lin received his B.Sc. degree from School of Software, Tsinghua University, Beijing, China in 2011, and currently is a Ph.D. candidate in Department of Computer Science and Technology in the same campus. His research interests include multimedia information retrieval and machine learning.

Guiguang Ding received his Ph.D degree in electronic engineering from XiDian University. He is currently an associate professor of School of Software, Tsinghua University. Before joining School of Software in 2006, he worked as a postdoctoral researcher in Automation Department of Tsinghua University. His current research centers on the area of multimedia information retrieval and mining, in particular, visual object classification, automatic semantic annotation, content-based multimedia indexing, and personal recommendation. He has published about 40 research papers in international conferences and journals and applied for 18 Patent Rights in China.

Jungong Han received the Ph.D. degree in telecommunication and information system from XiDian University, Xian, China, in 2004. During his Ph.D. study, he spent one year with the Internet Media Group of Microsoft Research Asia, Beijing, China. From 2005 to 2010, he was with Signal Processing Systems Group, Technical University of Eindhoven, Eindhoven, The Netherlands. In 2010, he joined the Multiagent and Adaptive Computation Group, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands. In 2012, he served as a Senior Scientist with Civolution Technology, Eindhoven (a combining synergy of Philips Content Identification and Thomson STS). His current research interests include multimedia content identification, multisensor data fusion, and computer vision. He has authored and co-authored over 70 papers.

Jianmin Wang graduated from Peking University, China, in 1990, and received the ME and Ph.D. degrees in computer software from Tsinghua University, China, in 1992 and 1995, respectively. He is a full professor in the School of Software, Tsinghua University. His research interests include unstructured big data management, workflow and BPM technology, and large-scale data analytics. He has published 100 papers in major journals and conferences, including TKDE, DMKD, WWWJ, SIGMOD, VLDB, ICDE, SIGKDD, SIGIR, AAAI, CVPR, and ICCV. He led to develop a product data/lifecycle management system, which has been implemented in hundreds of enterprises in China. He leads to develop an unstructured big data management system, LaUDMS.