Learning to Hash for Big Data: A Tutorial

Wu-Jun Li

LAMDA Group
National Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University, P. R. China

Dec 11, 2015
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference
Nearest Neighbor Search (Retrieval)

- Given a query point \( q \), return the points closest (similar) to \( q \) in the database (e.g., image database).

- Underlying many machine learning, data mining, information retrieval problems.
Big Data

Big data has attracted much attention from both academia and industry.

- Facebook: 750 million users
- Flickr: 6 billion photos
- Wal-Mart: 267 million items/day; 4PB data warehouse
- Sloan Digital Sky Survey: New Mexico telescope captures 200 GB image data/day
Nearest Neighbor Search (NNS) for Big Data

Challenge in big data applications:
- Curse of dimensionality
- Storage cost
- Query speed
Similiarity Preserving Hashing

$h(\text{Statue of Liberty}) = 10001010$

$h(\text{Napoléon}) = 01100001$

$h(\text{Napoléon}) = 01100101$

Should be very different

Should be similar

flipped bit
Reduce Dimensionality and Storage Cost

10 million images → 20 GB → 512 values → Binary reduction → 160 MB
Fast Query Speed

- By using hash-code to construct index, we can achieve constant or sub-linear search time complexity.

- In some cases, exhaustive search with linear time complexity is also acceptable because the distance calculation cost is low with binary representation.
Two Stages of Hash Function Learning

Two main categories:

- **Category I:**
  - Projection Stage (Dimension Reduction)
    - Projected with real-valued projection function
    - Given a point $x$, each projected dimension $i$ will be associated with a real-valued projection function $f_i(x)$ (e.g., $f_i(x) = w_i^T x$)
  - Quantization Stage
    - Turn real into binary
    - Essential difference between metric learning and learning to hash

- **Category II:**
  - Binary-Code Learning Stage
  - Hash Function Learning Stage
Data-Independent Methods

The hash function family is defined independently of the training dataset.

- **Locality-sensitive hashing (LSH):** (Gionis et al., 1999; Andoni and Indyk, 2008) and its extensions (Datar et al., 2004; Kulis and Grauman, 2009; Kulis et al., 2009).

- **SIKH:** Shift invariant kernel hashing (SIKH) (Raginsky and Lazebnik, 2009).

- **MinHash** (Broder et al., 1998) and the extensions (Li and König, 2011)

Hash function: random projections or manually constructed.

They do not belong to learning to hash methods. Hence, this kind of methods will not be included in this tutorial.
Data-Dependent Methods

Hash functions are learned from a given training dataset.

- Compared with data-independent methods, data-dependent methods (also called learning to hash methods) can achieve comparable or even better accuracy with shorter binary codes.

Seminal papers: (Salakhutdinov and Hinton, 2007, 2009; Torralba et al., 2008; Weiss et al., 2008)

Two categories:

- Unimodal
  - Supervised methods
    given some supervised (semantic) information, such as pairwise labels $s_{ij}$, point-wise labels $y_i$ or triplet labels $(x_i, x_j, x_k)$
  - Unsupervised methods
- Multimodal
  - Supervised methods
  - Unsupervised methods
(Unimodal) Unsupervised Methods

No labels to denote the categories of the training points.

- **PCA-H**: principal component analysis.
- **SH**: (Weiss et al., 2008) eigenfunctions computed from the data similarity graph.
- **STH**: self-taught hashing (Zhang et al., 2010).
- **AGH**: anchor graph-based hashing (Liu et al., 2011).
- **ITQ**: (Gong and Lazebnik, 2011) orthogonal rotation matrix to refine the initial projection matrix learned by PCA.
- **IsoHash**: (Kong and Li, 2012b) orthogonal rotation matrix to make the variances of different directions isotropic (equal)
- **DGH**: (Liu et al., 2014) directly learn the discrete binary code
- **SGH**: (Jiang and Li, 2015) scalable graph hashing with feature transformation
(Unimodal) Supervised (semi-supervised) Methods

Class labels or pairwise constraints:

- **SSH (SPLH):** Semi-supervised hashing (Wang et al., 2010a,b) exploiting both labeled data and unlabeled data
- **MLH:** Minimal loss hashing (Norouzi and Fleet, 2011) based on the latent structural SVM framework
- **LDAHash:** Linear discriminant analysis based hashing (Strecha et al., 2012)
- **KSH:** Kernel-based supervised hashing (Liu et al., 2012)
- **LFH:** Latent factor models for supervised hashing (Zhang et al., 2014)
- **FastH:** (Lin et al., 2014) Supervised hashing using graph cut and decision trees
- **SDH:** (Shen et al., 2015) Supervised discrete hashing with point-wise labels
- **COSDISH:** (Kang et al., 2016) Scalable discrete hashing with pairwise supervision
The supervised information is ranking labels, such as triplets \((x_i, x_j, x_k)\).

- **HDML**: Hamming distance metric learning (Norouzi et al., 2012)
- **OPH**: Order preserving hashing for approximate nearest neighbor search (Wang et al., 2013b)
- **RSH**: Learning hash codes with listwise supervision (Wang et al., 2013a)
- **RPH**: Ranking preserving hashing for fast similarity search (Wang et al., 2015)
Multimodal Methods

- Multi-Source Hashing
- Cross-Modal Hashing
Multi-Source Hashing

- Aim at learning better codes than unimodal hashing by leveraging auxiliary views.
- Assume that all the views are provided for a query.

- **MFH**: Multiple feature hashing (Song et al., 2011)
- **CH**: Composite hashing (Zhang et al., 2011)
Cross-Modal Hashing

Given a query of either image or text, return images or texts similar to it.

- **CVH**: Cross view hashing (Kumar and Udupa, 2011)
- **MLBE**: Multimodal latent binary embedding (Zhen and Yeung, 2012a)
- **CRH**: Co-regularized hashing (Zhen and Yeung, 2012b)
- **IMH**: Inter-media hashing (Song et al., 2013)
- **RaHH**: Relation-aware heterogeneous hashing (Ou et al., 2013)
- **SCM**: Semantic correlation maximization (Zhang and Li, 2014)
- **CMFH**: Collective matrix factorization hashing (Ding et al., 2014)
- **QCH**: Quantized correlation hashing (Wu et al., 2015)
- **SePH**: Semantics-preserving hashing (Lin et al., 2015b)
Deep Hashing

Deep learning for hashing

- **CNNH**: Supervised hashing via image representation learning (Xia et al., 2014)
- **NINH**: Simultaneous feature learning and hash coding with deep neural networks (Lai et al., 2015)
- **DSRH**: Deep semantic ranking based hashing (Zhao et al., 2015)
- **DRSCH**: Bit-scalable deep hashing (Zhang et al., 2015)
- **DH**: Deep hashing for compact binary codes learning (Liong et al., 2015)
- **Deep learning of binary hash codes** (Lin et al., 2015a)
- **DPSH**: Feature learning based deep supervised hashing with pairwise labels (Li et al., 2015)
The quantization stage is at least as important as the projection stage:

- **DBQ**: Double-bit quantization (Kong and Li, 2012a)
- **MQ**: Manhattan quantization (Kong et al., 2012)
- **VBQ**: Variable bit quantization (Moran et al., 2013)
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference
Problem Definition

Input:
- Feature vectors: \(\{x_i\}\) (compact matrix form for all training points: \(X\)).

Output:
- Binary codes: \(\{b_i\}\) (compact matrix form for all training points: \(B\)). Instances similar in the original feature space should have similar binary codes.

or

- When \(x_i\) is close to \(x_j\), the Hamming distance between \(b_i\) and \(b_j\) should be low.
- When \(x_i\) is far away from \(x_j\), the Hamming distance between \(b_i\) and \(b_j\) should be high.
PCA Hashing (PCAH)

To generate a code of $m$ bits, PCAH performs PCA on $X$, and then use the top $m$ eigenvectors of the matrix $XX^T$ as columns of the projection matrix $W \in \mathbb{R}^{d \times m}$. Here, top $m$ eigenvectors are those corresponding to the $m$ largest eigenvalues $\{\lambda_k\}_{k=1}^m$, generally arranged with the non-increasing order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$. Let $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_m]^T$. Then

$$\Lambda = W^T XX^T W = \text{diag}(\lambda)$$

Define hash function

$$h(x) = \text{sgn}(W^T x)$$
Spectral Hashing (Weiss et al., 2008)

\[
\min_{\{y_i\}} \sum_{i,j} W_{ij} ||y_i - y_j||^2 \\
\text{subject to : } y_i \in \{-1, 1\}^k \\
\sum_i y_i = 0 \\
\frac{1}{n} \sum_i y_i y_i^T = I
\]

where \( W_{ij} \) is the similarity between \( x_i \) and \( x_j \), the constraint \( \sum_i y_i = 0 \) requires each bit to be fire 50\% of the time, and the constraint \( \frac{1}{n} \sum_i y_i y_i^T = I \) requires the bits to be uncorrelated.

NP hard problem!
Spectral Hashing (Weiss et al., 2008)

In matrix form, and by relaxation:

\[
\begin{align*}
\min & \quad \text{tr}(Y^T \mathcal{L} Y) \\
\text{subject to} & : Y^T \mathbf{1} = 0 \\
& \frac{1}{n} Y^T Y = \mathbf{I}
\end{align*}
\]

where \( Y \) is a real-valued \( n \times k \) matrix whose \( j \)th row is \( y_j^T \), \( \mathcal{L} = D - W \) is the Laplacian matrix, \( D \) is a diagonal matrix with \( D(i, i) = \sum_j W(i, j) \).

Solution: simply the \( k \) eigenvectors of \( \mathcal{L} \) with minimal eigenvalues after excluding the trivial eigenvector \( \mathbf{1} \) which has eigenvalue 0.

\( \text{sign}(Y) \) to get the binary codes (quantization stage).

Out-of-sample extension with eigenfunctions by simply fitting a multidimensional rectangle distribution to the data (by using PCA to align the axes, and then assuming a uniform distribution on each axis).
Spectral Hashing (Weiss et al., 2008)

Figure 5: Performance of different binary codes on the LabelMe dataset described in [3]. The data is certainly not uniformly distributed, and yet spectral hashing gives better retrieval performance than boosting and LSH.
Self-Taught Hashing (STH) (Zhang et al., 2010)

Figure 1: The proposed STH approach to semantic hashing.
Self-Taught Hashing (STH) (Zhang et al., 2010)

Figure 4: The precision-recall curve for retrieving original nearest neighbours.

Figure 5: The precision-recall curve for retrieving same-topic documents.
Anchor Graph Hashing (AGH) (Liu et al., 2011)

\[
\min \frac{1}{2} \sum_{i,j} ||Y_i - Y_j||^2 A_{ij} = \text{tr}(Y^T L Y)
\]

subject to: \(Y \in \{1, -1\}^{n \times r}, 1^T Y = 0, Y^T Y = nI_{r \times r}\)

where \(A_{ij}\) is the similarity between points \(i\) and \(j\).

The same objective function as that in SH.

Problem: the time complexity of direct eigen-decomposition is \(O(n^3)\).

Solution: K-means clustering to obtain \(m (m << n)\) cluster centers (anchors) \(U = \{u_j \in \mathbb{R}^d\}_{j=1}^m\)

\[
Z_{ij} = \begin{cases} 
\frac{\exp(-D^2(x_i, u_j)/t)}{\sum_{j' \in \langle i \rangle} \exp(-D^2(x_i, u_{j'}/t))}, & \forall j \in \langle i \rangle \\
0, & \text{otherwise}
\end{cases}
\]

where \(\langle i \rangle \subset [1 : m]\) denotes the \(s (s << m)\) nearest anchors of \(x_i\).
Anchor Graph Hashing (AGH) (Liu et al., 2011)

Anchor graph: \( \hat{A} = Z \Lambda Z^T \) where \( \Lambda = \text{diag}(Z^T 1) \).

The solution is the \( r \) graph Laplacian eigenvectors associated with the smallest eigenvalues, which are also eigenvectors of \( \hat{A} \) associated with the \( r \) largest eigenvalues (ignoring eigenvalue 1).

\[
Y = \sqrt{n} Z \Lambda^{-1/2} V \Sigma^{-1/2} = ZW
\]

where \( V \) and \( \Sigma \) are the eigenvectors and eigenvalues of the small \( m \times m \) matrix \( \Lambda^{-1/2} Z^T \Lambda^{-1/2} \).

The hash function: \( h(x) = \text{sign}(W^T z(x)) \)

Time complexity: Decrease from \( O(n^3) \) to \( O(nm^2) \).
Anchor Graph Hashing (AGH) (Liu et al., 2011)

Hierarchical quantization:

Figure 1. Hierarchical hashing on a data graph. $x_1, \cdots, x_8$ are data points and $y$ is a graph Laplacian eigenvector. The data points of filled circles take ‘1’ hash bit and the others take ‘-1’ hash bit. The entries with dark color in $y$ are positive and the others are negative. (a) The first-layer hash function $h^1$ uses threshold 0; (b) the second-layer hash functions $h^2$ use thresholds $b^+$ and $b^-$. 
### Anchor Graph Hashing (AGH) (Liu et al., 2011)

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST (70K)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAP</td>
<td>Train Time</td>
<td>Test Time</td>
</tr>
<tr>
<td></td>
<td>$r = 24$</td>
<td>$r = 48$</td>
<td>$r = 48$</td>
</tr>
<tr>
<td>$\ell_2$ Scan</td>
<td>0.4125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE $\ell_2$ Scan</td>
<td>0.5269</td>
<td>0.3909</td>
<td></td>
</tr>
<tr>
<td>LSH</td>
<td>0.1613</td>
<td>0.2196</td>
<td>1.8</td>
</tr>
<tr>
<td>PCAH</td>
<td>0.2596</td>
<td>0.2242</td>
<td>4.5</td>
</tr>
<tr>
<td>USPLH</td>
<td>0.4699</td>
<td>0.4930</td>
<td>163.2</td>
</tr>
<tr>
<td>SH</td>
<td>0.2699</td>
<td>0.2453</td>
<td>4.9</td>
</tr>
<tr>
<td>KLSH</td>
<td>0.2555</td>
<td>0.3049</td>
<td>2.9</td>
</tr>
<tr>
<td>SIKH</td>
<td>0.1947</td>
<td>0.1972</td>
<td>0.4</td>
</tr>
<tr>
<td>1-AGH</td>
<td>0.4997</td>
<td>0.3971</td>
<td>22.9</td>
</tr>
<tr>
<td>2-AGH</td>
<td>0.6738</td>
<td>0.6410</td>
<td>23.2</td>
</tr>
<tr>
<td>BRE</td>
<td>0.2638</td>
<td>0.3090</td>
<td>57.9</td>
</tr>
</tbody>
</table>
Iterative Quantization (ITQ) (Gong and Lazebnik, 2011)

\[
\max \text{tr}(W^T X X^T W)
\]
\[
\text{subject to}: W^T W = I
\]

It is just PCA. If \( W \) is a solution, then \( \tilde{W} = WR \) is also a solution where \( R \) is an orthogonal rotation matrix.

Let \( V = XW \) denote the projected data, and \( B \) denote the binary code. ITQ tries to minimize the following objective function:

\[
Q(B, R) = \|B - VR\|
\]

Learn by iterating the following two steps:

- Fix \( R \) and update \( B \): \( B = \text{sign}(VR) \);
- Fix \( B \) and update \( R \): first compute the SVD of \( B^T V \) as \( S\Omega\hat{S}^T \), then let \( R = \hat{S}\hat{S}^T \). It is the classic orthogonal Procrustes problem.
Iterative Quantization (ITQ) (Gong and Lazebnik, 2011)

Figure 3. Comparative evaluation on CIFAR dataset. (a) Performance is measured by mean average precision (mAP) for retrieval using top 50 Euclidean neighbors of each query point as true positives. Refer to Figure 4 for the complete recall-precision curves for the state-of-the-art methods. (b) Performance is measured by the averaged precision of top $p$ ranked images for each query where ground truth is defined by semantic class labels. Refer to Figure 5 for the complete class label precision curves for the state-of-the-art methods.
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Problem:
All existing methods use the same number of bits for different projected dimensions with different variances.

Possible Solutions:
- Different number of bits for different dimensions (Variable bit quantization (Moran et al., 2013))
- Isotropic (equal) variances for all dimensions: Isotropic hashing (IsoHash) (Kong and Li, 2012b)
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

To generate a code of $m$ bits, PCA hashing (PCAH) performs PCA on $X$, and then use the top $m$ eigenvectors of the matrix $XX^T$ as columns of the projection matrix $W \in \mathbb{R}^{d \times m}$. Here, top $m$ eigenvectors are those corresponding to the $m$ largest eigenvalues $\{\lambda_k\}_{k=1}^m$, generally arranged with the non-increasing order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$. Let $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_m]^T$. Then

$$\Lambda = W^TXX^TW = \text{diag}(\lambda)$$

Define hash function

$$h(x) = \text{sgn}(W^Tx)$$
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Weakness of PCA Hash:

Using the same number of bits for different projected dimensions is unreasonable because larger-variance dimensions will carry more information.

Solve it by making variances equal (isotropic)!
Isotropically Hashing (IsoHash) (Kong and Li, 2012b)

Idea of IsoHash:

- Learn an orthogonal matrix $Q \in \mathbb{R}^{m \times m}$ which makes $Q^T W^T X X^T W Q$ become a matrix with equal diagonal values.

- Effect of $Q$: to make each projected dimension has the same variance while keeping the Euclidean distances between any two points unchanged.
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Problem Definition:

\[
\text{tr}(Q^T W^T X X^T W Q) = \text{tr}(W^T X X^T W) = \text{tr}(\Lambda) = \sum_{i=1}^{m} \lambda_i
\]

\[
a = [a_1, a_2, \cdots, a_m] \text{ with } a_i = a = \frac{\sum_{i=1}^{m} \lambda_i}{m},
\]

and

\[
\mathcal{T}(z) = \{ T \in \mathbb{R}^{m \times m} | \text{diag}(T) = \text{diag}(z) \},
\]

Problem

The problem of IsoHash is to find an orthogonal matrix \( Q \) making \( Q^T W^T X X^T W Q \in \mathcal{T}(a) \).
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

IsoHash Formulation:

Because $Q^T \Lambda Q = Q^T [W^T X X^T W] Q$, let

$$\mathcal{M}(\Lambda) = \{ Q^T \Lambda Q | Q \in \mathcal{O}(m) \},$$

where $\mathcal{O}(m)$ is the set of all orthogonal matrices in $\mathbb{R}^{m \times m}$.

Then, the IsoHash problem is equivalent to:

$$||T - Z||_F = 0,$$

where $T \in \mathcal{T}(a)$, $Z \in \mathcal{M}(\Lambda)$, $|| \cdot ||_F$ denotes the Frobenius norm.
Lemma

[Schur-Horn Lemma (Horn, 1954)] Let \( \mathbf{c} = \{c_i\} \in \mathbb{R}^m \) and \( \mathbf{b} = \{b_i\} \in \mathbb{R}^m \) be real vectors in non-increasing order respectively, i.e.,

\[
c_1 \geq c_2 \geq \cdots \geq c_m, \quad b_1 \geq b_2 \geq \cdots \geq b_m.
\]

There exists a Hermitian matrix \( H \) with eigenvalues \( \mathbf{c} \) and diagonal values \( \mathbf{b} \) if and only if

\[
\sum_{i=1}^{k} b_i \leq \sum_{i=1}^{k} c_i, \quad \text{for any } k = 1, 2, \ldots, m,
\]

\[
\sum_{i=1}^{m} b_i = \sum_{i=1}^{m} c_i.
\]

So we can prove:
There exists a solution to the IsoHash problem. And this solution is in the intersection of \( \mathcal{T}(\mathbf{a}) \) and \( \mathcal{M}(\Lambda) \).
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Learning Methods:

Two methods: (Chu, 1995)

- Lift and projection (LP)

- Gradient Flow (GF)
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Learning Method: Lift and projection (LP)
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Learning Method: Gradient Flow

- **Objective function:**

  $$\min_{Q \in \mathcal{O}(m)} F(Q) = \frac{1}{2} \| \text{diag}(Q^T \Lambda Q) - \text{diag}(a) \|_F^2.$$  

- **The gradient $\nabla F$ at $Q$:**

  $$\nabla F(Q) = 2 \Lambda \beta(Q),$$

  where $\beta(Q) = \text{diag}(Q^T \Lambda Q) - \text{diag}(a)$.

- **The projection of $\nabla F(Q)$ onto $\mathcal{O}(m)$**

  $$g(Q) = Q[Q^T \Lambda Q, \beta(Q)]$$

  where $[A, B] = AB - BA$ is the Lie bracket.
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Learning Method: Gradient Flow

The vector field $\dot{Q} = -g(Q)$ defines a steepest descent flow on the manifold $O(m)$ for function $F(Q)$. Letting $Z = Q^T \Lambda Q$ and $\alpha(Z) = \beta(Q)$, we get

$$\dot{Z} = [Z, [\alpha(Z), Z]],$$

where $\dot{Z}$ is an isospectral flow that moves to reduce the objective function $F(Q)$. 
Isotropic Hashing (IsoHash) (Kong and Li, 2012b)

Accuracy (mAP):

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># bits</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td>IsoHash</td>
<td>0.2249</td>
</tr>
<tr>
<td>PCAH</td>
<td>0.0319</td>
</tr>
<tr>
<td>ITQ</td>
<td><strong>0.2490</strong></td>
</tr>
<tr>
<td>SH</td>
<td>0.0510</td>
</tr>
<tr>
<td>SIKH</td>
<td>0.0353</td>
</tr>
<tr>
<td>LSH</td>
<td>0.1052</td>
</tr>
</tbody>
</table>
Isotropichash (IsoHash) (Kong and Li, 2012b)

Training Time:

![Graph showing training time for IsoHash-GF, IsoHash-LP, ITQ, SH, SIKH, LSH, PCAH, and PCAH. The x-axis represents the number of training data, and the y-axis represents the training time in seconds. The graph shows a clear increase in training time as the number of training data increases.]
Discrete Graph Hashing (DGH) (Liu et al., 2014)

\[
\min_B \frac{1}{2} \sum_{ij}^{n} ||b_i - b_j||^2 A_{ij}^o = \text{tr}(B^T L^o B)
\]

subject to: \(B \in \{1, -1\}^{n \times r}, 1^T B = 0, B^T B = n I_r\)

Problem
- Solving this problem in the discrete code space is NP hard.
- Relaxing the discrete constraints will lead to poor performance.

Solution
- Leverage the anchor graph \(A\) like AGH.
- Soften the constraint \(1^T B = 0\) and the constraint \(B^T B = n I_r\).
Discrete Graph Hashing (DGH) (Liu et al., 2014)

Define a set $\Omega = \{Y \in \mathbb{R}^{n \times r} | 1^T Y = 0, Y^T Y = nI_r \}$

$$\max_B \text{tr}(B^T AB) - \frac{\rho}{2} \text{dist}^2(B, \Omega)$$

subject to: $B \in \{1, -1\}^{n \times r}$

where $\text{dist}(B, \Omega) = \min_{Y \in \Omega} \|B - Y\|_F$

Learn by iterating the following two steps:

- **B-subproblem:**
  $$\max_{B \in \{-1, +1\}^{n \times r}} f(B) := \text{tr}(B^T AB) + \rho \text{tr}(Y^T B)$$

- **Y-subproblem:**
  $$\max_{Y \in \mathbb{R}^{n \times r}} \text{tr}(B^T Y)$$

subject to: $1^T Y = 0, Y^T Y = nI_r$
Discrete Graph Hashing (DGH) (Liu et al., 2014)

B-subproblem

- Use an iterative ascent procedure called Signed Gradient Method
- Repeat \( B^{k+1} = sgn(C(2AB^k + \rho Y, B^k)) \) until \( B^k \) converges.
- Where
  \[
  C(x, y) = \begin{cases} 
  x, & x \neq 0 \\
  y, & x = 0 
  \end{cases}
  \]

Y-subproblem

- Solve Y-subproblem by using a closed-form solution:
  \[
  Y = \sqrt{n} \left[ \begin{array}{cc}
  U & \bar{U} \\
  V & \bar{V}
  \end{array} \right], \text{ where } U\Sigma V^T = (I_n - \frac{1}{n}11^T)B
  \]
### Table 1: Hamming ranking performance on YouTube Faces and Tiny-1M.

$r$ denotes the number of hash bits used in the hashing methods. All training and test times are in seconds.

<table>
<thead>
<tr>
<th>Method</th>
<th>YouTube Faces</th>
<th>Tiny-1M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean Precision / Top-20K</td>
<td>TrainTime</td>
</tr>
<tr>
<td>r = 48</td>
<td>r = 96</td>
<td>r = 128</td>
</tr>
<tr>
<td>$\ell_2$ Scan</td>
<td>0.7591</td>
<td>–</td>
</tr>
<tr>
<td>LSH</td>
<td>0.0830</td>
<td>0.1005</td>
</tr>
<tr>
<td>KLSH</td>
<td>0.3982</td>
<td>0.5210</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.7017</td>
<td>0.7493</td>
</tr>
<tr>
<td>IsoH</td>
<td>0.6093</td>
<td>0.6962</td>
</tr>
<tr>
<td>SH</td>
<td>0.5897</td>
<td>0.6655</td>
</tr>
<tr>
<td>MDSH</td>
<td>0.6110</td>
<td>0.6752</td>
</tr>
<tr>
<td>IMH</td>
<td>0.3150</td>
<td>0.3641</td>
</tr>
<tr>
<td>1-AGH</td>
<td>0.7138</td>
<td>0.7571</td>
</tr>
<tr>
<td>2-AGH</td>
<td>0.6727</td>
<td>0.7377</td>
</tr>
<tr>
<td>BRE</td>
<td>0.5564</td>
<td>0.6238</td>
</tr>
<tr>
<td>DGH-I</td>
<td>0.7086</td>
<td>0.7644</td>
</tr>
<tr>
<td>DGH-R</td>
<td>0.7245</td>
<td>0.7672</td>
</tr>
</tbody>
</table>
Discrete Graph Hashing (DGH) (Liu et al., 2014)

(a) Hash lookup F-measure @ CIFAR-10

(b) Hash lookup F-measure @ SUN397
Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

Problem

- The memory cost and time complexity are at least $O(n^2)$ for graph hashing if all pairwise similarities are explicitly computed.
- How to utilize the whole graph and avoid $O(n^2)$ complexity?

Scalable Graph Hashing (SGH)

- A feature transformation (Shrivastava and Li, 2014) method to effectively approximate the whole graph without explicitly computing it.
- A sequential method for bit-wise complementary learning.
- Linear complexity.
Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

Objective function

$$\min_W \|c \tilde{S} - \text{sgn}(K(X)W^T)\text{sgn}(K(X)W^T)^T\|^2_F$$

s.t. \(WK(X)^T K(X)W^T = I\)

\(\forall x, \text{define: } K(x) = \left[\phi(x, x_1) - \sum_{i=1}^n \phi(x_i, x_1)/n, \ldots, \phi(x, x_m) - \sum_{i=1}^n \phi(x_i, x_m)/n\right]\)

\(\tilde{S}_{ij} = 2S_{ij} - 1 \in (-1, 1].\)

Notation

- \(X = \{x_1, \ldots, x_n\}^T \in \mathbb{R}^{n \times d}: n \text{ data points.}\)
- Pairwise similarity metric defined as: \(S_{ij} = e^{-\|x_i - x_j\|^2_F/\rho} \in (0, 1]\)
Feature transformation

- ∀x, define \( P(x) \) and \( Q(x) \):

\[
P(x) = \left[ \sqrt{\frac{2(e^2 - 1)}{e\rho}} e^{\frac{-||x||^2_F}{\rho}} x; \sqrt{\frac{e^2 + 1}{e}} e^{\frac{-||x||^2_F}{\rho}} ; 1 \right]
\]

\[
Q(x) = \left[ \sqrt{\frac{2(e^2 - 1)}{e\rho}} e^{\frac{-||x||^2_F}{\rho}} x; \sqrt{\frac{e^2 + 1}{e}} e^{\frac{-||x||^2_F}{\rho}} ; -1 \right]
\]

- ∀\( x_i, x_j \in X \)

\[
P(x_i)^T Q(x_j) = 2\left[ \frac{e^2 - 1}{2e} \times \frac{2x_i^T x_j}{\rho} + \frac{e^2 + 1}{2e} \right] e^{-\frac{||x_i||^2_F + ||x_j||^2_F}{\rho}} - 1
\]

\[\approx 2e^{-\frac{||x_i||^2_F - ||x_j||^2_F + 2x_i^T x_j}{\rho}} - 1 = \tilde{S}_{ij}\]


Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

- Here, we use an approximation: \( \frac{e^2-1}{2e} x + \frac{e^2+1}{2e} \approx e^x \)

We assume \(-1 \leq \frac{2}{\rho} x_i^T x_j \leq 1\). It is easy to prove that \( \rho = 2 \max\{\|x_i\|_F^2\}_{i=1}^n \) can make \(-1 \leq \frac{2}{\rho} x_i^T x_j \leq 1\).

Then we have \( \tilde{S} \approx P(X)^T Q(X) \)
Direct relaxation may lead to poor performance. We adopt a sequential learning strategy in a bit-wise complementary manner.

Residual definition:
\[ R_t = c\tilde{S} - \sum_{i=1}^{t-1} \text{sgn}(K(X)w_i)\text{sgn}(K(X)w_i)^T \]

Objective function:
\[
\min_{w_t} \|R_t - \text{sgn}(K(X)w_t)\text{sgn}(K(X)w_t)^T\|_F^2
\]
\[
s.t. \quad w_t^T K(X)^T K(X)w_t = 1
\]

By relaxation, we can get:
\[
\max_{w_t} \text{tr}(w_t^T K(X)^T R_t K(X)w_t)
\]
\[
s.t. \quad w_t^T K(X)^T K(X)w_t = 1
\]
Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

- Then we obtain a generalized eigenvalue problem:

\[ K(X)^T R_t K(X) w_t = \lambda K(X)^T K(X) w_t \]

- Key component:

\[ cK(X)^T \tilde{S} K(X) = cK(X)^T P(X)^T Q(X) K(X) \]

\[ = c[K(X)^T P(X)^T][Q(X)K(X)] \]

- Adopting \( R_t = c\tilde{S} - \sum_{i=1, i \neq t}^c \text{sgn}(K(X)w_i)\text{sgn}(K(X)w_i)^T \), we continue the sequential learning procedure.

- The information in \( \tilde{S} \) is fully used, but is not explicitly computed. Time complexity is decreased from \( O(n^2) \) to \( O(n) \).
Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

Algorithm 1 Sequential learning algorithm for SGH

Input: Feature vectors $\mathbf{X} \in \mathbb{R}^{n \times d}$; code length $c$; number of kernel bases $m$.
Output: Weight matrix $\mathbf{W} \in \mathbb{R}^{c \times m}$.

Procedure

- Construct $P(\mathbf{X})$ and $Q(\mathbf{X})$;
- Construct $K(\mathbf{X})$ based on the kernel bases, which are $m$ points randomly selected from $\mathbf{X}$;
- $A_0 = [K(\mathbf{X})^T P(\mathbf{X})^T] [Q(\mathbf{X}) K(\mathbf{X})]$;
- $A_1 = c A_0$;
- $Z = K(\mathbf{X})^T K(\mathbf{X}) + \gamma \mathbf{I}_d$;
- for $t = 1 \rightarrow c$ do
  - Solve the following generalized eigenvalue problem
    $A_t \mathbf{w}_t = \lambda Z \mathbf{w}_t$;
    $U = [K(\mathbf{X})^T \text{sgn}(K(\mathbf{X}) \mathbf{w}_t)] [K(\mathbf{X})^T \text{sgn}(K(\mathbf{X}) \mathbf{w}_t)]^T$;
    $A_{t+1} = A_t - U$;
- end for

$\hat{A}_0 = A_{c+1}$

Randomly permutate $\{1, 2, \cdots, c\}$ to generate a random index set $\mathcal{M}$;

for $t = 1 \rightarrow c$ do
  - $\hat{t} = \mathcal{M}(t)$;
  - $\hat{A}_0 = \hat{A}_0 + K(\mathbf{X})^T \text{sgn}(K(\mathbf{X}) \mathbf{w}_{\hat{t}}) \text{sgn}(K(\mathbf{X}) \mathbf{w}_{\hat{t}})^T K(\mathbf{X})$;
  - Solve the following generalized eigenvalue problem
    $\hat{A}_0 \mathbf{v} = \lambda Z \mathbf{v}$;
  - Update $\mathbf{w}_{\hat{t}} \leftarrow \mathbf{v}$
  - $\hat{A}_0 = \hat{A}_0 - K(\mathbf{X})^T \text{sgn}(K(\mathbf{X}) \mathbf{w}_{\hat{t}}) \text{sgn}(K(\mathbf{X}) \mathbf{w}_{\hat{t}})^T K(\mathbf{X})$;
end for
Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

Top-1k precision @TINY-1M.

<table>
<thead>
<tr>
<th>Method</th>
<th>32 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
<th>256 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGH</td>
<td>0.4697</td>
<td>0.5742</td>
<td>0.6299</td>
<td>0.6737</td>
<td>0.7357</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.4289</td>
<td>0.4782</td>
<td>0.4947</td>
<td>0.4986</td>
<td>0.5003</td>
</tr>
<tr>
<td>AGH</td>
<td>0.3973</td>
<td>0.4402</td>
<td>0.4577</td>
<td>0.4654</td>
<td>0.4767</td>
</tr>
<tr>
<td>DGH-I</td>
<td>0.3974</td>
<td>0.4536</td>
<td>0.4737</td>
<td>0.4874</td>
<td>0.4969</td>
</tr>
<tr>
<td>DGH-R</td>
<td>0.3793</td>
<td>0.4554</td>
<td>0.4871</td>
<td>0.4989</td>
<td>0.5276</td>
</tr>
<tr>
<td>PCAH</td>
<td>0.2457</td>
<td>0.2203</td>
<td>0.2000</td>
<td>0.1836</td>
<td>0.1421</td>
</tr>
<tr>
<td>LSH</td>
<td>0.2507</td>
<td>0.3575</td>
<td>0.4122</td>
<td>0.4529</td>
<td>0.5212</td>
</tr>
</tbody>
</table>

Top-1k precision @MIRFLICKR-1M.

<table>
<thead>
<tr>
<th>Method</th>
<th>32 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
<th>256 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGH</td>
<td>0.4919</td>
<td>0.6041</td>
<td>0.6677</td>
<td>0.6985</td>
<td>0.7584</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.5177</td>
<td>0.5776</td>
<td>0.5999</td>
<td>0.6096</td>
<td>0.6228</td>
</tr>
<tr>
<td>AGH</td>
<td>0.4299</td>
<td>0.4741</td>
<td>0.4911</td>
<td>0.4998</td>
<td>0.506</td>
</tr>
<tr>
<td>DGH-I</td>
<td>0.4299</td>
<td>0.4806</td>
<td>0.5001</td>
<td>0.5111</td>
<td>0.5253</td>
</tr>
<tr>
<td>DGH-R</td>
<td>0.4121</td>
<td>0.4776</td>
<td>0.5054</td>
<td>0.5196</td>
<td>0.5428</td>
</tr>
<tr>
<td>PCAH</td>
<td>0.2720</td>
<td>0.2384</td>
<td>0.2141</td>
<td>0.1950</td>
<td>0.1508</td>
</tr>
<tr>
<td>LSH</td>
<td>0.2597</td>
<td>0.3995</td>
<td>0.466</td>
<td>0.5160</td>
<td>0.6072</td>
</tr>
</tbody>
</table>
Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

(a) 64 bits @TINY-1M

(b) 64 bits @MIRFLICKR-1M
Unsupervised Hashing

Scalable Graph Hashing (SGH) (Jiang and Li, 2015)

Training time @TINY-1M. Here, \( t_1 = 1438.60 \)

<table>
<thead>
<tr>
<th>Method</th>
<th>32 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
<th>256 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGH</td>
<td>34.49</td>
<td>52.37</td>
<td>71.53</td>
<td>89.65</td>
<td>164.23</td>
</tr>
<tr>
<td>ITQ</td>
<td>31.72</td>
<td>60.62</td>
<td>89.01</td>
<td>149.18</td>
<td>322.06</td>
</tr>
<tr>
<td>AGH</td>
<td>18.60 + ( t_1 )</td>
<td>19.40 + ( t_1 )</td>
<td>20.08 + ( t_1 )</td>
<td>22.48 + ( t_1 )</td>
<td>25.09 + ( t_1 )</td>
</tr>
<tr>
<td>DGH-I</td>
<td>187.57 + ( t_1 )</td>
<td>296.99 + ( t_1 )</td>
<td>518.57 + ( t_1 )</td>
<td>924.08 + ( t_1 )</td>
<td>1838.30 + ( t_1 )</td>
</tr>
<tr>
<td>DGH-R</td>
<td>217.06 + ( t_1 )</td>
<td>360.18 + ( t_1 )</td>
<td>615.74 + ( t_1 )</td>
<td>1089.10 + ( t_1 )</td>
<td>2300.10 + ( t_1 )</td>
</tr>
<tr>
<td>PCAH</td>
<td>4.29</td>
<td>4.54</td>
<td>4.75</td>
<td>5.85</td>
<td>6.49</td>
</tr>
<tr>
<td>LSH</td>
<td>1.68</td>
<td>1.77</td>
<td>1.84</td>
<td>2.55</td>
<td>3.76</td>
</tr>
</tbody>
</table>

Training time @MIRFLICKR-1M. Here, \( t_2 = 1564.86 \)

<table>
<thead>
<tr>
<th>Method</th>
<th>32 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
<th>256 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGH</td>
<td>41.51</td>
<td>59.02</td>
<td>74.86</td>
<td>97.25</td>
<td>168.35</td>
</tr>
<tr>
<td>ITQ</td>
<td>36.17</td>
<td>64.61</td>
<td>89.50</td>
<td>132.71</td>
<td>285.10</td>
</tr>
<tr>
<td>AGH</td>
<td>17.99 + ( t_2 )</td>
<td>18.80 + ( t_2 )</td>
<td>20.30 + ( t_2 )</td>
<td>19.87 + ( t_2 )</td>
<td>21.60 + ( t_2 )</td>
</tr>
<tr>
<td>DGH-I</td>
<td>85.81 + ( t_2 )</td>
<td>143.68 + ( t_2 )</td>
<td>215.41 + ( t_2 )</td>
<td>352.73 + ( t_2 )</td>
<td>739.56 + ( t_2 )</td>
</tr>
<tr>
<td>DGH-R</td>
<td>116.25 + ( t_2 )</td>
<td>206.24 + ( t_2 )</td>
<td>308.32 + ( t_2 )</td>
<td>517.97 + ( t_2 )</td>
<td>1199.44 + ( t_2 )</td>
</tr>
<tr>
<td>PCAH</td>
<td>7.65</td>
<td>7.90</td>
<td>8.47</td>
<td>9.23</td>
<td>10.42</td>
</tr>
<tr>
<td>LSH</td>
<td>2.44</td>
<td>2.43</td>
<td>2.71</td>
<td>3.38</td>
<td>4.21</td>
</tr>
</tbody>
</table>
Comments on Unsupervised Methods

Although some methods are mainly for *unsupervised setting*, they can be adapted for supervised setting if the label information is available.

For example, we can also use the label information to construct the graph in SH and AGH to get the supervised counterparts.

However, these adapted unsupervised methods can not make effective use of both $X$ and $Y$ for training.

The *supervised methods* introduced later typically try to consider both $X$ and $Y$ for training.
Problem Definition

Input:
- Feature vectors: \( \{x_i\} \) (compact form: \( X \)).
- Class labels: \( \{y_i\} \) (compact form: \( Y \)).
  or Pairwise constraints: \( \{s_{ij}\} \) (Compact form: \( S = \{s_{ij}\} \))
  - \( s_{ij} = 1 \) if points \( i \) and \( j \) belong to the same class. must link
  - \( s_{ij} = 0 \) if points \( i \) and \( j \) belong to different classes. cannot link

Output:
- Binary codes: \( \{b_i\} \) (compact form: \( B \)).
  Instances with similar labels should have similar binary codes.
  or
  - When \( s_{ij} = 1 \), the Hamming distance between \( b_i \) and \( b_j \) should be low.
  - When \( s_{ij} = 0 \), the Hamming distance between \( b_i \) and \( b_j \) should be high.
Class labels or pairwise constraints:

- **SSH (SPLH):** Semi-supervised hashing (Wang et al., 2010a,b) exploiting both labeled data and unlabeled data
- **MLH:** Minimal loss hashing (Norouzi and Fleet, 2011) based on the latent structural SVM framework
- **LDAHash:** Linear discriminant analysis based hashing (Strecha et al., 2012)
- **KSH:** Kernel-based supervised hashing (Liu et al., 2012)
- **LFH:** Latent factor models for supervised hashing (Zhang et al., 2014)
- **FastH:** (Lin et al., 2014) Supervised hashing using graph cut and decision trees
- **SDH:** (Shen et al., 2015) Supervised discrete hashing with point-wise labels
- **COSDISH:** (Kang et al., 2016) Scalable discrete hashing with pairwise supervision
Semi-Supervised Hashing (SSH) (Wang et al., 2010a,b)

\[ J(W) = \frac{1}{2} \text{tr}\{W^T X_l S X_l^T W\} + \frac{\eta}{2} \text{tr}[W^T X X^T W] \]
\[ = \frac{1}{2} \text{tr}\{W^T [X_l S X_l^T + \eta X X^T] W\} \]
\[ = \frac{1}{2} \text{tr}\{W^T M W\} \]

where \( S \) contains the supervised information:

\[
S_{ij} = \begin{cases} 
1 : (x_i, x_j) \in M \\
-1 : (x_i, x_j) \in C \\
0 : otherwise
\end{cases}
\]
Semi-Supervised Hashing (SSH) (Wang et al., 2010a,b)

Sequential projection learning algorithm:

**Algorithm 1** Semi-supervised sequential projection learning for hashing (S3PLH)

**Input:** data $X$, pairwise labeled data $X_l$, initial pairwise labels $S_1$, length of hash codes $K$, constant $\alpha$

for $k = 1$ to $K$ do

- Compute adjusted covariance matrix:
  $$M_k = X_l S_k X_l^T + \eta XX^T$$

- Extract the first eigenvector $e$ of $M_k$ and set:
  $$w_k = e$$

- Update the labels from vector $w_k$:
  $$S_{k+1} = S_k - \alpha T \left( \tilde{S}^k, S_k \right)$$

- Compute the residual:
  $$X = X - w_k w_k^T X$$

end for
Semi-Supervised Hashing (SSH) (Wang et al., 2010a,b)

Figure 3. Results on MNIST dataset. a) MAP for different number of bits using Hamming ranking; b) Precision within Hamming radius 2 using hash lookup.
Supervised Hashing

Minimal Loss Hashing (MLH) (Norouzi and Fleet, 2011)

\[ b(x, w) = sign(Wx) \]
\[ \mathcal{L}(w) = \sum_{(i,j) \in S} L(b(x_i; w), b(x_j; w), s_{ij}) \]
\[ L(h, g, s) = \ell_\rho(||h - g||_H, s) \]

where \( \ell_\rho() \) is a hinge-like loss function:

\[ \ell_\rho(m, s) = \begin{cases} 
\max(m - \rho + 1, 0), & \text{for } s = 1 \\
\lambda \max(\rho - m + 1, 0), & \text{for } s = 0
\end{cases} \]

Formulated as a structural SVM problem.
Minimal Loss Hashing (MLH) (Norouzi and Fleet, 2011)
LDAHash (Strecha et al., 2012)

Given a positive-pair set \( P = \{(x, x')\} \) and a negative-pair set \( N = \{(x, x')\} \), learn the hash function parameterized by \( P \) and \( t \):

\[
y = \text{sign}(Px + t)
\]

\[
\Sigma_P = \mathbb{E}\{(x - x')(x - x')^T|P\}
\]

\[
\Sigma_N = \mathbb{E}\{(x - x')(x - x')^T|N\}
\]

**Linear Discriminant Analysis (LDA)**

\[
\tilde{L} \propto \text{tr}\left\{ P \Sigma_N^{-1/2} \Sigma_P \Sigma_N^{-T/2} P^T \right\}
\]

\[
= \text{tr}\left\{ P \Sigma_P \Sigma_N^{-1} P^T \right\} = \text{tr}\left\{ P \Sigma_R P^T \right\},
\]

**Difference of Covariances (DIF)**

\[
\tilde{L} = \text{tr}\left\{ P \Sigma_D P^T \right\}
\]

\[
\Sigma_D = \alpha \Sigma_P - \Sigma_N
\]

\[
P = (\Sigma_D)_m^{-1/2}
\]

\[
\min_{t_i} \mathbb{E}\{\text{sign}((p_i^T x + t_i)(p_i^T x' + t_i))|N\} - \alpha \mathbb{E}\{\text{sign}((p_i^T x + t_i)(p_i^T x' + t_i))|P\}
\]
LDAHash (Strecha et al., 2012)
Supervised Hashing with Kernels (KSH) (Liu et al., 2012)
Supervised Hashing with Kernels (KSH) (Liu et al., 2012)

\[ f(x) = \sum_{j=1}^{m} \left( \kappa(x_{(j)}, x) - \frac{1}{n} \sum_{i=1}^{n} \kappa(x_{(j)}, x_{i}) \right) a_j \]

\[ = a^T \tilde{k}(x), \]

\[ \tilde{k}(x) = [\kappa(x_{(1)}, x) - \mu_1, \ldots, \kappa(x_{(m)}, x) - \mu_m]^T, \]

\[ S_{ij} = \begin{cases} 
1, & (x_i, x_j) \in M \\
-1, & (x_i, x_j) \in C \\
0, & \text{otherwise}.
\end{cases} \]

\[ \min_{A \in \mathbb{R}^{m \times r}} Q(A) = \left\| \frac{1}{r} \text{sgn}(\tilde{K}_l A) (\text{sgn}(\tilde{K}_l A))^T - S \right\|_F^2 \]

where \( \tilde{K}_l = [\tilde{k}(x_1), \ldots, \tilde{k}(x_l)]^T \in \mathbb{R}^{l \times m} \)

and \( A = [a_1, \ldots, a_r] \in \mathbb{R}^{m \times r} \)

\[ R_{k-1} = rS - \sum_{t=1}^{k-1} \text{sgn}(\tilde{K}_l a_t^*) (\text{sgn}(\tilde{K}_l a_t^*))^T \]

\[ \left\| \text{sgn}(\tilde{K}_l a_k) (\text{sgn}(\tilde{K}_l a_k))^T - R_{k-1} \right\|_F^2 \]

\[ \max_{a_k} (\tilde{K}_l a_k)^T R_{k-1} (\tilde{K}_l a_k) \]

s.t. \( (\tilde{K}_l a_k)^T (\tilde{K}_l a_k) = l \)

\[ \tilde{K}_l^T R_{k-1} \tilde{K}_l a = \lambda \tilde{K}_l^T \tilde{K}_l a \]
Supervised Hashing with Kernels (KSH) (Liu et al., 2012)
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Motivation

Existing supervised methods:

- High training complexity
- Semantic information is poorly utilized
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Contribution

- A novel likelihood function based on latent factor models
- A learning algorithm with convergence guarantee
- A linear-time stochastic learning strategy for scalable training
- State-of-the-art accuracy
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Model

The likelihood on the observed similarity labels $S$ is defined as:

$$p(S \mid B) = \prod_{s_{ij} \in S} p(s_{ij} \mid B)$$

$$p(s_{ij} \mid B) = \begin{cases} a_{ij}, & s_{ij} = 1 \\ 1 - a_{ij}, & s_{ij} = 0 \end{cases}$$

$a_{ij}$ is defined as $a_{ij} = \sigma(\Theta_{ij})$ with:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\Theta_{ij} = \frac{1}{2} b_i^T b_j$$

Relationship between the Hamming distance and the inner product:

$$\text{dist}_H(b_i, b_j) = \frac{1}{2}(Q - b_i^T b_j) = \frac{1}{2}(Q - 2\Theta_{ij})$$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Hardness

The posteriori of $B$ can be computed as follows:

$$p(B \mid S) \sim p(S \mid B)p(B)$$

We can learn the optimal $B$ through maximum a posteriori (MAP) estimation.

However, directly optimizing on $B$ is an NP-hard problem. Thus we optimize $B$ through two stages:

1. **Relax** $B$ to be a real valued matrix $U \in \mathbb{R}^{N \times Q}$ (Latent factors). Then learn an optimal $U$ under the same probabilistic framework as that for $B$.

2. Get optimal $B$ from $U$ through some rounding techniques.
Relaxation

Re-defined $\Theta_{ij}$ as:

$$\Theta_{ij} = \frac{1}{2} U_{i*}^T U_{j*}$$

$p(S \mid B), p(B), p(B \mid S)$ becomes $p(S \mid U), p(U), p(U \mid S)$. Define a normal distribution of $p(U)$ as:

$$p(U) = \prod_{d=1}^{Q} \mathcal{N}(U_{*d} \mid 0, \beta I)$$

The log posteriori of $U$ can be derived as:

$$L = \log p(U \mid S) = \sum_{s_{ij} \in S} (s_{ij} \Theta_{ij} - \log(1 + e^{\Theta_{ij}})) - \frac{1}{2\beta} \|U\|_F^2 + c$$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Surrogate algorithm

- Choose an initial point $x_0$
- Construct a concave lower bound of $f(x)$ at $x_0$
- Maximize this lower bound (at $x_1$)
- Repeat
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Alternating projection

Optimize each row of $\mathbf{U}$ (denoted by $\mathbf{U}_{i*}$) at a time with other rows fixed using surrogate algorithm.

Gradient vector:

$$
\frac{\partial L}{\partial \mathbf{U}_{i*}^T} = \frac{1}{2} \sum_{j:s_{ij} \in S} (s_{ij} - a_{ij}) \mathbf{U}_{j*}^T + \frac{1}{2} \sum_{j:s_{ji} \in S} (s_{ji} - a_{ji}) \mathbf{U}_{j*}^T - \frac{1}{\beta} \mathbf{U}_{i*}^T
$$

Hessian matrix:

$$
\frac{\partial^2 L}{\partial \mathbf{U}_{i*}^T \partial \mathbf{U}_{i*}} = -\frac{1}{4} \sum_{j:s_{ij} \in S} a_{ij}(1 - a_{ij}) \mathbf{U}_{j*}^T \mathbf{U}_{j*} - \frac{1}{4} \sum_{j:s_{ji} \in S} a_{ji}(1 - a_{ji}) \mathbf{U}_{j*}^T \mathbf{U}_{j*} - \frac{1}{\beta} \mathbf{I}
$$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Alternating projection

Define $H_i$ as:

$$H_i = -\frac{1}{16} \sum_{j:s_{ij} \in S} U_j^T U_j^* - \frac{1}{16} \sum_{j:s_{ji} \in S} U_j^T U_j^* - \frac{1}{\beta} I$$

We can prove that:

$$\frac{\partial^2 L}{\partial U_{i*}^T \partial U_{i*}} \succeq H_i$$

$A \succeq B$ means $A - B$ is a positive semi-definite matrix.
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Surrogate function

Constructed lower bound of $L(U_{i*})$:

$$
\tilde{L}(U_{i*}) = L(U_{i*}(t)) + (U_{i*} - U_{i*}(t))\frac{\partial L}{\partial U_{i*}^T}(t)
$$

$$
+ \frac{1}{2}(U_{i*} - U_{i*}(t))H_i(t)(U_{i*} - U_{i*}(t))^T
$$

Update rule for $U_{i*}$: (by setting the gradient of $\tilde{L}(U_{i*})$ with respect to $U_{i*}$ to 0)

$$
U_{i*}(t+1) = U_{i*}(t) - \left(\frac{\partial L}{\partial U_{i*}^T}(t)\right)^T H_i(t)^{-1}
$$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Optimize $U$ (Cont’d)

**Algorithm 2** Optimizing $U$ using surrogate algorithm

**Input:** $X \in \mathbb{R}^{N \times D}$, $S = \{s_{ij}\}$, $Q, T \in \mathbb{N}^+$, $\beta, \varepsilon \in \mathbb{R}^+$

Initializing $U$ by performing PCA on $X$.

**for** $t = 1 \rightarrow T$ **do**

**for** $i = 1 \rightarrow N$ **do**

Compute $U_{i*}$ following the update rule.

end for

Compute $L$ with the updated $U$.

Terminate the iterative process when the change of $L$ is smaller than $\varepsilon$.

end for

**Output:** $U \in \mathbb{R}^{N \times Q}$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Rounding

The real values of $U$ are quantized into the binary codes of $B$ by taking their signs:

$$B_{ij} = \begin{cases} 1, & U_{ij} > 0 \\ -1, & \text{otherwise} \end{cases}$$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Out-of-sample extension

Goal: Compute the binary code $b$ for a query $x$ not in the training set in order to perform ANN search. Find $W \in \mathbb{R}^{D \times Q}$ that maps $x$ to $u$, then perform rounding on $u$ to obtain $b$:

$$u = W^T x$$

Solve $W$ using linear regression:

$$L_e = \|U - XW\|_F^2 + \lambda_e \|W\|_F^2$$

$$W = (X^T X + \lambda_e I)^{-1} X^T U$$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Complexity analysis

Total time cost:

- Optimize $U$: $\mathcal{O}(T(NQ^3 + |S|Q^2))$ ($T$ is the number of iterations)
- Rounding: $\mathcal{O}(NQ)$
- Out-of-sample extension: $\mathcal{O}(ND^2 + D^3 + NDQ)$ (reduced to $\mathcal{O}(ND^2)$ when $Q < D \ll N$)
- Query: $\mathcal{O}(DQ)$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Stochastic learning

- $O(N^2)$ for full $S$: both time cost and the memory cost are unacceptable

- Choose only a subset of $S$ for updating inspired by the idea used in stochastic gradient descent method

- For storage efficiency, compute only the needed subset of $S$ during each iteration instead of pre-computing the full $S$
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Stochastic learning

During each iteration, we randomly sample a subset of $S$ with $\mathcal{O}(NQ)$ entries. Then the time cost to update $\mathbf{U}$ by one iteration is reduced to $\mathcal{O}(NQ^3)$.
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Stochastic learning

Furthermore, if we choose the subset of $S$ by randomly selecting $O(Q)$ of its columns and rows, we can further reduce the time cost to $O(NQ^2)$ per iteration.
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Datasets

- **CIFAR-10**
  - 60,000 color images from 80M tiny image collection.
  - 512-D GIST feature vector.
  - manually classified into 10 mutually exclusive classes.
  - ground-truth neighbors defined based on whether share the same class label.

- **NUS-WIDE**
  - 269,648 images collected from Flickr.
  - 1134-D low-level feature vector.
  - manually assigned with some of the 81 concept tags.
  - ground-truth neighbors defined based on whether share the same tags.
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Baselines

- Locality-sensitive hashing (LSH)
- Shift-invariant kernels hashing (SIKH)
- Spectral hashing (SH)
- Anchor graph hashing (AGH)
- Principal component analysis based hashing (PCAH)
- Iterative quantization (ITQ)
- Minimal loss hashing (MLH)
- Kernel-based supervised hashing (KSH)
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Convergence of learning

Objective Function

MAP

Li (http://cs.nju.edu.cn/lwj)
Latent Factor Hashing (LFH) (Zhang et al., 2014)

MAP (CIFAR-10)
Latent Factor Hashing (LFH) (Zhang et al., 2014)

MAP (NUS-WIDE)

![Graph showing MAP values for different hashing methods against code length.](image-url)
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Training time (CIFAR-10)
Latent Factor Hashing (LFH) (Zhang et al., 2014)

MAP (Full supervised information)
Latent Factor Hashing (LFH) (Zhang et al., 2014)

Training time (full supervised information)

![Bar graph showing training time for different code lengths and methods]

- LFH–Full
- LFH–Stochastic
- KSH–Full
- MLH–Full

Li (http://cs.nju.edu.cn/lwj)
FastH (Lin et al., 2014)

Overall objective

\[
\min_{\mathbf{z} \in \{-1, 1\}^{m \times n}} \sum_{i=1}^{n} \sum_{j=1}^{n} |y_{ij}| \left( m y_{ij} - \sum_{k=1}^{m} z_{k, i} z_{k, j} \right)^2 ; \quad (3a)
\]

\[
\min_{\Phi(\cdot)} \sum_{k=1}^{m} \sum_{i=1}^{n} \delta(z_{k, j} = h_k(x_i)). \quad (3b)
\]

where \( y_{ij} \in \{-1, 0, 1\} \) is the pairwise label between points \( i \) and \( j \)

Optimization on a block in one bit

\[
\min_{z_{k, \mathcal{B}} \in \{-1, 1\}^{|\mathcal{B}|}} \sum_{i \in \mathcal{B}} u_i z_{k, i} + \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}} v_{ij} z_{k, i} z_{k, j}, \quad (8a)
\]

where, \( v_{ij} = -|y_{ij}| (k y_{ij} - \sum_{p=1}^{k-1} z^*_{p, i} z^*_{p, j}) \), \( u_i = -2 \sum_{j \notin \mathcal{B}} \hat{z}_{k, j} |y_{ij}| (k y_{ij} - \sum_{p=1}^{k-1} z^*_{p, i} z^*_{p, j}) \). \quad (8c)
### Algorithm 3: FastHash

**Input**: Training data points: \( \{x_1, \ldots, x_n\} \); Affinity matrix: \( Y \); bit length: \( m \); blocks: \( \{B_1, B_2, \ldots\} \).

**Output**: Hash functions: \( \Phi = [h_1, \ldots, h_m] \)

1. for \( k = 1, \ldots, m \) do
   2. Step-1: call Algorithm 2 to obtain binary codes of \( k \)-th bit;
   3. Step-2: train trees in (9) to obtain hash function \( h_k \);
   4. update the binary codes of \( k \)-th bit by the output of \( h_k \);

### Algorithm 2: Step 1: Block GraphCut for binary code inference

**Input**: Affinity matrix: \( Y \); bit length: \( k \); max inference iteration; blocks: \( \{B_1, B_2, \ldots\} \); binary codes: \( \{z_1, \ldots, z_{k-1}\} \).

**Output**: Binary codes of one bit: \( z_k \)

1. repeat
   2. Randomly permute all blocks;
   3. for each \( B_i \) do
      4. Solve the inference in (8a) on \( B_i \) using GraphCut;
   5. until max iteration is reached;
FastH (Lin et al., 2014)
Supervised Discrete Hashing (SDH) (Shen et al., 2015)

Objective

\[
\min_{B, W, F} \|Y - W^T B\|^2 + \lambda \|W\|^2 + \nu \|B - F(X)\|^2 \tag{7}
\]
\[
\text{s.t. } B \in \{-1, 1\}^{L \times n}.
\]

where \(Y = \{y_i\}_{i=1}^n \in \{0, 1\}^{C \times n}\) is the point-wise label matrix. \((y_{ki} = 1 \text{ if } x_i \text{ belongs to class } k \text{ and } y_{ki} = 0 \text{ otherwise})\)

Alternating optimization:

\[
P = (\phi(X)\phi(X)^T)^{-1}\phi(X)B^T. \tag{5}
\]

\[
W = (BB^T + \lambda I)^{-1}BY^T. \tag{8}
\]

\[
z = \text{sgn}(q - B^T W'v). \tag{15}
\]

\[
b_i = \text{sgn}(F'(x_i) + \frac{\delta}{2} \sum_{k=1}^C w^{(ki)}). \tag{22}
\]
Supervised Discrete Hashing (SDH) (Shen et al., 2015)

Algorithm 1 Supervised Discrete Hashing (SDH)

**Input:** Training data \( \{x_i, y_i\}_{i=1}^n \); code length \( L \); number of anchor points \( m \); maximum iteration number \( t \); parameters \( \lambda \) and \( \nu \).

**Output:** Binary codes \( \{b_i\}_{i=1}^n \in \{-1, 1\}^{L \times n} \); hash function \( H(x) = \text{sgn}(F(x)) \).

1. Randomly select \( m \) samples \( \{a_j\}_{j=1}^m \) from the training data and get the mapped training data \( \phi(x) \) via the RBF kernel function.
2. Initialize \( b_i \) as a \( \{-1, 1\}^L \) vector randomly, \( \forall i \).
3. Loop until converge or reach maximum iterations:

   - **G-Step:** Calculate \( W \) using equation (8) or multi-class SVM.
   - **F-Step:** Compute \( P \) using (5) to form \( F(x) \).
   - **B-Step:** For the \( \ell_2 \) loss, iteratively learn \( \{b_i\}_{i=1}^n \) bit by bit using the DCC method with equation (15); for the hinge loss, compute \( b_i \) by (22).
Supervised Hashing

Supervised Discrete Hashing (SDH) (Shen et al., 2015)

![Graph showing performance comparison of different hashing methods on CIFAR dataset. The x-axis represents code length (16, 32, 64, 96, 128), and the y-axis represents MAP (Mean Average Precision). The methods compared include SDH, FastHash, KSH, CCA–ITQ, SSH, BRE, and MLH. SDH performs the best across all code lengths.]
Supervised Discrete Hashing (SDH) (Shen et al., 2015)

The diagram illustrates the performance of different hashing methods on the NUSWIDE dataset. The x-axis represents the code length ranging from 16 to 128, while the y-axis shows the MAP (Mean Average Precision). The methods compared include SDH, KSH, CCA–ITQ, BRE, and MLH. The graph shows how MAP changes with different code lengths for each method.
Column Sampling based Discrete Supervised Hashing (COSDISH) (Kang et al., 2016)

Motivation

- Learning to hash is essentially a discrete optimization problem
- Most existing methods solve relaxed continuous problems
- FastH (Lin et al., 2014) illustrates better accuracy with discrete optimization, but it cannot utilize all training points due to high time complexity.

We propose a discrete supervised hashing method which can leverage all training points:
- COlumn Sampling based DIscrrete Supervised Hashing (COSDISH).
Column Sampling based Discrete Supervised Hashing (COSDISH) (Kang et al., 2016)

KSH (Liu et al., 2012), TSH (Lin et al., 2013), FastH (Lin et al., 2014) all optimize the following objective function:

$$\min_{B \in \{-1, 1\}^{n \times q}} \|qS - BB^T\|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} (qS_{ij} - B_{i*}B_{j*}^T)^2$$

where $S_{ij} \in \{-1, 1\}$ denotes the supervised label.

The inner product reflects the opposite of the Hamming distance:

$$\text{dist}_H(B_{i*}, B_{j*}) = (q - B_{i*}B_{j*}^T)/2$$
Column Sampling based Discrete Supervised Hashing (COSDISH) (Kang et al., 2016)

Column Sampling

In each iteration, we randomly choose a subset $\Omega$ of $\mathcal{N} = \{1, 2, ..., n\}$, and sample $|\Omega|$ columns of $\mathbf{S}$ with the column numbers indexed by $\Omega$. We use $\tilde{\mathbf{S}} \in \{-1, 1\}^{n*|\Omega|}$ to denote the sampled sub-similarity matrix.

Let $\Gamma = \mathcal{N} - \Omega$, we can split $\tilde{\mathbf{S}}$ and $\mathbf{B}$ into two parts:

$\Omega$ part: $\tilde{\mathbf{S}}^{\Omega} \in \{-1, 1\}^{\Omega \times |\Omega|}$, and $\mathbf{B}^{\Omega} \in \{-1, 1\}^{\Omega \times q}$

$\Gamma$ part: $\tilde{\mathbf{S}}^{\Gamma} \in \{-1, 1\}^{\Gamma \times |\Omega|}$, and $\mathbf{B}^{\Gamma} \in \{-1, 1\}^{\Gamma \times q}$

Based on sampled columns in each iteration, the objective can be reformulated as follows:

$$\min_{\mathbf{B}^{\Omega}, \mathbf{B}^{\Gamma}} \|q\tilde{\mathbf{S}}^{\Gamma} - \mathbf{B}^{\Gamma} [\mathbf{B}^{\Omega}]^T \|_F^2 + \|q\tilde{\mathbf{S}}^{\Omega} - \mathbf{B}^{\Omega} [\mathbf{B}^{\Omega}]^T \|_F^2.$$

Our learning strategy is to alternatively optimize $\mathbf{B}^{\Omega}$ and $\mathbf{B}^{\Gamma}$.
Update $\mathbf{B}^\Gamma$ with $\mathbf{B}^\Omega$ Fixed

$$\mathbf{B}^\Gamma = \text{sgn}(\tilde{\mathbf{S}}^\Gamma \mathbf{B}^\Omega)$$ reaches the minimum of $f_1(\cdot)$ which changes the $F$-norm to $L_1$ norm.

Theorem

*Suppose that $f_1(\mathbf{F}_1^*)$ and $f_2(\mathbf{F}_2^*)$ reach their minimum at the points $\mathbf{F}_1^*$ and $\mathbf{F}_2^*$, respectively. We have $f_2(\mathbf{F}_1^*) \leq 2q f_2(\mathbf{F}_2^*)$. 

Column Sampling based Discrete Supervised Hashing (COSDISH) (Kang et al., 2016)

Update $\mathbf{B}_\Omega$ with $\mathbf{B}_\Gamma$ Fixed
When $\mathbf{B}_\Gamma$ is fixed, the sub-problem of $\mathbf{B}_\Omega$ is given by:

$$
\min_{\mathbf{B}_\Omega \in \{-1,1\}^{|\Omega| \times q}} \| q\tilde{\mathbf{S}}^{\Gamma} - \mathbf{B}^{\Gamma}[\mathbf{B}_\Omega]^T \|_F^2 + \| q\tilde{\mathbf{S}}^{\Omega} - \mathbf{B}_\Omega[\mathbf{B}_\Omega]^T \|_F^2.
$$

We can transform the above problem to $q$ binary quadratic programming (BQP) problems. The optimization of the $k$th bit of $\mathbf{B}_\Omega$ is given by:

$$
\min_{\mathbf{b}^k \in \{-1,1\}^{|\Omega|}} [\mathbf{b}^k]^T \mathbf{Q}^{(k)} \mathbf{b}^k + [\mathbf{b}^k]^T \mathbf{p}^{(k)}
$$

where $\mathbf{b}^k$ denotes the $k$th column of $\mathbf{B}_\Omega$, and

$$
Q_{i,j}^{(k)} = -2(q\tilde{S}_{i,j}^{\Omega} - \sum_{m=1}^{k-1} b_i^m b_j^m), Q_{i,i}^{(k)} = 0,
$$

$$
p_i^{(k)} = -2 \sum_{l=1}^{|\Gamma|} B_{l,k}^{\Gamma} (q\tilde{S}_{l,i}^{\Gamma} - \sum_{m=1}^{k-1} B_{l,m}^{\Gamma} B_{i,m}^{\Omega})
$$
Out-of-Sample Extension
We have only learned binary codes for training points. How to learn a hash function for query points?

- Train $q$ binary classifiers as hash functions.

Classifiers:

- **Linear Classifier**  SPLH, LFH, COSDISH
- **Kernel Vector**  KSH, SDH
- **SVM**  TSH
- **CNN**  CNNH, NINH
- **Boosted Tree**  FastH, COSDISH\_BT
Column Sampling based Discrete Supervised Hashing (COSDISH) (Kang et al., 2016)

Accuracy (MAP)

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (60K)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8-bits</td>
</tr>
<tr>
<td>COSDISH</td>
<td>0.4986</td>
</tr>
<tr>
<td>SDH</td>
<td>0.2642</td>
</tr>
<tr>
<td>LFH</td>
<td>0.2908</td>
</tr>
<tr>
<td>TSH</td>
<td>0.2365</td>
</tr>
<tr>
<td>KSH</td>
<td>0.2334</td>
</tr>
<tr>
<td>SPLH</td>
<td>0.1588</td>
</tr>
<tr>
<td>COSDISH BT</td>
<td><strong>0.5856</strong></td>
</tr>
<tr>
<td>FastH</td>
<td>0.4230</td>
</tr>
</tbody>
</table>
Column Sampling based Discrete Supervised Hashing (COSDISH) (Kang et al., 2016)

Training Time

<table>
<thead>
<tr>
<th>Method</th>
<th>3K</th>
<th>10K</th>
<th>50K</th>
<th>100K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSDISH</td>
<td>5.6</td>
<td>8.0</td>
<td>33.7</td>
<td>67.7</td>
<td>162.2</td>
</tr>
<tr>
<td>SDH</td>
<td>3.9</td>
<td>11.8</td>
<td>66.2</td>
<td>126.9</td>
<td>248.2</td>
</tr>
<tr>
<td>LFH</td>
<td>14.3</td>
<td>16.3</td>
<td>27.8</td>
<td>40.8</td>
<td>85.9</td>
</tr>
<tr>
<td>TSH</td>
<td>922.2</td>
<td>27360</td>
<td>&gt;50000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>KSH</td>
<td>1104</td>
<td>4446</td>
<td>&gt;50000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SPLH</td>
<td>25.3</td>
<td>185</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>COSDISH_BT</td>
<td>60.2</td>
<td>69.1</td>
<td>228.3</td>
<td>422.6</td>
<td>893.3</td>
</tr>
<tr>
<td>FastH</td>
<td>172.3</td>
<td>291.6</td>
<td>1451</td>
<td>3602</td>
<td>-</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference
Ranking-based Methods

The supervised information is ranking labels, such as triplets \((x_i, x_j, x_k)\).

- **HDML**: Hamming distance metric learning (Norouzi et al., 2012)
- **OPH**: Order preserving hashing for approximate nearest neighbor search (Wang et al., 2013b)
- **RSH**: Learning hash codes with listwise supervision (Wang et al., 2013a)
- **RPH**: Ranking preserving hashing for fast similarity search (Wang et al., 2015)
Hamming Distance Metric Learning (Norouzi et al., 2012)

The objective is given by:

\[ L(w) = \sum_{(x, x^+, x^-) \in D} \ell_{\text{triplet}}(b(x; w), b(x^+, w); b(x^-, w)) + \frac{\lambda}{2} \|w\|_2^2 \]

discontinuous and non-convex!

Construct a continuous upper bound on the loss:

\[ \ell_{\text{triplet}}(b(x; w), b(x^+, w); b(x^-, w)) \leq \max_{g,g^+,g^-} \{ \ell_{\text{triplet}}(g, g^+, g^-) + g^T f(x; w) + g^{+T} f(x^+; w) + g^{-T} f(x^-; w) \} \]

\[ - \max_h \{ h^T f(x; w) \} - \max_{h^+} \{ h^{+T} f(x^+; w) \} - \max_{h^-} \{ h^{-T} f(x^-; w) \} \]

where,

\[ b(x; w) = \text{sign}(f(x; w)) \]
\[ = \arg \max_{h \in H} h^T f(x; w) \]
Hamming Distance Metric Learning (Norouzi et al., 2012)

To use the upper bound, we need to find the binary codes given by:

\[(\hat{g}, \hat{g}^+, \hat{g}^-) = \arg \max_{(g,g^+,g^-)} \{ \ell_{\text{triplet}}(g, g^+, g^-) + g^T f(x) + g^{+T} f(x^+) + g^{-T} f(x^-) \} \]

Assume \(d(g, g^+, g^-) = m\), \(m\) is an integer between \(-q\) and \(q\), \(q\) is code length. The problem is converted to

\[(\hat{g}, \hat{g}^+, \hat{g}^-) = \ell'(m) + \max_{g,g^+,g^-} \{ g^T f(x) + g^{+T} f(x^+) + g^{-T} f(x^-) \} \]

where \(\ell'(\alpha) = [\alpha - 1]_+\)

It is straightforward that the solution is the largest of those \(2q + 1\) maxima.
Hamming Distance Metric Learning (Norouzi et al., 2012)

Perceptron-like learning

Randomly initialize $w^{(0)}$, then use the following procedure to update $w^{(t+1)}$:
1. Select a random triplet $(x, x^+, x^-)$ from dataset D.
2. Compute $(\hat{h}, \hat{h}^+, \hat{h}^-) = (b(x; w^{(t)}), b(x^+; w^{(t)}), b(x^-; w^{(t)}))$
3. Compute $(\hat{g}, \hat{g}^+, \hat{g}^-)$
4. Update model parameters using

$$w^{(t+1)} = w^{(t)} + \eta \left[ \frac{\partial f(x)}{\partial w} (\hat{h} - \hat{g}) + \frac{\partial f(x^+)}{\partial w} (\hat{h}^+ - \hat{g}^+) + \frac{\partial f(x^-)}{\partial w} (\hat{h}^- - \hat{g}^-) - \lambda w^{(t)} \right]$$
Hamming Distance Metric Learning (Norouzi et al., 2012)

<table>
<thead>
<tr>
<th>Hashing, Loss</th>
<th>Distance</th>
<th>$k\text{NN}$</th>
<th>64 bits</th>
<th>128 bits</th>
<th>256 bits</th>
<th>512 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear, pairwise hinge [24]</td>
<td>H</td>
<td>7 NN</td>
<td>72.2</td>
<td>72.8</td>
<td>73.8</td>
<td>74.6</td>
</tr>
<tr>
<td>Linear, pairwise hinge</td>
<td>AH</td>
<td>8 NN</td>
<td>72.3</td>
<td>73.5</td>
<td>74.3</td>
<td>74.9</td>
</tr>
<tr>
<td>Linear, triplet ranking</td>
<td>H</td>
<td>2 NN</td>
<td>75.1</td>
<td>75.9</td>
<td>77.1</td>
<td>77.9</td>
</tr>
<tr>
<td>Linear, triplet ranking</td>
<td>AH</td>
<td>2 NN</td>
<td>75.7</td>
<td>76.8</td>
<td>77.5</td>
<td>78.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Baseline</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>One-vs-all linear SVM [6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Euclidean 3NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>One-vs-all linear SVM [6]</td>
<td>77.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Euclidean 3NN</td>
<td>59.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Recognition accuracy on the CIFAR-10 test set ($H \equiv$ Hamming, $AH \equiv$ Asym. Hamming).
Order Preserving Hashing (Wang et al., 2013b)

Assuming the binary code length is $m$, OPH tries to align the following two kinds of categories:

- $(C^h_0, C^h_1, \ldots, C^h_m)$: ordered categories according to the $m+1$ Hamming distance values.
- $(C^e_0, C^e_1, \ldots, C^e_m)$: target categories computed from the original space.

Two principles:

- **Category order alignment**: categories computed from the Hamming space and those from the original space are matched, i.e., $C^h_k = C^e_k$.
- **Bucket balance**: the points are uniformly distributed in all hash buckets.
The objective function:

\[
\min_{W} \frac{1}{N} \sum_{n=1}^{N} \sum_{i=0}^{m-1} L_{ni}(W) \\
\text{s.t. } w_k^T w_k = 1, \forall k = 1, \ldots, m
\]

where,

\[
L_{ni}(W) = \sum_{x' \in \mathcal{N}_{ni}^e} \phi_{\gamma_1} (d_{RH}(x_n, x') - i) + \lambda \sum_{x' \notin \mathcal{N}_{ni}^e} \phi_{\gamma_2} (i + 1 - d_{RH}(x_n, x'))
\]

\[
d_{RH}(x, x') = \| \phi_\beta(W^T x) - \phi_\beta(W^T x') \|_2^2
\]

\[
\phi_\alpha(z) = \frac{1}{1 + e^{-\alpha z}}
\]

**Solution:** Quadratic penalty, mini-batch gradient descent and active set are utilized.
Table 1: Comparison in terms of mean average precision (%). The best scores are in bold font. Our approach OPH obtains a significant gain on larger data sets (SIFT100K, SIFT1M, GIST1M) and comparable performance or superior on smaller data sets (Labelme, Peekaboom).

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Code Length</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OPH</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>33.94</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>44.36</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>56.85</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>23.00</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>34.14</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>45.42</td>
</tr>
<tr>
<td>GIST1M</td>
<td>32</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4.12</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>6.97</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>10.43</td>
</tr>
<tr>
<td>SIFT100K</td>
<td>32</td>
<td>12.40</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>25.17</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>37.41</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>51.57</td>
</tr>
<tr>
<td>SIFT1M</td>
<td>32</td>
<td>5.07</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>13.58</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>26.00</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>39.88</td>
</tr>
</tbody>
</table>
Ranking-based supervised hashing (RSH) defines a rank triplet $S_{mij}$:

$$S_{mij} = S(q_m; x_i, x_j) = \begin{cases} 
1 & : r^{q}_i < r^{q}_j \\
-1 & : r^{q}_i > r^{q}_j \\
0 & : r^{q}_i = r^{q}_j
\end{cases}$$

The loss function can be represented as:

$$L_H = - \sum_m \sum_{i,j} H(q_m)^T [H(x_i) - H(x_j)] S_{mij}$$
Ranking-based Hashing

Learning Hash Codes with Listwise Supervision (Wang et al., 2013a)

Utilizing a linear hash function, the relaxed objective function is given by:

$$\min_W - \sum_m \sum_{i,j} q_m^T WW^T (x_i - x_j) S_{mi,j}$$

$$s.t. \quad WW^T = I$$

Solution: augmented Lagrangian multiplier is utilized.
Learning Hash Codes with Listwise Supervision (Wang et al., 2013a)

Figure 3. Performance evaluation on NUSWIDE dataset using different number of hash bits. a) NDCG of the Hamming ranking; b) ACG within Hamming radius 3.
The NDCG objective is a non-convex non-smooth optimization problem, since it depends on the ranking positions of data.
Ranking Preserving Hashing (Wang et al., 2015)

Utilizing a tractable probabilistic framework with a logistic function to approximate it:

\[
\hat{\pi}_j(x_i) = 1 + E \left[ \sum_{k=1}^{n} I(c_{q_j}^T (c_k - c_i) > 0) \right] \\
= 1 + \sum_{k=1}^{n} \Pr(c_{q_j}^T (c_k - c_i) > 0) \\
= 1 + \sum_{k=1}^{n} \frac{1}{1 + \exp(-q_j^T W^T W(x_k - x_i))}
\]

The overall objective is given by:

\[
J(W) = - \sum_{j=1}^{m} \frac{1}{Z_j} \sum_{i=1}^{n} \frac{2r_i^j - 1}{\log(1 + \hat{\pi}_j(x_i))} + \alpha \|WW^T - I\|_F^2
\]

**Solution:** This guarantees the objective is smooth and differentiable. Then L-BFGS quasi-Newton method is applied to solve the optimization problem.
Ranking Preserving Hashing (Wang et al., 2015)

Figure 2: Performance evaluation on both datasets with different number of hashing bits. (a)-(b): NDCG@10 using Hamming Ranking. (c)-(d): ACG with Hamming radius 2 using Hash Lookup.
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference
Multimodal Hashing

Multi-Source Hashing:
- **MFH**: Multiple feature hashing (Song et al., 2011)
- **CH**: Composite hashing (Zhang et al., 2011)

Cross-Modal Hashing:
- **CVH**: Cross view hashing (Kumar and Udupa, 2011)
- **MLBE**: Multimodal latent binary embedding (Zhen and Yeung, 2012a)
- **CRH**: Co-regularized hashing (Zhen and Yeung, 2012b)
- **IMH**: Inter-media hashing (Song et al., 2013)
- **RaHH**: Relation-aware heterogeneous hashing (Ou et al., 2013)
- **SCM**: Semantic correlation maximization (Zhang and Li, 2014)
- **CMFH**: Collective matrix factorization hashing (Ding et al., 2014)
- **QCH**: Quantized correlation hashing (Wu et al., 2015)
- **SePH**: Semantics-preserving hashing (Lin et al., 2015b)
Multiple Feature Hashing (Song et al., 2011)
Multiple Feature Hashing (Song et al., 2011)

MFH define $v$ affinity matrices:

$$(A^g)_{pq} = \begin{cases} 
1, & \text{if } (x^g)_p \in \mathcal{N}_k((x^g)_q) \text{ or } (x^g)_q \in \mathcal{N}_k((x^g)_p) \\
0, & \text{else}
\end{cases}$$

The relaxed objective function is given by

$$\min_{Y,Y^g,W,b} \sum_{g=1}^{v} \sum_{p,q=1}^{n} (A^g)_{pq} \| (y^g)_p - (y^g)_q \|_F^2 + \gamma \sum_{g=1}^{v} \sum_{t=1}^{n} \| y_t - (y^g)_t \|_F^2 + \alpha (\| X^T W + 1b - Y \|_F^2 + \beta \| W \|_F^2)$$

s.t. $YY^T = I$
**Multiple Feature Hashing (Song et al., 2011)**

Table 2: Comparison of MAP and time on Combined Dataset

<table>
<thead>
<tr>
<th>Methods</th>
<th>MAP</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPH</td>
<td>0.5941</td>
<td>0.4907</td>
</tr>
<tr>
<td>GF</td>
<td>0.6466</td>
<td>1.3917</td>
</tr>
<tr>
<td>STH</td>
<td>0.7536</td>
<td>0.6439</td>
</tr>
<tr>
<td>MFH_lbp</td>
<td>0.7526</td>
<td>0.6445</td>
</tr>
<tr>
<td>MFH_hsv</td>
<td>0.8042</td>
<td>0.4508</td>
</tr>
<tr>
<td>MFH</td>
<td>0.8656</td>
<td>0.5533</td>
</tr>
</tbody>
</table>
Composite Hashing (Zhang et al., 2011)

Composite Hashing follows SH’s idea and it learns hashing codes from $M$ information sources.

For $t$-th source, the affinity matrix is defined by

$$S_{i,j}^{(t)} = \begin{cases} e^{-\frac{||x_{i}^{(t)}-x_{j}^{(t)}||^2}{\delta_{i,j}^2}} & \text{if } x_{i}^{(t)} \in N_k(x_{j}^{(t)}) \text{ or } x_{j}^{(t)} \in N_k(x_{i}^{(t)}) \\ 0 & \text{otherwise} \end{cases}$$

The overall objective function of Composite Hashing combines the similarity preservation part and the consistency part

$$\min_{Y,\tilde{W},\alpha} tr(C_1Y^T \sum_{t=1}^{M} \tilde{L}^{(t)}Y) + C_2||Y - \sum_{t=1}^{M} \alpha_t(\tilde{W})^T \tilde{X}^{(t)}||^2 + ||\tilde{W}||^2$$

s.t. $Y \in \{-1, 1\}^{K \times n}$

$Y \mathbf{1} = 0$, $YY^T = I$, $\alpha^T \mathbf{1} = 1$, $\alpha \geq 0$
Composite Hashing (Zhang et al., 2011)

Figure 3: Precision and Recall Curve on Reuters and the healthcare dataset, with fixed bit number 32. CHMIS-AW demonstrates its superior performance over the other three algorithms.
Cross View Hashing (CVH) (Kumar and Udupa, 2011)

CVH extends the SH model to cross view settings. The Hamming distance summed over all the views

\[
d_{ij} = \sum_{k=1}^{K} d(y_i^{(k)}, y_j^{(k)}) + \sum_{k=1}^{K} \sum_{k' > k} d(y_i^{(k)}, y_j^{(k')})
\]

The objective function of CVH is a generalization of SH

\[
\min \bar{d} = \sum_{i=1}^{n} \sum_{i=1}^{n} W_{ij} d_{ij} = \sum_{k=1}^{K} \text{Tr}(Y^{(k)} L' Y^{(k)T}) - 2 \sum_{k=1}^{K} \sum_{k' > k} \text{Tr}(Y^{(k)} W Y^{(k')})
\]

s.t. \( Y^{(k)} e = 0, \) for \( k = 1, \ldots, K \)
\[
\frac{1}{n} Y^{(k)} Y^{(k)T} = I_d, \text{ for } k = 1, \ldots, K
\]
\[
Y_{ij}^{(k)} \in \{-1, 1\}, \text{ for } k = 1, \ldots, K
\]
Cross View Hashing (CVH) (Kumar and Udupa, 2011)

Interesting special cases:

- $K = 1$

$$X^{(1)}LX^{(1)T}A^{(1)} = X^{(1)}X^{(1)T}A^{(1)}\Lambda$$

This formulation is similar in structure to Locality Preserving Indexing (Cai et al., 2007).

- $W = I_{n \times n}$

$$X^{(1)}X^{(2)T}A^{(2)} = X^{(1)}X^{(1)T}A^{(1)}\Lambda'$$

$$X^{(2)}X^{(1)T}A^{(1)} = X^{(2)}X^{(2)T}A^{(2)}\Lambda'$$

This is the generalized eigenvalue formulation of Canonical Correlation Analysis.
### Cross View Hashing (CVH) (Kumar and Udupa, 2011)

<table>
<thead>
<tr>
<th>Training Data</th>
<th>ENG-HIN</th>
<th>ENG-KAN</th>
<th>ENG-TAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENG-HIN</td>
<td>0.725</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ENG-KAN</td>
<td>-</td>
<td>0.712</td>
<td>-</td>
</tr>
<tr>
<td>ENG-TAM</td>
<td>-</td>
<td>-</td>
<td>0.71</td>
</tr>
<tr>
<td>ENG-HIN-KAN</td>
<td>0.716</td>
<td>0.722</td>
<td>-</td>
</tr>
<tr>
<td>ENG-HIN-KAN-TAM</td>
<td>0.69</td>
<td>0.678</td>
<td>0.708</td>
</tr>
</tbody>
</table>
MLBE proposed a probabilistic model for multimodal hash function learning.

Figure 1: Graphical model representation of MLBE
Multimodal Hashing

Multimodal Latent Binary Embedding (MLBE) (Zhen and Yeung, 2012a)

\[
p(S^x | U, W^x, \theta_x) = \prod_{i=1}^{I} \prod_{i'=1}^{I} \mathcal{N}(S_{ii'}^x | u_i^T W^x u_{i'}, \frac{1}{\theta_x}),
\]

\[
p(S^y | V, W^y, \theta_y) = \prod_{j=1}^{J} \prod_{j'=1}^{J} \mathcal{N}(S_{jj'}^y | v_j^T W^y v_{j'}, \frac{1}{\theta_y}),
\]

\[
p(S^{xy} | U, V, w) = \prod_{i=1}^{I} \prod_{j=1}^{J} \left[ \text{Bern}(S_{ij}^{xy} | \gamma(w u_i^T v_j)) \right]^{O_{ij}}
\]

\[
p(U_{ik} | \pi_{ik}) = \text{Bern}(U_{ik} | \pi_{ik}),
\]

\[
p(\pi_{ik} | \alpha_u, \beta_u) = \text{Beta}(\pi_{ik} | \alpha_u, \beta_u),
\]

\[
p(U | \alpha_u, \beta_u) = \prod_{i=1}^{I} \prod_{k=1}^{K} \text{Bern}(U_{ik} | \frac{\alpha_u}{\alpha_u + \beta_u}).
\]

\[
p(V | \alpha_v, \beta_v) = \prod_{j=1}^{J} \prod_{k=1}^{K} \text{Bern}(V_{jk} | \frac{\alpha_v}{\alpha_v + \beta_v}).
\]

\[
p(W^x | \phi_x) = \prod_{k=1}^{K} \prod_{d=k}^{K} \mathcal{N}(W_{kd}^x | 0, \frac{1}{\phi_x}),
\]

\[
p(W^y | \phi_y) = \prod_{k=1}^{K} \prod_{d=k}^{K} \mathcal{N}(W_{kd}^y | 0, \frac{1}{\phi_y}).
\]
Multimodal Latent Binary Embedding (MLBE) (Zhen and Yeung, 2012a)

Table 2: mAP comparison on Flickr

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$K = 8$</td>
</tr>
<tr>
<td>Image Query vs. Text Database</td>
<td>MLBE</td>
<td>0.6322</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.5361</td>
</tr>
<tr>
<td></td>
<td>CMSSH</td>
<td>0.5155</td>
</tr>
<tr>
<td>Text Query vs. Image Database</td>
<td>MLBE</td>
<td>0.5626</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.5260</td>
</tr>
<tr>
<td></td>
<td>CMSSH</td>
<td>0.5093</td>
</tr>
</tbody>
</table>
Multimodal Latent Binary Embedding (MLBE) (Zhen and Yeung, 2012a)
Co-Regularized Hashing (CRH) (Zhen and Yeung, 2012b)

CRH is a novel multimodal hash function learning method based on a boosted co-regularization framework. The objective function is

\[
\mathcal{O} = \frac{1}{I} \sum_{i=1}^{I} l^x_i + \frac{1}{J} \sum_{j=1}^{J} l^y_j + \gamma \sum_{n=1}^{N} \omega_n l^*_n + \frac{\lambda_x}{2} ||w_x||^2 + \frac{\lambda_y}{2} ||w_y||^2
\]

The intra-modality loss \( l^x_i \) and \( l^y_j \) are

\[
l^x_i = [1 - f(x_i)(w^T_x x_i)]_+ = [1 - |w^T_x x_i|]_+
\]

\[
l^y_j = [1 - g(y_j)(w^T_y y_j)]_+ = [1 - |w^T_y y_j|]_+
\]

The inter-modality loss \( l^*_n \) is

\[
l^*_n = s_n d_n^2 + (1 - s_n) \tau(d_n)
\]

where \( d_n = w^T_x x_{an} - w^T_y y_{bn} \), \( \tau \) is the smoothly clipped inverted squared deviation (SCISD) function.
Co-Regularized Hashing (CRH) (Zhen and Yeung, 2012b)

The CRH formulation can be solved by using CCCP to minimize the upper bound w.r.t. $w_x$

$$\mathcal{O}_x = \frac{\lambda_x ||w_x||^2}{2} + \gamma \sum_{n=1}^{N} \omega_n (s_n d_n^2 + (1 - s_n) \zeta_n^x) + \frac{1}{I} \sum_{i=1}^{I} l_i^x$$

where

$$\zeta_n^x = \tau_1 (d_n) - \tau_2 (d_n^{(t)}) - d_n^{(t)} x_{an}^T (w_x - w_x^{(t)})$$

$$d_n^{(t)} = (w_x^{(t)})^T x_{an} - w_y^T y_{bn}$$
## Co-Regularized Hashing (CRH) (Zhen and Yeung, 2012b)

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$K = 24$</td>
</tr>
<tr>
<td>Image Query vs. Text Database</td>
<td>CRH</td>
<td>0.5259 ± 0.0094</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.4717 ± 0.0035</td>
</tr>
<tr>
<td></td>
<td>CMSSH</td>
<td><strong>0.5287 ± 0.0123</strong></td>
</tr>
<tr>
<td>Text Query vs. Image Database</td>
<td>CRH</td>
<td>0.5364 ± 0.0021</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.4598 ± 0.0020</td>
</tr>
<tr>
<td></td>
<td>CMSSH</td>
<td>0.5029 ± 0.0321</td>
</tr>
</tbody>
</table>

**Table 2: mAP comparison on Flickr**
Co-Regularized Hashing (CRH) (Zhen and Yeung, 2012b)

Figure 2: Results on Flickr
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Supervised Multimodal Similarity Search
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Motivation

- Existing supervised methods are not scalable
- Train hash functions on large-scale multimodal dataset

Contribution

- Avoiding explicitly computing the pairwise similarity matrix.
  - Linear-time complexity w.r.t. the size of training data.
- A sequential learning method with closed-form solution to each bit.
  - No hyper-parameters and stopping conditions are needed.
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Notations

- \( n \) - the number of training entities (data points)
- \( d_x, d_y \) - the dimensions of feature space in each modality
- \( \{x_1, x_2, ..., x_n | x_i \in \mathbb{R}^{d_x}\}, \{y_1, y_2, ..., y_n | y_i \in \mathbb{R}^{d_y}\} \)
  - denote the feature vectors of the two modalities
- \( X \in \mathbb{R}^{n \times d_x}, Y \in \mathbb{R}^{n \times d_y} \)
  - feature vectors form the rows of the data matrix
- \( m \) - number of semantic categories
- \( \{l_1, l_2, ..., l_n | l_i \in \{0, 1\}^{m}\} \)
  - semantic labels, \( l_{i,k} = 1 \) denotes that the \( i \)th entity belongs to the \( k \)th semantic category
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Similarity Definition

Cosine similarity between the semantic label vectors. The similarity between the $i$th entity and the $j$th entity is defined as follows:

$$\tilde{S}_{ij} = \frac{l_i \cdot l_j}{\|l_i\|_2 \|l_j\|_2},$$

$$\tilde{L}_{ik} = \frac{l_{i,k}}{\|l_i\|_2}, \text{ then } \tilde{S} = \tilde{L}\tilde{L}^T$$

Perform element-wise linear transformation on $\tilde{S}$ to get our final semantic similarity matrix $S \in [-1, 1]^{n \times n}$

$$S = 2\tilde{S} - E = 2\tilde{L}\tilde{L}^T - 1_n 1_n^T$$
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Objective

The two hash functions should preserve the semantic similarity cross modalities. We try to reconstruct the semantic similarity matrix by the learned hash codes. Hence, the objective function of our model is to minimize the following square error:

$$\min_{f,g} \sum_{i,j} \left( \frac{1}{c} f(x_i) g(y_j) - S_{ij} \right)^2$$

s.t. \( \sum_i f(x_i)f(x_i)^T = nI_c \)

\( \sum_j g(y_j)g(y_j)^T = nI_c, \)

The constraints try to make the bits between different hash functions uncorrelated.
Multimodal Hashing

Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

In matrix form, we can rewrite the problem as follows

$$\min_{W_x, W_y} \| \text{sgn}(XW_x)\text{sgn}(YW_y)^T - cS \|_F^2$$

s.t. \(\text{sgn}(XW_x)^T\text{sgn}(XW_x) = nI_c\)

\(\text{sgn}(YW_y)^T\text{sgn}(YW_y) = nI_c.\)

With simple algebra, we can transform the objective function into the following form

$$\| \text{sgn}(XW_x)\text{sgn}(YW_y)^T - cS \|_F^2$$

$$= \text{tr}[(\text{sgn}(XW_x)\text{sgn}(YW_y)^T - cS) \times (\text{sgn}(XW_x)\text{sgn}(YW_y)^T - cS)^T]$$

$$= \text{tr}(\text{sgn}(XW_x)\text{sgn}(YW_y)^T\text{sgn}(YW_y)\text{sgn}(XW_x)^T)$$

$$- 2c \cdot \text{tr}(\text{sgn}(XW_x)^T S \text{sgn}(YW_y)) + \text{tr}(c^2 S^2)$$

$$= - 2c \cdot \text{tr}(\text{sgn}(XW_x)^T S \text{sgn}(YW_y)) + \text{const},$$
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Then, we can reformulate the problem as the following equivalent problem:

$$\max_{W_x, W_y} \text{tr} \left( \text{sgn}(XW_x)^T S \text{sgn}(YW_y) \right)$$

s.t. $\text{sgn}(XW_x)^T \text{sgn}(XW_x) = nI_c$

$\text{sgn}(YW_y)^T \text{sgn}(YW_y) = nI_c$
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Semantic Correlation Maximization

Since it is NP hard to solve the problem, we apply the spectral relaxation trick to our problem

\[
\begin{align*}
\max_{W_x,W_y} \quad & \text{tr}(W_x^T X^T SY W_y) \\
\text{s.t.} \quad & W_x^T X^T X W_x = I_c \\
\quad & W_y^T Y^T Y W_y = I_c.
\end{align*}
\]

The term \(X^T SY\) actually measures the correlation between the two modalities with respect to the semantic labels. This correlation is called semantic correlation.

The objective function will degenerate to the CCA formulation when \(S = I_n\).
Multimodal Hashing

Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Generalized Eigenvalue Problem

The problem is equivalent to a generalized eigenvalue problem
Let $C_{xy} = X^T S Y$, $C_{xx} = X^T X$, and $C_{yy} = Y^T Y$.

The optimal solution of $W_x$ is the eigenvectors corresponding to the $c$ largest eigenvalues of

$$C_{xy}C_{yy}^{-1}C_{xy}^T W_x = \Lambda^2 C_{xx} W_x$$

$W_y$ can be obtained by $W_y = C_{yy}^{-1}C_{xy}^T W_x \Lambda^{-1}$. 
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Solved perfectly? No!

The quantization loss from projection space to Hamming space should not be neglected by spectral relaxation.

In eigen-decomposition, larger-variance projected dimensions carry more information. Hence, using each eigenvector to generate one bit of hash code is not reasonable (Kong and Li, 2012b).
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Sequential Strategy

Assuming that the projection vectors $w_x^{(1)}, \ldots, w_x^{(t-1)}$ and $w_y^{(1)}, \ldots, w_y^{(t-1)}$ have been learned. To learn the next projection vectors $w_x^{(t)}$ and $w_y^{(t)}$, we define a residue matrix

$$R_t = cS - \sum_{k=1}^{t-1} sgn(Xw_x^{(k)})sgn(Yw_y^{(k)})^T.$$ 

Objective function can be written as

$$\min_{w_x^{(t)}, w_y^{(t)}} \left\| sgn(Xw_x^{(t)})sgn(Yw_y^{(t)})^T - R_t \right\|_F^2.$$
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

The above objective function can be solved as a generalized eigenvalue problem which can be got by substituting \( w_x(t) \), \( w_y(t) \), and \( R_t \) for \( W_x \), \( W_y \), and \( S \) in the original formulation.

Note that the semantic correlation can be still efficiently calculated in linear time

\[
C_{xy}^{(t)} = X^T R_t Y
\]

\[
= cX^T SY - \sum_{k=1}^{t-1} X^T \text{sgn}(Xw_x^{(k)}) \text{sgn}(Yw_y^{(k)})^T Y
\]

\[
= C_{xy}^{(t-1)} - X^T \text{sgn}(Xw_x^{(t-1)}) \text{sgn}(Yw_y^{(t-1)})^T Y,
\]
Algorithm 3 Learning Algorithm of SCM Hashing Method.

\[
\begin{align*}
C_{xy}^{(0)} & \leftarrow 2(X^T \tilde{L})(Y^T \tilde{L})^T - (X^T 1_n)(Y^T 1_n)^T; \\
C_{xy}^{(1)} & \leftarrow c \times C_{xy}^{(0)}; \\
C_{xx} & \leftarrow X^T X + \gamma I_{d_x}; \\
C_{yy} & \leftarrow Y^T Y + \gamma I_{d_y}; \\
& \text{for } t = 1 \rightarrow c \text{ do} \\
& \quad \text{Solving the following generalized eigenvalue problem} \\
& \quad C_{xy}^{(t)} C_{yy}^{-1} [C_{xy}]^T w_x = \lambda^2 C_{xx} w_x, \\
& \quad \text{we can obtain the optimal solution } w_x^{(t)} \text{ corresponding to the largest eigenvalue } \lambda_{\max}; \\
& \quad w_y^{(t)} \leftarrow \frac{C_{yy}^{-1} C_{xy}^T w_x^{(t)}}{\lambda_{\max}}; \\
& \quad h_x^{(t)} \leftarrow \text{sgn}(X w_x^{(t)}); \\
& \quad h_y^{(t)} \leftarrow \text{sgn}(Y w_y^{(t)}); \\
& \quad C_{xy}^{(t+1)} \leftarrow C_{xy}^{(t)} - (X^T \text{sgn}(X w_x^{(t)}))(Y^T \text{sgn}(Y w_y^{(t)}))^T; \\
& \quad \text{end for}
\end{align*}
\]
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Scalability

Table: Training time (in seconds) on NUS-WIDE dataset by varying the size of training set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Size</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
<th>3000</th>
<th>5000</th>
<th>10000</th>
<th>20000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCM</td>
<td></td>
<td>276</td>
<td>249</td>
<td>303</td>
<td>222</td>
<td>236</td>
<td>260</td>
<td>248</td>
<td>228</td>
<td>230</td>
</tr>
<tr>
<td>CCA</td>
<td></td>
<td>25</td>
<td>20</td>
<td>23</td>
<td>22</td>
<td>25</td>
<td>22</td>
<td>28</td>
<td>38</td>
<td>44</td>
</tr>
<tr>
<td>CVH</td>
<td></td>
<td>62</td>
<td>116</td>
<td>123</td>
<td>149</td>
<td>155</td>
<td>170</td>
<td>237</td>
<td>774</td>
<td>1630</td>
</tr>
<tr>
<td>CRH</td>
<td></td>
<td>68</td>
<td>253</td>
<td>312</td>
<td>515</td>
<td>760</td>
<td>1076</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MLBE</td>
<td></td>
<td>67071</td>
<td>126431</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Scalability

Figure: MAP on NUS-WIDE dataset by varying the size of training set.
Semantic Correlation Maximization (SCM) (Zhang and Li, 2014)

Table: MAP results on small-scale training set of NUS-WIDE

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$c = 16$</td>
</tr>
<tr>
<td>Image Query v.s. Text Database</td>
<td>SCM</td>
<td>0.4385</td>
</tr>
<tr>
<td></td>
<td>CCA</td>
<td>0.3625</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.3608</td>
</tr>
<tr>
<td></td>
<td>CRH</td>
<td>0.3957</td>
</tr>
<tr>
<td></td>
<td>MLBE</td>
<td>0.3697</td>
</tr>
<tr>
<td>Text Query v.s. Image Database</td>
<td>SCM</td>
<td>0.4273</td>
</tr>
<tr>
<td></td>
<td>CCA</td>
<td>0.3619</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.3640</td>
</tr>
<tr>
<td></td>
<td>CRH</td>
<td>0.3926</td>
</tr>
<tr>
<td></td>
<td>MLBE</td>
<td>0.3877</td>
</tr>
</tbody>
</table>

Table: MAP results on large-scale training set of NUS-WIDE

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$c = 16$</td>
</tr>
<tr>
<td>Image Query v.s. Text Database</td>
<td>SCM</td>
<td>0.5451</td>
</tr>
<tr>
<td></td>
<td>CCA</td>
<td>0.4078</td>
</tr>
<tr>
<td>Text Query v.s. Image Database</td>
<td>SCM</td>
<td>0.5147</td>
</tr>
</tbody>
</table>
|                             | CCA    | 0.4038      | 0.3934      | 0.3861
Collective Matrix Factorization Hashing (Ding et al., 2014)

Employ collective matrix factorization (CMF) to learn cross-view hash functions.

Figure 2. Framework of CMFH, illustrated with toy data.
Collective Matrix Factorization Hashing (Ding et al., 2014)

Decompose $X^{(1)}$ and $X^{(2)}$ jointly with the constraint $V_1 = V_2 = V$

$$\lambda \|X^{(1)} - U_1 V\|_F^2 + (1 - \lambda) \|X^{(2)} - U_2 V\|_F^2$$

The overall objective function contains the collective matrix factorization, the linear embedding and the regularization term:

$$\min_{U_1, U_2, P_1, P_2, V} G(U_1, U_2, P_1, P_2, V)$$

where

$$G = \lambda \|X^{(1)} - U_1 V\|_F^2 + (1 - \lambda) \|X^{(2)} - U_2 V\|_F^2$$

$$+ \mu (\|V - P_1 X^{(1)}\|_F^2 + \|V - P_2 X^{(2)}\|_F^2)$$

$$+ \gamma R(U_1, U_2, P_1, P_2, V)$$
Collective Matrix Factorization Hashing (Ding et al., 2014)

Mean Average Precision (MAP) on Wiki dataset and NUS-WIDE dataset.

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Wiki</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
<td>64 bits</td>
<td>128 bits</td>
<td>16 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>View1 to View2</td>
<td>CVH</td>
<td>0.2041</td>
<td>0.1604</td>
<td>0.1296</td>
<td>0.1308</td>
<td>0.3722</td>
<td>0.3632</td>
</tr>
<tr>
<td></td>
<td>IMH</td>
<td>0.2017</td>
<td>0.2119</td>
<td>0.2025</td>
<td>0.1926</td>
<td>0.4721</td>
<td>0.4716</td>
</tr>
<tr>
<td></td>
<td>CMSSH</td>
<td>0.2026</td>
<td>0.2113</td>
<td>0.2011</td>
<td>0.2084</td>
<td>0.4997</td>
<td>0.5189</td>
</tr>
<tr>
<td></td>
<td>CHMIS</td>
<td>0.2207</td>
<td>0.2189</td>
<td>0.2126</td>
<td>0.2035</td>
<td>0.4870</td>
<td>0.4896</td>
</tr>
<tr>
<td></td>
<td>CMFH</td>
<td>0.2538</td>
<td>0.2582</td>
<td>0.2619</td>
<td>0.2648</td>
<td>0.5591</td>
<td>0.5698</td>
</tr>
<tr>
<td>View2 to View1</td>
<td>CVH</td>
<td>0.2962</td>
<td>0.1944</td>
<td>0.1337</td>
<td>0.1125</td>
<td>0.4042</td>
<td>0.3991</td>
</tr>
<tr>
<td></td>
<td>IMH</td>
<td>0.4865</td>
<td>0.5295</td>
<td>0.4935</td>
<td>0.4628</td>
<td>0.4793</td>
<td>0.4800</td>
</tr>
<tr>
<td></td>
<td>CMSSH</td>
<td>0.2928</td>
<td>0.2732</td>
<td>0.2753</td>
<td>0.2814</td>
<td>0.5015</td>
<td>0.5103</td>
</tr>
<tr>
<td></td>
<td>CHMIS</td>
<td>0.2207</td>
<td>0.2189</td>
<td>0.2126</td>
<td>0.2035</td>
<td>0.4870</td>
<td>0.4896</td>
</tr>
<tr>
<td></td>
<td>CMFH</td>
<td>0.6116</td>
<td>0.6298</td>
<td>0.6398</td>
<td>0.6477</td>
<td>0.6614</td>
<td>0.6921</td>
</tr>
</tbody>
</table>
Quantized Correlation Hashing (Wu et al., 2015)

Contributions

- First attempt to integrate hash function learning with quantization together for cross-modal hashing
- Multi-modality objective function is transformed to a single-modality formulation

Framework

![Diagram of cross-modal hashing algorithm]

Figure 1: The framework of cross-modal hashing algorithm
Quantized Correlation Hashing (Wu et al., 2015)

Objective function

\[
\min O(B, W) = \|B - ZW\|_F^2 - \text{tr}(W^T Z^T \tilde{S} Z W)
\]

where \( W = \begin{bmatrix} W_x \\ W_y \end{bmatrix} \), \( \tilde{S} = \begin{bmatrix} \beta L_x & \alpha S \\ \alpha S^T & \beta L_y \end{bmatrix} \), \( Z = \begin{bmatrix} X \\ Y \end{bmatrix} \), \( B = \begin{bmatrix} B_x \\ B_y \end{bmatrix} \)

Learn hash function

Adopt an alternating optimization procedure to iteratively optimize \( W \) and \( B \)

- Optimize \( B \) using \( B = \text{sgn}(ZW) \) when fixing \( W \)
- Iteratively optimize \( W \) using Crank-Nicolson-like scheme (Smith, 1985) when fixing \( B \)
Quantized Correlation Hashing (Wu et al., 2015)

Precision-Recall curve on Wiki dataset and 58W-CIFAR dataset

Precision-Recall curves for Wiki image->text and Wiki text->image at 16 bits. (b) Wiki @ 16 bits

Precision-Recall curves for 58W-CIFAR image->text and 58W-CIFAR text->image at 8 bits. (e) 58W-CIFAR @ 8 bits
Semantics-Preserving Hashing (Lin et al., 2015b)

Training of SePH: 1) Learning semantics-preserving hash codes of the training data (red dotted rectangle), 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 using a novel probabilistic approach
Semantics-Preserving Hashing (Lin et al., 2015b)

Objective function: the KL-divergence between two distributions $\mathcal{P}$ and $\mathcal{Q}$

$$
\Psi = \min_{\hat{H} \in \mathbb{R}^{n \times d_c}} \sum_{i \neq j} p_{i,j} \log \frac{p_{i,j}}{q_{i,j}} + \frac{\alpha}{C} \|\|\hat{H}\| - I\|^2
$$

where $\hat{H}$ is the relaxed hash-code matrix, $p_{i,j} = \frac{A_{i,j}}{\sum_{i \neq j} A_{i,j}}$ with $A_{i,j}$ being the supervised affinity between points $i$ and $j$,

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

Learning algorithm:

- Utilize gradient descent based optimization methods.
- The gradient w.r.t. $\hat{H}_{i,\cdot}$ can be derived as follows

$$
\frac{\partial \Psi}{\partial \hat{H}_{i,\cdot}} = \sum_{j \neq i} (p_{i,j} - q_{i,j})(1 + \frac{1}{4}\|\hat{H}_{i,\cdot} - \hat{H}_{j,\cdot}\|^2_2)^{-1} (\hat{H}_{i,\cdot} - \hat{H}_{j,\cdot})
$$

$$
+ \frac{2\alpha}{C} (\|\hat{H}_{i,\cdot}\| - 1^T) \odot \sigma(\hat{H}_{i,\cdot})
$$
Semantics-Preserving Hashing (Lin et al., 2015b)

Learn hash functions
After getting the hash code, kernel logistic regression is used to learn the hash functions:

$$\Theta = \min_{w^{(k)}} \sum_{i=1}^{n} \log(1 + e^{-h^{(k)}_i \phi(X_i, \cdot)w^{(k)}}) + \lambda \|w^{(k)}\|_2^2$$

Generate hash codes
- For any unseen instance with only one view observed, utilize
  $$p(c_k^x = b|x) = (1 + e^{-b \phi(x)\hat{v}^T(k)})^{-1}$$
- For those with both views observed, utilize
  $$c_k = sgn(p(c_k = 1|x)p(c_k = 1|y) - p(c_k = -1|x)p(c_k = -1|y))$$
  to generate hash code.
## Semantics-Preserving Hashing (Lin et al., 2015b)

### Mean Average Precision (MAP):

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>MIRFlickr</th>
<th>NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
<td>64 bits</td>
</tr>
<tr>
<td>CMSSH [1]</td>
<td>0.1877</td>
<td>0.1771</td>
<td>0.1646</td>
</tr>
<tr>
<td>CVH [9]</td>
<td>0.1257</td>
<td>0.1212</td>
<td>0.1215</td>
</tr>
<tr>
<td>IMH [14]</td>
<td>0.1573</td>
<td>0.1575</td>
<td>0.1568</td>
</tr>
<tr>
<td>LSSH [25]</td>
<td>0.2141</td>
<td>0.2216</td>
<td>0.2218</td>
</tr>
<tr>
<td>CMFH [3]</td>
<td>0.2132</td>
<td>0.2259</td>
<td>0.2362</td>
</tr>
<tr>
<td>KSH-CV [26]</td>
<td>0.1965</td>
<td>0.1839</td>
<td>0.1701</td>
</tr>
<tr>
<td>SCM-Orth [19]</td>
<td>0.1598</td>
<td>0.1460</td>
<td>0.1383</td>
</tr>
<tr>
<td>SCM-Seq [19]</td>
<td>0.2210</td>
<td>0.2337</td>
<td>0.2442</td>
</tr>
<tr>
<td>SePH_{rnd}</td>
<td>0.2762</td>
<td>0.2965</td>
<td>0.3049</td>
</tr>
<tr>
<td>SePH_{km}</td>
<td>0.2787</td>
<td>0.2956</td>
<td>0.3064</td>
</tr>
<tr>
<td>Text Query v.s.</td>
<td>Wiki</td>
<td>MIRFlickr</td>
<td>NUS-WIDE</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
<td>64 bits</td>
</tr>
<tr>
<td>CMSSH [1]</td>
<td>0.1630</td>
<td>0.1617</td>
<td>0.1539</td>
</tr>
<tr>
<td>CVH [9]</td>
<td>0.1185</td>
<td>0.1034</td>
<td>0.1024</td>
</tr>
<tr>
<td>IMH [14]</td>
<td>0.1463</td>
<td>0.1311</td>
<td>0.1290</td>
</tr>
<tr>
<td>LSSH [25]</td>
<td>0.5031</td>
<td>0.5224</td>
<td>0.5293</td>
</tr>
<tr>
<td>CMFH [3]</td>
<td>0.4884</td>
<td>0.5132</td>
<td>0.5269</td>
</tr>
<tr>
<td>KSH-CV [26]</td>
<td>0.1710</td>
<td>0.1665</td>
<td>0.1696</td>
</tr>
<tr>
<td>SCM-Orth [19]</td>
<td>0.1553</td>
<td>0.1389</td>
<td>0.1262</td>
</tr>
<tr>
<td>SCM-Seq [19]</td>
<td>0.2134</td>
<td>0.2366</td>
<td>0.2479</td>
</tr>
<tr>
<td>SePH_{rnd}</td>
<td>0.6312</td>
<td>0.6581</td>
<td>0.6637</td>
</tr>
<tr>
<td>SePH_{km}</td>
<td>0.6318</td>
<td>0.6577</td>
<td>0.6646</td>
</tr>
</tbody>
</table>

|                   | 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 |

Table 2. Cross-view retrieval performance of the proposed SePH (i.e. SePH_{rnd} and SePH_{km}) and compared baselines on all benchmark datasets with different hash code lengths, in terms of mAP.
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference
Deep Hashing

Deep learning for hashing

- **CNNH**: Supervised hashing via image representation learning (Xia et al., 2014)
- **NINH**: Simultaneous feature learning and hash coding with deep neural networks (Lai et al., 2015)
- **DSRH**: Deep semantic ranking based hashing (Zhao et al., 2015)
- **DRSCH**: Bit-scalable deep hashing (Zhang et al., 2015)
- **DH**: Deep hashing for compact binary codes learning (Liong et al., 2015)
- Deep learning of binary hash codes (Lin et al., 2015a)
- **DPSH**: Feature learning based deep supervised hashing with pairwise labels (Li et al., 2015)
Supervised Hashing via Image Representation Learning (CNNH) (Xia et al., 2014)

Two-stage framework

Figure 1: Overview of the proposed two-stage method. In stage 1, the pairwise similarity matrix $S$ is decomposed into a product $HH^T$, where $H$ is a matrix of approximate target hash codes. In stage 2, we use a convolutional network to learn the feature representation for the images as well as a set of hash functions. The network consists of three convolution-pooling layers, a fully connected layer and an output layer. The output layer can be simply constructed with the learned hash codes in $H$ (the red nodes). If the image tags are available in training, one can add them in the output layer (the black nodes) so as to help to learn a better shared representation of the images. By inputting an test image to the trained network, one can obtain the desired hash code from the values of the red nodes in the output layer.
Supervised Hashing via Image Representation Learning (CNNH) (Xia et al., 2014)

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST (MAP) 12 bits</th>
<th>MNIST (MAP) 24 bits</th>
<th>MNIST (MAP) 32 bits</th>
<th>MNIST (MAP) 48 bits</th>
<th>CIFAR10 (MAP) 12 bits</th>
<th>CIFAR10 (MAP) 24 bits</th>
<th>CIFAR10 (MAP) 32 bits</th>
<th>CIFAR10 (MAP) 48 bits</th>
<th>NUS-WIDE (MAP) 12 bits</th>
<th>NUS-WIDE (MAP) 24 bits</th>
<th>NUS-WIDE (MAP) 32 bits</th>
<th>NUS-WIDE (MAP) 48 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNNH+</td>
<td>0.969</td>
<td>0.975</td>
<td>0.971</td>
<td>0.975</td>
<td>0.465</td>
<td>0.521</td>
<td>0.521</td>
<td>0.532</td>
<td>0.623</td>
<td>0.630</td>
<td>0.629</td>
<td>0.625</td>
</tr>
<tr>
<td>CNNH</td>
<td>0.957</td>
<td>0.963</td>
<td>0.956</td>
<td>0.960</td>
<td>0.439</td>
<td>0.511</td>
<td>0.509</td>
<td>0.522</td>
<td>0.611</td>
<td>0.618</td>
<td>0.625</td>
<td>0.608</td>
</tr>
<tr>
<td>KSH</td>
<td>0.872</td>
<td>0.891</td>
<td>0.897</td>
<td>0.900</td>
<td>0.303</td>
<td>0.337</td>
<td>0.346</td>
<td>0.356</td>
<td>0.556</td>
<td>0.572</td>
<td>0.581</td>
<td>0.588</td>
</tr>
<tr>
<td>ITQ-CCA</td>
<td>0.659</td>
<td>0.694</td>
<td>0.714</td>
<td>0.726</td>
<td>0.264</td>
<td>0.282</td>
<td>0.288</td>
<td>0.295</td>
<td>0.435</td>
<td>0.435</td>
<td>0.435</td>
<td>0.435</td>
</tr>
<tr>
<td>MLH</td>
<td>0.472</td>
<td>0.666</td>
<td>0.652</td>
<td>0.654</td>
<td>0.182</td>
<td>0.195</td>
<td>0.207</td>
<td>0.211</td>
<td>0.500</td>
<td>0.514</td>
<td>0.520</td>
<td>0.522</td>
</tr>
<tr>
<td>BRE</td>
<td>0.515</td>
<td>0.593</td>
<td>0.613</td>
<td>0.634</td>
<td>0.159</td>
<td>0.181</td>
<td>0.193</td>
<td>0.196</td>
<td>0.485</td>
<td>0.525</td>
<td>0.530</td>
<td>0.544</td>
</tr>
<tr>
<td>SH</td>
<td>0.265</td>
<td>0.267</td>
<td>0.259</td>
<td>0.250</td>
<td>0.131</td>
<td>0.135</td>
<td>0.133</td>
<td>0.130</td>
<td>0.433</td>
<td>0.426</td>
<td>0.426</td>
<td>0.423</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.388</td>
<td>0.436</td>
<td>0.422</td>
<td>0.429</td>
<td>0.162</td>
<td>0.169</td>
<td>0.172</td>
<td>0.175</td>
<td>0.452</td>
<td>0.468</td>
<td>0.472</td>
<td>0.477</td>
</tr>
<tr>
<td>LSH</td>
<td>0.187</td>
<td>0.209</td>
<td>0.235</td>
<td>0.243</td>
<td>0.121</td>
<td>0.126</td>
<td>0.120</td>
<td>0.120</td>
<td>0.403</td>
<td>0.421</td>
<td>0.426</td>
<td>0.441</td>
</tr>
</tbody>
</table>

Table 1: MAP of Hamming ranking w.r.t different number of bits on three datasets. For NUS-WIDE, we calculate the MAP values within the top 5000 returned neighbors. The results of CNNH / CNNH+ are the average of 5 trials.
Supervised Hashing via Image Representation Learning (CNNH) (Xia et al., 2014)

Figure 3: The results on CIFAR10. (a) precision curves within Hamming radius 2; (b) precision-recall curves of Hamming ranking with 48 bits; (c) precision curves with 48 bits w.r.t. different number of top returned samples
Simultaneous Feature Learning and Hash Coding with Deep Neural Networks (NINH) (Lai et al., 2015)

Figure 1. Overview of the proposed deep architecture for hashing. The input to the proposed architecture is in the form of triplets, i.e., $(I, I^+, I^-)$ with a query image $I$ being more similar to an image $I^+$ than to another image $I^-$. Through the proposed architecture, the image triplets are first encoded into a triplet of image feature vectors by a shared stack of multiple convolution layers. Then, each image feature vector in the triplet is converted to a hash code by a divide-and-encode module. After that, these hash codes are used in a triplet ranking loss that aims to preserve relative similarities on images.
Simultaneous Feature Learning and Hash Coding with Deep Neural Networks (NINH) (Lai et al., 2015)

The loss function:

\[
\ell_{\text{triplet}}(F(I), F(I^+), F(I^-)) \\
= \max(0, \|F(I) - F(I^+)\|_2^2 - \|F(I) - F(I^-)\|_2^2 + 1) \\
\text{s.t. } F(I), F(I^+), F(I^-) \in [0, 1]^q.
\]
Simultaneous Feature Learning and Hash Coding with Deep Neural Networks (NINH) (Lai et al., 2015)

Table 2. MAP of Hamming ranking w.r.t different numbers of bits on three datasets. For NUS-WIDE, we calculate the MAP values within the top 5000 returned neighbors. The results of CNNH is directly cited from [27]. CNNH* is our implementation of the CNNH method in [27] using Caffe, by using a network configuration comparable to that of the proposed method (see the text in Section 4.1 for implementation details).

<table>
<thead>
<tr>
<th>Method</th>
<th>SVHN (MAP)</th>
<th>CIFAR-10 (MAP)</th>
<th>NUS-WIDE (MAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12 bits</td>
<td>24 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>Ours</td>
<td>0.899</td>
<td>0.914</td>
<td>0.925</td>
</tr>
<tr>
<td>CNNH*</td>
<td>0.897</td>
<td>0.903</td>
<td>0.904</td>
</tr>
<tr>
<td>CNNH [27]</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KSH [12]</td>
<td>0.469</td>
<td>0.539</td>
<td>0.563</td>
</tr>
<tr>
<td>ITQ-CCA [4]</td>
<td>0.428</td>
<td>0.488</td>
<td>0.489</td>
</tr>
<tr>
<td>MLH [16]</td>
<td>0.147</td>
<td>0.247</td>
<td>0.261</td>
</tr>
<tr>
<td>BRE [8]</td>
<td>0.165</td>
<td>0.206</td>
<td>0.230</td>
</tr>
<tr>
<td>SH [26]</td>
<td>0.140</td>
<td>0.138</td>
<td>0.141</td>
</tr>
<tr>
<td>ITQ [4]</td>
<td>0.127</td>
<td>0.132</td>
<td>0.135</td>
</tr>
<tr>
<td>LSH [2]</td>
<td>0.110</td>
<td>0.122</td>
<td>0.120</td>
</tr>
</tbody>
</table>
Deep Semantic Ranking Based Hashing for Multi-Label Image Retrieval (DSRH) (Zhao et al., 2015)

Figure 1. The proposed deep semantic ranking based hashing. Solid and hollow arrows indicate forward and backward propagation directions of features and gradients respectively. Hash functions consist of deep convolutional neural network (CNN) and binary mappings of the feature representation from the top hidden layers of CNN. Multilevel semantic ranking information is used to learn such deep hash functions to preserve the semantic structure of multi-label images.

Figure 2. The structure of deep hash functions. An input image is first transformed to a fixed size, and then goes through five convolution layers and two fully-connected layers, which provides a deep feature representation. Finally, the hash layer generates a compact binary code. The hash layer is also directly connected to the first fully-connected layer (FCa) in order to utilize diverse feature information biased toward visual appearance.
Deep Hashing

Deep Semantic Ranking Based Hashing for Multi-Label Image Retrieval (DSRH) (Zhao et al., 2015)

The major part:

- **Hash functions:**
  \[ h(x; w) = \text{sign}(w^T [f_a(x); f_b(x)]) , \]

- **Semantic ranking supervision**

- **Optimization with surrogate loss (weighted triplet loss):**
  \[ F(W) = \sum_{q \in D, \{x_i\}_{i=1}^M \subset D} L_\omega(h(q; W), \{h(x_i; W)\}_{i=1}^M) \]
  \[ + \frac{\alpha}{2} \left\| \text{mean}(h(q; W)) \right\|_2^2 + \frac{\beta}{2} \|W\|_2^2 . \]
Deep Semantic Ranking Based Hashing for Multi-Label Image Retrieval (DSRH) (Zhao et al., 2015)

Figure 4. Comparison of ranking performance of our DSRH and other hashing methods based on hand-crafted features on two datasets: (a) MIRFLICKR-25K and (b) NUS-WIDE.
Deep Semantic Ranking Based Hashing for Multi-Label Image Retrieval (DSRH) (Zhao et al., 2015)

Figure 5. Comparison of ranking performance of our DSRH and other hashing methods based on activation features of pre-trained CNN on two datasets: (a) MIRFLICKR-25K and (b) NUS-WIDE.
Bit-Scalable Deep Hashing (DRSCH) (Zhang et al., 2015)

Fig. 2. The bit-scalable deep hashing learning framework. The bottom panel shows the deep architecture of neural network that produces the hashing code with the weight matrix by taking raw images as inputs. The training stage is illustrated in the left up panel, where we train the network with triplet-based similarity learning. An example of hashing retrieval is presented in the right up panel, where the similarity is measured by the Hamming affinity.
### Bit-Scalable Deep Hashing (DRSCH) (Zhang et al., 2015)

**Table I**

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST (MAP %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>DRSCH</td>
<td>96.92</td>
</tr>
<tr>
<td>DSCH</td>
<td>96.51</td>
</tr>
<tr>
<td>DSRH [40]</td>
<td>96.48</td>
</tr>
<tr>
<td>KSH-CNN [7]</td>
<td>83.89</td>
</tr>
<tr>
<td>MLH-CNN [12]</td>
<td>71.03</td>
</tr>
<tr>
<td>BRE-CNN [39]</td>
<td>61.00</td>
</tr>
<tr>
<td>KSH [7]</td>
<td>82.85</td>
</tr>
<tr>
<td>MLH [12]</td>
<td>45.77</td>
</tr>
<tr>
<td>BRE [39]</td>
<td>41.96</td>
</tr>
<tr>
<td>PCA-RR [14]</td>
<td>35.96</td>
</tr>
<tr>
<td>ITQ [14]</td>
<td>34.44</td>
</tr>
<tr>
<td>LSH [18]</td>
<td>22.65</td>
</tr>
<tr>
<td>Euclidean</td>
<td>89.55</td>
</tr>
</tbody>
</table>

**Table II**

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (MAP %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>DRSCH</td>
<td>61.46</td>
</tr>
<tr>
<td>DSCH</td>
<td>60.87</td>
</tr>
<tr>
<td>DSRH [40]</td>
<td>60.84</td>
</tr>
<tr>
<td>KSH-CNN [7]</td>
<td>40.08</td>
</tr>
<tr>
<td>KSH [7]</td>
<td>32.15</td>
</tr>
<tr>
<td>MLH [12]</td>
<td>13.33</td>
</tr>
<tr>
<td>BRE [39]</td>
<td>12.19</td>
</tr>
<tr>
<td>Euclidean</td>
<td>35.46</td>
</tr>
</tbody>
</table>

*Image retrieval results (Mean Average Precision) with various number of bits on the MNIST dataset. The scale of test query set is 10K. Our method outperforms the state-of-the-art methods.*

*Image retrieval results (Mean Average Precision) with various number of bits on the CIFAR-10 dataset. The scale of test query set is 10K (1K per class). The proposed method outperforms the state-of-the-art methods.*
Bit-Scalable Deep Hashing (DRSCH) (Zhang et al., 2015)

Fig. 4. The results on the CIFAR-10 dataset. (a) Precision curves within Hamming radius 2; (b) Precision curves with top 500 returned; (c) Precision curves with 64 hash bits.
Deep Supervised Hashing with Pairwise Labels (Li et al., 2015)

Motivation

- Most existing hashing methods are based on hand-crafted features which might not be optimally compatible with the hashing procedure.
- Recently, deep hashing is proposed for simultaneous feature learning and hash-code learning, with better performance.
- Most existing deep hashing methods are supervised with triplet labels. For pairwise labels, there have not existed methods for simultaneous feature learning and hash-code learning.

Our contribution:

- An end-to-end framework, called deep pairwise-supervised hashing (DPSH), to perform simultaneous feature learning and hash-code learning for applications with pairwise labels.
Deep Supervised Hashing with Pairwise Labels (Li et al., 2015)

VGG-F + hash-code: 7 + 1 = 8 layers.
Five conv+pooling, two 4096 fully-connected, one fully-connected for hash-code
Deep Supervised Hashing with Pairwise Labels (Li et al., 2015)

The same pairwise loss function as LFH:

\[
L = - \log p(S|B) = - \sum_{s_{ij} \in S} \log p(s_{ij}|B) \\
= - \sum_{s_{ij} \in S} (s_{ij} \Theta_{ij} - \log(1 + e^{\Theta_{ij}})),
\]

where \(s_{ij} \in \{0, 1\}\) is the supervised label, \(B = \{b_i\}_{i=1}^n\) is the binary codes, \(\Theta_{ij} = \frac{1}{2} b_i^T b_j\).

Similar continuous relaxation as that in LFH is adopted for learning.
# Deep Supervised Hashing with Pairwise Labels (Li et al., 2015)

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (MAP)</th>
<th>NUS-WIDE (MAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12-bits</td>
<td>24-bits</td>
</tr>
<tr>
<td>DPSH</td>
<td>0.682</td>
<td>0.686</td>
</tr>
<tr>
<td>DPSH0</td>
<td>0.458</td>
<td>0.512</td>
</tr>
<tr>
<td>NINH [CVPR15]</td>
<td>0.552</td>
<td>0.566</td>
</tr>
<tr>
<td>CNNH [AAAI14]</td>
<td>0.439</td>
<td>0.476</td>
</tr>
<tr>
<td>FastH</td>
<td>0.305</td>
<td>0.349</td>
</tr>
<tr>
<td>SDH</td>
<td>0.285</td>
<td>0.329</td>
</tr>
<tr>
<td>KSH</td>
<td>0.303</td>
<td>0.337</td>
</tr>
<tr>
<td>LFH</td>
<td>0.278</td>
<td>0.435</td>
</tr>
<tr>
<td>LFH0</td>
<td>0.176</td>
<td>0.231</td>
</tr>
<tr>
<td>SPLH</td>
<td>0.171</td>
<td>0.173</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.162</td>
<td>0.169</td>
</tr>
<tr>
<td>SH</td>
<td>0.127</td>
<td>0.128</td>
</tr>
</tbody>
</table>
Deep Supervised Hashing with Pairwise Labels (Li et al., 2015)

Compare to baselines with triplet labels:

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (MAP)</th>
<th>NUS-WIDE (MAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16-bits</td>
<td>24-bits</td>
</tr>
<tr>
<td>DPSH</td>
<td>0.742</td>
<td>0.764</td>
</tr>
<tr>
<td>DRSCH [TIP15]</td>
<td>0.615</td>
<td>0.622</td>
</tr>
<tr>
<td>DSCH [TIP15]</td>
<td>0.609</td>
<td>0.613</td>
</tr>
<tr>
<td>DSRH [CVPR15]</td>
<td>0.608</td>
<td>0.611</td>
</tr>
</tbody>
</table>

Li (http://cs.nju.edu.cn/lwj)
Deep Supervised Hashing with Pairwise Labels (Li et al., 2015)
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference
Double Bit Quantization (Kong and Li, 2012a)

Point distribution of the real values computed by PCA on 22K LabelMe data set, and different coding results based on the distribution:

- (a) single-bit quantization (SBQ);
- (b) hierarchical hashing (HH) (Liu et al., 2011);
- (c) double-bit quantization (DBQ).
Double Bit Quantization (Kong and Li, 2012a)

Precision-recall curve on 22K LabelMe data set

SH 32 bits

SH 64 bits

SH 128 bits

SH 256 bits
Double Bit Quantization (Kong and Li, 2012a)

mAP on LabelMe data set

<table>
<thead>
<tr>
<th># bits</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBQ</td>
<td>HH</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.2926</td>
<td>0.2592</td>
</tr>
<tr>
<td>SH</td>
<td>0.0859</td>
<td>0.1329</td>
</tr>
<tr>
<td>PCA</td>
<td>0.0535</td>
<td>0.1009</td>
</tr>
<tr>
<td>LSH</td>
<td>0.1657</td>
<td>0.105</td>
</tr>
<tr>
<td>SIKH</td>
<td>0.0590</td>
<td>0.0712</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># bits</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBQ</td>
<td>HH</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.3675</td>
<td>0.4032</td>
</tr>
<tr>
<td>SH</td>
<td>0.1730</td>
<td>0.2034</td>
</tr>
<tr>
<td>PCA</td>
<td>0.0323</td>
<td>0.1083</td>
</tr>
<tr>
<td>LSH</td>
<td>0.3579</td>
<td>0.3311</td>
</tr>
<tr>
<td>SIKH</td>
<td>0.2792</td>
<td>0.3147</td>
</tr>
</tbody>
</table>
Manhattan Quantization (Kong et al., 2012)

Quantization Stage

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>0</th>
<th>BC</th>
<th>DE</th>
<th>1</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>01</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
</tr>
</tbody>
</table>

Figure 1: Different quantization methods: (a) single-bit quantization (SBQ); (b) hierarchical quantization (HQ); (c) 2-bit Manhattan quantization (2-MQ); (d) 3-bit Manhattan quantization (3-MQ).
Manhattan Quantization (Kong et al., 2012)

Natural Binary Code (NBC)

(a) Hamming distance

(b) Decimal distance with NBC
Manhattan Quantization (Kong et al., 2012)

Manhattan Distance

Let $\mathbf{x} = [x_1, x_2, \cdots, x_d]^T$, $\mathbf{y} = [y_1, y_2, \cdots, y_d]^T$, the Manhattan distance between $\mathbf{x}$ and $\mathbf{y}$ is defined as follows:

$$d_m(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|,$$

where $|x|$ denotes the absolute value of $x$. 
Manhattan Quantization (Kong et al., 2012)

- We divide each projected dimension into $2^q$ regions and then use $q$ bits of natural binary code to encode the index of each region.

- For example, If $q = 3$, the indices of regions are \{0, 1, 2, 3, 4, 5, 6, 7\} and the natural binary codes are \{000, 001, 010, 011, 100, 101, 110, 111\}
Manhattan quantization (MQ) with \( q \) bits is denoted as \( q \)-MQ.

For example, if \( q = 2 \),

\[
d_m(000100, 110000) = d_d(00, 11) + d_d(01, 00) + d_d(00, 00)
\]
\[
= 3 + 1 + 0
\]
\[
= 4.
\]
Manhattan Quantization (Kong et al., 2012)

Figure: Precision-recall curve on 22K LabelMe data set
Manhattan Quantization (Kong et al., 2012)

Table: mAP on ANN_SIFT1M data set. The best mAP among SBQ, HQ and 2-MQ under the same setting is shown in bold face.

<table>
<thead>
<tr>
<th># bits</th>
<th>32</th>
<th>64</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBQ</td>
<td>HQ</td>
<td>2-MQ</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.1657</td>
<td>0.2500</td>
<td>0.2750</td>
</tr>
<tr>
<td>SIKH</td>
<td>0.0394</td>
<td>0.0217</td>
<td>0.0570</td>
</tr>
<tr>
<td>LSH</td>
<td>0.1163</td>
<td>0.0961</td>
<td>0.1173</td>
</tr>
<tr>
<td>SH</td>
<td>0.0889</td>
<td>0.2482</td>
<td>0.2771</td>
</tr>
<tr>
<td>PCA</td>
<td>0.1087</td>
<td>0.2408</td>
<td>0.2882</td>
</tr>
</tbody>
</table>
Figure 1: Left: Data points with identical shapes are 1-NN. Two hyperplanes $h_1$, $h_2$ are shown alongside their associated normal vectors ($n_1$, $n_2$). Right top: Projection of points onto the normal vectors $n_1$ and $n_2$ of the hyperplanes (arrows denote projections). Right middle: Positioning of the points along normal vector $n_2$. Three quantisation thresholds ($t_1$, $t_2$, $t_3$, and consequently 2 bits) can maintain the neighbourhood structure. Right bottom: the high degree of mixing between the 1-NN means that this hyperplane ($h_1$) is likely to have 0 bits assigned (and therefore be discarded entirely).
Variable Bit Quantization (Moran et al., 2013)

Table 1: Area under the Precision Recall curve (AUPRC) for all five projection methods. Results are for 32 bits (images) and at 128 bits (text). The best overall score for each dataset is shown in bold face.
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference
Conclusion

- Hashing can significantly improve searching speed and reduce storage cost.
- Projections with isotropic variances will be better than those with anisotropic variances. (IsoHash)
- Avoiding pairwise computation via feature transformation can be used for scalable graph hashing. (SGH)
- Stochastic learning or avoiding pairwise computation can be used for scalable supervised hashing. (LFH/SCM)
- Discrete hashing methods might outperform continuous relaxation methods. (COSDISH)
- Deep hashing can improve accuracy by integrating feature learning into code learning procedure. (DPSH).
- The quantization stage is at least as important as the projection stage. (DBQ/MQ)
Q & A

Thanks!

Question?

A Learning to Hash website:  http://cs.nju.edu.cn/lwj/L2H.html

Some code available at:  http://cs.nju.edu.cn/lwj
Outline

1. Introduction
2. Unsupervised Hashing
3. Supervised Hashing
4. Ranking-based Hashing
5. Multimodal Hashing
6. Deep Hashing
7. Quantization
8. Conclusion
9. Reference

Li (http://cs.nju.edu.cn/lwj)


A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions.


W. Kong and W.-J. Li. Isotropic hashing. In *Proceedings of the 26th*


M. Ou, P. Cui, F. Wang, J. Wang, W. Zhu, and S. Yang. Comparing...


R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and

