Semantic Structure-based Unsupervised Deep Hashing

Erkun Yang¹, Cheng Deng¹*, Tongliang Liu², Wei Liu³, Dacheng Tao²
¹ School of Electronic Engineering, Xidian University, Xi’an 710071, China
² UBTECH Sydney AI Centre, SIT, FEIT, University of Sydney, Australia
³ Tencent AI Lab, Shenzhen, China
ekyang@stu.xidian.edu.cn, chdeng.xd@gmail.com, tongliang.liu@sydney.edu.au, wliu@ee.columbia.edu, dacheng.tao@sydney.edu.au

Abstract

Hashing is becoming increasingly popular for approximate nearest neighbor searching in massive databases due to its storage and search efficiency. Recent supervised hashing methods, which usually construct semantic similarity matrices to guide hash code learning using label information, have shown promising results. However, it is relatively difficult to capture and utilize the semantic relationships between points in unsupervised settings. To address this problem, we propose a novel unsupervised deep framework called Semantic Structure-based unsupervised Deep Hashing (SSDH). We first empirically study the deep feature statistics, and find that the distribution of the cosine distance for point pairs can be estimated by two half Gaussian distributions. Based on this observation, we construct the semantic structure by considering points with distances obviously smaller than the others as semantically similar and points with distances obviously larger than the others as semantically dissimilar. We then design a deep architecture and a pair-wise loss function to preserve this semantic structure in Hamming space. Extensive experiments show that SSDH significantly outperforms current state-of-the-art methods.

1 Introduction

The explosive growth of visual data (e.g., photos and videos) has led to renewed interest in efficient indexing and searching algorithms [Liu et al., 2016; Andoni and Indyk, 2006; Weiss et al., 2009; Gong et al., 2013; Dai et al., 2017; Ma et al., 2017; You et al., 2017a; Deng et al., 2018; Li et al., 2018]. However, since exact nearest neighbor searching is typically time-consuming and sometimes infeasible for big data applications, approximate nearest neighbor (ANN) searching [Andoni and Indyk, 2006], which balances retrieval performance and computational cost, has recently become more popular. Representative ANN solutions include tree-based [Guttman, 1984] and hash-based methods [Andoni and Indyk, 2006; Gui et al., 2017; Liu et al., 2014; Weiss et al., 2009; Liu et al., 2012]. In real applications, visual data are usually represented by high-dimensional features, e.g., SIFT-based bag-of-words features [Lowe, 2004] and deep features. Traditional tree-based methods are known to suffer from high feature space dimensionality, and their performance has been theoretically shown to degrade to that of the linear scan in many cases [Toth et al., 2004]. Therefore, focus has shifted to hash-based methods for large-scale ANN searching.

Based on whether the hash learning processes are aware of the data distributions, hashing methods can be divided into data-independent hashing, also known as locality-sensitive hashing (LSH), and data-dependent hashing, also called learn-to-hash (L2H). The seminal LSH in [Andoni and Indyk, 2006] formulates a paradigm for the locality-sensitive hashing technique and guarantees that similar data points will be mapped to similar hash codes in Hamming space. However, since the LSH hash functions are usually generated independently, long hash bits are often needed to achieve a specific retrieval performance. Conversely, data-dependent methods [Weiss et al., 2009; Liu et al., 2011; 2011; Gong et al., 2013; Dai et al., 2017], which learn hash functions from data distributions, usually perform well with shorter binary codes, increasing their value.

For data-dependent methods, several studies [Lin et al., 2016; Yang et al., 2018] have focused on learning hash codes under supervised settings in which data labels (e.g., instance labels, pair-wise labels, or triplet labels) are assumed to be available. In these methods, the hash codes are usually learned to be consistent with these high-level supervised signals. However, manually labeling data is often time-consuming, laborious, and expensive, hindering the applications of these methods in practice. To make better use of widely-available unlabeled data, efforts have been made to exploit unsupervised hashing.

Most traditional unsupervised hashing methods are based on shallow architectures that consider feature learning and hash code learning as two separate processes. Representative unsupervised shallow methods include spectral hashing (SH) [Weiss et al., 2009], anchor graph hashing (AGH) [Liu
et al., 2011], iterative quantization (ITQ) [Gong et al., 2013], and stochastic generative hashing (SGH) [Dai et al., 2017].

SH utilizes spectral graph partitioning to interpret the hash code learning and then relax the original problem by using a spectral method that can be solved efficiently. AGH attempts to approximate the data structure by constructing anchor graphs, which can also accelerate the computation of graph Laplacian eigenvectors. ITQ tries to map the data to the vertices of a zero-centered binary hypercube by finding an orthogonal rotation matrix that minimizes the quantization error. SGH adopts a generative approach to compress the dataset into hash codes using the minimum description length principle. The learned hash codes can also be used to regenerate the inputs. Though these methods have progressed the field, they usually depend on pre-defined features and cannot simultaneously optimize the feature and hash code learning processes, thus missing an opportunity to learn more effective hash codes.

Recently, deep learning has revolutionized computer vision, machine learning, and other related areas [You et al., 2017b]. Deep unsupervised hashing methods have also been proposed, which adopt deep architectures to extract features and perform hash mapping. Representative unsupervised deep hashing methods include semantic hashing [Salakhutdinov and Hinton, 2009], deep auto-encoder hashing [Krizhevsky and Hinton, 2011], and deep binary descriptors (DeepBit) [Lin et al., 2016]. Semantic hashing uses pre-trained restricted Boltzmann machines (RBMs) to construct an auto-encoder network, which is then used to generate efficient hash codes and reconstruct the original inputs. Deep auto-encoder hashing designs very deep auto-encoders to map inputs to binary codes, and a reconstruction loss is also used to guide hash code learning. DeepBit considers original images and the corresponding rotated images as similar pairs and tries to learn hash codes to preserve this similarity. By integrating the feature and hash code learning processes, deep unsupervised hashing methods usually produce better results.

However, most existing unsupervised deep hashing methods such as semantic hashing, deep auto-encoder hashing, and DeepBit, usually learn hash codes based on the reconstruction loss or by preserving the similarity between rotated images. They cannot capture and utilize high-level semantic relationships between different data points. In reality, in supervised settings, most deep methods [Yang et al., 2017] leverage the supervised signals to construct a semantic similarity matrix and learn hash codes to preserve this semantic relationship. However, these methods are unsuitable for unsupervised settings.

Here we ask: can we learn semantic structures without labels? Our approach is inspired by recent studies [Girshick et al., 2014], which show that features extracted from pre-trained deep architectures contain rich semantic information. We empirically analyze the deep feature statistics for the NUSWIDE and FLICKR25K datasets and find that the distribution of the cosine distance for different data points can be estimated by two half Gaussian distributions. Since deep features contain rich semantic information, we assume that data points with distances obviously smaller than the others are semantically similar and data points with distances obviously larger than the others are semantically dissimilar. Based on the above observation and assumptions, we construct a semantic structure that denotes whether data point pairs are similar. We then design a pair-wise loss function to preserve this semantic information. To integrate the feature and hash code learning processes, we design a unified, end-to-end trainable deep framework. Extensive experiments demonstrate that our proposed method significantly outperforms the current state-of-the-art.

2 Notation and Problem Definition

In this paper, boldface uppercase letters (such as $A$) are used to denote matrices. $A_{i,:}$ denotes the $i$-th column of $A$. $A_{i,j}$ denotes the $(i,j)$-th element of $A$. Further, $\|A\|_F$ and $A^\top$ are used to denote the Frobenius norm and the transpose of matrix $A$, respectively. The $\odot$ symbol is used to denote the Hadamard product and $\text{sign}(\cdot)$ represents the element-wise signum function as:

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{if } x < 0. \end{cases}$$

Assume that we have $m$ database points denoted $X = \{x_i\}_{i=1}^m$. The goal of unsupervised hashing is to learn binary hash codes for database points and a hash function that can be used to generate hash codes for query data points. We use $B = \{b_i\}_{i=1}^m$ to denote the hash codes for $X$, and $b_i \in \{-1, +1\}^K$ corresponds to the hash code for point $x_i$, where $K$ denotes the hash code length.

3 Model Formulation

In this section, we introduce our Semantic Structure-based unsupervised Deep Hashing (SSDH) approach in detail, including its network architecture, semantic structure learning, and hash code learning.

3.1 Network Architecture

We apply the VGG-F model from [Simonyan and Zisserman, 2014] to perform feature and hash code learning. This model has been used in many other hashing methods [Lin et al., 2016; Yang et al., 2018]. The original VGG-F model contains five convolutional layers ($\text{conv1} - \text{conv5}$) and three fully-connected layers ($\text{fc6-fc8}$), the details of which can be found in [Simonyan and Zisserman, 2014]. We modify the original VGG-F model by replacing the last fully-connected layer with a fully-connected layer with $k$ hidden units to incorporate the hash code learning process in this deep learning framework. Note that we adopt VGG-F network for illustration purposes only, and any other network can easily be integrated into our framework.

3.2 Semantic Structure Learning

Since hashing methods usually aim to map the original points into Hamming space where the semantic relationship across different data points can be preserved, constructing a semantic similarity matrix using label information in supervised settings is natural. However, for unsupervised hashing learning, it is difficult to capture the semantic structure without...
any supervised information. To address this problem, and inspired by recent studies [Girshick et al., 2014] showing that features extracted from pre-trained deep architectures contain rich semantic information, we first extract features from a pre-trained deep architecture and analyze their statistics. Based on this analysis, we can then learn a semantic structure that explicitly captures the semantic relationships across different data points.

Specifically, we first randomly sample 10,000 data points from the FLICKR and NUSWIDE datasets and then extract their fc7-layer features from the VGG-F network pre-trained with ImageNet [Deng et al., 2009]. To analyze the semantic relationships between these data points, we calculate the cosine distance for each pair of points based on the extracted features. Figure 1(a) shows the distance cumulative histogram, and Figure 1(b) is the corresponding distance histogram over all data point pairs. It can be seen that most data point pairs have relatively large distances, and each distance histogram is similar to two half Gaussian distributions. Since deep features contain rich semantic information, we assume that data points with distances obviously smaller than the others are semantically similar and data points with distances obviously larger than the others are semantically dissimilar.

Based on the above observation and assumptions, we split the distance histogram into two parts based on the maximum value and fit each part with a half Gaussian distribution. The results for the NUSWIDE dataset are shown in Figures 1(c) and 1(d), which clearly show that the distance histogram can be approximately estimated by two half Gaussian distributions. We use \( m_l \) and \( \sigma_l \) to denote the mean and standard deviation of the Gaussian distribution for the left part and \( m_r \) and \( \sigma_r \) for the right part. To obtain the semantically similar data points, we set a distance threshold \( d_s = (m_l - \alpha \sigma_l) \), and consider data points with distance smaller than \( d_s \) as semantically similar data points. \( \alpha \) is a hyper-parameter to control the value of the distance threshold \( d_s \), and it also indicates the percentage of similar points from all data point pairs. For example, if we only consider points from the left part and choose \( \alpha \) as 2, then from the properties of Gaussian distributions we know that only about 2.2\% of the distances are smaller than \((m_l - 2\sigma_l)\). To obtain semantically dissimilar data points, we set a distance threshold as \( d_d = (m_r + \beta \sigma_r) \), and consider data points with distance larger than \( d_d \) as semantically dissimilar. \( \beta \) is a hyper-parameter to control the value of the distance threshold \( d_d \), which also indicates the percentage of dissimilar points.

Based on the above two thresholds, we construct a semantic structure \( S \) as:

\[
S_{ij} = \begin{cases} 
1, & \text{if } d(i, j) \leq d_s, \\
0, & \text{if } d_s < d(i, j) < d_d, \\
-1, & \text{if } d(i, j) \geq d_d,
\end{cases}
\]

(2)

where \( d(i, j) \) represents the cosine distance of the deep representations for points \( x_i \) and \( x_j \), and \( S_{ij} \) is set to 1 if \( x_i \) and \( x_j \) are semantically similar, and -1 if they are semantically dissimilar. If we cannot decide whether they are semantically similar or dissimilar, we set \( S_{ij} \) to 0.

### 3.3 Hash Code Learning

In order to preserve the learned semantic structure, we try to map semantically similar data points into similar hash codes, semantically dissimilar data points into dissimilar hash codes, and discard pairs of data points if we cannot decide whether they are semantically similar or dissimilar. We first construct a similarity structure \( H \) by using the inner product of different hash codes:

\[
H_{ij} = \frac{1}{K} b_i \cdot b_j, \quad b_i \in \{-1, +1\}^K,
\]

(3)

where \( b_i \) is the hash code for data point \( x_i \). To explicitly enforce semantic structure-preserving criteria, we minimize the \( L_2 \) loss between this similarity structure and the learned
semantic structure, which can be formulated as:

$$\min_B J(B) = \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} |S_{ij}| (H_{ij} - S_{ij})^2.$$  \hspace{1cm} (4)

We set $b_i = \text{sign}(F(x_i; \Theta))$, where $F(x_i; \Theta)$ denotes the output of the neural network, and $\Theta$ is the parameters. Therefore we can integrate the above loss function into the deep architecture. However, binary values make standard backpropagation for the deep neural network infeasible, which is known as the ill-posed gradient problem. So, here we relax the binary constraint, use tanh($\cdot$) to approximate the sign($\cdot$) function, and adopt the following objective function:

$$\min_\Theta J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{m} |S_{ij}| (H_{ij} - S_{ij})^2,$$

$$\text{s.t. } H_{ij} = \frac{1}{K} v_i^\top v_j, \quad v_i = \tanh(F(x_i; \Theta)),$$

where we use $v_i$ to denote the relaxed binary representation.

4 Optimization

To optimize the problem in Equation 5, we first extract 10,000 samples from the dataset before constructing the semantic structure using Equation 2. Then, we minimize Equation 5 using the mini-batch stochastic gradient descent (SGD) method. The whole learning procedure is summarized in Algorithm 1.

When $S$ is given, we use the mini-batch SGD method to learn the neural network parameter $\Theta$. Specifically, we first sample a mini-batch of the training data points and then update parameter $\Theta$ based on this mini-batch data. For the sake of simplicity, we define $z_i = F(x_i; \Theta)$. Then we calculate the gradient of the loss function with regard to $z_i$ as:

$$\frac{\partial J(\Theta)}{\partial z_i} = \frac{2}{K m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} \left(1 - v_i^2 \right) \odot |S_{ij}| \left(\frac{1}{K} v_i^\top v_j - S_{ij}\right)
\cdot v_j + \left(\frac{1}{K} v_i^\top v_i - 1\right) v_j.$$  \hspace{1cm} (6)

The gradients for loss function $J(\Theta)$ with regard to other parameters can be readily computed using the chain rule, and all the parameters can be updated using mini-batch SGD.

We performed experiments to update the semantic structure based on updating image features in each epoch. Constructing this semantic structure was time-consuming, and the update strategy did not significantly impact performance. So, in this work, we first obtain the semantic structure and then fix it when updating the neural network parameters.

5 Out of Sample Extension

For any point $q_i$ not in the training set, we can obtain its hash code $b_i$ by directly forward propagating it through the learned neural network as follows:

$$b_i = \text{sign}(F(q_i; \Theta)).$$  \hspace{1cm} (7)

Algorithm 1: SSDH

Training Stage

Input: Training images $X$, code length $K$, parameters $\alpha$ and $\beta$, mini-batch size $N$.

Output: Parameters $\Theta$ for the neural network and hash codes $B$ for training images.

Procedure:
1. Initialize parameters for the VGG-F network $\Theta$.
2. Extract their fc7-layer features based on the VGG-F network.
3. Construct the semantic structure $S$ by using Equation 2.

repeat
3.1 Randomly sample $N$ points from $X$ to construct a mini-batch.
3.2 Calculate the outputs by forward-propagating through the VGG-F network.
3.3 Update parameters of the VGG-F network by (6).
until convergence;

Testing Stage

Input: Image query $q$, parameters for the VGG-F network.

Output: Hash codes for the query.

Procedure:
1. Calculate the output of the neural network by directly forward-propagating the input images.
2. Calculate the hash codes by using Equation (7).

6 Experiments

We evaluate our method on two popular benchmark datasets, NUSWIDE and FLICKR25K. We provide extensive evaluations of the proposed hash codes and demonstrate their performance. We first introduce the datasets and then present and discuss our experimental results together with comparative evaluations with current state-of-the-art methods.

6.1 Datasets

- NUSWIDE contains 269,648 images, with each data point annotated with multiple labels based on 81 concepts. The subset belonging to the 10 most popular concepts are used here. We randomly select 5,000 images as a test set. The remaining images are used as a retrieval set, from which we randomly select 5,000 images as a training set.
- FLICKR25K contains 25,000 images collected from the Flickr website. Each image is manually annotated with at least one of 24 unique provided labels. We randomly select 2,000 images as the test set. The remaining images are used as the retrieval set, from which we randomly select 10,000 images as the training set.

For our proposed approach and deep learning-based comparator methods, we use raw pixels as inputs. For traditional shallow methods, we extract 4096-dimensional feature vectors from the fc7 layer using the VGG-F architecture pre-trained on ImageNet for fair comparison.
Table 1: Comparison with baselines in terms of MAP. The highest accuracy is shown in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>FLICKR25K</th>
<th>NUSWIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.6492</td>
<td>0.6546</td>
</tr>
<tr>
<td>SH</td>
<td>0.6091</td>
<td>0.6033</td>
</tr>
<tr>
<td>DSH</td>
<td>0.6452</td>
<td>0.6451</td>
</tr>
<tr>
<td>SpH</td>
<td>0.6119</td>
<td>0.6381</td>
</tr>
<tr>
<td>SGH</td>
<td>0.6362</td>
<td>0.6283</td>
</tr>
<tr>
<td>DeepBit</td>
<td>0.5934</td>
<td>0.5933</td>
</tr>
<tr>
<td>Ours</td>
<td>0.7240</td>
<td>0.7276</td>
</tr>
</tbody>
</table>

Table 2: Comparison with baselines in terms of MAP. The highest accuracy is shown in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>NUSWIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.5270</td>
</tr>
<tr>
<td>SH</td>
<td>0.4350</td>
</tr>
<tr>
<td>DSH</td>
<td>0.5123</td>
</tr>
<tr>
<td>SpH</td>
<td>0.4458</td>
</tr>
<tr>
<td>SGH</td>
<td>0.4994</td>
</tr>
<tr>
<td>DeepBit</td>
<td>0.3844</td>
</tr>
<tr>
<td>Ours</td>
<td>0.6374</td>
</tr>
</tbody>
</table>

6.2 Implementation Details

We implement our approach using the open source TensorFlow [Abadi et al., 2016], and optimize our model by minibatch SGD with momentum. The mini-batch size is set to 224 x 224 as the inputs. The first seven layers of our neural network are fine-tuned from the model pre-trained with ImageNet, and the last fully-connected layer is learnt from scratch. The learning rate is fixed at 0.001.

6.3 Protocols and Baseline Methods

Three evaluation criteria are adopted to evaluate our method’s performance, namely mean average precision (MAP), precision-recall, and TopN-precision.

MAP is one of the most widely-used criteria to evaluate retrieval accuracy. Given a query and a list of R ranked retrieval results, the average precision (AP) for this query is defined as

\[ AP = \frac{1}{N} \sum_{r=1}^{R} P(r) \delta(r), \]

where N is the number of ground-truth relevant instances in the database for the query, and \( P(r) \) represents the precision for the top r retrieved instances. \( \delta(r) = 1 \) when the r-th retrieval instance is relevant to the query, otherwise \( \delta(r) = 0 \).

MAP is defined as the average of APs for all queries. R is set to 5,000 in our experiments. The ground-truth relevant instances for a query are defined as those sharing at least one label with the query. Precision-recall reports recall and the corresponding precision values. The TopN-precision denotes the precision at different numbers of retrieved instances.

We compare SSDH with several other unsupervised hashing methods including ITQ [Gong et al., 2013], SH [Weiss et al., 2009], DSH [Jin et al., 2014], SpH [Heo et al., 2012], DeepBit [Lin et al., 2016], and SGH [Dai et al., 2017], where ITQ, SH, DSH, SpH and SGH are shallow architecture-based methods, and DeepBit is a deep architecture-based method. All codes of these methods are kindly provided by the authors, and configurations for all methods are set according to the original papers.

6.4 Results and Discussion

As a global evaluation, we first present the MAP values for SSDH and all comparative methods across different hash code lengths on the two datasets. We then draw precision-recall and TopN-precision curves for SSDH and all baseline methods with 32 and 64 hash code lengths as a more comprehensive comparison.

Table 1 shows the MAP results for SSDH and all comparative methods on FLICKR25K with code lengths varying from 16 to 128. It can be seen that the proposed method consistently obtains the best results across different hash bit lengths. Specifically, the SSDH MAP obtains a relative improvement over the next-best baseline methods of 7.48%, 7.29%, 8.26%, and 7.66% for 16, 32, 64, and 128 bit lengths, respectively. Table 2 shows the MAP results for all methods on NUSWIDE. Again, our proposed method outperforms all other baseline methods in all cases. Specifically, the MAP of our proposed method obtains a relative improvement over the next-best baseline methods of 11.04%, 15.27%, 14.95%, and 14.33% for 16, 32, 64, and 128 bit lengths, respectively. The results presented in Table 1 and 2 well demonstrate the superiority of our proposed method.

Figures 3(c), 3(d), 4(c), and 4(d) show TopN-precision curves of SSDH and all comparative methods on FLICKR25K and NUSWIDE, respectively. It can clearly be seen that SSDH always achieves the highest precision. Since MAP values and TopN-precision curves are all based on Hamming ranking, taken together the above analysis shows that SSDH can achieve superior performance for Hamming ranking-based evaluations. To illustrate hash lookup results, we plot the recall-precision curves for SSDH and other comparative methods, as shown in Figures 3(a), 3(b), 4(a), and 4(b), where Figures 3(a) and 3(b) present the 32 bit length results and Figures 4(a) and 4(b) presents the 64 bit length results. It can again be seen that SSDH always achieves the best performance by a large margin, further demonstrating the superiority of our proposed method.

6.5 Parameter Sensitivity

We next investigate the sensitivity of hyper-parameters \( \alpha \) and \( \beta \). Figure 5 shows the effect of these two hyper-parameters in SSDH on NUSWIDE dataset with hash code lengths of 32 and 64 bits. We first fix \( \beta \) to 1 and evaluate the MAP values by varying \( \alpha \) between 0 and 3, the results are presented in Figure 5(a). SSDH performance first increases and then decreases as \( \alpha \) varies, showing a desirable bell-shaped curve. The best result is obtained when \( \alpha = 2 \), so we fix \( \alpha \) to 2 in our other experiments. According to a Gaussian distribution, only about 2.2% points are smaller than the threshold \( (m_1 - 2m_1) \), consistent setting data points with distances obviously smaller than the others as semantically similar points.
Here we present a novel deep unsupervised hashing method called semantic structure-based unsupervised deep hashing (SSDH). Our empirical study of the deep feature statistics shows that the cosine distance histogram distributions can be estimated by two half Gaussian distributions. Based on this observation, we construct a semantic structure to capture the semantic relationships and design a pair-wise loss function to preserve the relationships in Hamming space. Comparative studies on two benchmark datasets show that SSDH significantly outperforms current state-of-the-art methods.

6.6 Encoding Time
In this subsection, we compare the encoding time. For shallow methods, the overall time includes the time used for feature extraction. The results are shown in Figure 6, from which we can see that SSDH takes almost the same amount of time as DeepBit and both these two methods have comparative encoding time with ITQ, SH, DSH, and SpH. From the results, we can get that deep learning based methods enable as efficient encoding as most shallow methods.

7 Conclusion
Here we present a novel deep unsupervised hashing method called semantic structure-based unsupervised deep hashing (SSDH).
References


