Hamming Compatible Quantization for Hashing

Zhe Wang, Ling-Yu Duan, Jie Lin, Xiaofang Wang, Tiejun Huang, Wen Gao
The Institute of Digital Media, Peking University, Beijing, China
{zhew, lingyu, jielin, xiaofangwang, tjhuang, wgao}@pku.edu.cn

Abstract

Hashing is one of the effective techniques for fast Approximate Nearest Neighbour (ANN) search. Traditional single-bit quantization (SBQ) in most hashing methods incurs lots of quantization error which seriously degrades the search performance. To address the limitation of SBQ, researchers have proposed promising multi-bit quantization (MBQ) methods to quantize each projection dimension with multiple bits. However, some MBQ methods need to adopt specific distance for binary code matching instead of the original Hamming distance, which would significantly decrease the retrieval speed. Two typical MBQ methods Hierarchical Quantization and Double Bit Quantization retain the Hamming distance, but both of them only consider the projection dimensions during quantization, ignoring the neighborhood structure of raw data inherent in Euclidean space. In this paper, we propose a multi-bit quantization method named Hamming Compatible Quantization (HCQ) to preserve the capability of similarity metric between Euclidean space and Hamming space by utilizing the neighborhood structure of raw data. Extensive experiment results have shown our approach significantly improves the performance of various state-of-the-art hashing methods while maintaining fast retrieval speed.

1 Introduction

Approximate Nearest Neighbour (ANN) search plays an important role in many applications such as computer vision and information retrieval. In recent years there has been growing interest in hashing techniques for efficient ANN search. The goal of hashing is to map data points into compact binary codes whose Hamming distance approximates the similarity. In hashing, data points are usually represented as short codes, such as 32 or 64 bits, which are cost effective in data storage. What's more, under Hamming distance metric, retrieval speed can be very fast (10^9 XOR and POPCOUNT operations per second in modern CPU [He *et al.*, 2013]). Such properties make hashing an ideal method to handle the ANN search challenges in big data [Torralba *et al.*, 2008].

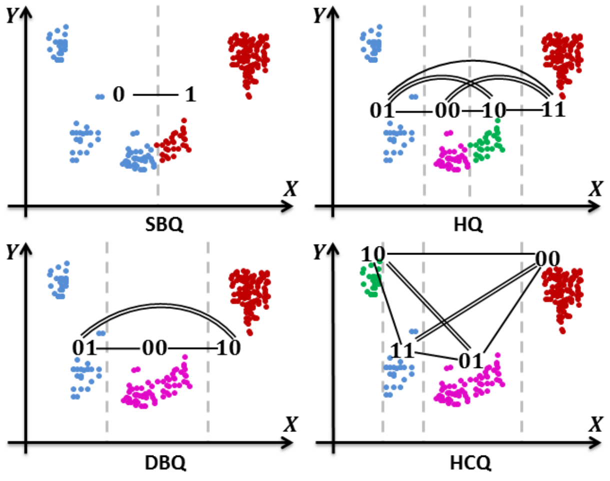


Figure 1: Examples of four quantization methods SBQ, HH, DBQ and HCQ to quantize the X-axis projection dimension on a set of 2D random points. A single line indicates the Hamming distance between two regions is 1, and a double line indicates 2. This figure is best viewed in color version.

Hashing methods typically involve two stages [Kong and L, 2012]: projection and quantization. Firstly, data points are transformed into low dimensional vectors with a linear or non-linear projection. Secondly, each projection dimension is quantized into binary code by a quantizer. Most of works focus on the improvement of the projection stage, with an aim to find good projection functions. Local Sensitive Hashing (LSH) [Andoni and Indyk, 2006] adopts a random Gaussian matrix as projection function. Principle Component Analysis Hashing (PCAH) [Wang *et al.*, 2006] uses the projection matrix learnt by principal component analysis. Iterative Quantization (ITQ) [Gong and Lazebnik, 2011] figures out an orthogonal rotation matrix to refine the initial projection matrix. Weighted Component Hashing (WCH) [Duan *et al.*, 2015] utilizes the dependency within each dimensionality to produce discriminative codes. Other typical projection methods include RBM [Hinton *et al.*, 2006], SH [Weiss *et al.*, 2008], KMH [He *et al.*, 2013] and SSH [Wang *et al.*, 2010].

Compared to projection, the quantization stage in hashing has attracted much less attention. Most existing hashing methods adopt single-bit quantization (SBQ) at the second stage [Kong and L, 2012], see figure 1. Specifically, given a projection dimension $Y^{(i)}$ in projected vector $Y \in R^q$ $(1 \leq i \leq q)$, $\mathbf{b}(Y^{(i)})$ is 1 if $Y^{(i)}$ is larger than a threshold θ. Otherwise, $\mathbf{b}(Y^{(i)})$ is 0. $\mathbf{b}$ denotes the quantizer in SBQ. SBQ may incur lots of quantization error [Kong and L, 2012], resulting in poor performance. We argue that quantization is equally important as projection. A good quantization mechanism may significantly improve the performance [Kong *et al.*, 2012]. Unfortunately, relatively little efforts were devoted to better quantization. In this paper, we will perform in-depth analysis regarding the quantization stage and further improve the performance of Hashing techniques.

Recently, researchers have attempted to address the limitation of SBQ by proposing multiple bit quantization (MBQ) methods to reduce the quantization error. Representative MBQ methods include Hierarchical Quantization (HQ) in [Liu *et al.*, 2011] and Double Bit Quantization (DBQ) in [Kong and L, 2012]. As shown in figure 1, HQ divides a projection dimension into four regions and encodes each region by a 2-bit code 01, 00, 10 and 11 respectively. However, the neighborhood structure would be destroyed. For instance, let's consider two elements, one in the first region and the other in the third region. Their Hamming distance equals $\mathbf{d}(01, 10) = 2$, which is larger than the Hamming distance between the first region and the fourth region , as $\mathbf{d}(01, 11) = 1$, which is unreasonable as two closer elements get a larger distance. Thus, Kong et al. proposed DBQ [Kong and L, 2012] to divide the projection dimension into three regions and encode them 01, 00 and 10 respectively. However, DBQ does not maximize the use of code space as two bits can represent four different values, leading to insufficient region division of the space.

In this work, we propose a novel multi-bit quantization method named Hamming Compatible Quantization (HCQ). The proposed method HCQ attempts to maximize the use of code space. In HQ and DBQ, each projection dimension is dealt with independently during quantization, ignoring the neighborhood structure of raw data in Euclidean space. Some data points may be actually far away although the distances in some individual projection dimensions are shorter. Hence, HCQ introduces a distance error function to describe the similarity preservation between Euclidean distance space and Hamming distance space, and maximize the similarity preservation. Moreover, a fast optimization algorithm is proposed to significantly reduce the computation complexity from $O(n^5)$ to $O(n^3)$, where n is the training data size. Experiments have shown HCQ significantly outperform HQ and DBQ. The main contributions are threefold:

(1) The proposed multi-bit quantization method with Hamming distance metric is essential for hashing techniques to solve the problem of fast ANN search. To the best of our knowledge, this is the first work to explicitly formulate and address the multi-bit quantization problem by incorporating practically important Hamming distance metric towards effective and efficient hashing techniques.

(2) We formulate the multi-bit quantization as an optimization problem by a minimizing distance error function, which elegantly preserves the capability of similarity metric between Euclidean space and Hamming space by utilizing the neighborhood structure of raw data. Moreover, a fast algorithm is proposed to solve the optimization problem.

(3) Extensive experiments on benchmark datasets have shown that our method significantly improves the performance of state-of-the-art multi-bit quantization methods and further advances the hashing technologies.

2 Related Work

Projection Methods. Most of existing hashing algorithms fall into this category and can be divided into two groups: (1) data-independent hashing, such as Locality Sensitive Hashing (LSH) [Andoni and Indyk, 2006] and Shift Invariant Kernel Hashing (SIKH) [Raginsky and Lazebnik, 2009], generates random projections independent of dataset. Data-independent methods normally generate long binary codes for satisfactory performance. (2) data-dependent hashing, such as Spectral Hashing (SH) [Weiss *et al.*, 2008], Principle Component Analysis Hashing (PCAH) [Wang *et al.*, 2006], Semi-supervised Hashing (SSH) [Wang *et al.*, 2010], Iterative Quantization (ITQ) [Gong and Lazebnik, 2011] and Restricted Boltzmann Machine (RBM) [Hinton *et al.*, 2006], aims to learn the projection functions from training data. In general, data-dependent hashing outperforms data-independent hashing. In this work, our focus is on the quantization stage, which is a valuable complement to the state-of-the-art projection methods for hashing.

Quantization Methods. Besides the aforementioned multi-bit quantization methods HQ and DBQ, previous works have studied more sophisticated multi-bit strategies.

Kong et al. proposed a multi-bit quantization method called Manhattan Hashing [Kong *et al.*, 2012] (MH). MH encodes each projection dimension into natural binary code (NBC) and adopts Manhattan distance to measure the similarity. For m-bits binary code, the range of Manhattan distance can be 0 to $2^m - 1$, which is more wider than the range of Hamming distance(0 to m), thereby yielding high retrieval performance. Moran et al. further improved MH by introducing an affinity matrix with F-measure criterion and proposing Neighborhood Preserving Quantization [Moran *et al.*, 2013a] (NPQ). Likewise, Lee et al. present a Quadra-Embedding [Lee *et al.*, 2012] (QE) quantization methods by adopting a specialized 'quadra' distance.

The above methods have achieved significant performance improvement over SBQ. However, those methods have fixed the bits assignment of each projection dimension, without allowing to vary the bit assignment across dimensions. Towards varying bit assignment, Moran et al. firstly proposed a variable-bit quantization [Moran *et al.*, 2013b] named VBQ to allocate bits for different projection dimensions. Then, Xiong et al. proposed an adaptive quantization [Xiong *et al.*, 2014] (AQ) strategy that assigns varying numbers of bits to different dimensions based on their information content.

However, all these methods need to adopt a special distance instead of the Hamming distance. Thus, retrieval speed would be much slower. Figure 2 compares the computing procedure of different distance. Given two codes A and B, only

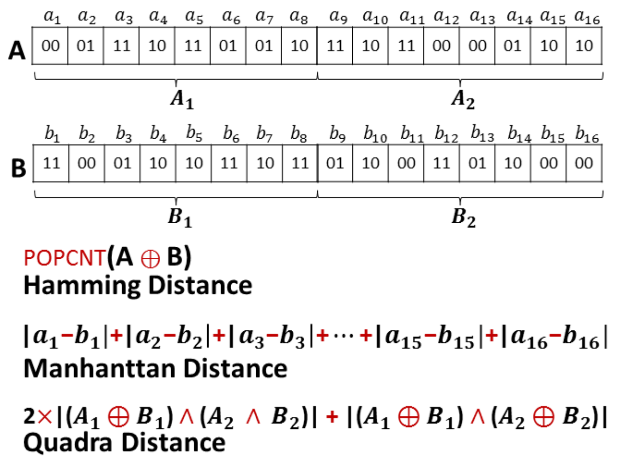

A
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13} a_{14} a_{15} a_{16}
00 01 11 10 11 01 01 10 11 10 11 00 00 01 10 10
A_1
A_2
B
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15} b_{16}
11 00 01 10 10 11 10 11 01 10 00 11 01 10 00 00
B_1
B_2
POPCNT(A ⊕ B)
Hamming Distance
$|a_1-b_1|+|a_2-b_2|+|a_3-b_3|+\cdots+|a_{15}-b_{15}|+|a_{16}-b_{16}|$
Manhanttan Distance
$2\times|(A_1 \oplus B_1)\wedge(A_2 \wedge B_2)| + |(A_1 \oplus B_1)\wedge(A_2 \oplus B_2)|$
Quadra Distance

Figure 2: The procedure of computing different distance to compare two 16×2 bits binary code A and B. Hamming distance only involves 2 atomic instructions: 1 OR and 1 POPCNT. Quadra distance involves 8 operations: 3 OR, 3 AND, 1 PLUS and 1 MULT. Manhanttan Distance involves 31 operations: 16 SUBTRACTION, 15 PLUS.

2 atomic machine instructions are involved to compute their Hamming distance. However, Quadra distance and Manhattan distance involve 8 and 31 CPU operations respectively. At 64 bits code size, Hamming distance is at least 46 times faster than Manhattan distance and 10 times faster than Quadra distance. Readers are referred to Section 4 for more details on retrieval speed.

To the best of our knowledge, only two multi-bit quantization methods HQ and DBQ retain the Hamming distance. Distinct from HQ and DBQ, our work proposes a more efficient and effective quantization method by incorporating the distance information of the original data points.

3 Hamming Compatible Quantization

Assume a projected vector $Y \in R^q$ is given. As our focus is the quantization stage, we don't consider how Y is generated. Given $Y^{(u)}$ $(1 \leq u \leq q)$, we aim to figure out a mapping

$$\mathbf{b} : R \rightarrow \{0, 1\}^b \tag{1}$$

to quantize $Y^{(u)}$ into binary code $\mathbf{b}(Y^{(u)})$.

In our method, we set $b = 2$. That is, each element is quantized into 2 bits. The reason is that the computation complexity of threshold learning will be much heavier when b increases. Moreover, setting b to more than 2 cannot incur noticeable performance improvement. Sometimes it even causes performance drop, which was shown in MH [Kong *et al.*, 2012]. Note that our related works, such as HQ [Liu *et al.*, 2011], DBQ [Kong and L, 2012] and NPQ [Moran *et al.*, 2013a], all assign b to 2.

Let $T = \{T_i | 1 \leq i \leq n, T_i \in R^q\}$ denote the training samples. The set of the u-th dimension projection of T is expressed as

$$S = \{s_i | 1 \leq i \leq n, s_i \in R\}, \tag{2}$$

where $s_i = T_i^{(u)}$, $1 \leq i \leq n$. Without loss of generality, s_i $(1 \leq i \leq n)$ are supposed to be in order and then we have $s_1 \leq s_2 \leq ... \leq s_n$ (if not, we rank T by the value of $T^{(u)}$).

3.1 Formulation

Our goal is to find $m - 1$ boundaries $c_1, c_2, ..., c_{m-1}$ to divide S into $m = 2^b = 4$ groups, $G_1 = \{s_1, ..., s_{c_1}\}$, $G_2 = \{s_{c_1+1}, ..., s_{c_2}\}$, ..., $G_m = \{s_{c_{m-1}+1}, ..., s_n\}$ and assign an index I_i to each group G_i, where $1 \leq i \leq m$. Then, any element $x \in G_i$ will be quantized to I_i, i.e.

$$\mathbf{b}(x) = I_i, x \in G_i. \tag{3}$$

Here, I_i is a binary code, and $\{I_1, I_2, ..., I_m\}$ is a permutation of the integers $\{0, 1, ..., 2^b - 1\}$. The left and right boundary of group G_i are denoted as G_i^L and G_i^R, where $G_i^L = min\{j | s_j \in G_i\}$ and $G_i^R = max\{j | s_j \in G_i\}$.

Given any $s_i, s_j \in S$, the Euclidean distance between their data points is

$$\mathbf{E}(s_i, s_j) = \mathbf{d}(T_i, T_j), \tag{4}$$

where $\mathbf{d}(.)$ denotes the Euclidean distance. After quantization, the distance between s_i and s_j will be approximated as

$$\mathbf{H}(s_i, s_j) = \mathbf{d_h}(\mathbf{b}(s_i), \mathbf{b}(s_j)), \tag{5}$$

where $\mathbf{d_h}(.)$ denotes the Hamming distance.

Considering the similarity preservation between Euclidean distance space and Hamming distance space, a good hashing method should hash close data points to close binary codes. When two data points are faraway in Euclidean distance space, their corresponding hashing codes should get a large distance in Hamming space as well [Andoni and Indyk, 2006]. An optimal quantization function $\mathbf{b}(.)$ is to minimize the distance error introduced by the Hamming approximation:

$$\min_{\mathbf{b}(.)} \sum_{x \in S} \sum_{y \in S} (\mathbf{E}(x, y) - \lambda \mathbf{H}(x, y))^2. \tag{6}$$

Here, λ is a constant scalar factor to balance the ranges of $\mathbf{E}$ and $\mathbf{H}$. The value of λ will be discussed in section 4. We call Eqn.6 as the distance error function.

There are two groups of variables in $\mathbf{b}(.)$, $m-1$ boundaries $\{c_i | 1 \leq i \leq m - 1\}$ and m indices $\{I_i | 1 \leq i \leq m\}$. Solving Eqn.6 consists of two steps: finding $m - 1$ boundaries $\{c_i\}$ to divide S into m groups, and assigning a binary code I_i to each groups G_i $(1 \leq i \leq m)$. For simplicity, we firstly fix the indices $\{I_i\}$ and then update $\{c_i\}$ to minimize the distance error function subject to the given assignment.

3.2 Optimization

A naive method to update $\{c_i\}$ would be: enumerate all possible values of c_1, c_2 and c_3, and calculate the distance error in Eqn.6. Those values yielding minimum distance error are selected as the solution. However, this brute-force algorithm has a $O(n^5)$ time complexity including $O(n^3)$ for enumerating c_i and $O(n^2)$ for calculating the distance error function, which is too heavy to solve a practical problem. The size of training data is usually a few thousands or more. When $n = 10000$, the order of magnitude of time complexity will increase up to 10^{20}. In practice, we even could not get the answer for this magnitude after nearly 4 days running on a Intel(R) Core(TM) i5 3470 CPU at 3.20GHz.

To update $\{c_i\}$ efficiently, we propose a very fast algorithm by reducing redundant operations. Let $\mathbf{F}(i,j)$ denote the sum of distance error between group G_i and G_j, where

$$\mathbf{F}(i,j) = \sum_{x \in G_i} \sum_{y \in G_j} (\mathbf{E}(x,y) - \lambda \mathbf{H}(x,y))^2. \tag{7}$$

Then, Eqn.6 can be rewritten as:

$$\min_{\mathbf{d}(.)} \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{F}(i,j). \tag{8}$$

Optimizing Eqn.6 is equivalent to optimize Eqn.8.

Next, we discuss how to efficiently calculate $\mathbf{F}(i,j)$. We expand the square item in Eqn.7,

$$\begin{aligned}\mathbf{F}(i,j) = \sum_{x \in G_i} \sum_{y \in G_j} \{&\mathbf{E}^2(x,y) + \lambda^2 \mathbf{H}^2(x,y) \\ &- 2\lambda \mathbf{E}(x,y)\mathbf{H}(x,y)\}.\end{aligned} \tag{9}$$

In Eqn.9, $\mathbf{H}(x,y)$ is a constant and $\mathbf{H}(x,y) = \mathbf{d_h}(I_i, I_j)$. Let $\mathbf{A}$ and $\mathbf{B}$ denote the sum of $\mathbf{E}^2(x,y)$ and $\mathbf{E}(x,y)$ respectively, where $x \in G_i$ and $y \in G_j$. Then Eqn.9 is represented as

$$\mathbf{F}(i,j) = \mathbf{A} - 2\lambda\sigma\mathbf{B} + \lambda^2\sigma^2 |G_i||G_j|, \tag{10}$$

where σ denotes the constant term $\mathbf{d_h}(I_i, I_j)$. In Eqn.10, we can efficiently solve $\mathbf{F}(i,j)$ by directly reading the values of $\mathbf{A}$ and $\mathbf{B}$. As λ, σ, $|G_i|$ and $|G_j|$ are all constant items, once $\mathbf{A}$ and $\mathbf{B}$ are given, $\mathbf{F}(i,j)$ can be calculated in $O(1)$ time. Then, we define two function

$$\begin{aligned}\mathbf{J}(i,j) =& \sum_{p=1}^{i} \sum_{q=1}^{j} \mathbf{E}(s_p, s_q) = \sum_{p=1}^{i} \sum_{q=1}^{j} \mathbf{D}(T_p, T_q) \\ \mathbf{K}(i,j) =& \sum_{p=1}^{i} \sum_{q=1}^{j} \mathbf{E}^2(s_p, s_q) = \sum_{p=1}^{i} \sum_{q=1}^{j} \mathbf{D}^2(T_p, T_q).\end{aligned} \tag{11}$$

Let $l_1 = G_i^L - 1$, $r_1 = G_i^R$, $l_2 = G_j^L - 1$ and $r_1 = G_j^R$, and we have

$$\mathbf{A} = \mathbf{J}(r_1, r_2) - \mathbf{J}(l_1, r_2) - \mathbf{J}(r_1, l_2) + \mathbf{J}(l_1, l_2). \tag{12}$$

Likewise, factor $\mathbf{B}$ can be modified as

$$\mathbf{B} = \mathbf{K}(r_1, r_2) - \mathbf{K}(l_1, r_2) - \mathbf{K}(r_1, l_2) + \mathbf{K}(l_1, l_2). \tag{13}$$

By pre-computing $\mathbf{J}(.)$ and $\mathbf{K}(.)$, the results of $\mathbf{A}$ and $\mathbf{B}$ can be directly obtained from the look-up tables. $\mathbf{F}(i,j)$ can be efficiently calculated in $O(1)$ time. Algorithm 1 shows the pseudo-code.

In our work, since $\{I_i\}$ is a permutations of $\{00, 01, 10, 11\}$ and there are only in total $4! = 24$ instances, we just assign all permutations of $\{00, 01, 10, 11\}$ to I_i and solve Eqn.8 for each assignment. The time complexity is $O(24(n^2q + 16n^3)) = O(n^2q + n^3)$, including $O(n^2q)$ for computing $\mathbf{J}(.)$ and $\mathbf{K}(.)$, and $O(16n^3)$ for finding the minimal value. We just enumerate 24 instances. Since q is usually much smaller than n, the time complexity can be considered as $O(n^3)$. The space complexity is $O(n^2)$, due to $n \times n$ lookup tables $\mathbf{J}(.)$ and $\mathbf{K}(.)$.

Algorithm 1 The fast algorithm to solve the distance error function in Eqn.8.

```
Input: The training dataset $S$ and $T$, index $I_i (1 \le i \le m)$, factor $\lambda$.
Output: The solution $c_1$, $c_2$ and $c_3$.
  Initialize the variable of minimal distance error $min = \infty$.
  Define $B_1 = \{(i,j,k) | 1 \le i < j < k \le n, n = |s|\}$ and $B_2 = \{(i,j) | 1 \le i,j \le 2^2\}$.
  Compute the results of $\mathbf{J}(i,j)$ and $\mathbf{K}(i,j)$, $0 \le i,j \le n$.
  for each $(c_1^*, c_2^*, c_3^*) \in B_1$ do
    Initialize $value \leftarrow 0$, $G_1 \leftarrow \{s_k | 1 \le k \le c_1^*\}$, $G_2 \leftarrow \{s_k | c_1^* < k \le c_2^*\}$, $G_3 \leftarrow \{s_k | c_2^* < k \le c_3^*\}$ and $G_4 \leftarrow \{s_k | c_3^* < k \le n\}$.
    for each $(i,j) \in B_2$ do
      $\sigma = \mathbf{d_h}(I_i, I_j)$.
      $l_1 = G_i^L - 1$, $r_1 = G_i^R$, $l_2 = G_j^L - 1$ and $r_2 = G_j^R$.
      $\mathbf{A} = \mathbf{J}(r_1, r_2) - \mathbf{J}(l_1, r_2) - \mathbf{J}(r_1, l_2) + \mathbf{J}(l_1, l_2)$.
      $\mathbf{B} = \mathbf{K}(r_1, r_2) - \mathbf{K}(l_1, r_2) - \mathbf{K}(r_1, l_2) + \mathbf{K}(l_1, l_2)$.
      $\mathbf{F} = \mathbf{A} - 2\lambda\sigma\mathbf{B} + \lambda^2\sigma^2|G_i||G_j|$.
      $value \leftarrow value + \mathbf{F}$.
    end for
    if $value < min$ then
      $min \leftarrow value$.
      $(c_1, c_2, c_3) = (c_1^*, c_2^*, c_3^*)$.
    end if
  end for
```

4 Experiment

4.1 Dataset and Evaluation Protocol

Extensive experiments were carried out over three widely used retrieval benchmark datasets, LabelMe22K [Torralba *et al.*, 2008], CIFAR-10 [Krizhevsky, 2009] and NUS-WIDE [Chua *et al.*, 2009]. The CIFAR-10 dataset contains 60,000 images of size 32x32 and has been categorized into 10 classes. The LabelMe22K dataset contains 22,019 images. As setup in [Kong and L, 2012], [Kong *et al.*, 2012] and [Moran *et al.*, 2013a], we represent each image with a 512 dimensional gray-scale GIST descriptor [Aude and Torralba, 2001]. The NUS-WIDE dataset contains 269,648 images. We use the 225-D block-wise color feature to represent each image.

We report the ANN search results using Euclidean nearest neighbours as the ground-truth. Thresholding is applied to determine whether a returned database image is true positive or not. Following [Kong and L, 2012; Kong et al., 2012; Moranet al., 2013a; 2013b], a threshold is set as the average distance between each of randomly selected 100 data points and their returned the 50th nearest neighbours. We randomly select 1000 images as queries, and the remaining images as reference database images. We use mean Average Precision (mAP) to evaluate the search accuracy. We repeated the retrieval experiments by 10 times. In addition, we measure the search time on an Intel(R) Core(TM) i5 3470 CPU at 3.20GHz with a single thread.

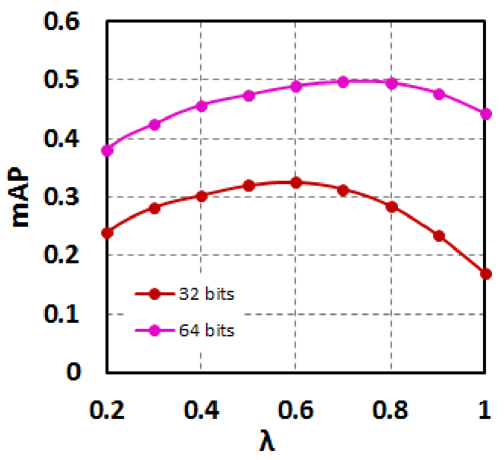


Figure 3: Impact of parameter λ over Lableme-22K dataset at code size 32 and 64. ITQ projection is used.

4.2 Baselines

We choose five representative projection methods, ITQ [Gong and Lazebnik, 2011], SH [Weiss *et al.*, 2008], PCAH [Wang *et al.*, 2006], LSH [Andoni and Indyk, 2006] and SIKH [Raginsky and Lazebnik, 2009], and combine them with different quantization methods for hashing. Note that ITQ, SH and PCAH are data-dependent methods, while LSH and SIKH are data-independent methods.

By leveraging the aforementioned projection methods, we study the ANN search accuracy of our method versus state-of-the-art single/multiple bit quantization schemes:

- SBQ: single bit quantization.
- HQ [Liu *et al.*, 2011]: hierarchical hashing uses k-means clustering to divide each dimension into 4 regions, and encodes the regions as 01, 00, 10, 11 from left to right.
- DBQ [Kong and L, 2012]: double-bit quantization with 01, 00, 10 from left to right.
- HCQ: the proposed Hamming Compatible Quantization.

To the best of our knowledge, only two multi-bit quantization methods HQ and DBQ retaining the Hamming distance were reported and we select both of them as baselines. We compare the search time cost of the proposed quantization method with Hamming distance and other typical quantization algorithms with Non-Hamming distance, i.e., MH [Kong *et al.*, 2012] and NPQ [Moran *et al.*, 2013a] with Manhattan distance, and QE [Lee *et al.*, 2012] with Quadra distance. In our experiments, we also compare with Manhattan Hashing [Kong *et al.*, 2012] which adopts Manhattan distance.

4.3 Impact of parameters

Training Sample n. Table 1 shows the impact of the training sample size n for our HCQ. Learning over a small training set (say 1000) has been sufficient for stable and promising performance. More training instances won't incur noticeable improvements while fewer instances may degrade the performance. In practice, we select 1000 images from the database to learn the quantization boundaries. The overall training stage is very efficient and can be finished in just five minutes.

Code Size	Training Sample n				
	200	500	1000	2000	5000
32 bits	0.125	0.203	0.346	0.344	0.350
64 bits	0.235	0.323	0.471	0.473	0.469
128 bits	0.357	0.465	0.608	0.612	0.611
256 bits	0.398	0.512	0.641	0.637	0.639

Table 1: Impact of the training sample size n for HCQ. We adopt the ITQ projection, and evaluate the retrieval performance with different n at code size 32, 64, 128 and 256, respectively.

#bits	32	64	128	256
Hamming Distance	0.15	0.23	0.39	0.81
Manhattan Distance	5.93	10.70	18.81	37.52
Quadra Distance	1.45	2.53	4.97	10.51

Table 2: The retrieval time cost (s) on LableMe22K with different distance metric, including Hamming distance, Manhattan distance and Quadra distance.

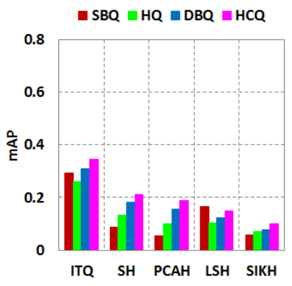
Parameter λ. Figure 3 shows the effect of scalar constant factor λ in Eq. 6 at code size of 32 bits and 64 bits on LabelMe22K dataset. As one can see, for both 32 bits and 64 bits, there is significantly search performance drop (in mAP) when λ becomes smaller (close to 0) or larger (close to 1). Similar trends have been observed at code size 128 bits and 256 bits. It is observed that the optimal λ increases with the code size, e.g., the best mAP is achieved when $\lambda = 0.6$ at 32 bits and $\lambda = 0.7$ at 64 bits. This is reasonable since the range of Hamming distance increases with the code size, thereby we should enlarge λ to make sure that the Euclidean distance and the Hamming distance have comparable order of magnitude. In the following experiments, we set $\lambda = 0.6, 0.7, 0.8, 0.9$ at code size 32, 64, 128, 256, respectively.

4.4 Performance

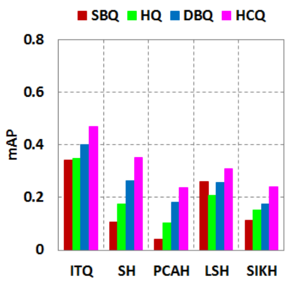
Comparison with State-of-the-art. Figure 4, figure 5 and figure 6 list the search accuracy mAP when combining different quantization methods and different projection methods on dataset LabelMe22K, CIFAR-10 and NUS-WIDE respectively. When combining SBQ with different projection methods, we have the original baseline algorithms (i.e., ITQ, SH, PCA, LSH and SIKH). For multi-bit quantization HH, DBQ and HCQ, we use double bits to encode each projected dimension.

For fair comparison, HCQ firstly projects data points into c/2 dimension at code size c, which is also applied to other two multi-bit quantization methods HQ and DBQ. Experiment results show that HCQ significantly outperforms SBQ although only half of projection dimensions are used. This has validated the importance of the quantization stage to promote hashing performance.

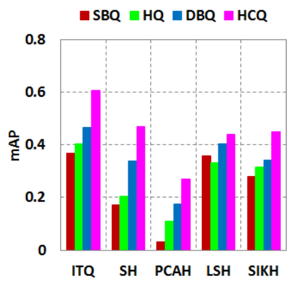
As listed in figure 4, figure 5 and figure 6, the proposed HCQ consistently outperforms SBQ for all settings even at code size 32. At small code size, HQ and DBQ perform worse



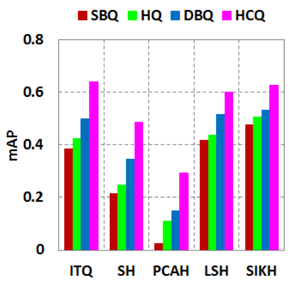
(a) LabelMe-22K 32-bit



(b) LabelMe-22K 64-bit

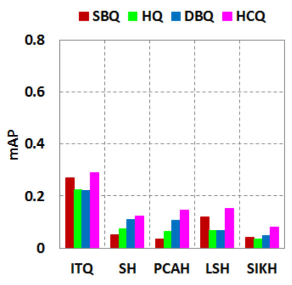


(c) LabelMe-22K 128-bit

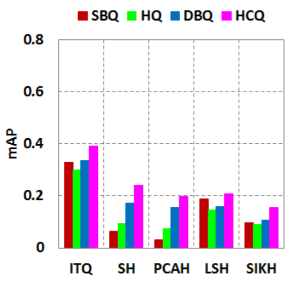


(d) LabelMe-22K 256-bit

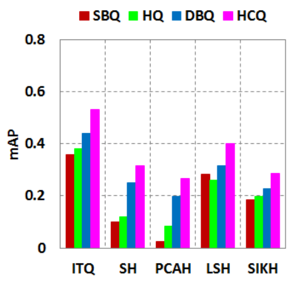
Figure 4: The results of mAP on LableMe-22K dataset at code size of 32 bits, 64 bits, 128 bits and 256 bits.



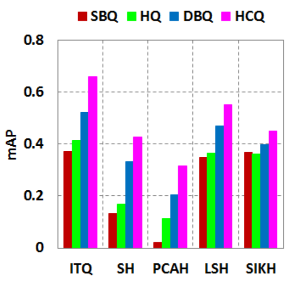
(a) CIFAR-10 32-bit



(b) CIFAR-10 64-bit



(c) CIFAR-10 128-bit



(d) CIFAR-10 256-bit

Figure 5: The results of mAP on CIFAR-10 dataset at code size of 32 bits, 64 bits, 128 bits and 256 bits.

#bits	32		64		128	
Method	MH	HCQ	MH	HCQ	MH	HCQ
mAP	0.354	0.346	0.488	0.471	0.591	0.608
Time(s)	5.93	0.15	10.70	0.23	18.81	0.39

Table 3: Comparison with Manhattan Hashing on LabelMe22K dataset at code size 32, 64 and 128.

than SBQ in some cases (e.g., HQ+LSH and DBQ+LSH perform worse than SBQ+LSH at 32 bits on LabelMe22K and CIFAR-10), while HCQ outperforms SBQ expect just one point (i.e., HCQ+LSH at 32 bits on LabelMe22K dataset). At code size 32, data points are projected into 16 dimension and lots of information would be lost. But HCQ still outperforms SBQ. This demonstrates the effectiveness of HCQ.

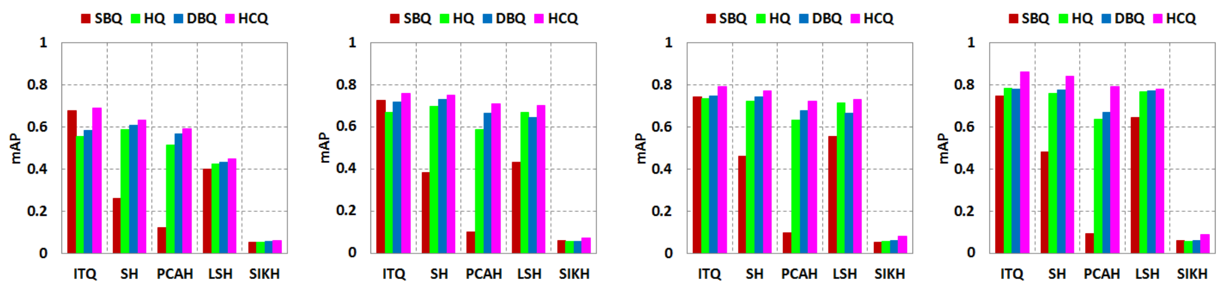
Referring to the listed results, HCQ achieves much better performance than other two multi-bit quantization methods HQ and DBQ. The performance gap becomes larger as the code size increases. At 32 bits, HCQ is about +9% mAP better than HQ and +4% better than DBQ on LabelMe22K with ITQ projection. While, at 256 bits, HCQ is +21% mAP better than HQ and 14% mAP better than DBQ on LabelMe22K with ITQ projection. At 256 bits, HCQ is even +25% mAP better than SBQ on LabelMe22K with ITQ projection(i.e., 64.11% vs 38.46%). Considerable mAP improvements are yielded over CIFAR-10 and NUS-WIDE as well. The performance gain from quantization is significant to promote hashing techniques.

Retrieval Speed. Table 2 lists the search time on LabelMe22K dataset using different metrics including Manhattan distance in MH [Kong *et al.*, 2012] and NPQ [Moran *et al.*, 2013a] and Quadra distance in QE [Lee *et al.*, 2012]. The Hamming distance computation is much faster than others at the same code size. For instance, at 32 bits, Hamming distance is about 40 times faster than Manhanttan distance and 10 times faster than Quadra distance. More importantly, the search speedup increases with the code size. This is consistent with the theoretical analysis in Section 2.

Comparison with Manhattan Hashing. We also compare HCQ with another multi-bit quantization method Manhattan Hashing (MH) which adopts Manhattan distance. Note that Manhattan distance has natural advantages over Hamming distance due to much wider distance range, but Manhattan distance incurs more computation complexity. In our experiments, MH firstly projects data points into c/2 dimension and then quantizes each of them into 2 bits at code size c. As listed in Table 3, HCQ achieves comparable retrieval performance with MH, and even better than MH at code size 128. However, the proposed HCQ is with more than 40 times faster retrieval speed than MH. At code size 128, HCQ even yields slightly better results than MH. The performance advantages have again shown that quantization is crucial for improving hashing techniques.

5 Conclusion

Quantization is crucial for performance, and Hamming distance metric contributes to fast retrieval speed. The proposed Hamming Compatibility Quantization not only benefits the



(a) NUS-WIDE 32-bit (b) NUS-WIDE 64-bit (c) NUS-WIDE 96-bit (d) NUS-WIDE 128-bit

Figure 6: The results of mAP on NUS-WIDE dataset at code size of 32 bits, 64 bits, 96 bits and 128 bits.

multi-bit quantization but also supports highly efficient Hamming distance metric. HCQ has significantly improved the performance of state-of-the-art hashing methods.

Acknowledgments

This work was supported by the Chinese Natural Science Foundation under Contracts No. 61271311, No. 61390515, No. 61421062 and by the National Hightech R&D Program of China (863 Program) under Grant No. 2015AA016302. Ling-Yu Duan is the correspondence author.

References

[Andoni and Indyk, 2006] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *In IEEE FOCS*, 2006.

[Aude and Torralba, 2001] Oliva Aude and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 2001.

[Chua *et al.*, 2009] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao Zheng. Nus-wide: A real-world web image database from national university of singapore. *ACM International Conference on Image and Video Retrieval*, 2009.

[Duan *et al.*, 2015] Ling-Yu Duan, Jie Lin, Zhe Wang, Tiejun Huang, and Wen Gao. Weighted component hashing of binary aggregated descriptors for fast visual search. *IEEE Trans. Multimedia*, 2015.

[Gong and Lazebnik, 2011] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. *CVPR*, 2011.

[He *et al.*, 2013] Kaiming He, Fang Wen, and Jian Sun. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. *CVPR*, 2013.

[Hinton *et al.*, 2006] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 2006.

[Kong and L, 2012] Weihao Kong and Wu-Jun L. Double-bit quantization for hashing. *In Proceedings of the Twenty-Sixth AAAI*, 2012.

[Kong *et al.*, 2012] Weihao Kong, Wu-Jun Li, and Minyi Guo. Manhattan hashing for large-scale image retrieval. *In Proceedings of the 35th international ACM SIGIR*, 2012.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Tech report, University of Torontos*, 2009.

[Lee *et al.*, 2012] Youngwoon Lee, Jae-Pil Heo, and Sung-Eui Yoon. Quadra-embedding: Binary code embedding with low quantization error. *In Proceedings of ACCV*, 2012.

[Liu *et al.*, 2011] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. *ICML*, 2011.

[Moran *et al.*, 2013a] Sean Moran, Victor Lavrenko, and Miles Osborne. Neighbourhood preserving quantisation for lsh. *In Proceedings of the 36th international ACM SIGIR*, 2013.

[Moran *et al.*, 2013b] Sean Moran, Victor Lavrenko, and Miles Osborne. Variable bit quantisation for lsh. *In Proceedings of ACL*, 2013.

[Raginsky and Lazebnik, 2009] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. *In NIPS*, 2009.

[Torralba *et al.*, 2008] Antonio Torralba, Robert Fergus, and Yair Weiss. Small codes and large image databases for recognition. *In Proceedings of CVPR*, 2008.

[Wang *et al.*, 2006] Xin-Jing Wang, Lei Zhang, Feng Jing, and Wei-Ying Ma. Annosearch: Image auto-annotation by search. *CVPR*, 2006.

[Wang *et al.*, 2010] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. *CVPR*, 2010.

[Weiss *et al.*, 2008] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. *In NIPS*, 2008.

[Xiong *et al.*, 2014] Caiming Xiong, Wei Chen, Gang Chen, David Johnson, and Jason J. Corso. Adaptive quantization for hashing: An information-based approach to learning binary codes. *In Proceedings of ICDM*, 2014.