Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Affinity Preserving Quantization for Hashing: A Vector Quantization Approach to Learning Compact Binary Codes

Zhe Wang, Ling-Yu Duan, Tiejun Huang, Wen Gao
Institute of Digital Media, Peking University
No.5 Yiheyuan Road Haidian District, Beijing, P.R.China 100871
{zhew,lingyu,tjhuang,wgao}@pku.edu.cn

Abstract

Hashing techniques are powerful for approximate nearest neighbour (ANN) search. Existing quantization methods in hashing are all focused on scalar quantization (SQ) which is inferior in utilizing the inherent data distribution. In this paper, we propose a novel vector quantization (VQ) method named affinity preserving quantization (APQ) to improve the quantization quality of projection values, which has significantly boosted the performance of state-of-the-art hashing techniques. In particular, our method incorporates the neighbourhood structure in the pre- and post-projection data space into vector quantization. APQ minimizes the quantization errors of projection values as well as the loss of affinity property of original space. An effective algorithm has been proposed to solve the joint optimization problem in APQ, and the extension to larger binary codes has been resolved by applying product quantization to APQ. Extensive experiments have shown that APQ consistently outperforms the state-of-the-art quantization methods, and has significantly improved the performance of various hashing techniques.

Introduction

Approximate nearest neighbour (ANN) search is widely used in machine learning, computer vision and information retrieval. In the past years, there has been increasing interest in learning binary codes to fulfill efficient and effective ANN search by hashing techniques. Hashing aims to learn compact binary representation for data which is supposed to preserve the similarity structure in the original space (Weiss, Torralba, and Fergus 2008). The compact representation is efficient in reducing memory usage as well as improving retrieval speed, so that hashing is becoming one of the most powerful techniques in dealing with ANN search in massive data (Torralba, Fergus, and Weiss 2008).

Typically, hashing methods consist of two basic stages: projection and quantization (Weiss, Torralba, and Fergus 2008). Hashing methods firstly transform data point $\mathbf{v} \in \mathbb{R}^D$ into a low dimensional vector $\mathbf{x} = [\mathrm{f}_1(\mathbf{v}), ..., \mathrm{f}_d(\mathbf{v})] \in \mathbb{R}^d$ where real-valued functions $\{\mathrm{f}_i(.)\}_{i=1}^d$ are usually called projection functions, and then quantize the projection values into binary codes. Many great research efforts have been devoted to the projection stage, with an aim to learn powerful projections via machine learning approaches, such as linear learning (Andoni and Indyk 2006), spectral learning (Weiss, Torralba, and Fergus 2008), kernal learning (Raginsky and Lazebnik 2009), manifold learning (Irie et al. 2014), graph learning (Liu et al. 2011) and others (He, Wen, and Sun 2013). Compared with the projection stage, how to make high quality quantization has received less research efforts, which, however, may significantly impact ANN search performance (Kong and Li 2012). Most hashing works adopt a single bit quantization (SBQ) (Kong and Li 2012) method to quantize projection values. SBQ simply applies thresholds to quantize each projection element $\mathbf{x}(i) = \mathrm{f}_i(\mathbf{v})$ into a single bit 0 or 1.

Recent works have reported the impact of quantization on the hashing performance. As single bit quantization (SBQ) may incur serious quantization distortion, the hashing performance would degenerate much (Kong and Li 2012; Kong, Li, and Guo 2012). To address the quantization distortion in SBQ, several promising multiple bit quantization (MBQ) methods have been proposed such as double bit quantization (DBQ) (Kong and Li 2012), manhattan quantization (MQ) (Kong, Li, and Guo 2012), adaptive quantization (AQ) (Xiong et al. 2014), hamming compatible quantization (HCQ) (Wang et al. 2015), etc. Different from SBQ, MBQ methods assign multiple bits to each projection element, which helps to reduce the information loss in quantization stage. Experiment results have shown that MBQ methods greatly improve the performance of various hashing methods (Kong and Li 2012; Kong, Li, and Guo 2012; Wang et al. 2015). High quality quantization is crucial in improving hashing performance.

In this work, we study how to further improve the performance of quantization stage in hashing from the vector quantization (VQ) (Allen and Gray 2012) point of view. All the previous SBQ and MBQ methods fall into the category of scalar quantization (SQ) (Allen and Gray 2012), in which each projection dimension (or element) is quantized separately. However, a projection value is actually a vector presentation, which renders SQ approaches inferior in utilizing the inherent vector data distribution. By contrast, VQ excels in adapting a quantizer to the true data space in terms of real vectors. Figure 1 shows an example of different scalar and vector quantization methods. In this example, data points are projected into a 2D space at the projection stage. Scalar

quantization approaches (SBQ or MBQ) quantize each element of projection values by using a couple of thresholds to divide each projection dimension into several regions, which works like two groups of horizontal and vertical cutting lines to partition the 2D space. No matter how to adjust the setting of thresholds, any scalar quantizer always result in rectangular quantization cells (Figure 1(a) and 1(b)). In contrast, a vector quantizer is free from geometrical restrictions and may have arbitrary quantization cell shapes as indicated in the examples of Figures 1(c) and 1(d). VQ is much more flexible in space partition than SQ. In addition, both the SBQ and MBQ methods have a limited distance range in terms of similarity measurement. After encoding data points into binary codes, they compute the distance (Hamming distance or Manhattan distance) between binary codes to approximate the similarity between the original data points. Both the ranges of Hamming distance and Manhattan distance are limited. For example, the Hamming distance over 128-bits codes is only up to 128, while the actual distance between any two points is a real value.

Unlike the SBQ or MBQ methods, the proposed VQ technique, in the context of hashing, deals with each projection vector as an elementary unit. The practice is to partition the vector space into Voronoi cells based on the data distribution and quantize the vector space into finite codewords (Jegou, Douze, and Schmid 2011). The distance range in VQ can be extended to any real-value codeword distance (see section 3 for details). More importantly, we propose a novel vector quantization approach named affinity preserving quantization (APQ) which minimizes the quantization errors of projection values as well as the loss of affinity property of original data points. The proposed APQ is meant to explicitly preserve the affinity in the original data space through reducing the distance bias between pre- and post-projection values from quantization. As shown in Fig.1, the neighborhood structure in original space may be already partially damaged at the projection stage (seeing those dots in red or pink colors), and thus the projection values cannot exactly tell the neighborhood structure in the original space. Hence, the proposed APQ utilizes the affinity in the original space to refine the VQ quantizer, which tends to quantize those close data points into similar codes (Fig 1(c) vs Fig 1(d)). We summarize the main contributions of this paper as follows:

- We have proposed to incorporate vector quantization into hashing to address the issues of non-adaptive projection space partition and limited distance range of the previous SBQ and MBD methods (scalar quantization), which significantly impact the performance of state-of-the-art hashing techniques. To the best of our knowledge, this is the first work on revealing the positive effects of vector quantization on hashing techniques.
- Furthermore, we proposed a novel affinity preserving quantization approach to strengthen the maintenance of neighborhood structure over the course of hashing, in terms of similarity distance of close points in the original space, which can greatly enhance the quantization quality.
- Extensive experiments over three large-scale benchmarks have shown that the proposed APQ approach has consistently outperformed state-of-the-art SBQ and MBQ methods, and significantly improved the elementary quantization stage of hashing techniques.

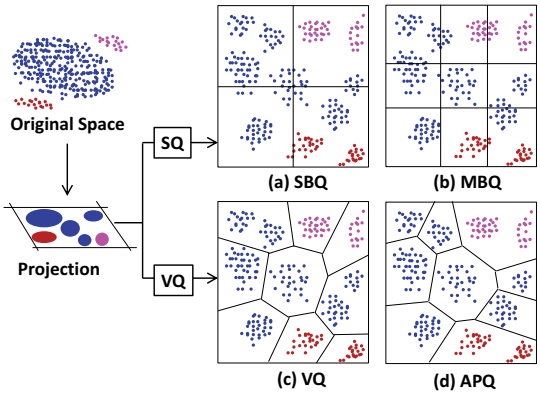


Figure 1: Examples of scalar quantization methods such as single bit quantization (SBQ), multiple bit quantization (MBQ), and vector quantization methods such as baseline VQ and the proposed APQ. Data points are transformed into a 2D space at the projection stage. The dots in red (or pink) color denote the near neighbors in the original space.

Related Work

In this section, we discuss the existing quantization works in the context of hashing. Let $\mathbf{v} \in \mathbb{R}^D$ denote a data point. Hashing methods first project $\mathbf{v}$ into a low dimensional vector $\mathbf{x} \in \mathbb{R}^d$ ($d < D$), and then quantize the projection vector $\mathbf{x}$ into binary codes. Single bit quantization (Kong and Li 2012) applies a threshold to quantize each element of projection vectors into a single bit. Specifically, the i-th projection element $\mathbf{x}(i)$ ($1 \leq i \leq d$) is encoded as 1 if $\mathbf{x}(i) > \theta$. Otherwise, 0. This strategy incurs lots of quantization distortions, which results in less discriminative codes (Kong and Li 2012; Kong, Li, and Guo 2012).

To alleviate the quantization distortion issue of SBQ, researchers have proposed promising multiple bits quantization (MBQ) methods (Kong and Li 2012; Kong, Li, and Guo 2012). MBQ methods quantize each element of a projection value into multiple bits to reduce the information loss during the quantization stage. In general, we may categorize bits assignment schemes into uniform quantization (Kong and Li 2012) and non-uniform quantization (Moran, Lavrenko, and Osborne 2013b). Uniform quantization quantizes all the elements of a projection vector at a fixed number of bits, while non-uniform quantization may allocate variable number of bits to different elements.

Most MBQ methods fall into the category of uniform quantization, such as double bit quantization (DBQ) (Kong and Li 2012), manhattan quantization (MQ) (Kong, Li, and Guo 2012) and Hamming compatible quantization (HCQ) (Wang et al. 2015). DBQ (Kong and Li 2012) divides each projection dimension into three regions and then uses a 2-bit code to represent each element. MQ (Kong, Li, and Guo 2012) uses natural binary code (NBC) and

adopt Manhattan distance to compute the distance between the NBC codes. Neighbourhood preserving quantization (NPQ) (Moran, Lavrenko, and Osborne 2013a) works in a similar way as MQ, which uses F-measure criterion to learn thresholds under Manhattan distance measurement. Hamming compatible quantization (HCQ) (Wang et al. 2015) utilizes a distance error function to preserve the capability of similarity metric between Euclidean space and Hamming space. Other uniform quantization methods include hierarchical quantization (HQ) (Liu et al. 2011) and Quadra embedding quantization (EQ) (Lee, Heo, and Yoon 2012).

The above uniform quantization strategies do not allow for the assignment of different bits across elements. Non-uniform quantization addresses this limitation. Typical non-uniform works include variable bit quantization (VBQ) (Moran, Lavrenko, and Osborne 2013b) and adaptive quantization (AQ) (Xiong et al. 2014). VBQ dynamically allocates the number of bits across different elements of a projection vector based on a combination of F-measure score (Moran, Lavrenko, and Osborne 2013b). AQ adaptively assigns varying numbers of bits to each projection dimension based on their informative content as well.

In summary, MBQ methods have significantly improved SBQ. In addition, the adaptive bits allocation mechanism across different elements of a projection vector may benefit the quantization. However, all these methods work on scalar quantization. In this paper, our focus is on vector quantization, which elegantly combines the advantages of multiple bits quantization and the joint optimization of multiple dimensions of projection vectors.

Affinity Preserving Quantization

Vector quantization is a classical signal quantization technique (Allen and Gray 2012). Formally, a vector quantizer is defined as a function

$$\mathrm{q} : \mathbb{R}^d \rightarrow \{\mathbf{c}_0, \mathbf{c}_1, ..., \mathbf{c}_{N-1}\} \tag{1}$$

that maps d-dimension vectors in the vector space $\mathbb{R}^d$ into a finite set of vectors $\mathcal{C} = \{\mathbf{c}_i : i = 0, 1, ..., N-1, \mathbf{c}_i \in \mathbb{R}^d\}$. Each vector $\mathbf{c}_i$ is called a codeword, and the set of all the codewords $\mathcal{C}$ forms a codebook, which is derived by offline training. Associated with each codeword $\mathbf{c}_i$, its nearest neighbor region R_i called Voronoi cell is defined by

$$R_i = \{\mathbf{x} \in \mathbb{R}^d : \forall_{j \neq i} \|\mathbf{x} - \mathbf{c}_i\| < \|\mathbf{x} - \mathbf{c}_j\|\}. \tag{2}$$

The Voronoi cells partition the entire space $\mathbb{R}^d$ such that:

$$(i \neq j) \Rightarrow R_i \cap R_j = \emptyset \quad and \quad \bigcup_{i=0}^{N-1} R_i = \mathbb{R}^d. \tag{3}$$

Given an input vector, the codeword located in the same Voronoi cell as the input vector is chosen, and the input vector is encoded as the index of the codeword in codebook.

Different from conventional multiple bits quantization methods, a vector quantization approach employs the indices of codewords to encode data points. Given two binary codes b_1 and b_2, the distance is represented by the distance of the two codewords $\mathbf{c}_{b_1}$ and $\mathbf{c}_{b_2}$. The distance space of $N = 2^b$ codewords yields in total $N^2 = 2^{2b}$ situations. Clearly, a VQ based coding strategy is much more effective than SBQ and DBQ methods in representing the similarity of data points. Moreover, a lookup strategy may make the distance computation very efficient. We precompute the distance between any two codewords and save the results in a $N \times N$ lookup table offline. Then, we can directly read the distance by two indices from the lookup table online.

Formulation

Let $\mathbf{V} = \{\mathbf{v}_i\}_{i=1}^l$ denote the sample set and $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^l$ be the projection vectors, where $\mathbf{x}_i$ corresponds to the projection vector of $\mathbf{v}_i$ ($1 \leq i \leq n$). For an input vector $\mathbf{x}_i$, the smaller the quantization error between $\mathbf{x}_i$ and $\mathrm{q}(\mathbf{x}_i)$, the better the quantization value $\mathrm{q}(\mathbf{x}_i)$ is to preserve the information of $\mathbf{x}_i$. VQ methods typically work out an optimal quantization by minimizing the mean squared error (MSE) (Jegou, Douze, and Schmid 2011) between input vectors and quantization values as:

$$\arg\min_{\mathcal{C}} \sum_{i=1}^{l} \|\mathbf{x}_i - \mathrm{q}(\mathbf{x}_i)\|^2 / l \tag{4}$$

As introduced before, at the projection stage, the neighborhood structure between data points have been damaged more or less. The projection vectors can't exactly reflect the distance between the original data points. The objective in (4) is to reduce the quantization errors of projection vectors, while ignoring the capability of maximally preserving the neighborhood structure in original space.

For any two data point v_i and v_j, VQ approximates their distance by using the codeword distance:

$$\mathrm{d}(\mathbf{v}_i, \mathbf{v}_j) \simeq \mathrm{d}(\mathrm{q}(\mathbf{x}_i), \mathrm{q}(\mathbf{x}_j)) = \mathrm{d}(\mathbf{c}_{\mathrm{i}(\mathbf{x}_i)}, \mathbf{c}_{\mathrm{i}(\mathbf{x}_j)}) \tag{5}$$

where $\mathrm{i}(\mathbf{x})$ denotes the index of the Voronoi cell that contains $\mathbf{x}$. To preserve the affinity property in original space, we want to minimize the bias of the distance approximation to maintain the affinity of the original data points. Hence, we define the distance bias between all sample point pairs:

$$\sum_{i=1}^{l} \sum_{j=1}^{l} (\hat{\mathrm{d}}(\mathbf{v}_i, \mathbf{v}_j) - \mathrm{d}(\mathbf{c}_{\mathrm{i}(\mathbf{x}_i)}, \mathbf{c}_{\mathrm{i}(\mathbf{x}_j)}))^2. \tag{6}$$

Here, $\hat{\mathrm{d}}(.)$ is the scaled distance between two data points where $\hat{\mathrm{d}}(\mathbf{v}_i, \mathbf{v}_j) = \|\mathbf{v}_i - \mathbf{v}_j\|/\sigma$, $\mathrm{d}(.)$ denotes the Euclidean distance where $\mathrm{d}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$. For simplicity, let $w_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|/\sigma$. By incorporating the distance bias in Equation (6) and quantization errors in Equation (4), the proposed affinity preserving vector quantization thereby formulates the objective function

$$\arg\min_{\mathcal{C}} \sum_{i} \|\mathbf{x}_i - \mathrm{q}(\mathbf{x}_i)\|^2 + \lambda \sum_{i,j} (w_{ij} - \|\mathbf{c}_{\mathrm{i}(\mathbf{x}_i)} - \mathbf{c}_{\mathrm{i}(\mathbf{x}_j)}\|^2)^2. \tag{7}$$

Optimization

The optimization problem in Equation (7) is computational difficult (NP-hard). To quickly resolve a local optimum, we

Algorithm 1 The algorithm to optimize Equation (7).

Input: the samples $\mathbf{V} = \{\mathbf{v}_i\}_{i=1}^{l}$, the projection vectors $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{l}$, the bit number b.
Output: the codebook $\mathcal{C} = \{\mathbf{c}_i\}_{i=0}^{N-1}$, $N = 2^b - 1$.
Initialize the codeword $\mathbf{c}_i$ via kmeans clustering.
while not convergence **do**
Step 1: fix codebook $\mathcal{C}$, update $\mathrm{q}(\mathbf{x}_i)$ and R_i.
Step 2: fix $\mathrm{q}(\mathbf{x}_i)$ and R_i, separately update each $\mathbf{c}_t$ in (10) by using quasi-Newton method.
end while

adopt iterative optimization technique. In a similar way as k-means (MacQueen 1967) clustering algorithm, the proposed algorithm involves an assignment step and an update step in each iteration. At the assignment step, we fix codebook $\mathcal{C}$ and update quantization values $\mathrm{q}(\mathbf{x}_i)$. At the update step, we fix quantization values $\mathrm{q}(\mathbf{x}_i)$ and update codebook $\mathcal{C}$. The details are given below.

Assignment step: By fixing $\mathcal{C}$, we optimize $\mathrm{q}(\mathbf{x}_i)$. We quantize each $\mathbf{x}_i$ into its nearest centroid

$$\mathrm{q}(\mathbf{x}_i) = \{\mathbf{c}_j : \|\mathbf{c}_j - \mathbf{x}_i\| < \|\mathbf{c}_{j'} - \mathbf{x}_i\|, j^{'} \neq j\} \quad (8)$$

After that, we update Voronoi cell R_i as

$$R_i = \{\mathbf{x}_i : \mathrm{q}(\mathbf{x}_i) = \mathbf{c}_i\}, \quad i = 1, 2, ..., N. \quad (9)$$

Update step: By fixing $\mathrm{q}(\mathbf{x}_i)$, we optimize $\mathcal{C}$. Different from K-means, we have to incorporate the pairwise affinity in (6) into the objective function in (7).

We sequentially minimize the objective by updating each codeword $\mathbf{c}_t$ with others $\mathbf{c}_i$ $(i \neq t)$ fixed. To update $\mathbf{c}_t$, we optimize the following function with $\mathbf{y} = \mathbf{c}_t$

$$\arg\min_{\mathbf{y}} \sum_{\mathbf{x}_i \in R_t} \|\mathbf{y} - \mathbf{x}_i\|^2 + 2\lambda \sum_{\mathbf{x}_i \in R_t} \sum_{\mathbf{x}_j \notin R_t} (w_{ij} - \|\mathbf{y} - \mathbf{c}_{\mathrm{i}(\mathbf{x}_j)}\|^2)^2. \quad (10)$$

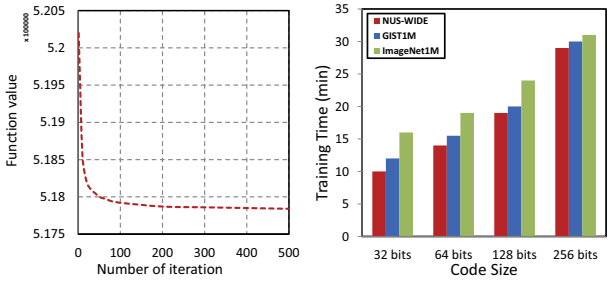
As this function is an unary polynomial about $\mathbf{y}$, it can be solved by quasi-Newton method (Shanno 1970).

We use the K-means clustering algorithm (MacQueen 1967) to initialize the codebooks. Our algorithm converges in 50-100 iterations and we empirically set the iteration number $t = 200$ to ensure good convergence (see figure 1). For each iteration, it takes $O(ldN)$ to assign $\mathrm{q}(\mathbf{x}_i)$ and $O(ld + dN^2)$ to update $\mathcal{C}$. The overall time complexity is $O(tdN(l + N))$. Algorithm 1 shows the pseudo-code.

Generalization to Product Space.

To produce b bits codes, a VQ quantizer may involve 2^b codewords. For a large code (say 128 bits or more), the codebook size would be unaffordable. Below we discuss the generalization of APQ in the product space (Jegou, Douze, and Schmid 2011). Specifically, we split the input vector $\mathbf{x}$ into m disjoint sub-vectors $\mathbf{x} = [\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^m]$. Each sub-codebook $\mathcal{C}_i$ is independently trained in the i-th sub-space. Given an input vector $\mathbf{x}$, the quantizer can be mapped to the product space as:

$$\mathrm{q}(\mathbf{x}) = [\mathrm{q}^1(\mathbf{x^1}), \mathrm{q}^2(\mathbf{x^2}), ..., \mathrm{q}^{\mathrm{m}}(\mathbf{x^m})] \quad (11)$$



(a) Convergence curve (b) Training time

Figure 2: (a) The convergence curve at different iteration numbers. (b) The training time over different datasets.

where $q^i(.)$ denotes the quantizer in the i-th subspace. The distance between two data points $\mathbf{x}$ and $\mathbf{y}$ is approximated by the distances between the codewords in sub-codebooks

$$d(x, y) \approx \sum_{i=1}^{m} (\mathrm{q}^i(\mathbf{x}^i) - \mathrm{q}^i(\mathbf{y}^i))^2 \quad (12)$$

In this way, the entire space $\mathbb{R}^d$ is therefore decomposed as the Cartesian product of sub-codebooks

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times ... \times \mathcal{C}_m. \quad (13)$$

To produce b bits codes, we only need to generate m sub-codebooks with $2^{b/m}$ codewords, with in total $m2^{b/m}$ codewords. The codebook size becomes more affordable. In addition, if we set $m = d$, one codebook is assigned to each projection dimension, in which, our method degenerates to a scalar quantization method.

The APQ generalization to product space can be summarized as follows. We firstly decompose the space $\mathbb{R}^d$ into m sub-spaces $\mathbb{R}^{d/m}$. At the offline training stage, we apply the APQ algorithm to learn a sub-codebook $\mathcal{C}_i$ for the i-th sub-space($1 \leq i \leq m$), each sub-codebook containing $2^{b/m}$ codewords. At the online stage, given a vector $\mathbf{x} \in \mathbb{R}^d$, we first divide $\mathbf{x}$ into m sub-vectors. For sub-vector $\mathbf{x}^i$, we find its nearest codeword from sub-codebook $\mathcal{C}_i$ and use the index of the nearest codeword to encode $\mathbf{x}^i$. These m indices are concatenated into a binary code to encode vector $\mathbf{x}$.

Experiment

Extensive experiments are carried out over three widely used large scale benchmarks NUS-WIDE (Chua et al. 2009), GIST1M (Jegou, Douze, and Schmid 2011) and ImageNet1M (Deng et al. 2009). NUS-WIDE dataset contains 269,648 images. Each image is represented by a 634-D feature. GIST1M contains 1 million 960-D GIST (Aude and Torralba 2001) features. ImageNet1M contains 1 million images which is a subset of the ImageNet database (Deng et al. 2009). We use the 2048-D fisher vector (Florent, Sanchez, and Mensink 2010) to present each image to evaluate the performance over high dimensionality space. The CIFAR-10 (Krizhevsky 2009) dataset contains 60,000 images of size

Table 1: Comparison with state-of-the-art single bit quantization and multiple bit quantization methods over dataset NUS-WIDE (Chua et al. 2009) at code size 64 bits and 128 bits.

#bits	64					128				
	ITQ	LSH	PCAH	SH	SIKH	ITQ	LSH	PCAH	SH	SIKH
SBQ	0.2415	0.1531	0.1032	0.1090	0.1370	0.3112	0.2201	0.1009	0.1432	0.1912
HQ	0.2681	0.1459	0.1452	0.1689	0.1441	0.3615	0.2453	0.2551	0.2115	0.2217
DBQ	0.2932	0.1510	0.2160	0.1505	0.0912	0.3871	0.2743	0.2390	0.1953	0.1876
MQ	0.3812	0.1901	0.2430	0.3297	0.1570	0.4574	0.3123	0.2709	0.3989	0.2970
HCQ	0.3327	0.1756	0.2235	0.2895	0.1442	0.4216	0.2917	0.2655	0.3618	0.2557
NPQ	0.4012	0.1988	0.3635	0.3219	0.1599	0.4613	0.3356	0.2755	0.3841	0.2513
AQ	0.3476	0.1951	0.3260	0.2814	0.1625	0.5063	0.3082	0.4210	0.4484	0.2630
APQ*(PQ)	0.4259	0.2657	0.4019	0.3945	0.1600	0.5189	0.4013	0.4516	0.4895	0.2754
APQ	0.4825	0.3016	0.5013	0.4519	0.1856	0.6013	0.4589	0.5655	0.5574	0.3078

Table 2: Comparison with state-of-the-art single bit quantization and multiple bit quantization methods over dataset GIST1M (Jegou, Douze, and Schmid 2011) at code size 64 bits and 128 bits.

#bits	64					128				
	ITQ	LSH	PCAH	SH	SIKH	ITQ	LSH	PCAH	SH	SIKH
SBQ	0.2815	0.2443	0.1103	0.1636	0.1874	0.3244	0.2856	0.1270	0.2101	0.2385
HQ	0.2157	0.2418	0.1885	0.1456	0.2012	0.3852	0.2857	0.2013	0.2435	0.2213
DBQ	0.3223	0.2354	0.2409	0.1722	0.1834	0.4032	0.2907	0.2405	0.2341	0.2430
MQ	0.3435	0.2670	0.2230	0.2690	0.2261	0.4865	0.2900	0.2476	0.3517	0.2654
HCQ	0.3215	0.2661	0.2013	0.2597	0.2150	0.4326	0.2819	0.2367	0.3471	0.2701
NPQ	0.3316	0.2716	0.2256	0.2798	0.2351	0.4731	0.3026	0.2517	0.3468	0.2851
AQ	0.3964	0.2447	0.3498	0.3292	0.2319	0.5133	0.3265	0.3701	0.3800	0.2503
APQ*(PQ)	0.4018	0.3256	0.3915	0.3156	0.2516	0.5826	0.3519	0.4215	0.4016	0.3174
APQ	0.4645	0.3754	0.4812	0.3757	0.2899	0.6571	0.3887	0.5078	0.4323	0.3532

32x32 and has been categorized into 10 classes. The LabelMe22K dataset (Torralba, Fergus, and Weiss 2008) contains 22,019 images. As setup in (Kong and Li 2012), (Kong, Li, and Guo 2012) and (Moran, Lavrenko, and Osborne 2013a), we represent each image with a 512 dimensional gray-scale GIST descriptor (Aude and Torralba 2001).

We follow the evaluation protocols in recent papers (Kong and Li 2012; Kong, Li, and Guo 2012; Xiong et al. 2014; Wang et al. 2015) and adopt the Euclidean distance based neighbors in the original space as the ground truth. For each dataset, we randomly select 1000 data points as queries and leave the rest as the database. The top 50/100 nearest data points in terms of Euclidean distance of a query are selected as the ground truth. We use mean Average Precision (mAP) to evaluate the search accuracy at different code sizes. We report the averaged results of 5 runs.

Baseline

We study the impact of quantization in five representative hashing methods, in which the projection stage is maintained, and different quantization methods are injected. The projections include spectral hashing (SH) (Weiss, Torralba, and Fergus 2008), local sensitive hashing (LSH) (Andoni and Indyk 2006), shift invariant kernels hashing (SIKH) (Raginsky and Lazebnik 2009), principal component analysis hashing (PCAH) (Wang et al. 2006) and iterative quantization (ITQ) (Gong and Lazebnik 2011).

Based on different projections, we compare the proposed VQ methods (including baseline VQ and APQ) with seven state-of-the-art quantization methods (including SBQ and a couple of MBQ methods):

- SBQ: single bit quantization.
- HQ: hierarchical quantization (Liu et al. 2011).
- DBQ: double bit quantization (Kong and Li 2012).
- MQ: Manhattan quantization (Kong, Li, and Guo 2012).
- NPQ: neighborhood preserving quantization (Moran, Lavrenko, and Osborne 2013a).
- HCQ: Hamming compatible quantization (Wang et al. 2015).
- AQ: adaptive quantization (Xiong et al. 2014).

Setting and Configuration

We discuss the empirical settings and parameter impact in this subsection. Figure 2a illustrates the convergence process of training APQ codebooks. In practice, the algorithm can quickly converge in 50-100 iterations and we empirically set the maximum iteration number $t = 200$ to ensure good convergency. Figure 2b shows the training time cost over dataset NUS-WIDE (Chua et al. 2009), GIST1M (Jegou, Douze, and Schmid 2011) and ImageNet1M (Deng et al. 2009). It takes about 16.2 minutes, 19.3 minutes, 24.5 minutes and 31.7 minutes to train the codebook for learning codes at 32, 64, 128, and 256 bits over dataset ImageNet1M.

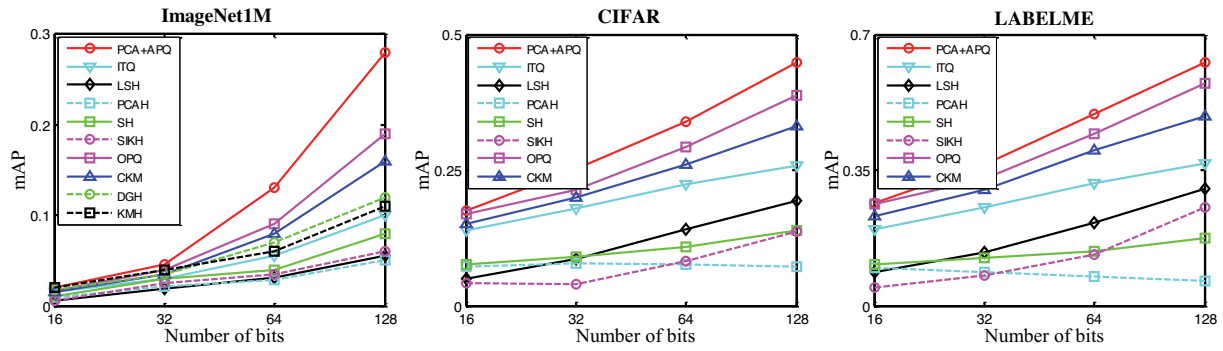


Figure 3: The results of mAP on ImageNet1M, CIFAR and LABELME at code size of 16 bits, 32 bits, 64 bits and 128 bits.

We apply product quantization, and then train $2^8 = 256$ codewords for each sub-codebook. The running time cost is tested on a workstation with an Intel(R) Core(TM) i5 3470 CPU and 32G memory.

Results on NUS-WIDE and GIST1M

Table 1 and table 2 list the performance comparison by combining different quantization methods and typical projection methods used in representative hashing methods over dataset NUS-WIDE (Chua et al. 2009) and GIST1M (Jegou, Douze, and Schmid 2011), respectively.

We present two sets of VQ quantization results. The first set is to evaluate the impact of baseline vector quantization on hashing performance, which just minimizes the mean square error (MSE) in Equation 4. For simplicity, we name this baseline set as APQ*. Note that APQ* applies product quantization to reduce the codebook size. The second one sets the complete version of the proposed APQ including preserving the affinity of original data points. The results show that APQ* has already achieve promising results and outperforms advanced MBQ methods in most cases. APQ* only works a bit worse than AQ (Xiong et al. 2014) on SIKH projection at code size 64 bits and MQ (Kong, Li, and Guo 2012) on SIKH projection at code size 128 bits over dataset NUS-WIDE (Chua et al. 2009). These promising results have demonstrated the superiority of vector quantization in improving hashing performance. Beyond the baseline vector quantization, the APQ led to the best hashing performance and consistently outperformed others. The positive results can be partially attributed to the use of affinity preservation in original data points.

APQ consistently outperforms the state-of-the-art multiple bit quantization methods. Let's compare the results over dataset NUS-WIDE (Chua et al. 2009). AQ (Xiong et al. 2014) and NPQ (Moran, Lavrenko, and Osborne 2013a) are the most competitive amongst the baseline methods. However, beyond the AQ performance, APQ further achieves significant mAP gains of +12.2% and +10.9% on average at code size 64 bits and 128 bits, respectively. Compared to NPQ, APQ achieves even more significant mAP gains of +9.56% and +15.66% on average at 64 bits and 128 bits, respectively. Referring to Table 2, over dataset GIST1M (Jegou, Douze, and Schmid 2011), APQ have achieved great performance improvements as well.

Results on ImageNet1M, CIFAR and LABELME

We combine the proposed APQ with PCA projection, and then compare it with state-of-the-art binary coding methods including ITQ (Gong and Lazebnik 2011), LSH (Andoni and Indyk 2006), PCAH (Wang et al. 2006), SH (Weiss, Torralba, and Fergus 2008), SIKH (Raginsky and Lazebnik 2009), OPQ (Ge et al. 2013), CKM (Norouzi and Fleet 2013), etc. Note that PCA is the most common projection, which is adopted by many hashing method such as SH, PCAH, ITQ, etc. Rather than focusing on advanced projection, we simply adopt PCA projection and study the impact of proper quantization. In addition, the performance comparison has been extended to the latest DGH (Liu et al. 2014) and KMH (He, Wen, and Sun 2013) methods over ImageNet1M.

Figure 3 shows the results over three datasets ImageNet1M, CIFAR and LABELME. PCA+APQ consistently outperforms others. For example, compared to the competitive OPQ, PCA+APQ achieves mAP gains of +1.0%, +1.5%, +4.0% and +8.2% at code size 16 bits, 32 bits, 64 bits and 128 bits over dataset ImageNet1M, respectively. Considerable improvements are yielded over CIFAR and LABELME as well. Figure 4 provides comparison examples. Previous hashing methods such SH and LSH focus more on projection than quantization, whereas, directly applying a quantization method such OPQ and CKM would yield suboptimal binary codes. The reported promising performance is attributed to the combination of PCA projection and APQ quantization.

Conclusion

Incorporating vector quantization into hashing can address the issues of non-adaptive projection space partition and limited distance range of scalar quantization methods. We have proposed a novel affinity preserving vector quantization approach (APQ) to strengthen the maintenance of neighborhood structure in original space over the course of hashing. Extensive experiments have shown that the proposed APQ approach has significantly improved the quantization stage of hashing techniques.

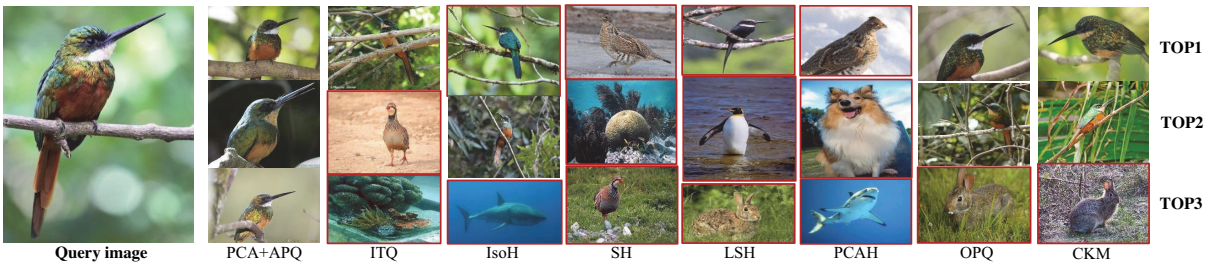


Figure 4: Image examples of the top 3 retrieved results of eight state-of-the-arts binary coding methods on ImageNet1M using 32 bits. Red rectangle denotes false positive. Best viewed in color.

Acknowledgments

This work was supported by the National Basic Research Program of China (973 Program): 2015CB351806, the National Hightech R&D Program of China (863 Program): 2015AA016302, and Chinese Natural Science Foundation: 61271311, 61390515, 61421062. Ling-Yu Duan is the corresponding author.

References

Allen, G., and Gray, R. M. 2012. Vector quantization and signal compression. *Springer Science and Business Media*.

Andoni, A., and Indyk, P. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *In IEEE FOCS*.

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

Chua, T.-S.; Tang, J.; Hong, R.; Li, H.; Luo, Z.; and Zheng, Y.-T. 2009. Nus-wide: A real-world web image database from national university of singapore. *ACM International Conference on Image and Video Retrieval*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Li, F.-F. 2009. Imagenet: A large-scale hierarchical image database. *In Proceedings of CVPR*.

Florent, P.; Sanchez, J.; and Mensink, T. 2010. Improving the fisher kernel for large-scale image classification. *ECCV*.

Ge, T.; He, K.; Ke, Q.; and Sun, J. 2013. Optimized product quantization for approximate nearest neighbor search. *CVPR*.

Gong, Y., and Lazebnik, S. 2011. Iterative quantization: A procrustean approach to learning binary codes. *CVPR*.

He, K.; Wen, F.; and Sun, J. 2013. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. *CVPR*.

Irie, G.; Li, Z.; Wu, X.-M.; and Chang, S.-F. 2014. Locally linear hashing for extracting non-linear manifolds. *In CVPR*.

Jegou, H.; Douze, M.; and Schmid, C. 2011. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Analysis and Machine Intelligence*.

Kong, W., and Li, W.-J. 2012. Double-bit quantization for hashing. *In Proceedings of the Twenty-Sixth AAAI*.

Kong, W.; Li, W.-J.; and Guo, M. 2012. Manhattan hashing for large-scale image retrieval. *In Proceedings of the 35th international ACM SIGIR*.

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

Lee, Y.; Heo, J.-P.; and Yoon, S.-E. 2012. Quadra-embedding: Binary code embedding with low quantization error. *In Proceedings of ACCV*.

Liu, W.; Wang, J.; Kumar, S.; and Chang, S.-F. 2011. Hashing with graphs. *ICML*.

Liu, W.; Mu, C.; Kumar, S.; and Chang, S.-F. 2014. Discrete graph hashing. *In NIPS*.

MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, 281–297. Berkeley, Calif.: University of California Press.

Moran, S.; Lavrenko, V.; and Osborne, M. 2013a. Neighbourhood preserving quantisation for lsh. *In Proceedings of the 36th international ACM SIGIR*.

Moran, S.; Lavrenko, V.; and Osborne, M. 2013b. Variable bit quantisation for lsh. *In Proceedings of ACL*.

Norouzi, M., and Fleet, D. J. 2013. Cartesian k-means. *IEEE Computer Vision and Pattern Recognition (CVPR)*.

Raginsky, M., and Lazebnik, S. 2009. Locality-sensitive binary codes from shift-invariant kernels. *In NIPS*.

Shanno, D. F. 1970. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation, 24(111):647656*.

Torralba, A.; Fergus, R.; and Weiss, Y. 2008. Small codes and large image databases for recognition. *In CVPR*.

Wang, X.-J.; Zhang, L.; Jing, F.; and Ma, W.-Y. 2006. Annosearch: Image auto-annotation by search. *CVPR*.

Wang, Z.; Duan, L.-Y.; Lin, J.; Wang, X.; Huang, T.; and Gao, W. 2015. Hamming compatible quantization for hashing. *IJCAI*.

Weiss, Y.; Torralba, A.; and Fergus, R. 2008. Spectral hashing. *In NIPS*.

Xiong, C.; Chen, W.; Chen, G.; Johnson, D.; and Corso, J. J. 2014. Adaptive quantization for hashing: An information-based approach to learning binary codes. *In ICDM*.