박사 학위논문
Ph. D. Dissertation

대규모 이미지 검색을 위한 고차원 데이터 간략화

Compact Representation of High-Dimensional Data for Large-Scale Image Search

허 재 필 (許 載 弼 Heo, Jae-Pil)
전산학과
Department of Computer Science

KAIST

2015

대규모 이미지 검색을 위한 고차원 데이터 간략화

Compact Representation of High-Dimensional Data for Large-Scale Image Search

Compact Representation of High-Dimensional Data for Large-Scale Image Search

Advisor : Professor Yoon, Sung-Eui

by

Heo, Jae-Pil
Department of Computer Science
KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science . The study was conducted in accordance with Code of Research Ethics[1].

2014. 11. 24.

Approved by

Professor Yoon, Sung-Eui

[Advisor]

---

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

대규모 이미지 검색을 위한 고차원 데이터 간략화

허 재 필

위 논문은 한국과학기술원 박사학위논문으로
학위논문심사위원회에서 심사 통과하였음.

2014년 11월 24일

심사위원장 윤 성 의 (인)

심사위원 강 유 (인)

심사위원 김 명 호 (인)

심사위원 양 현 승 (인)

심사위원 Zhe Lin (인)

DCS
20105431

허 재 필. Heo, Jae-Pil. Compact Representation of High-Dimensional Data for Large-Scale Image Search. 대규모 이미지 검색을 위한 고차원 데이터 간략화. Department of Computer Science . 2015. 56p. Advisor Prof. Yoon, Sung-Eui. Text in English.

ABSTRACT

Approximate Nearest Neighbor (ANN) search has been an active research problem across many fields in computer science including computational geometry, data mining, information retrieval, and computer vision. The problem is especially important for high-dimensional and large-scale cases due to the efficiency requirement by many practical applications. Traditional hierarchical approaches including the kd-trees have been used for low-dimensional data, however, these techniques are not scalable to high-dimensional data. Hence compact code representations of high-dimensional data have been actively studied recently, since they can provide efficient similarity search and are suitable for handling large scale databases.

In this dissertation, two compact representations of high-dimensional data and search scheme based on the compact codes are proposed: 1) Spherical Hashing and 2) Distance Encoded Product Quantization. In the Spherical Hashing, a novel hypersphere-based hashing functions are proposed to map more spatially coherent data into a binary code compared to hyperplane-based methods. We also propose a new binary code distance function tailored for our hypersphere-based binary code encoding scheme, and an efficient iterative optimization process to achieve both balanced partitioning for each hashing function and independence between hashing functions. Furthermore, we generalize the Spherical Hashing to support various similarity measures define by kernel functions. We also propose a novel compact code encoding scheme that distributes the available bit budget to encode both the cluster index and the quantized distance between point and its cluster center in the Distance Encoded Product Quantization (DPQ). We also propose two different distance metrics tailored to the Distance Encoded Product Quantization. All the proposed schemes are extensively evaluated against the state-of-the-art techniques with various large-scale benchmarks consisting of high-dimensional image descriptors.

This dissertation includes two ongoing works: 1) shortlist computation in the inverted file and 2) tag-driven feature learning. In the first ongoing work, a distance estimator based on the orthogonality property in a high-dimensional space is proposed. We also propose a distance aware indexing method and a shortlist construction scheme using the distance estimator. In the second work we learn an image feature from a large-scale tagged image database. Specifically, we define pseudo classes from the tag information and train a neural network for those pseudo classes.

Contents

Abstract i
Contents ii
List of Tables iv
List of Figures v

Chapter 1. Introduction **1**

Chapter 2. Related Work **4**
  2.1 Image Descriptors 4
  2.2 Hierarchical Methods 4
  2.3 Binary Code Embedding Methods 4
    2.3.1 Projection based methods 4
    2.3.2 Quantization based methods. 5
  2.4 Distance based Indexing Methods 6

Chapter 3. Spherical Hashing **7**
  3.1 Overview 7
  3.2 Spherical Hashing 7
    3.2.1 Binary Code Embedding Function 7
    3.2.2 Distance between Binary Codes 10
    3.2.3 Independence between Hashing Functions 11
    3.2.4 Iterative Optimization 11
    3.2.5 Max-Margin based Distance Thresholding 16
  3.3 Generalized Spherical Hashing 17
    3.3.1 Kernelized Binary Code Embedding Function 17
    3.3.2 Kernelized Iterative Optimization 18
  3.4 Evaluation 19
    3.4.1 Datasets 19
    3.4.2 Evaluation on Euclidean Space 22
    3.4.3 Evaluation on Kernel Space 25
    3.4.4 Evaluation on Image Retrieval 26
    3.4.5 Discussion 26
  3.5 Conclusion and Future Work 27

Chapter 4. Distance Encoded Product Quantization 30
4.1 Overview 30
4.2 Background and Motivations 31
    4.2.1 Motivations 32
4.3 Our Approach 33
    4.3.1 Our Binary Code Encoding Scheme 33
    4.3.2 Distance Metrics 34
4.4 Evaluation 37
    4.4.1 Protocol 38
    4.4.2 Results 39
4.5 Conclusion 40

Chapter 5. Accurate Shortlist Computation in Inverted File 42
5.1 Overview 42
5.2 Background 42
5.3 Our Approach 43
    5.3.1 Motivation 43
    5.3.2 Distance Estimator for High-Dimensional Space 43
    5.3.3 Shortlist Construction 44
5.4 Evaluation 46

Chapter 6. Large-Scale Tag-Driven Feature Learning 47
6.1 Overview 47
6.2 Tag-Driven Pseudo Classes 47
6.3 Evaluation 48
6.4 Regression Network 51

Chapter 7. Conclusion 52

References 53

Summary (in Korean) 57

List of Tables

3.1 Experimental results of hyperplane based methods combined with SHD. 10

3.2 Comparisons between SHD and SHD-SUB described in Sec. 3.2.2 11

3.3 These two tables show the effect of our max-margin based distance thresholding (Sec. 3.2.5). **SHD** and **HD** indicate the binary code distance metric type, and **M** indicates the max-margin based distance thresholding scheme. The max-margin based distance thresholding improves mAPs 3.8% on average over the median based distance thresholding across various settings of experiments. 16

3.4 These two tables show the effect of **SHD** (Sec. 3.2.2) with generalized spherical hashing (Sec. 3.3). **SHD** and **HD** indicate binary code distance metric types. **SHD** improves mAPs of generalized spherical hashing 3.5% on average over **HD**. 27

4.1 This table shows mAPs with different bit budgets. mAPs are measured with the asymmetric distances in **GIST-1M-960D** dataset. 'OPQ baseline' is provided here to see how much our scheme improves the performance by using quantized distances. 8 bit l_c for OPQ is used in other tests. 38

6.1 This table shows experimental results of image tagging task. 25 nearest neighbors are considered in the voting scheme that determines tags of given query. 48

List of Figures

1.1 This figure shows a conceptual procedure of the image search. In general, the extracted image features are high-dimensional vectors. The high dimensionality makes the search significantly complicated (*i.e. curse of dimensionality*). This dissertation addresses indexing, encoding, and searching (red rectangle) for those high-dimensional data by proposing novel compact data representations. 2

3.1 These figures show a main difference between our hypersphere-based binary code embedding method and hyperplane-based one. The left and right figures show partitioning examples of hypersphere-based and hyperplane-based methods respectively, for 3 bit binary codes in 2-D space. Each function h_i determines the value of i-th bit of binary codes. The hypersphere-based binary code embedding scheme gives more tightly closed regions compared to hyperplane-based one. 8

3.2 The left figure shows how the avg. of the max. distances among points having the same binary code changes with different code lengths based on hyperspheres or hyperplanes. We randomly sample 1000 different binary codes to compute avg. of the max. distances. The right figure shows how having more common +1 bits in our method effectively forms tighter closed regions. For the right curve we randomly sample one million pairs of binary codes. For each pair of binary codes (b_i, b_j) we compute the max. distance between pairs of points, (x_i, x_j), where $H(x_i) = b_i$ and $H(x_j) = b_j$. We report the avg. of the max. distances as a function of the number of common +1 bits, i.e. $|b_i \wedge b_j|$. Both figures are obtained with GIST-1M-960D dataset (Sec. 3.4.1). 9

3.3 These figures give a high-level intuition of the spherical hamming distance. If both two points x_i and x_j are located in one of colored regions of (a), (b), or (c), then their binary codes b_i and b_j have at least 1, 2, or 3 common +1 bits, respectively. As the number of common +1 bits of b_i and b_j increases, the area (or volume) of regions where two points x_i and x_j can be located in is getting smaller. As a result, the expected distance between x_i and x_j is getting smaller, as the number of common +1 bits of b_i and b_j increases. . . . 9

3.4 These two images show how a force between two pivots is computed. In the left image a repulsive force is computed since their overlap $o_{i,j}$ is larger than the desired amount. On the other hand, the attractive force is computed in the right image because their overlap is smaller than the desired amount. 12

3.5 This graph shows mean Average Precision (mAP) curves for k-nearest neighbor search with respect to various parameters. A pair of values in x-axis are used for two parameters of ϵ_m and ϵ_s, and y-axis represents their corresponding mAP values. Each legend consists of four experiment settings 'dataset / k / binary code length / distance metric type (HD: Hamming distance, SHD: spherical Hamming distance)'. 13

3.6 This graph shows a convergence rate of our iterative optimization. The left and right y-axises indicate how the average and std. dev. of $o_{i,j}$ approach our termination condition in the scale of our error tolerances, ϵ_m and ϵ_s respectively. In this case, we set ϵ_m as 0.1 and ϵ_s as 0.15 for terminating our optimization. This result is obtained with the **GIST-1M-384D** dataset at the 64-bit code length. 15

3.7 Convergence rates of our iterative optimization with three individual trials. The optimization processes are finished within 30 iterations when we se we set ϵ_m as 0.1 and ϵ_s as 0.15. This graph also shows that both objectives converge to 0 when we increase the number of iterations. This result is obtained with the **GIST-1M-384D** dataset at the 64-bit code length. 15

3.8 These figures visualize results of our optimization process with three hyperspheres and 500 points in 2D space. The leftmost and rightmost figures show an initial state and converged state of the optimization process, respectively, while two middle figures show intermediate states. 15

3.9 Comparison between our method and the state-of-the-art methods with the GIST-1M-384D dataset when $k = 100$ 19

3.10 Comparison between our method and the state-of-the-art methods with the GIST-1M-960D dataset when $k = 1,000$ 20

3.11 Comparison between our method and the-state-of-the-art methods with the **VLAD-1M-8192D** dataset when $k = 1,000$. 20

3.12 Comparison between our method and the-state-of-the-art methods with the **L2-normalized GIST-1M-960D** dataset when $k = 1,000$. 21

3.13 Comparison between our method and the state-of-the-art methods on GIST-1M-384D dataset when $k = 100$. Refer to Fig. 3.9 for the mAP curves. 21

3.14 Comparison between our method and the state-of-the-art methods on GIST-1M-960D dataset when $k = 1,000$. Refer to Fig. 3.10 for the mAP curves. 21

3.15 Recall curves of different methods when $k = 100$ for the **GIST-1M-384D** dataset. Each hash table is constructed by 64 bits code lengths. The recall (y-axis) represents search accuracy and the number of retrieved samples (x-axis) represents search time. Graphs are best viewed with colors. 23

3.16 Comparison between our method and the-state-of-the-art methods with the **GIST-75M-384D** dataset when $k = 10,000$. 23

3.17 This figure shows how each component of our method affects the accuracy. The mAP curves are obtained with GIST-1M-384D dataset when $k = 100$. 24

3.18 k-nearest neighbor search performances on three different datasets when $k = 1000$. . . . 28

3.19 Image retrieval performances on **ILSVRC** dataset. Exact-CS and Exact-HI are recognition rates obtained by the exact 1000-nearest neighbor classifier on corresponding kernel functions. 29

4.1 The left and right figures show toy examples of partitioned space and assigned binary codes for PQ and ours, respectively, when using 4 bit binary codes. Our method allocates first two bits for encoding a cluster index and another underlined two bits for quantized distances, while PQ allocates all the bits for encoding a cluster index. 31

4.2 The left figure shows the empirical quantization distortions as a function of the number of clusters in each subspace. M indicates the number of subspaces used. The right figure shows mAP curves of 1000-nearest neighbor search with varying numbers of clusters for each subspace. We use OPQ on the GIST-960D dataset for the experiments. 32

4.3 This figure visualizes errors of the symmetric distances. We sample two random points, x and y, in a randomly selected subspace. The x-axis indicates the distance between x^i and its corresponding cluster's center $q^i(x^i)$, and y-axis shows similar information for y. The vertical axis is the difference between the actual distance $d(x^i, y^i)$ and its estimated distance $d(q^i(x^i), q^i(y^i))$. The errors of estimated distances tend to be higher as the distance between data points and their corresponding clusters becomes larger. We use OPQ to define subspaces and clusters with the GIST-960D dataset. 33

4.4 This figure shows the distribution of differences from the ground-truth distances to estimated results from statistics based distance metrics used for our method and PQ. We draw 100 K pairs of samples randomly chosen from the GIST-1M-960D dataset. Our method shows the bias of 0.0091, which is reduced from 0.0096 of PQ. Similarly, the variance of our method is 0.0099, while PQ has 0.0133. 35

4.5 This figure shows two points x and y on the hyper-spheres centered at c_x and c_y respectively. r_x and r_y represent the radii of hyper-spheres. 36

4.6 This figure shows the recall curves with respect to the number of retrieved data. Our geometry based distance metric, Ours w/ GMAD, improves our statistics based metric, Ours w/ ECAD, and the PQ method, PQ w/ ECAD. The results are obtained with GIST-1M-960D dataset and the number of ground truth K=100. We use 64 bits in this experiment and PQ uses 8 bits for each subspace to define 256 centers. Our method uses 7 bits for the center and 1 bit for the distance quantization. 38

4.7 Comparisons on **GIST-1M-960D**. The left and right graphs show the mAP curves with symmetric and asymmetric distances, respectively. 39

4.8 Comparisons on **GIST-1M-384D**. The left and right graphs show the mAP curves with symmetric distances and asymmetric distances, respectively. 40

4.9 Comparisons on **BoW-1M-1024D**. The left and right graphs show results for original and L_2 normalized data with symmetric and asymmetric distance metrics. 41

5.1 These two figures illustrate examples of shortlists for given query y. The shortlists consist of points inside the red-dotted shape. The left figure shows a limitation of the shortlist construction method described in Sec. 5.2. The green points are close to the query y but they are missed in the shortlist since c_3 is farther than c_1 and c_2 from y. In the other hand, the shortlist in the right figure is more accurately constructed. Our goal is to develop a method to construct accurate shortlist as the right figure. 43

5.2 These two figures show an example of the procedure described in Sec. 5.3.3 (a) The data points in each list L_i is sorted according to the squared distance to the center $d(x, c_i)^2$, and range of the squared distances $[R_0, R_{max}]$ is uniformly divided with fixed interval length ΔR. (b) A lookup table W is defined to $W(i, j)$ be the number of data points in L_i whose squared distance to the center $d(x, c_i)^2$ is less than R_j. For instance, the number of data points within the green box in (a) is $W(2, 5) = 20$ in the lookup table. 45

5.3 These two figure show the experimental results with different K which is number of defined true neighbors. x-axis and y-axis represent the shortlist size (T) and Recall@T respectively. 46

6.1 This figure shows our framework to define pseudo classes. 47
6.2 These figures show examples of three pseudo classes that confirm the visual coherency within each pseudo classes 49
6.3 These figures show that three image search results of our method and the compared method. The leftmost picture of each sub-figure is the provided query image. 50

Chapter 1. Introduction

Thanks to rapid advances of digital camera and various image processing tools, we can easily create new pictures and images for various purposes. This in turn results in a huge amount of images available online. These huge image databases pose a significant challenge in terms of scalability to many computer vision applications, especially those applications that require efficient similarity search such as the image retrieval.

Image search system consists of two main parts, 1) feature extraction and 2) indexing, encoding and searching the extracted features, as illustrated in Fig. 1.1. Since the features used in the image search problem are high-dimensional vectors in general. This dissertation is mostly focusing on the data encoding and searching.

Approximate Nearest Neighbor (ANN) search has been an active research problem across many fields in computer science including computational geometry, data mining, information retrieval, and computer vision. The problem is especially important for high-dimensional and large-scale cases due to the efficiency requirement by many practical applications.

For similarity search, nearest neighbor search techniques have been widely studied and tree-based techniques [1, 2, 3, 4] have been used for low-dimensional data. Unfortunately, these techniques are not scalable to high-dimensional data. Hence recently compact data representations have been actively studied to provide efficient solutions for such high-dimensional data [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22].

Encoding high-dimensional data points into compact binary codes based on hashing techniques enables higher scalability thanks to both its compact data representation and efficient indexing mechanism. Similar high-dimensional data points are mapped to similar binary codes and thus by looking into only those similar binary codes (based on the Hamming distance), we can efficiently identify approximate nearest neighbors.

Existing hashing techniques can be broadly categorized as data-independent and data-dependent schemes. In data-independent techniques, hashing functions are chosen independently from the data points. Locality-Sensitive Hashing (LSH) [5] is one of the most widely known techniques in this category. This technique is extended to various hashing functions [6, 7, 10, 11, 12]. Recent research attentions have been shifted to developing data-dependent techniques to consider the distribution of data points and design better hashing functions. Notable examples include spectral hashing [9], semi-supervised hashing [16], iterative quantization [19], joint optimization [20], and random maximum margin hashing [21].

In all of these existing hashing techniques, hyperplanes are used to partition the data points into two sets and assign two different binary codes (e.g., -1 or $+1$) depending on which set each point is assigned to. Departing from this conventional approach, we propose a novel hypersphere-based scheme, *spherical hashing*, for computing binary codes in Chapter 3. Intuitively, hyperspheres provide much stronger power in defining a tighter closed region than hyperplanes. For example, at least $d + 1$ hyperplanes are needed to define a closed region for a d-dimensional space, while only a single hypersphere can form such a closed region even in an arbitrarily high dimensional space.

In the other hands, Product Quantization (PQ) [23] and its recent improvement, Optimized Product Quantization (OPQ) [24], have shown the-state-of-the-art performance. Its high accuracy and efficiency

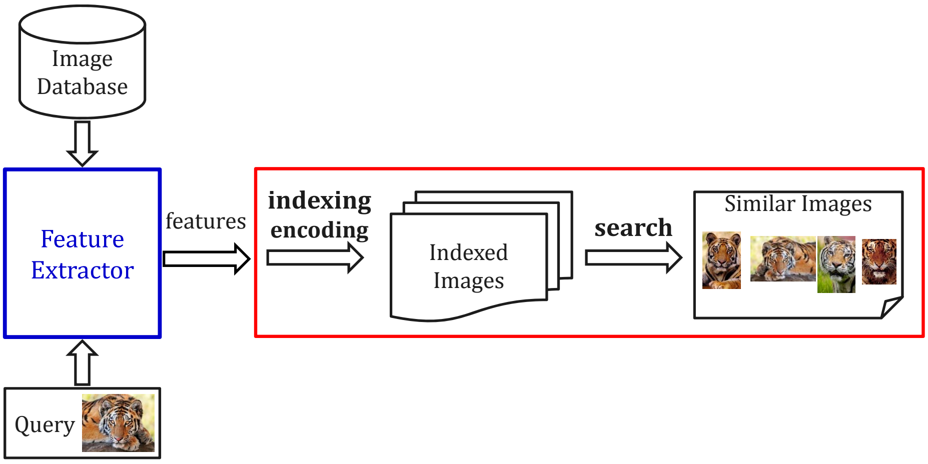


Figure 1.1: This figure shows a conceptual procedure of the image search. In general, the extracted image features are high-dimensional vectors. The high dimensionality makes the search significantly complicated (*i.e. curse of dimensionality*). This dissertation addresses indexing, encoding, and searching (red rectangle) for those high-dimensional data by proposing novel compact data representations.

is mainly because (1) quantization in subspaces offers a strong representational power and (2) distances between two data points can be computed via a look-up table. Its input feature space is divided into several disjoint subspaces, and each subspace is partitioned into clusters independently. A data point is then represented by an index of its corresponding cluster in each subspace. The distance between two data points is approximated by the sum of distances of their cluster centers computed in each subspace.

PQ and OPQ can effectively generate many clusters in each subspace and thereby reduce the quantization distortion. Nonetheless, we have found that their approach shows marginal accuracy improvement in practice, as we increase the number of clusters in each subspace. This is mainly because they encode only clusters containing data points, but are not designed to consider how far data points are located away from cluster centers. To address aforementioned problems, we propose a new binary code encoding scheme, Distance-encoded Product Quantization (DPQ), and two different distance metrics tailored to the scheme in Chapter 4.

The inverted file structure has been applied to the PQ framework to avoid expensive exhaustive distance estimations [23]. Specifically, in order to prevent linear scan on whole database coarse quantization is performed with given a query to compute candidate search results called shortlist which is a small fraction of the whole data. However, the conventional shortlist construction scheme has a limitation that some close points to the query can be missed in the shortlist. To address this, we propose a distance aware indexing method and a shortlist construction scheme using the distance estimator designed for the high-dimensional data in Chapter 5.

We also investigate about the image feature. The deep convolutional neural network trained on the ImageNet dataset [25] shows the state-of-the-art performance in the image classification task. The image features computed in the intermediate layers in the deep convolutional neural network can be used in many computer vision applications including visual search and tagging. Specifically, the vectors in the fully connected layers in the network is known as one of the best image features. However, the neural network trained on the ImageNet dataset has a few disadvantages. First, most of images in the ImageNet dataset contain a single object and are strictly assigned to a single object label. This can bring training bias to the pre-defined fixed number of object categories. And the trained network is weak at images containing drawings, graphics, or illustrations since most images in the training set are real photos. In Chapter 6, we propose a new scheme to train a neural network with large-scale image dataset that are weakly annotated with multiple text tags.

Chapter 2. Related Work

In this chapter we discuss prior work related to image descriptors and nearest neighbor search techniques.

2.1 Image Descriptors

To identify visually similar images for a query image, many image descriptors have been studied [26]. Examples of the most popular schemes include the Bag-of-visual-Words representation [27] which is a weighted histogram of visual words computed by local image descriptors such as SIFT [28], and GIST descriptor [29] which have been known to work well in practice. These image descriptors have high dimensionality (e.g. hundreds to thousands) and identifying similar images is typically reduced to finding nearest neighbor points in those high dimensional, image descriptor spaces [30]. Since these image descriptor spaces have high dimensions, finding nearest neighbor image descriptors has been known to be very hard because of the 'curse of dimensionality' [5].

2.2 Hierarchical Methods

Space partitioning based tree structures such as kd-trees [1, 2], R-trees [3], Vantage Point Trees (VPT) [31] have been used to find nearest neighbors. Excellent surveys for such tree-based indexing and nearest neighbor search methods are available [32, 33]. Using kd-trees is one of the most popular approaches, and thus there have been a lot of optimization efforts such as randomized kd-trees [34], relaxed orthogonality of partitioning axes [35], and minimizing probabilistic search cost [36]. It has been widely known, however, that kd-tree based search can run slower even than the linear scan for high dimensional data. Nistér and Stewénius [37] proposed another tree-based nearest neighbor search scheme based on hierarchical k-means trees. Muja and Lowe [4] have proposed an automatic parameter selection algorithm of some of techniques mentioned in above.

Although these techniques achieve reasonably high accuracy and efficiency, they have been demonstrated in small image databases consisting of about one million images. Also, these techniques do not consider compressions of image descriptors to handle large-scale image databases.

2.3 Binary Code Embedding Methods

Recently binary code embedding methods have been actively studied, since they provide a high compression rate by encoding high-dimensional data into compact binary codes, and fast distance (i.e., similarity) computation with simple bit-string operations or a pre-computed lookup table. We categorize binary code embedding methods into two categories: projection and clustering based methods.

2.3.1 Projection based methods

These techniques map high-dimensional data to the Hamming space by using projection functions. They can be categorized further into data-independent and data-dependent methods. In data-independent methods, the hash functions are defined independently from the data. One of the most popular hashing techniques in this category is Locality Sensitive Hashing (LSH) [5]. Its hash function is based on projection onto random vectors drawn from a specific distribution. Many variations and extensions of LSH have been proposed for L_p norms [7], learned metrics [11], min-hash [10], inner products [6], multi-probe [38]. Kulis et al. [30] generalized LSH to Kernelized LSH that supports arbitrary kernel functions defining similarity. Raginsky and Lazebnik [12] have proposed a binary code embedding scheme based on random Fourier features for shift-invariant kernels. According to Johnson-Lindenstrauss lemma [39, 40], LSH preserves the distances among the data points within a small relative error bound by using a sufficient number of projections. Specifically, at least $O(\ln n/\epsilon^2)$ projections are required for a relative error rate ϵ, where n is the number of data points. Hence, these data-independent methods can be inefficient especially for short binary codes computed with a small number of projections.

There have been a number of research efforts to develop data-dependent hashing methods that reflect data distributions to improve the performance. Weiss et al. [9] have proposed spectral hashing motivated by spectral graph partitioning. Liu et al. [41] applied the graph Laplacian technique by interpreting a nearest neighbor structure as an anchor graph. Strecha et al. [42] used Linear Discriminant Analysis (LDA) for binarization of image descriptors. Wang et al. [16] proposed a semi-supervised hashing method to improve image retrieval performance by exploiting label information of the training set. Gong and Lazebnik [19] introduced a procrustean approach that directly minimizes quantization error by rotating zero-centered PCA-projected data. He et al. [20] presented a hashing method that jointly optimizes both search accuracy and search time by incorporating a similarity preserving term into the Independent Component Analysis (ICA). Joly and Buisson [21] constructed hash functions by using large margin classifiers such as the support vector machine (SVM) with arbitrarily sampled data points that are randomly separated into two sets. In most cases, data-dependent methods outperform data-independent ones with short binary codes.

The efficiency of each hash function in data-dependent methods is, however, getting lower as they allocate longer binary codes. The main cause of this trend is the growing difficulty of defining independent and informative set of projections as the number of hash functions increases. To avoid the issue there have been a few approaches that use a single hash function to determine multiple bits and use less hash functions [41, 43, 44, 45].

2.3.2 Quantization based methods.

These techniques are closely related to clustering. In these methods, a binary code of a data point encodes the index of a cluster containing the data point. Product Quantization (PQ) [23] decomposes the original data space into lower-dimensional subspaces and quantizes each subspace separately using k-means clustering. It then computes a binary code as a concatenation of cluster indices, encoded in subspaces.

He et al. [46] have proposed k-means hashing, which optimizes cluster centers and their cluster indices in a way that the Hamming distance between encoded cluster indices reflects distances between cluster centers. Recently, Ge et al. have proposed Optimized PQ (OPQ) [24] that optimizes PQ by minimizing quantization distortions with respect to the space decomposition and code books. OPQ shows the state-of-the-art results over other quantization and projection based methods. Norouzi and Fleet have presented Cartesian k-means [47] that also reduces the quantization distortions of PQ in a similar manner to OPQ.

Hamming embedding [48] uses an orthogonal projection and thresholding projected values for computing binary codes only within a cluster. This approach provides higher accuracy within each cluster and works for image retrieval. On the other hand, this method is not designed for accurately measuring distances between points that are contained in different clusters.

2.4 Distance based Indexing Methods

The database community has been designing efficient techniques for indexing high dimensional points and supporting various proximity queries. Filho et al. [49] index points with distances from fixed *pivot* points. As a result, a region with the same index given a pivot becomes a ring shape. This method reduces the region further by using multiple pivot points. They then built various hierarchical structures (e.g., R-tree [3]) to support various proximity queries. The efficiency of this method highly depends on the locations of pivots. For choosing pivots, Jagadish et al. [50] used k-means clustering and Venkateswaran et al. [51] adopted other heuristics such as maximizing the variance of distances from pivots to data points.

This line of work uses a similar concept to ours in terms of using distances from pivots for indexing high-dimensional points. However, our approach is drastically different from these techniques, since ours aims to compute compact binary codes preserving the original metric spaces by using hashing, while theirs targets for designing hierarchical indexing structures supporting efficient proximity queries.

Chapter 3. Spherical Hashing

3.1 Overview

In most of prior existing hashing techniques, hyperplanes are used to partition the data points into two sets and assign two different binary codes (e.g., -1 or $+1$) depending on which set each point is assigned to. Departing from this conventional approach, we propose a novel hypersphere-based scheme, *spherical hashing*, for computing binary codes. Intuitively, hyperspheres provide much stronger power in defining a tighter closed region than hyperplanes. For example, at least $d+1$ hyperplanes are needed to define a closed region for a d-dimensional space, while only a single hypersphere can form such a closed region even in an arbitrarily high dimensional space.

Our paper has the following contributions:

1. We propose a novel spherical hashing scheme, analyze its ability in terms of similarity search, and compare it against the state-of-the-art hyperplane-based techniques (Sec. 3.2.1).
2. We develop a new binary code distance function tailored for the spherical hashing method (Sec. 3.2.2).
3. We formulate an optimization problem that achieves both balanced partitioning for each hashing function and the independence between any two hashing functions (Sec. 3.2.3). Also, an efficient, iterative process is proposed to construct spherical hashing functions (Sec. 3.2.4).
4. We generalize spherical hashing to support arbitrary kernel functions, and reformulate the optimization process into a kernelized one (Sec. 3.3).

In order to highlight benefits of our method, we have tested our method against different benchmarks that consist of one to 75 million image descriptors with varying dimensions. We have also compared our method with many state-of-the-art techniques and found that our method significantly outperforms all the tested techniques, confirming the superior ability of defining closed regions with tighter bounds compared to conventional hyperplane-based hashing functions (Sec. 3.4).

3.2 Spherical Hashing

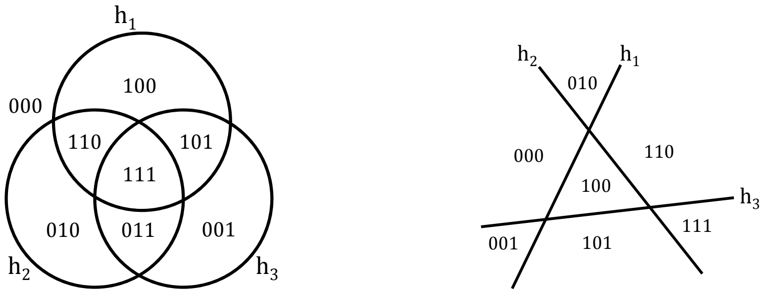
Let us first define notations. Given a set of N data points in a D-dimensional space, we use $X = \{x_1, ..., x_N\}$, $x_i \in \mathbb{R}^D$ to denote those data points. A binary code corresponding to each data point x_i is defined by $b_i = \{-1, +1\}^l$, where l is the length of the code[1].

3.2.1 Binary Code Embedding Function

Our binary code embedding function $H(x) = (h_1(x), ..., h_l(x))$ maps points in $\mathbb{R}^D$ into the binary cube $\{-1, +1\}^l$. We use a hypersphere to define a spherical hashing function. Each spherical hashing

---

[1] $(-1, +1)^*$ codes are conceptual expression. Codes are stored and processed as $(0, 1)^*$ codes in practice.



(a) (b)

Figure 3.1: These figures show a main difference between our hypersphere-based binary code embedding method and hyperplane-based one. The left and right figures show partitioning examples of hypersphere-based and hyperplane-based methods respectively, for 3 bit binary codes in 2-D space. Each function h_i determines the value of i-th bit of binary codes. The hypersphere-based binary code embedding scheme gives more tightly closed regions compared to hyperplane-based one.

function $h_i(x)$ is defined by a pivot $p_i \in \mathbb{R}^D$ and a distance threshold $t_i \in \mathbb{R}^+$ as the following:

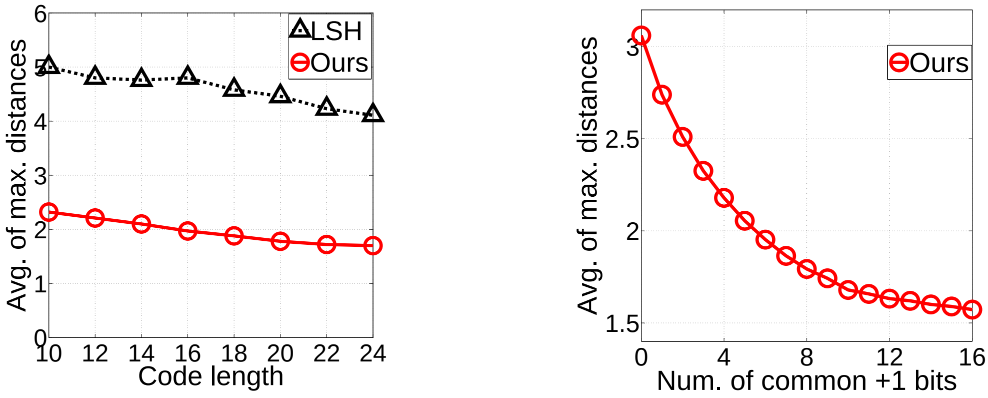
$$h_i(x) = \begin{cases} -1 & \text{when } d(p_i, x) > t_i \\ +1 & \text{when } d(p_i, x) \le t_i, \end{cases} \tag{3.1}$$

where $d(\cdot,\cdot)$ denotes the Euclidean distance between two points in $\mathbb{R}^D$; various distance metrics (e.g., L_p metrics) can be used instead of the Euclidean distance. The value of each spherical hashing function $h_i(x)$ indicates whether the point x is inside the hypersphere whose center is p_i and radius is t_i. Fig. 3.1(a) shows an example of a space partitioning and assigned binary codes with three hyperspheres in 2-D space.

The key difference between using hyperplanes and hyperspheres for computing binary codes is their abilities to define a closed region in $\mathbb{R}^D$ that can be indexed by a binary code. To define a closed region in a d-dimensional space, at least $d+1$ hyperplanes are needed, while only a single hypersphere is sufficient to form such a closed region in an arbitrarily high dimensional space. Furthermore, unlike using multiple hyperplanes a higher number of closed regions can be constructed by using multiple hyperspheres, while the distances between points located in each region are bounded. For example, the number of bounded regions by having l hyperspheres goes up to $\binom{l-1}{d} + \sum_{i=0}^{d} \binom{l}{i}$ [52]. In addition, we can approximate a hyperplane with a large hypersphere that has a large radius and a far-away center.

In nearest neighbor search the capability of forming closed regions with tighter distance bounds is very important in terms of effectively locating nearest neighbors from a query point. When we construct such tighter closed regions, a region indexed by the binary code of the query point can contain more promising candidates for the nearest neighbors.

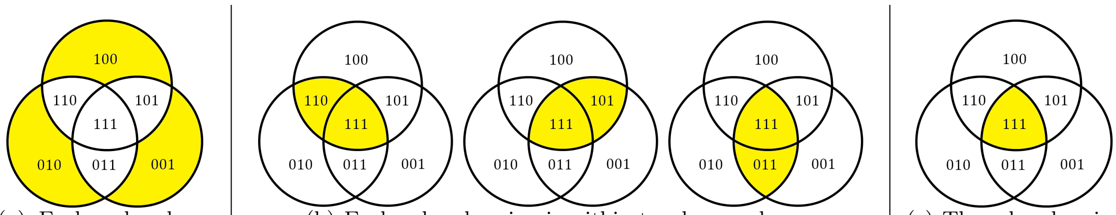
We also empirically measure how tightly hyperspheres and hyperplanes bound regions. For this purpose, we measure the maximum distance between any two points that have the same binary code and take the average of the maximum distances among different binary codes. As can be seen in Fig. 3.2(a), hyperspheres bound regions of binary codes more tightly compared to hyperplanes used in LSH [7]. Across all the tested code lengths, hyperspheres show about two times tighter bounds over the hyperplane-based approach.



(a)

(b)

Figure 3.2: The left figure shows how the avg. of the max. distances among points having the same binary code changes with different code lengths based on hyperspheres or hyperplanes. We randomly sample 1000 different binary codes to compute avg. of the max. distances. The right figure shows how having more common +1 bits in our method effectively forms tighter closed regions. For the right curve we randomly sample one million pairs of binary codes. For each pair of binary codes (b_i, b_j) we compute the max. distance between pairs of points, (x_i, x_j), where $H(x_i) = b_i$ and $H(x_j) = b_j$. We report the avg. of the max. distances as a function of the number of common +1 bits, i.e. $|b_i \wedge b_j|$. Both figures are obtained with GIST-1M-960D dataset (Sec. 3.4.1).



(a) Each colored region is within one hypersphere.

(b) Each colored region is within two hyperspheres.

(c) The colored region is within three hyperspheres.

Figure 3.3: These figures give a high-level intuition of the spherical hamming distance. If both two points x_i and x_j are located in one of colored regions of (a), (b), or (c), then their binary codes b_i and b_j have at least 1, 2, or 3 common +1 bits, respectively. As the number of common +1 bits of b_i and b_j increases, the area (or volume) of regions where two points x_i and x_j can be located in is getting smaller. As a result, the expected distance between x_i and x_j is getting smaller, as the number of common +1 bits of b_i and b_j increases.

1000-NN mAP with GIST-1M-960D

# bits	32	64	128	256
RMMH-SHD	0.0279	0.0603	0.0976	0.1466
RMMH-HD	0.0266	0.0576	0.0993	0.1483
ITQ-SHD	0.0385	0.0578	0.0860	0.1060
ITQ-HD	0.0380	0.0620	0.0875	0.1101

Table 3.1: Experimental results of hyperplane based methods combined with SHD.

3.2.2 Distance between Binary Codes

Most hyperplane-based binary code embedding methods use the Hamming distance between two binary codes, which measures the number of different bits, i.e. $|b_i \oplus b_j|$, where $\oplus$ is the XOR bit operation and $|\cdot|$ denotes the number of $+1$ bit in a given binary code. This distance metric measures the number of hyperplanes that two given points reside in the opposing side of them. The Hamming distance, however, does not well reflect the property related to defining closed regions with tighter bounds, which is the core benefit of using our spherical hashing functions.

To fully utilize desirable properties of our spherical hashing function, we propose the following distance metric, *spherical Hamming distance* (SHD) ($d_{SHD}(b_i, b_j)$), between two binary codes b_i and b_j computed by spherical hashing:

$$d_{SHD}(b_i, b_j) = \frac{|b_i \oplus b_j|}{|b_i \wedge b_j|},$$

where $|b_i \wedge b_j|$ denotes the number of common $+1$ bits between two binary codes which can be easily computed with the AND bit operations.

Having the common $+1$ bits in two binary codes gives us tighter bound information than having the common -1 bits in our spherical hashing functions. This is mainly because each common $+1$ bit indicates that two data points are inside its corresponding hypersphere, giving a stronger cue in terms of distance bounds of those two data points; see Fig. 3.3 for intuition. In order to see the relationship between the distance bound and the number of the common $+1$ bits, we measure the average distance bounds of data points as a function of the number of the common $+1$ bits. As can be seen in Fig. 3.2(b), the average distance bound decreases as the number of the common $+1$ bits in two binary codes increases. As a result, we put $|b_i \wedge b_j|$ in the denominator of our spherical Hamming distance.

In implementation we add a small value (e.g. 0.1) to the denominator to avoid the division-by-zero. Also, we can construct a pre-computed SHD table $T(|b_i \wedge b_j|, |b_i \oplus b_j|)$ whose size is $(l+1)^2$ and refer the table, when computing SHD to avoid expensive division operations.

The common $+1$ bits between two binary codes define a closed region with a distance bound as mentioned above. Within this closed region we can further differentiate the distance between two binary codes based on the Hamming distance $|b_i \oplus b_j|$, the numerator of our distance function. The numerator affects our distance function in the same manner to the Hamming distance, since the distance between two binary codes increases as we have more different bits between two binary codes.

In hyperplane based methods, the common $+1$ bits do not give strong cue on estimating the real distance. As a resulot, SHD does not provide any benefit for hyperplane based methods as reported in Table. 3.1.

An alternative definition of SHD can be constructed based on the subtraction as following:

$$d_{SHD-SUB}(b_i, b_j) = |b_i \oplus b_j| - |b_i \wedge b_j|. \tag{3.2}$$

100-NN mAP with GIST-1M-384D

# bits	32	64	128	256	512
SHD	0.0153	0.0426	0.981	0.1760	0.2572
SHD-SUB	0.0139	0.0398	0.0931	0.1656	0.2402

Table 3.2: Comparisons between SHD and SHD-SUB described in Sec. 3.2.2

SHD-SUB is intuitive and free from the division by zero. However, the estimated distance is linearly decreasing with respect to $|b_i \wedge b_j|$ and thus a little bit different from our observation in Fig. 3.2(b). We also provide experimental comparison between SHD and SHD-SUB in Table. 3.2. Since SHD provides slightly better performance compared to SHD-SUB, we have used SHD in all the experiments in this paper instead of SHD-SUB.

3.2.3 Independence between Hashing Functions

Achieving balanced partitioning of data points for each hashing function and the independence between hashing functions has been known to be important [9,20,21], since independent hashing functions distribute points in a balanced manner to different binary codes. It has been known that achieving such properties lead to minimizing the search time [20] and improving the accuracy even for longer code lengths [21]. We also aim to achieve this independence between our spherical hashing functions.

We define each hashing function h_i to have the equal probability for $+1$ and -1 bits respectively as the following:

$$Pr[\ h_i(x) = +1\ ] = \frac{1}{2}, \quad x \in X, \quad 1 \leq i \leq l \tag{3.3}$$

Let us define a probabilistic event V_i to represent the case of $h_i(x) = +1$. Two events V_i and V_j are independent if and only if $Pr[V_i \cap V_j] = Pr[V_i] \cdot Pr[V_j]$. Once we achieve balanced partitioning of data points for each bit (Eq. 3.3), then the independence between two bits can satisfy the following equation given $x \in X$ and $1 \leq i < j \leq l$:

$$\begin{aligned} &Pr[h_i(x) = +1,\ h_j(x) = +1] \\ &\quad = Pr[h_i(x) = +1] \cdot Pr[h_j(x) = +1] = \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{1}{4} \end{aligned} \tag{3.4}$$

In general the pair-wise independence between hashing functions does not guarantee the higher-order independence among three or more hashing functions. We can also formulate the independences among more than two hashing functions and aim to satisfy them in addition to constraints shown in Eq. 3.3 and Eq. 3.4. However we found that considering such higher-order independence hardly improves the search quality.

3.2.4 Iterative Optimization

We now propose an iterative process for computing l different hyperspheres, i.e. their pivots p_i and distance thresholds t_i. During this iterative process we construct hyperspheres to satisfy constraints shown in Eq. 3.3 and Eq. 3.4.

As the first phase of our iterative process, we sample a subset $S = \{s_1, s_2, ..., s_n\}$ from data points X to approximate its distribution. We then initialize the pivots of l hyperspheres with randomly chosen l data points in the subset S; we found that other alternatives of initializing the pivots (e.g., using center

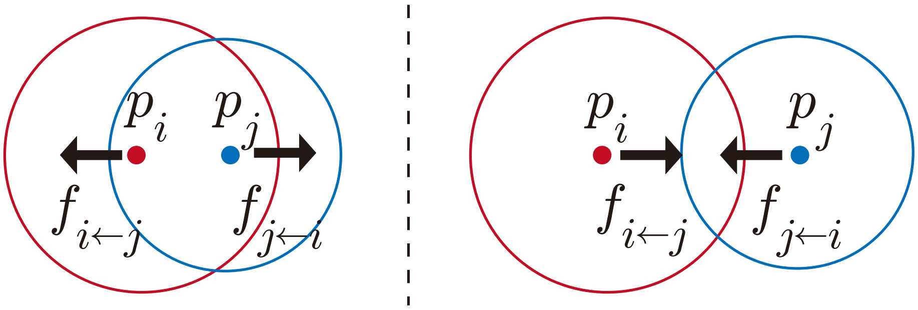


Figure 3.4: These two images show how a force between two pivots is computed. In the left image a repulsive force is computed since their overlap $o_{i,j}$ is larger than the desired amount. On the other hand, the attractive force is computed in the right image because their overlap is smaller than the desired amount.

points of K-means clustering performed on the subset S) do not affect the results of our optimization process. However, we observe that the optimization process converges slightly quicker, when initial pivots are closely located in the center of the training points. This is mainly because by locating hyperspheres closely to each other, we can initialize hyperspheres to have overlaps. For this acceleration, we set the pivot position of a hypersphere to be the median of randomly chosen multiple samples, i.e. $p_i = \frac{1}{g}\sum_{j=1}^{g} q_j$, where q_j are randomly selected points from S and g is the number of such points. Too small g does not locate pivots closely to the data center, and too large g locates pivots to be in almost similar positions. In practice, $g = 10$ provides a reasonable acceleration rate, given its trade-off space.

As the second phase of our iterative process, we refine pivots of hyperspheres and compute their distance thresholds. To help these computations, we compute the following two variables, o_i and $o_{i,j}$, given $1 \le i, j \le l$:

$$\begin{aligned} o_i &= |\ \{s_g | h_i(s_g) = +1, 1 \le g \le n\}\ |, \\ o_{i,j} &= |\ \{s_g | h_i(s_g) = +1, h_j(s_g) = +1, 1 \le g \le n\}\ |, \end{aligned}$$

where $|\cdot|$ is the cardinality of the given set. o_i measures how many data points in the subset S have $+1$ bit for i-th hashing function and will be used to satisfy balanced partitioning for each bit (Eq. 3.3). Also, $o_{i,j}$ measures the number of data points in the subset S that are contained within both of two hyperspheres corresponding to i-th and j-th hashing functions. $o_{i,j}$ will be used to satisfy the independence between i-th and j-th hashing functions during our iterative optimization process.

Once we compute these two variables with data points in the subset of S, we adopt two alternating steps to refine pivots and distance thresholds for hyperspheres. First, we adjust the pivot positions of two hyperspheres in a way that $o_{i,j}$ becomes closer to or equal to $\frac{n}{4}$. Intuitively, for each pair of two hyperspheres i and j, when $o_{i,j}$ is greater than $\frac{n}{4}$, a repulsive force is applied to both pivots of those two hyperspheres (i.e. p_i and p_j) to place them farther away. Otherwise an attractive force is applied to locate them closer. Second, once pivots are computed, we adjust the distance threshold t_i of ith hypersphere such that o_i becomes $\frac{n}{2}$ to meet balanced partitioning of the data points for the hypersphere (Eq. 3.3).

We perform our iterative process until the computed hyperspheres do not make further improvements in terms of satisfying constraints. Specifically, we consider the sample mean and standard deviation of $o_{i,j}$ as a measure of the convergence of our iterative process. Ideal values for the mean and standard deviation of $o_{i,j}$ are $\frac{n}{4}$ and zero respectively. However, in order to avoid over-fitting, we stop our iterative

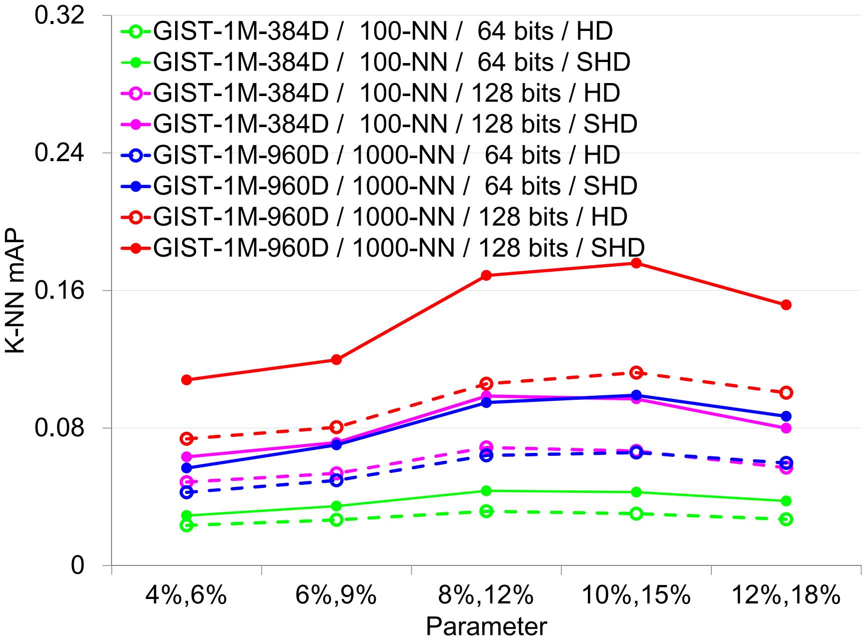


Figure 3.5: This graph shows mean Average Precision (mAP) curves for k-nearest neighbor search with respect to various parameters. A pair of values in x-axis are used for two parameters of ϵ_m and ϵ_s, and y-axis represents their corresponding mAP values. Each legend consists of four experiment settings 'dataset / k / binary code length / distance metric type (HD: Hamming distance, SHD: spherical Hamming distance)'.

process when the mean and standard deviation of $o_{i,j}$ are within ϵ_m% and ϵ_s%, error tolerances, of the ideal mean of $o_{i,j}$ respectively.

For these parameters, we conducted the following experimental tests to find suitable values. We compute mean Average Precisions (mAPs) of k-nearest neighbor search with various experiment settings, and they are shown in Fig. 3.5. According to the experimental results, we pick ϵ_m and ϵ_s that provide the empirical maximum. Based on these experimental tests, we have chosen (ϵ_m=10%, ϵ_s=15%) for GIST-1M-384D, GIST-1M-960D, and 1000 dimensional BoW descriptors. We have, however, found that we need stricter termination conditions of the optimization process for higher dimensional data. The convergence rate of the objective functions is much faster in higher dimensional space, since we have more degrees of freedom of pivot positions, and this can cause an undesired under-fitting. We have therefore chosen (ϵ_m=4%, ϵ_s=6%) for 8192 dimensional VLAD descriptors (Sec. 3.4.1).

Force computation: A (repulsive or attractive) force from p_j to p_i, $f_{i\leftarrow j}$, is defined as the following (Fig. 3.4):

$$f_{i\leftarrow j} = \frac{1}{2}\frac{o_{i,j} - n/4}{n/4}(p_i - p_j). \tag{3.5}$$

An accumulated force, f_i, is then the average of all the forces computed from all the other pivots as the following:

$$f_i = \frac{1}{l}\sum_{j=1}^{l} f_{i\leftarrow j}.$$

Once we apply the accumulated force f_i to p_i, then p_i is updated simply as $p_i + f_i$. Our iterative

Algorithm 1 Our iterative optimization process

```
Input: sample points $S = \{s_1, ..., s_n\}$, error tolerances $\epsilon_m$ and $\epsilon_s$, and the number of hash functions $l$
Output: pivot positions $p_1, ..., p_l$ and distance thresholds $t_1, ..., t_l$ for $l$ hyperespheres
Initialize $p_1, ..., p_l$ with randomly chosen $l$ data points from the set $S$
(It can be replaced with $p_i = \frac{1}{g}\sum_{j=1}^{g} q_j$ for quicker convergence, where $q_j$ are randomly selected points from $S$)
Determine $t_1, ..., t_l$ to satisfy $o_i = \frac{n}{2}$ (Sec. 3.2.5)
Compute $o_{i,j}$ for each pair of hashing functions
repeat
  for $i = 1$ to $l - 1$ do
    for $j = i + 1$ to $l$ do
      $f_{i \leftarrow j} = \frac{1}{2}\frac{o_{i,j} - n/4}{n/4}(p_i - p_j)$
      $f_{j \leftarrow i} = -f_{i \leftarrow j}$
    end for
  end for
  for $i = 1$ to $l$ do
    $f_i = \frac{1}{l}\sum_{j=1}^{l} f_{i \leftarrow j}$
    $p_i = p_i + f_i$
  end for
  Determine $t_1, ..., t_l$ to satisfy $o_i = \frac{n}{2}$ (Sec. 3.2.5)
  Compute $o_{i,j}$ for each pair of hashing functions
until $avg(\mid o_{i,j} - \frac{n}{4} \mid) \le \epsilon_m \frac{n}{4}$ and $std\text{-}dev(o_{i,j}) \le \epsilon_s \frac{n}{4}$
```

optimization process is shown in Algorithm 1. A simple example of the optimization process in 2-D space is presented in Fig. 3.8.

The time complexity of our iterative process is $O((l^2 + lD)n)$, which is comparable to those of the state-of-the-art techniques (e.g., $O(D^2n)$ of spectral hashing [9]). In practice, our iterative process is finished within 10 to 30 iterations. Also, its overall computation time is less than 30 seconds even for 128 code lengths. The convergence rate with respect to the number of iterations is shown in Fig. 3.6 and Fig. 3.7. Note that our iterative optimization process shares similar characteristics of the N-body simulation [53] designed for simulating various dynamic systems of particles (e.g., celestial objects interacting with each other under gravitational forces). Efficient numerical integration methods (e.g., fast multipole method) can be applied to accelerate our iterative optimization process.

One may wonder why we do not use k-means to compute centers of hyperspheres. Using k-means clustering to obtain centers of hyperspheres is very intuitive, since k-means locates the centers in dense regions and assigning the same hash value to those dense regions seems an appropriate direction. However, this alternative does not ensure the independence between hashing functions. The cluster centers obtained by k-means clustering in a high dimensional space are highly likely to be close to the data mean. It leads that hyperspheres are highly overlapped, and a high portion of regions are not covered by any hypersphere. As a result, the alternative optimization scheme does not meet our independence criteria, since hashing functions corresponding to highly overlapped hyperspheres will generated correlated hash values.

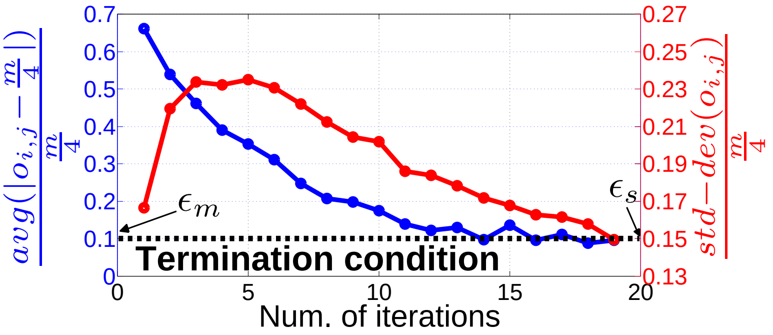


Figure 3.6: This graph shows a convergence rate of our iterative optimization. The left and right y-axises indicate how the average and std. dev. of $o_{i,j}$ approach our termination condition in the scale of our error tolerances, ϵ_m and ϵ_s respectively. In this case, we set ϵ_m as 0.1 and ϵ_s as 0.15 for terminating our optimization. This result is obtained with the **GIST-1M-384D** dataset at the 64-bit code length.

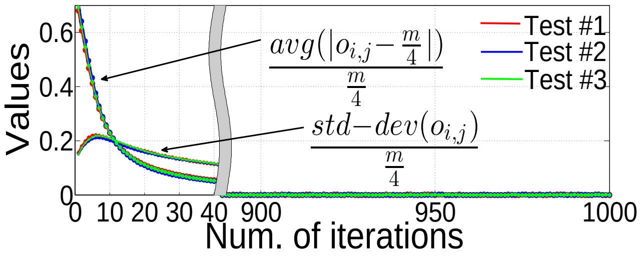


Figure 3.7: Convergence rates of our iterative optimization with three individual trials. The optimization processes are finished within 30 iterations when we se we set ϵ_m as 0.1 and ϵ_s as 0.15. This graph also shows that both objectives converge to 0 when we increase the number of iterations. This result is obtained with the **GIST-1M-384D** dataset at the 64-bit code length.

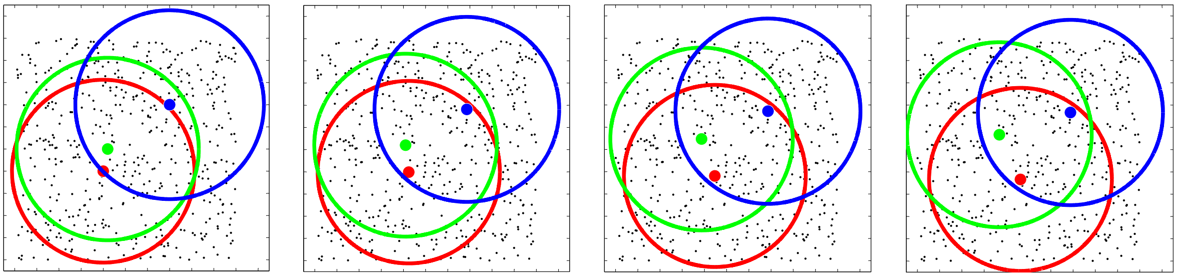


Figure 3.8: These figures visualize results of our optimization process with three hyperspheres and 500 points in 2D space. The leftmost and rightmost figures show an initial state and converged state of the optimization process, respectively, while two middle figures show intermediate states.

100-NN mAP with GIST-1M-384D

# bits	32	64	128	256	512
SHD+M	0.0153	0.0426	0.0981	0.1760	0.2572
SHD	0.0147	0.0409	0.0938	0.1678	0.2434
HD+M	0.0113	0.0310	0.0665	0.1152	0.1648
HD	0.0107	0.0290	0.0653	0.1113	0.1583

1000-NN mAP with GIST-1M-960D

# bits	32	64	128	256	512
SHD+M	0.0460	0.0982	0.1782	0.2738	0.3560
SHD	0.0439	0.0945	0.1756	0.2641	0.3398
HD+M	0.0322	0.0660	0.1132	0.1669	0.2103
HD	0.0310	0.0636	0.1126	0.1644	0.2058

Table 3.3: These two tables show the effect of our max-margin based distance thresholding (Sec. 3.2.5). **SHD** and **HD** indicate the binary code distance metric type, and **M** indicates the max-margin based distance thresholding scheme. The max-margin based distance thresholding improves mAPs 3.8% on average over the median based distance thresholding across various settings of experiments.

3.2.5 Max-Margin based Distance Thresholding

In each iteration step, we need to determine distance thresholds $t_1, ..., t_l$ to satisfy $o_i = \frac{n}{2}$ for the balanced partitioning. For this we could simply set each t_i as $d(p_i, s_{n/2})$ the distance from p_i to $s_{n/2}$, when samples of S are sorted into $s_1, ..., s_n$ in terms of distance from p_i. However, this simple approach could lead undesirable partitioning, especially when $s_{n/2}$ is located in a dense region. To ameliorate this concern, the distance threshold t_i is set to maximize a margin from points to to the hypersphere without significant loss on the balance partition criterion. For our max-margin based threshold optimization, we first sort samples of S into $s^s_1, ..., s^s_n$ according to $d(p_i, s^s_j)$ the distance to the pivot. Instead of simply using the median point $s^s_{n/2}$ with its index, $n/2$, indicating a sample in the ordered list, we compute a set J containing candidate indices near the median $\frac{n}{2}$ for the optimization:

$$J = \{j | (\frac{1}{2} - \beta)n \leq j \leq (\frac{1}{2} + \beta)n, j \in \mathbb{Z}^+\},$$

where β is a parameter that controls the degree of tolerance for breaking the balance partition criterion. We set $\beta = 0.05$ in practice. We then compute an index $\hat{j}$ of a sample among the sorted list that maximizes the margin to the hypersphere as the following:

$$\hat{j} = \arg\max_{j \in J} d(t_i, s^s_{j+1}) - d(t_i, s^s_j)$$

The distance threshold t_i is finally determined such that the hypersphere partitions $s^s_{\hat{j}}$ and $s^s_{\hat{j}+1}$ as the following:

$$t_i = \frac{1}{2}(d(t_i, s^s_{\hat{j}}) + d(t_i, s^s_{\hat{j}+1}))$$

Table 3.3 shows how much the max-margin based distance thresholding improves the performance of our method. Our margin-based distance threshold computation scheme improves mAPs 3.8% on average, compared with the simple median based distance thresholding.

3.3 Generalized Spherical Hashing

Many applications benefit from the use of domain-specific kernels that define data similarities [54,55]. In this section we generalize our basic spherical hashing (Sec. 3.2) to a kernelized one.

Let us first define notations. Given a set of N data elements in an input space $\mathcal{X}$, we use $X = \{x_1, x_2, ..., x_N\} \in \mathcal{X}$ to denote those data elements. We use a non-linear map $\Phi : \mathcal{X} \rightarrow \mathcal{F}$ from the input space $\mathcal{X}$ to a $feature\ space\ \mathcal{F}$. We denote $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$ as a kernel function corresponding to the map Φ, where $\langle \cdot, \cdot \rangle$ is the inner product operator.

3.3.1 Kernelized Binary Code Embedding Function

The squared distance between two points $\Phi(x)$ and $\Phi(y)$ in the feature space $\mathcal{F}$ can be expressed with the kernel function as the following:

$$\begin{aligned} &\parallel \Phi(x) - \Phi(y) \parallel^2 \\ &= \quad \langle \Phi(x), \Phi(x) \rangle - 2\langle \Phi(x), \Phi(y) \rangle + \langle \Phi(y), \Phi(y) \rangle \\ &= \quad k(x, x) - 2k(x, y) + k(y, y). \end{aligned} \tag{3.6}$$

Our binary code construction function $H(x) = (h_1(x), ..., h_l(x))$ maps a data element in the input space into the Hamming space $\{-1, +1\}^l$. Each kernelized spherical hashing function $h_i(x)$ is defined with the pivot point p_i in the feature space and distance threshold t_i as the following:

$$h_i(x) = \begin{cases} -1 & \text{when} \parallel \Phi(x) - p_i \parallel^2 > t_i^2 \\ +1 & \text{when} \parallel \Phi(x) - p_i \parallel^2 \leq t_i^2 \end{cases}.$$

Intuitively, each kernelized spherical hashing function $h_i(x)$ determines whether $\Phi(x)$, the point x mapped into the feature space, is inside the hypersphere defined by its center p_i and radius t_i.

To represent the center of a hypersphere in the feature space, we use a set of m landmark samples $Z = \{z_1, ..., z_m\} \in X$, where $m \ll n$. We now express the center p_i by a linear combination of $\{\Phi(z_1), ..., \Phi(z_m)\}$ as the following:

$$p_i = \sum_{j=1}^{m} w_j^i \Phi(z_j), \tag{3.7}$$

where $w_j^i \in \mathbb{R}$ denotes a weight of $\Phi(z_j)$ for p_i.

The squared distance between a point $\Phi(x)$ and the pivot p_i used in our kernelized spherical hashing function is computed as the following:

$$\begin{aligned}
& \| \Phi(x) - p_i \|^2 \\
= \;& \langle \Phi(x), \Phi(x) \rangle - 2\langle \Phi(x), p_i \rangle + \langle p_i, p_i \rangle \qquad \text{[by Eq. 3.6]} \\
= \;& \langle \Phi(x), \Phi(x) \rangle - 2\langle \Phi(x), \sum_{j=1}^{m} w_j^i \Phi(z_j) \rangle \\
& + \langle \sum_{j=1}^{m} w_j^i \Phi(z_j), \sum_{j=1}^{m} w_j^i \Phi(z_j) \rangle \qquad \text{[by Eq. 3.7]} \\
= \;& \langle \Phi(x), \Phi(x) \rangle - 2\sum_{j=1}^{m} w_j^i \langle \Phi(x), \Phi(z_j) \rangle \\
& + \sum_{j=1}^{m}\sum_{g=1}^{m} w_j^i w_g^i \langle \Phi(z_j), \Phi(z_g) \rangle \\
= \;& k(x,x) - 2\sum_{j=1}^{m} w_j^i k(x, z_j) + \sum_{j=1}^{m}\sum_{g=1}^{m} w_j^i w_g^i k(z_j, z_g)
\end{aligned}$$

Note that the last term $\sum_{j=1}^{m}\sum_{g=1}^{m} w_j^i w_g^i k(z_j, z_g)$ can be pre-computed for each hypersphere, since it is independent from x [56].

3.3.2 Kernelized Iterative Optimization

As did in Sec. 3.2.4 we first sample a training set $S = \{s_1, ..., s_n\}$ from X to approximate its distribution, and also sample a subset $Z = \{z_1, ..., z_m\}$ from S as landmarks that are used for defining center positions of hyperspheres, as described in Eq. 3.7.

Initial center positions p_i of hyperspheres are chosen randomly. Specifically speaking, we set each element of weight vectors w^i defining centers p_i of hyperspheres with randomly drawn values from the uniform distribution $U(-1, 1)$ and normalize the weight vectors according to L_2 norm.

To express two constraints of Eq. 3.3 and Eq. 3.4 with the training set X, let us recall the following two variables, o_i and $o_{i,j}$, given $1 \le i < j \le l$:

$$\begin{aligned}
o_i &= | \{ s_g | h_i(s_g) = +1, 1 \le g \le n \} |, \\
o_{i,j} &= | \{ s_g | h_i(s_g) = +1, h_j(s_g) = +1, 1 \le g \le n \} |,
\end{aligned}$$

where $|\cdot|$ is the cardinality of the given set. o_i, and $o_{i,j}$ measure how many data elements in the training set S are inside a single i-th hypersphere and inside the overlap region of i-th and j-th hyperspheres, respectively. The objectives of our optimization are then described as the following:

$$o_i = \frac{n}{2} \text{ and } o_{i,j} = \frac{n}{4}.$$

In each iteration of our optimization process, we first measure o_i and $o_{i,j}$. We then move centers of hyperspheres to meet $o_{i,j} = \frac{n}{4}$, followed by adjusting radii of hyperspheres to satisfy $o_i = \frac{n}{2}$.

For moving centers of hyperspheres, we compute a pair-wise force vector $f_{i \leftarrow j} = -f_{j \leftarrow i}$ between i-th and j-th hyperspheres. $f_{i \leftarrow j}$ is a force applied to i-th hypersphere caused by j-th hypersphere. We

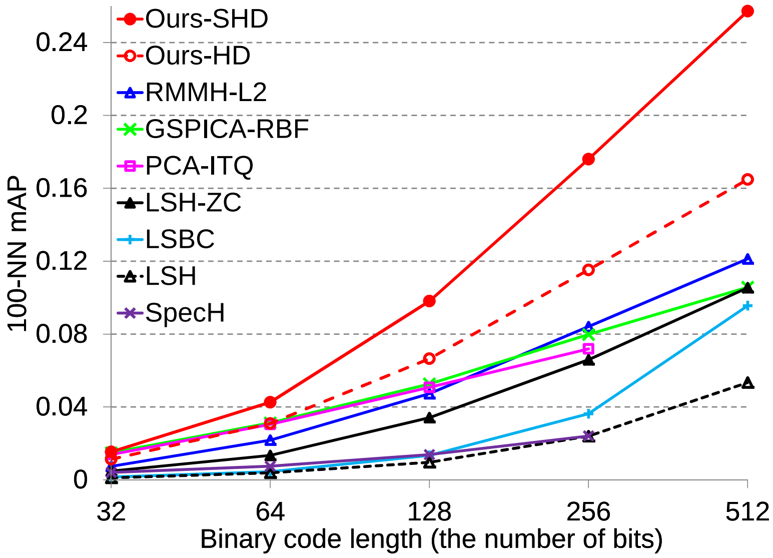


Figure 3.9: Comparison between our method and the state-of-the-art methods with the GIST-1M-384D dataset when $k = 100$

kernelize Eq. 3.5 as the following:

$$\begin{aligned} f_{i\leftarrow j} &= \frac{1}{2}\frac{o_{i,j}-n/4}{n/4}(p_i - p_j) \\ &= \frac{1}{2}\frac{o_{i,j}-n/4}{n/4}\left(\sum_{g=1}^{m} w_g^i \Phi(z_g) - \sum_{g=1}^{m} w_g^j \Phi(z_g)\right) \\ &= \frac{1}{2}\frac{o_{i,j}-n/4}{n/4}\sum_{g=1}^{m}(w_g^i - w_g^j)\Phi(z_g). \end{aligned}$$

Same as Algorithm 1, a pair of forces $f_{i\leftarrow j}$ and $f_{j\leftarrow i}$ between i-th and j-th hyperspheres attracts each other to increase their overlap when their overlap $o_{i,j}$ is less than $\frac{n}{4}$, or pushes them farther away in the other case. Once we compute all the forces, we compute an accumulated force $f_i = \frac{1}{l}\sum_{j=1}^{l} f_{i\leftarrow j}$ for each hypersphere and adjust its center p_i to $p_i + f_i$.

We iteratively perform our optimization process until the mean and standard deviation of $o_{i,j}$ are within ϵ_m% and ϵ_s% of $\frac{n}{4}$ to avoid over-fitting in the same manner to the basic spherical hashing method.

3.4 Evaluation

In this section we evaluate our method and compare it with the state-of-the-art methods [7, 9, 12, 19, 20, 21]. We use a machine consisting of Xeon X5690 and 144GB main memory to hold all the data in its main memory.

3.4.1 Datasets

We perform various experiments with the following four datasets:

- **GIST-1M-384D**: A set of 384 dimensional, one million GIST descriptors, which consist of a subset of Tiny Images [8].
- **GIST-1M-960D**: A set of 960 dimensional, one million GIST descriptors that are also used in [23].

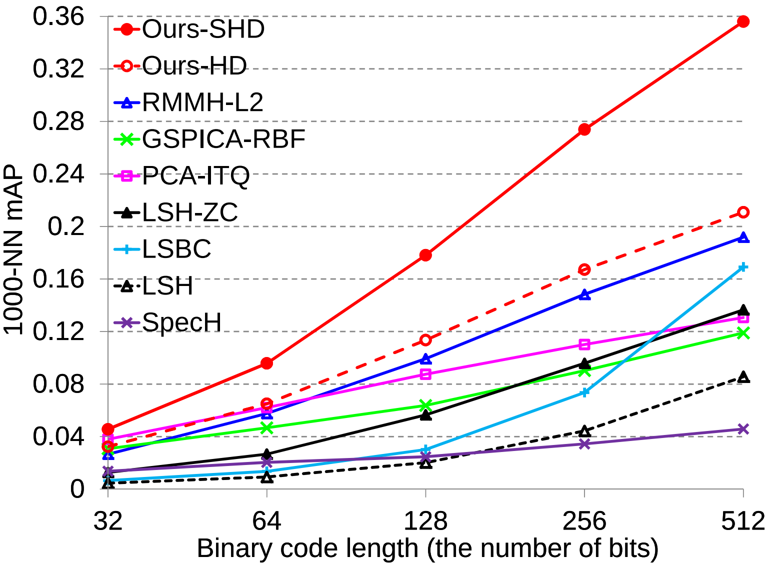


Figure 3.10: Comparison between our method and the state-of-the-art methods with the GIST-1M-960D dataset when $k = 1,000$

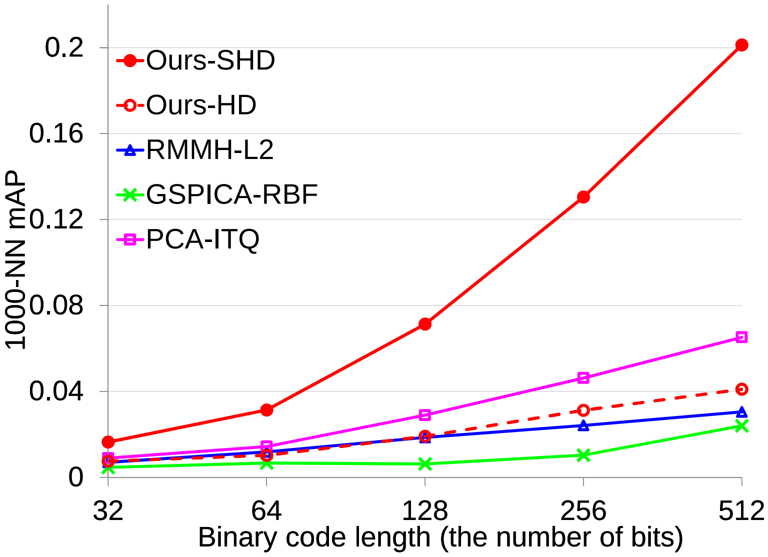


Figure 3.11: Comparison between our method and the-state-of-the-art methods with the **VLAD-1M-8192D** dataset when $k = 1,000$.

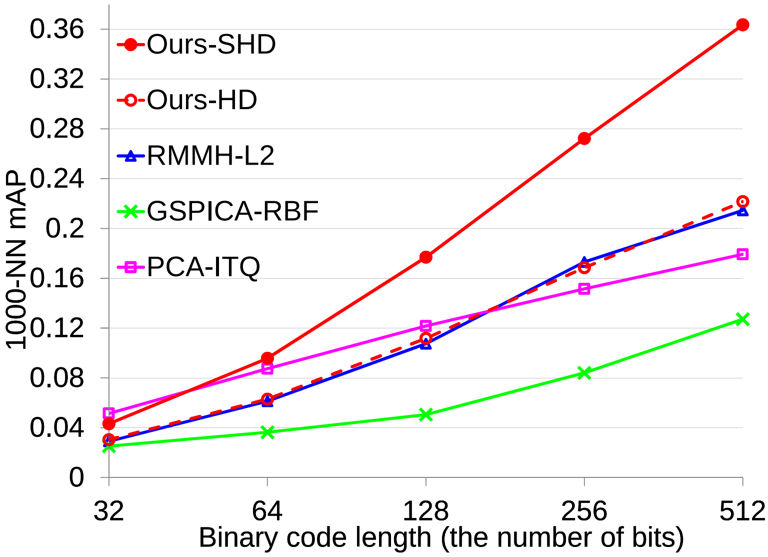
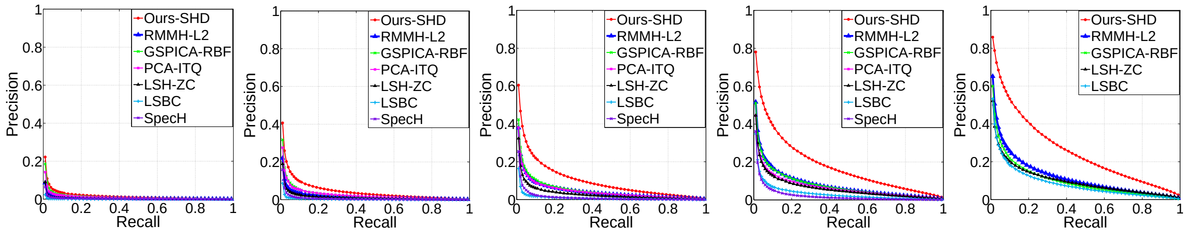
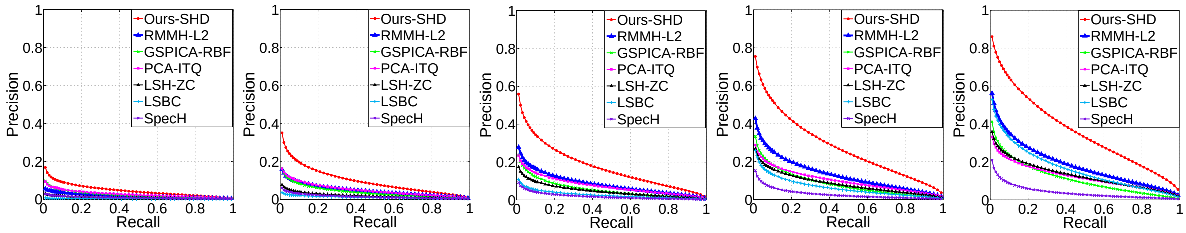


Figure 3.12: Comparison between our method and the-state-of-the-art methods with the **L2-normalized GIST-1M-960D** dataset when $k = 1,000$.



(a) 32 bits (b) 64 bits (c) 128 bits (d) 256 bits (e) 512 bits

Figure 3.13: Comparison between our method and the state-of-the-art methods on GIST-1M-384D dataset when $k = 100$. Refer to Fig. 3.9 for the mAP curves.



(a) 32 bits (b) 64 bits (c) 128 bits (d) 256 bits (e) 512 bits

Figure 3.14: Comparison between our method and the state-of-the-art methods on GIST-1M-960D dataset when $k = 1,000$. Refer to Fig. 3.10 for the mAP curves.

- **GIST-75M-384D**: A set of 384 dimensional, 75 million GIST descriptors, which consist of a subset of 80 million Tiny Images [8].
- **ILSVRC**: A set of 1000 dimensional, one million SIFT-based Bag of visual Words (BoW) histogram descriptors provided from ImageNet Large Scale Visual Recognition Challenge 2010 dataset, which is a subset of the ImageNet database [57].
- **VLAD-1M-8192D**: One million of 8192 dimensional VLAD [58] descriptors (128 dimensional SIFT features and 64 codebook vectors).

3.4.2 Evaluation on Euclidean Space

We first present results with the Euclidean space, followed by ones with the kernel space.

Protocol

Our evaluation protocol follows that of [21]. Specifically, we tested with randomly chosen 1000 queries for datasets **GIST-1M-384D** and **GIST-1M-960D**, **VLAD-1M-8192D** and 500 queries for **GIST-75M-384D** that do not have any overlap with data points. The performance is measured by mean Average Precision (mAP). The ground truth is defined by k nearest neighbors that are computed by the exhaustive, linear scan based on the Euclidean distance. When calculating precisions, we consider all the items having lower or the equal Hamming distance (or spherical Hamming distance) from given queries.

Compared Methods

- **LSH** and **LSH-ZC**: Locality Sensitive Hashing [7] with/without Zero Centered data points. The projection matrix is a Gaussian random matrix. As discussed in [19,59], centering the data around the origin (*i.e.* $\sum x_i = 0$) produces much better results over **LSH**. Hence, we transform data points such that their center is located at the origin for **LSH-ZC**.
- **LSBC**: Locality Sensitive Binary Codes [12]. The bandwidth parameter used in experiment is the inverse of the mean distance between the points in the dataset, as suggested in [59].
- **SpecH**: Spectral Hashing [9].
- **PCA-ITQ**: Iterative Quantization [19].
- **RMMH-L2**: Random Maximum Margin Hashing (RMMH) [21] with the triangular L2 kernel. We experiment RMMH with the triangular L2 kernel since the authors reported the best performance on k nearest neighbor search with this kernel. We use 32 for the parameter M that is the number of samples for each hash function, as suggested by [21].
- **GSPICA-RBF**: Generalized Similarity Preserving Independent Component Analysis (GSPICA) [20] with the RBF kernel. We experiment GSPICA with the RBF kernel, since the authors reported the best performance on k nearest neighbor search with this kernel. The parameter used in the RBF kernel is determined by the mean distance of kth nearest neighbors within training samples as suggested by [21]. The parameters γ and P are 1 and the dimensionality of the dataset respectively, as suggested in [20].

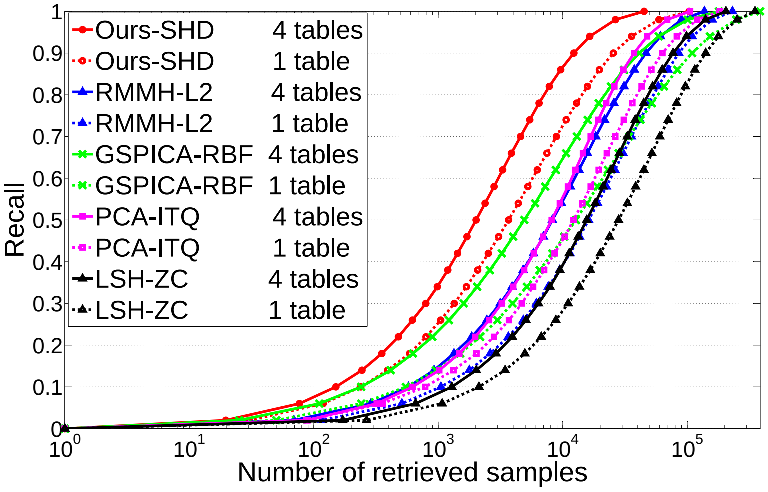


Figure 3.15: Recall curves of different methods when $k = 100$ for the **GIST-1M-384D** dataset. Each hash table is constructed by 64 bits code lengths. The recall (y-axis) represents search accuracy and the number of retrieved samples (x-axis) represents search time. Graphs are best viewed with colors.

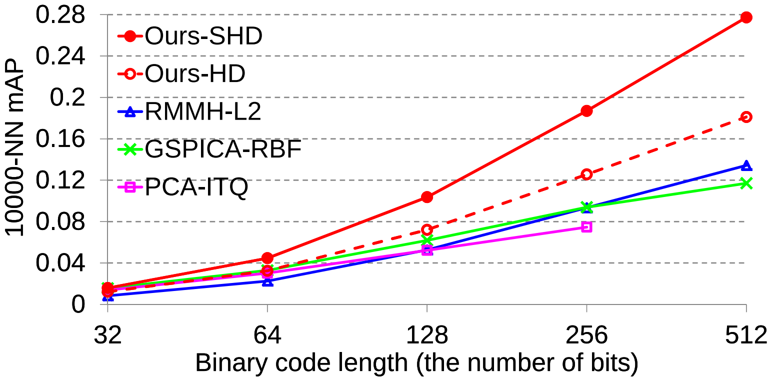


Figure 3.16: Comparison between our method and the-state-of-the-art methods with the **GIST-75M-384D** dataset when $k = 10,000$.

- **Ours-HD** and **Ours-SHD**: We have tested two different versions of our method. **Ours-HD** represents our method with the common Hamming distance, while **Ours-SHD** uses our spherical Hamming distance (Sec. 3.2.2). Max-margin based distance thresholding scheme (Sec. 3.2.5) is also applied to both versions of our method.

For all the data-dependent hashing methods, we randomly choose 100K data points from the original dataset as a training set. We also use the same training set to estimate parameters of each method. We report the average mAP and recall values by repeating all the experiments five times, in order to gain statistically meaningful values; for **GIST-75M-384D** benchmark, we repeat experiments only three times because of its long experimentation time. Note that we do not report results of two PCA-based methods **SpecH** and **PCA-ITQ** for 512 hash bits at 384 dimensional datasets, since they do not support bit lengths larger than the dimension of the data space.

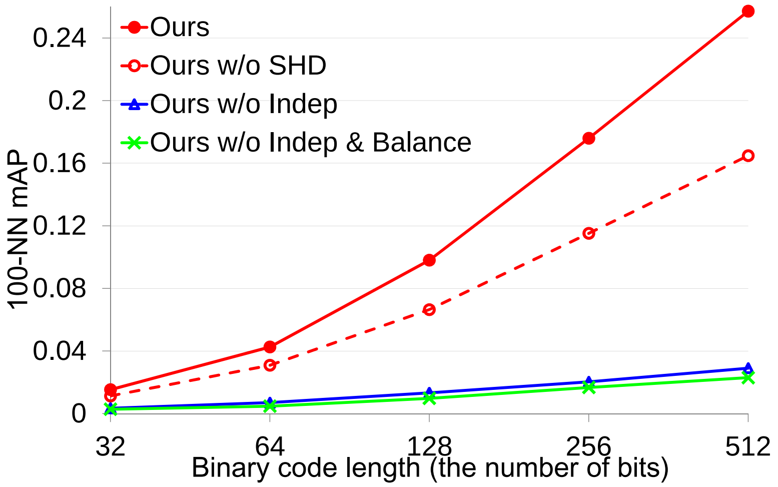


Figure 3.17: This figure shows how each component of our method affects the accuracy. The mAP curves are obtained with GIST-1M-384D dataset when $k = 100$.

Results

Fig. 3.9 shows the mAP of k nearest neighbor search of all the tested methods when $k = 100$. Our method with the spherical Hamming distance, **Ours-SHD**, shows better results over all the tested methods across all the tested bit lengths ranging from 32 bits to 512 bits. Furthermore, our method shows increasingly higher benefits over all the other tested methods as we allocate more bits. This increasing improvement is mainly because using multiple hyperspheres can effectively create closed regions with tighter distance bounds compared to hyperplanes.

Given 0.1 mAP in Fig. 3.9, our method needs to use 128 bits to encode each image. On the other hand, other tested methods should use more than 256 bits. As a result, our method provides over two times more compact data representations than other methods. We would like to point out that low mAP values of our method are still very meaningful, as discussed in [21]. Once we identify nearest neighbor images based on binary codes, we can employ additional re-ranking processes on those images. As pointed out in [21], 0.1 mAP given $k = 100$ nearest neighbors, for example, indicates that 1000 images on average need to be re-ranked.

Performances of our methods with two different binary code distance functions are also shown in Fig. 3.9. Our method with the Hamming distance **Ours-HD** shows better results than most of other methods across different bits, especially higher bits. Furthermore, the spherical Hamming distance **Ours-SHD** shows significantly improved results even than **Ours-HD**. The spherical Hamming distance function also shows increasingly higher improvement over the Hamming distance, as we add more bits for encoding images.

Our technique can be easily extended to use multiple hash tables; for example, we can construct a new hash table by recomputing S, the subset of the original dataset. Fig. 3.15 shows recall curves of different methods with varying numbers of hash tables, when we allocate 64 bits for encoding each image. Our method (with our spherical Hamming distance) improves the accuracy as we use more tables. More importantly, our method only with a single table shows significantly improved results over all the other tested methods that use four hash tables. For example, when we aim to achieve 0.5 recall rate, our method with one and four hash tables needs 3674 and 2013 images on average respectively. However, **GSPICA-RBF** [20] with four hash tables, the second-best method, needs to identify 4909 images, which are 33% and 143% more number of images than our method with one and four hash tables respectively.

We have also performed all the tests against the 960 dimensional, one million GIST dataset **GIST-1M-960D** with $k = 1000$ (Fig. 3.10). We have found that our method shows similar trends even with this dataset, compared to what we have achieved in **GIST-1M-384D**. The precision-recall curves corresponding to Fig. 3.9 and Fig. 3.10 are Fig. 3.13 and Fig. 3.14 respectively.

We have performed each test multiple times, since our method can be impacted by different initializations. However, our method shows robust results against different initializations. For example, the standard deviation of mAPs of five experiments with **GIST-1M-960D** when the code length is 64 bits is only 0.0017, while the average mAP is 0.0982. The standard deviation with 256 bits is 0.0035, while the average is 0.2738.

In order to evaluate our method with very high dimensional data, we have performed the tests with 8192 dimensional, one million VLAD dataset **VLAD-1M-8192D** with $k = 1000$ (Fig. 3.11). **Ours-SHD** consistently provides the best performance among the tested techniques.

We have also performed all the tests against the 384 dimensional, 75 million GIST dataset **GIST-75M-384D** with $k = 10,000$ (Fig. 3.16). We have found that our method shows significantly higher results than all the other tested methods across all the tested bit lengths even with this large-scale dataset.

In order to see how each component of our method affects the accuracy, we measure mAP by disabling the spherical Hamming distance, the independent constraint, and balanced partitioning in our method (Fig. 3.17). In the case of using 64 bits, mAP of our method goes down 28%, 83%, and 90% by disabling the spherical Hamming distance, the independence constraint, and the balanced partitioning/independence constraints respectively.

Finally, we also measure how efficiently our hypersphere-based hashing method generates binary codes given a query image. Our method takes 0.08 ms for generating a 256 bit-long binary code. This cost is same to that of the LSH technique generating binary codes based on hyperplanes.

3.4.3 Evaluation on Kernel Space

Datasets

We normalized **GIST-1M-384D** and **GIST-1M-960D** datasets according to L_2-norm to make exact k-nearest neighbors with the linear kernel to be equivalent to the k-nearest neighbors with the RBF kernel as did in [21]. As a result, we can compare results acquired by using two different kernels in the same ground. We also normalized **ILSVRC** dataset according to L_1-norm as suggested in [60]. For all the experiments, we tested with randomly chosen one thousand queries that do not overlap with data elements. We report the average result by repeating all the experiments three times.

Compared Methods

- **RMMH**: Random Maximum Margin Hashing [21]. We used 32 for the parameter M that is the number of samples for each hash function as suggested by [21].
- **GSPICA**: Generalized Similarity Preserving Independent Component Analysis [20]. We set the parameter P as the dimensionality of the dataset and γ to 1 as suggested in the paper.

There are a few more kernelized binary hashing methods such as KLSH [30]. According to papers of **RMMH** and **GSPICA**, their techniques have been reported to be better than KLSH. As a result, we compare our method only against them. For our method **Ours**, we set the number of landmarks m as the dimensionality of input data.

Used Kernels

We have tested our method with the following four popular kernels:

- **Linear**: Linear kernel, $k(x,y) = \langle x, y \rangle$.
- **RBF**: RBF kernel, $k(x,y) = exp(-\gamma \parallel x - y \parallel^2)$. We set the bandwidth parameter γ as an inverse of the mean distance between randomly sampled points as suggested by [59].
- **HI**: Histogram Intersection kernel, $k(x,y) = \sum_{i=1}^{D} min(x_i, y_i)$, where D is the dimensionality of x and y.
- **CS**: Chi Square kernel, $k(x,y) = 2\sum_{i=1}^{D} \frac{x_i y_i}{x_i + y_i}$.

Results

We evaluated our method with k-nearest neighbor search. The ground truth is defined by top k data elements based on each tested kernel function. The performance is measured by mAP. When calculating precisions, we consider all the items having lower or equal Hamming distance from given queries.

Fig. 3.18 shows the mAP of k-nearest neighbor search of all the tested methods when $k = 1000$; other cases (e.g. $k = 50, 100, 500, 2000$) show similar trends. We evaluated the performance of our method with **Linear** and **RBF** in **GIST384D** and **GIST960D** datasets (Fig. 3.18-(a) and -(b)); we report results of compared methods only with **RBF**, since **RBF** gives better results than **Linear**.

We also experimented with **CS** and **HI** in **ILSVRC** dataset (Fig. 3.18-(c)). Since **RMMH** performed better than **GSPICA** in this experiment, we report results of **RMMH** in this graph. We have found that our method consistently shows higher performance than the state-of-the-art methods in all the tested benchmarks with various kernels.

3.4.4 Evaluation on Image Retrieval

We evaluated image retrieval performance of our method in **ILSVRC**, which has 1000 different classes. We followed the evaluation protocol of [21]. For each query we run a k-nearest neighbor classifier on the top 1000 results retrieved by each method. As suggested in ILSVRC, we evaluated tested methods with the five best retrieved classes (i.e. recognition rate@5). Specifically we first perform 1000-nearest neighbor search for given a query with binary codes. We then determine the correctness of retrieval results by checking whether five most frequently occurred classes within 1000-nearest neighbor images contain the ground truth class.

Fig. 3.19 shows recognition rates of our method and **RMMH** with **CS** and **HI** kernels. Our method consistently gives better image retrieval performance with both kernels over the other tested methods.

3.4.5 Discussion

SHD (Sec. 3.2.2) drastically improves mAPs in the Euclidean space (Sec. 3.4.2). However, we observed that **SHD** does not provide significant accuracy improvements with the generalized spherical hashing. Table 3.4 shows how much the **SHD** improves mAPs of the generalized spherical hashing over **HD** with two popular kernels, and **SHD** provides 3.5% benefits on the mAPs over the Hamming

1000-NN mAP with GIST-1M-384D

# bits	32	64	128	256	512
RBF-SHD	0.0358	0.0789	0.1336	0.1992	0.2587
RBF-HD	0.0330	0.0725	0.1318	0.1997	0.2575
Linear-SHD	0.0345	0.0671	0.1385	0.1974	0.2424
Linear-HD	0.0325	0.0736	0.1295	0.1958	0.2450

1000-NN mAP with GIST-1M-960D

# bits	32	64	128	256	512
RBF-SHD	0.0316	0.0672	0.1198	0.1757	0.2451
RBF-HD	0.0308	0.0680	0.1220	0.1811	0.2331
Linear-SHD	0.0335	0.0733	0.1270	0.1932	0.2517
Linear-HD	0.0332	0.0682	0.1194	0.1824	0.2321

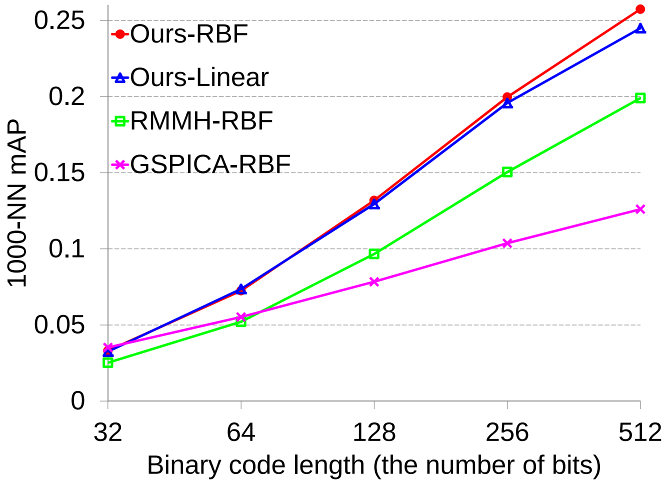
Table 3.4: These two tables show the effect of **SHD** (Sec. 3.2.2) with generalized spherical hashing (Sec. 3.3). **SHD** and **HD** indicate binary code distance metric types. **SHD** improves mAPs of generalized spherical hashing 3.5% on average over **HD**.

distance. The reason why **SHD** shows a relatively small benefit for generalized spherical hashing is that **SHD** does not directly reflect inner product, since it is designed to better reflect the Euclidean distance. Nonetheless, generalized spherical hashing with both of **HD** and **SHD** outperforms state-of-the-art kernelized binary code embedding methods.

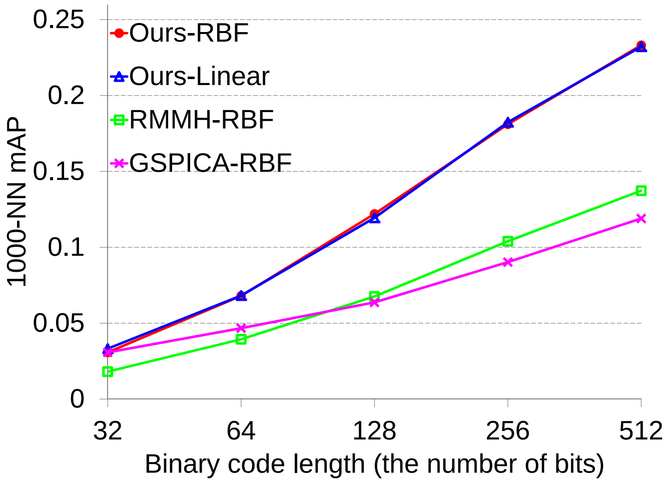
3.5 Conclusion and Future Work

In this work we have proposed a novel hypersphere-based binary embedding technique, spherical hashing, for providing a compact data representation and highly scalable nearest neighbor search with high accuracy. We have found that spherical hashing significantly outperforms the tested six state-of-the-art binary code embedding techniques based on hyperplanes with one and 75 million high-dimensional image descriptors. We have also proposed generalized spherical hashing to support various similarity metrics defined by arbitrary kernel functions, and we have demonstrated on three datasets with four popular kernels that generalized spherical hashing improves the state-of-the-art techniques.

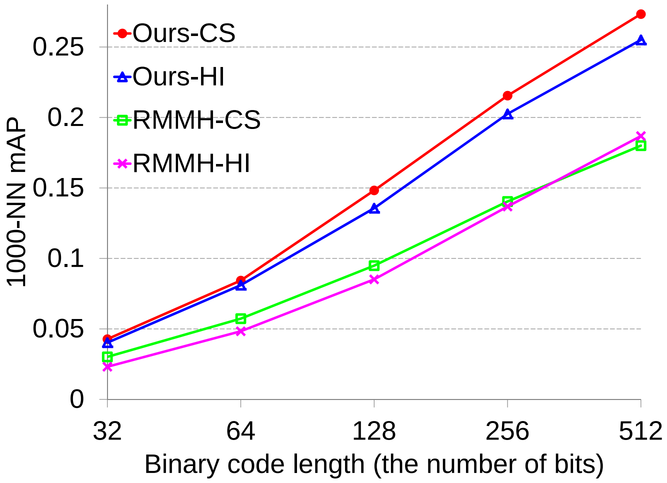
Many interesting future research direction lies ahead. We would like to further improve our spherical Hamming distance such that it shows higher improvement for kernel functions, as we achieved improvement for the Euclidean space. Also, we would like to incorporate the quantization error that is considered in the iterative quantization method [19] into our optimization process. We expect that by doing so, we can improve the search accuracy further. In addition, some of recent techniques [43, 44] use multiple bits per hashing function for achieving higher accuracy. We would like to design a effective scheme that allocates multiple bits for our spherical hashing functions and achieve higher accuracy.



(a) GIST-1M-384D



(b) GIST-1M-960D



(c) ILSVRC

Figure 3.18: k-nearest neighbor search performances on three different datasets when $k = 1000$.

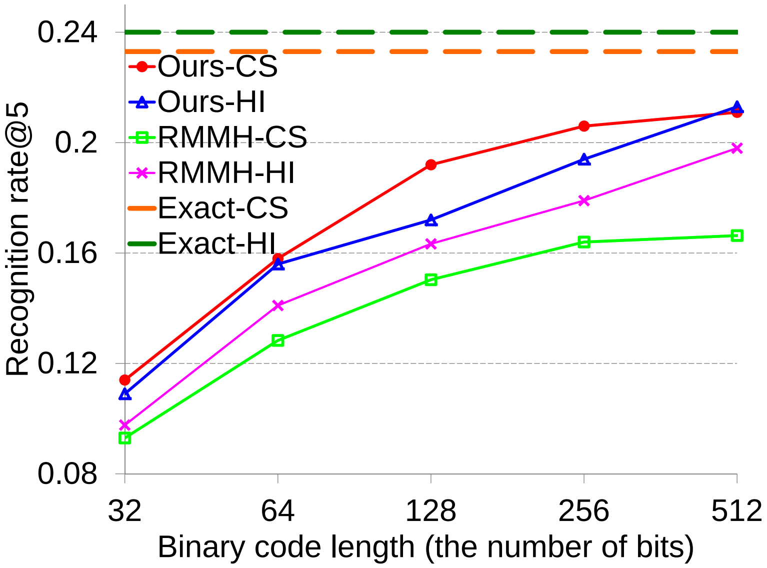


Figure 3.19: Image retrieval performances on **ILSVRC** dataset. Exact-CS and Exact-HI are recognition rates obtained by the exact 1000-nearest neighbor classifier on corresponding kernel functions.

Chapter 4. Distance Encoded Product Quantization

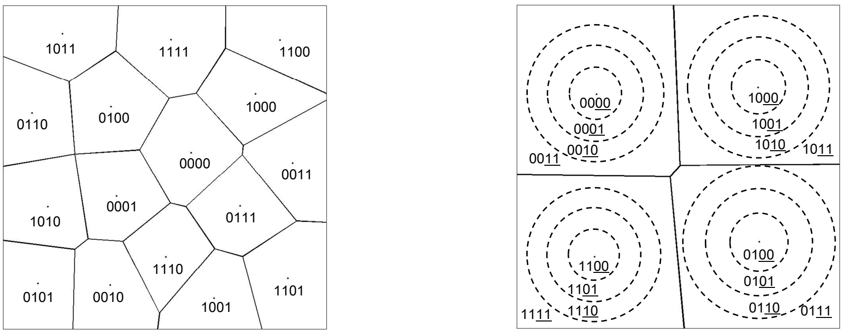
4.1 Overview

Among prior ANN techniques, Product Quantization (PQ) [23] and its recent improvement, Optimized Product Quantization (OPQ) [24], have shown the-state-of-the-art performance. Its high accuracy and efficiency is mainly because (1) quantization in subspaces offers a strong representational power and (2) distances between two data points can be computed via a look-up table. Its input feature space is divided into several disjoint subspaces, and each subspace is partitioned into clusters independently. A data point is then represented by an index of its corresponding cluster in each subspace. The distance between two data points is approximated by the sum of distances of their cluster centers computed in each subspace.

PQ and OPQ can effectively generate many clusters in each subspace and thereby reduce the quantization distortion. Nonetheless, we have found that their approach shows marginal accuracy improvement in practice, as we increase the number of clusters in each subspace. This is mainly because they encode only clusters containing data points, but are not designed to consider how far data points are located away from cluster centers.

Main contributions. To address aforementioned problems, we propose a new binary code encoding scheme, Distance-encoded Product Quantization (DPQ), and two different distance metrics tailored to the scheme. We follow exactly the same procedure as in PQ and OPQ to generate subspaces, quantize them with unsupervised clustering (i.e. k-means), and encode each data point with the index of its nearest cluster center in each subspace. The novelty of our method lies in that in addition to encoding the cluster index, we use additional bits to quantize the distance from the data point to its closest cluster center. Based on the new binary encoding scheme, we propose two distance metrics, statistics and geometry based distance metrics, for symmetric and asymmetric cases. Especially, our geometry based distance metric is based on novel geometric reasoning for high-dimensional data spaces.

We have applied our method to three standard benchmarks consisting of GIST and BoW descriptors. Results show that our encoding scheme with our distance metrics consistently outperforms the existing state of the art methods across tested benchmarks. Specifically, combined with PQ our method improves results of PQ significantly, and combined with OPQ, the improvement is even larger. This indicates that our subspace encoding scheme is more useful, when the subspace is constructed more optimally. These improvements are mainly caused by both quantizing distances of points from cluster centers and well estimated distance metrics. Overall our method is simple, but results in meaningful performance improvement over PQ and OPQ.



(a) PQ

(b) Ours

Figure 4.1: The left and right figures show toy examples of partitioned space and assigned binary codes for PQ and ours, respectively, when using 4 bit binary codes. Our method allocates first two bits for encoding a cluster index and another underlined two bits for quantized distances, while PQ allocates all the bits for encoding a cluster index.

4.2 Background and Motivations

Let us define notations that we will use throughout the paper. We use $X = \{x_1, ..., x_n\}$, $x_i \in \mathbb{R}^D$ to denote a set of n data points in a D-dimensional space, A binary code corresponding to each data point x_i is defined by $b_i = \{0, 1\}^L$, where L is the length of the code. We denote $d(x, y)$ as the Euclidean distance $\parallel x - y \parallel$.

We first briefly review Product Quantization (PQ) [23] that our work is built upon and its two distance measures. Let us denote a point $x \in \mathbb{R}^D$ as the concatenation of M subvectors, $x = [x^1, ..., x^M]$. For simplicity, we assume that the dimensionality of data D is divisible by the number of subspace M. Each i^{th} subspace is encoded by L/M bits and we thus have $k (= 2^{L/M})$ codebook vectors, $\{c^i_1, ..., c^i_k\}$.

A vector quantizer $q^i(x^i)$ given i^{th} subspace is defined as following:

$$q^i(x^i) = \arg\min_{c^i_j} d(x^i, c^i_j).$$

The sub-binary code, b^i, computed from i^{th} subspace elements of x is computed by encoding an codebook index of the nearest cluster:

$$b^i = B\big(\arg\min_j d(x^i, c^i_j), \frac{L}{M}\big),$$

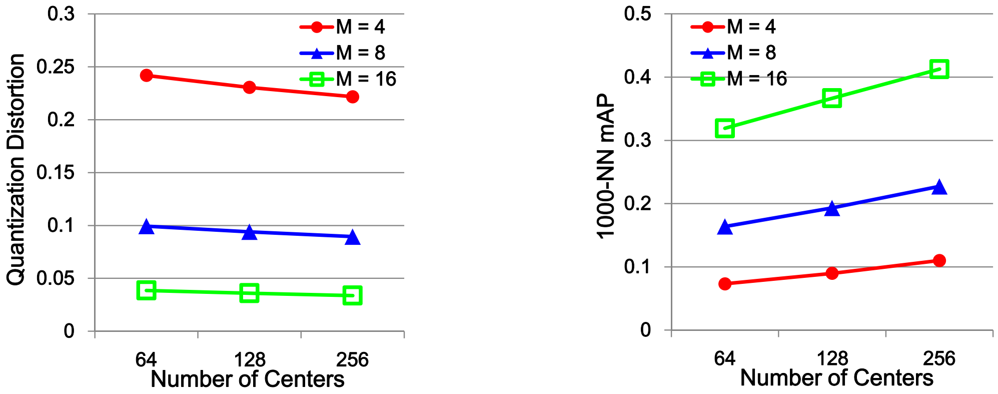
where the function $B(v, l)$ converts an integer $v-1$ to a binary string with a length l; e.g., $B(6, 4) = 0101$. PQ then maps x to the concatenation of sub-binary codes, $b = [b^1, ..., b^M]$.

PQ uses two distance computation schemes: Symmetric Distance (SD) and Asymmetric Distance (AD). SD is used, when both vectors x and y are encoded, and is defined as following:

$$d^{PQ}_{SD}(x, y) = \sqrt{\sum_{i=1}^{M} d\big(q^i(x), q^i(y)\big)^2}. \tag{4.1}$$

On the other hand, AD is used, when only data point x is encoded, but query y is not, and is defined as following:

$$d^{PQ}_{AD}(x, y) = \sqrt{\sum_{i=1}^{M} d\big(q^i(x), y\big)^2}. \tag{4.2}$$



(a) (b)

Figure 4.2: The left figure shows the empirical quantization distortions as a function of the number of clusters in each subspace. M indicates the number of subspaces used. The right figure shows mAP curves of 1000-nearest neighbor search with varying numbers of clusters for each subspace. We use OPQ on the GIST-960D dataset for the experiments.

Recently proposed Optimized PQ (OPQ) [24] has the same underlying scheme as PQ, but operating on a transformed feature space obtained with an optimized projection matrix. OPQ shows the state of the art performance in the field of approximate nearest neighbor search. Cartesian k-means (ck-means) and OPQ are based on a very similar generalization of PQ, as stated in [47]. Our method is independent from clustering and construction methods for subspaces. Our method, therefore, can be built on the subspaces created by ck-means in the same manner to what we do for OPQ. In this paper we explain our concept and its benefits of our method on top of PQ and OPQ for the sake of succinct explanations.

4.2.1 Motivations

Quantization distortion has been identified to be closely related to the search accuracy [24]. OPQ directly aims to reduce the quantization distortion of PQ. In general we can reduce the quantization distortion by allocating longer binary codes, i.e., having more clusters. In particular, we have studied the relationship between the number of clusters and quantization distortion, ξ, which is defined as follows [23, 24]:

$$\xi = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{n}\sum_{j=1}^{n} d\big(x_j^i, q^i(x_j^i)\big)^2.$$

We experimentally measure quantization distortions as a function of the number of clusters (Fig. 4.2(a)). As expected, the quantization distortion reduces as we have more bits. However we observe that the decreasing rate of the quantization distortion is marginal with respect to the number of centers. Similarly we observe the same diminishing return of having more clusters for the search accuracy, as shown in Fig. 4.2(b).

Once a data point is encoded as a compact code, a reconstructed position from the code is set as the center of the corresponding cluster of the code. Distances between encoded binary codes at the search phase are estimated only with such center positions. One can easily see that the error of estimated distances depends on the quantization distortion. Specifically, it has been shown previously that the distance is biased and the error is statistically bounded by two times of the quantization distortion [23].

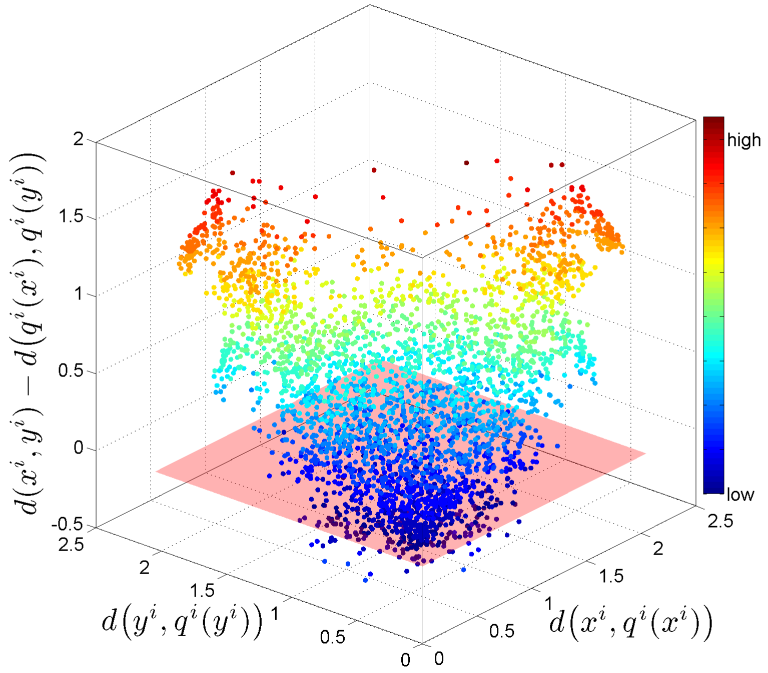


Figure 4.3: This figure visualizes errors of the symmetric distances. We sample two random points, x and y, in a randomly selected subspace. The x-axis indicates the distance between x^i and its corresponding cluster's center $q^i(x^i)$, and y-axis shows similar information for y. The vertical axis is the difference between the actual distance $d(x^i, y^i)$ and its estimated distance $d(q^i(x^i), q^i(y^i))$. The errors of estimated distances tend to be higher as the distance between data points and their corresponding clusters becomes larger. We use OPQ to define subspaces and clusters with the GIST-960D dataset.

It is also observed that error-corrected versions of distance measures can reduce the bias, but increase the distance variance, resulting in even worse search accuracy.

We have empirically studied a functional relationship between the errors of estimated distances and the actual distance of data points from centers of their corresponding clusters (Fig. 4.3). We have found that the estimated distances tend to have higher errors, as data points are further away from centers of their corresponding clusters.

These results suggest that by reducing quantization distortions, we can predict the distances between data points more reliably, i.e. lower variance. Motivated by this, we allocate additional bits to directly encode the distances of data points from their corresponding cluster centers in each subspace, instead of constructing more clusters and encoding data with them.

4.3 Our Approach

In this section we explain our binary code encoding scheme, Distance-encoded Product Quantization (DPQ), and two distance metrics tailored to the scheme.

4.3.1 Our Binary Code Encoding Scheme

Our encoding scheme can be used with any hashing techniques that encode cluster indices in computed binary codes. For simplicity we explain our method by following the PQ framework. Combining our method with OPQ is straightforward, since we only need to apply an estimated rotation projection to the input feature space.

Suppose that the binary code length assigned for encoding the information in each subspace is L/M bits, where L and M indicate the overall binary code length and the number of subspaces, respectively. In each subspace, our method encodes the distance of a data point from the center of its cluster containing the point as well as the index of the cluster. Fig. 4.1 shows a visual example of our encoding method. Specifically, we allocate l_c bits for encoding the cluster index, and l_d bits for the distance from its cluster center. We define $h(= 2^{l_d})$ different distance thresholds, $t^i_{j,1}, ..., t^i_{j,h}$, for c^i_j, the center of the cluster j in i^{th} subspace. The binary code of a data point, x, for the i^{th} subspace is then the concatenation of the nearest center index, $\hat{j}$, and the quantized distance index, $\hat{k}$, as follows:

$$b^i(x^i) = [B(\hat{j}, l_c), B(\hat{k}, l_d)],$$

where

$$\hat{j} = \arg\min_j d(x^i, c^i_j),$$

and $\hat{k}$ is the value satisfying the following:

$$t^i_{\hat{j},\hat{k}-1} \leq d(x^i, c^i_{\hat{j}}) < t^i_{\hat{j},\hat{k}}.$$

$t^i_{j,0}$ and $t^i_{j,h}$ are defined as 0 and ∞, respectively. We also use $P^i_{j,k}$ to denote a set of data points that are encoded by the cluster j with threshold k in i^{th} subspace. We use P^i_j to denote all the data points of the union of $P^i_{j,1}$, ... , $P^i_{j,h}$.

Computing thresholds. In order to choose distance thresholds determining h disjoint regions within each cluster, we identify points $|P^i_j|$ contained in the cluster j in i^{th} subspace. We then construct distances of those points from the cluster center, c^i_j. For choosing thresholds, we first compute h different regions in a way that we minimize the variances of distances of points contained in each region, i.e., minimizing the within-region variance.

It is also important to balance the number of points contained in each region. To achieve this, we enforce the number of points in each $P^i_{j,k}$ to be between $(|P^i_j|/h - |P^i_j|/h^2)$ and $(|P^i_j|/h + |P^i_j|/h^2)$; in this equation we use h^2 to achieve a near-balance among the numbers of points allocated to regions. Each cluster has a small number of points, and the search space of the candidate set for computing thresholds given the balancing criterion are small. As a result, we can efficiently find thresholds that minimize the within-region variance even by exhaustively searching the optimal one. Alternatively, we can also use balanced-clustering techniques such as [61] for accelerating the aforementioned process of computing thresholds.

4.3.2 Distance Metrics

We propose two distance metrics, statistics and geometry based metrics, that consider quantization distortions for achieving higher accuracy.

Statistics based distance metric. One can measure how far data points are located from the center of their corresponding cluster and use that information for improving the quality of estimated distances between quantized points. Jegou et al. [23] have discussed the quantization distortion of each cluster and suggested error corrected versions for Symmetric Distance (SD) and Asymmetric Distance (AD).

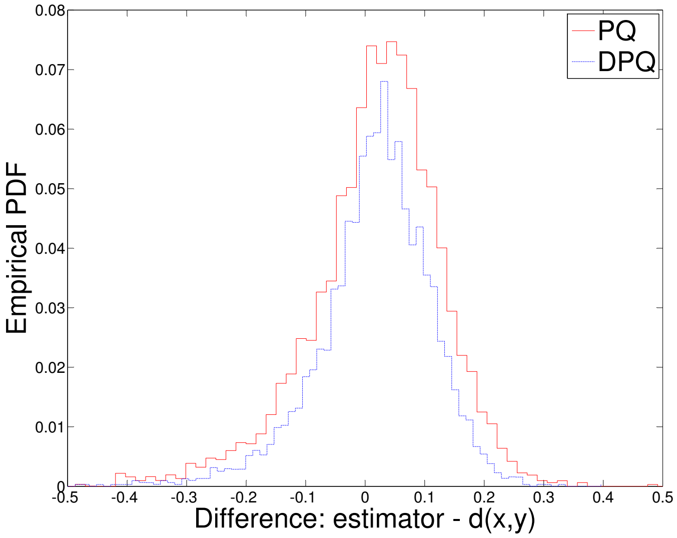


Figure 4.4: This figure shows the distribution of differences from the ground-truth distances to estimated results from statistics based distance metrics used for our method and PQ. We draw 100 K pairs of samples randomly chosen from the GIST-1M-960D dataset. Our method shows the bias of 0.0091, which is reduced from 0.0096 of PQ. Similarly, the variance of our method is 0.0099, while PQ has 0.0133.

For AD, we start with the following distance metric, Error-Corrected AD (ECAD), considering the quantization distortion of x [23] :

$$d_{ECAD}^{PQ}(x,y)^2 = d_{AD}^{PQ}(x,y)^2 + \sum_{i=1}^{M} \xi_j^i(x^i), \tag{4.3}$$

where $\xi_j^i(x^i)$ is a pre-computed error correcting term for the cluster j containing x^i. The error correcting term $\xi_j^i(\cdot)$ is defined as the average distortion of the cluster j in the i^{th} subspace:

$$\xi_j^i(x^i) = \frac{1}{|P_j^i|} \sum_{w=1}^{|P_j^i|} d(p_w, c_j^i)^2,$$

where p_w is w^{th} point of P_j^i. We can similarly define an Error-Corrected distance metric for SD (ECSD) considering quantization distortions of both x and y.

We can easily extend these error-corrected distance metrics to our encoding scheme. For our method we define a new error correcting term, $\xi_{j,k}^i(x^i)$, with $x^i \in P_{j,k}^i$, which contains points in the k^{th} region of the cluster j in the i^{th} subspace:

$$\xi_{j,k}^i(x^i) = \frac{1}{|P_{j,k}^i|} \sum_{w=1}^{|P_{j,k}^i|} d(p_w, c_j^i)^2. \tag{4.4}$$

Interestingly, [23] reported that the error-corrected distance metrics did not improve accuracy over metrics without the error correcting terms, mainly because the error-corrected distance metrics have higher variances. In contrast, our encoding scheme with our error-correcting terms (Eq. 4.4) shows higher accuracy over ours without the terms. In order to identify reasons why the similar error-correcting terms result in contrasting results between our encoding scheme and PQ, we have measured the bias and variance of these two distance estimators. As can be seen in Fig. 4.4, the variance and bias of our

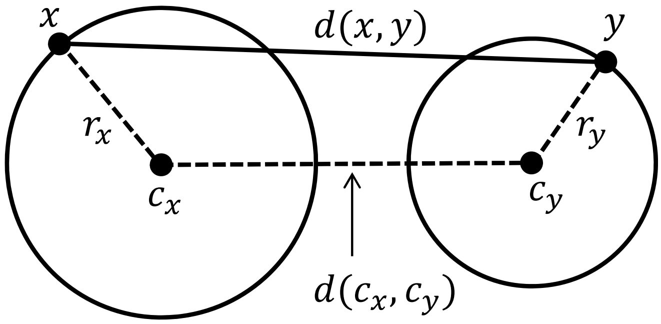


Figure 4.5: This figure shows two points x and y on the hyper-spheres centered at c_x and c_y respectively. r_x and r_y represent the radii of hyper-spheres.

error-corrected distance metric are reduced from those of PQ. Since our encoding scheme quantizes the distance of a data point from its corresponding cluster, our error correcting term (Eq. 4.4) reduces the bias and, more importantly, the variance of the distance estimates effectively.

Geometry based distance metric. We now propose a novel geometric approach to develop distance metrics for our encoding scheme. Suppose that two high dimensional points x and y are randomly chosen on the surfaces of two hyper-spheres centered at c_x and c_y, respectively, with r_x and r_y radii (Fig. 4.5). Given these geometric configurations, the vector $x - y$ is reformulated as:

$$x - y = (c_x - c_y) + (x - c_x) + (c_y - y). \tag{4.5}$$

Our goal is to estimate the length of the vector $x - y$ with available information within our encoding scheme.

As the dimension of data points goes higher, the surface area of the hyper-sphere becomes closer to the length of its equator. One may find this is counter-intuitive, but this has been proved for high dimensional spaces [62]. Given a D dimensional hyper-sphere, a cross section of the hyper-sphere against a horizontal hyperplane is $D - 1$ dimensional hyper-sphere. The length of the cross section is longest in the equator. It then exponentially decreases with a function of $D - 1$ degree, as the cross section gets closer to the north pole. As a result, as we have a higher dimensional space, the length of the equator takes a more dominant factor in the surface area of the hyper-sphere.

Given those x and y points, we rotate our randomly chosen points such that x is located at the north pole. By applying the above theorem, we have a higher probability that another point y is located on the equator of the rotated hyper-sphere, as we have higher dimensional space. As a result, we can conclude that it is highly likely that two vectors $x - c_x$ and $y - c_y$ are orthogonal, when these data points are in a high-dimensional space. Similarly, we can show that these two vectors are also orthogonal to another vector $c_y - c_x$. We have also experimentally checked its validity with a benchmark consisting of 960 dimensional GIST descriptors. For this we have measured the average angle between two randomly chosen points (i.e. 100K pairs) from a random cluster. On average their average angle is $89.81°$ with the standard deviation of $\pm 7.1°$.

Since $c_x - c_y$, $x - c_x$, and $c_y - y$ are highly likely to be mutually orthogonal in a high-dimensional space, the squared magnitude of the vector $x - y$ can be computed as follows:

$$\begin{aligned} \| x - y \|^2 &= \| (c_x - c_y) + (x - c_x) + (c_y - y) \|^2 \\ &\approx \|c_x - c_y\|^2 + \|x - c_x\|^2 + \|c_y - y\|^2 . \end{aligned} \tag{4.6}$$

The first term, $\| c_x - c_y \|^2$, is pre-computed as the distance between different clusters. The second and third terms indicate how far x and y are located from the centers of their corresponding clusters.

In our encoding scheme, the second term can be estimated by using points $p_w \in P^i_{j_x,k_x}$, where j_x and k_x are encoded cluster and threshold indices for x, respectively. Specifically, the second term is estimated as the average distance, $\overline{r^i_{j_x,k_x}}$, from the center $c^i_{j_x}$ to $p_w \in P^i_{j_x,k_x}$:

$$\overline{r^i_{j_x,k_x}} = \frac{1}{|P^i_{j_x,k_x}|} \sum_{w=1}^{|P^i_{j_x,k_x}|} d(p_w, c^i_{j_x}). \tag{4.7}$$

The third term of Eq. 4.6 is estimated in the same manner with points in $P^i_{j_y,k_y}$, where j_y and k_y are chosen cluster and threshold indices for y.

We then formulate our distance metric based on Eq. 4.6 and Eq. 4.7. Our GeoMetry based squared Symmetric Distance (GMSD) between two points x and y is defined as:

$$d^{DPQ}_{SD}(x,y)^2 = \sum_{i=1}^{M} \Big(d\big(q(x^i), q(y^i)\big)^2 + \overline{r^i_{j_x,k_x}}^2 + \overline{r^i_{j_y,k_y}}^2 \Big). \tag{4.8}$$

Our GeoMetry based squared Asymmetric Distance (GMAD) between encoded data x and query y is defined similarly as:

$$d^{DPQ}_{AD}(x,y)^2 = \sum_{i=1}^{M} \Big(d\big(q(x^i), y\big)^2 + \overline{r^i_{j_x,k_x}}^2 \Big). \tag{4.9}$$

Note that $\overline{r^i_{j,k}}$ is precomputed and stored as a lookup table as a function of i, j, and k values.

One may find that our geometry based distance metric using the average distance (Eq. 4.7) of points from their cluster have a similar form to our statistics based distance metric using the error correcting term (Eq. 4.4). One can theoretically show that our statistics based metric generates a distance equal or larger than that of the geometry based metric, and their value difference is the variance of distances between data points and their corresponding clusters. It is hard to theoretically tell which approach is better, but these two different metrics consider different aspects of input data points; the statistics based metric considers the variance of distances, while the geometry based one does not.

Empirically we, however, have observed that the geometry based metric shows better performance, 6%, on average over our statistics based metric (Fig. 4.6). Furthermore, when we integrate our geometry based distance metric within PQ, we have observed that our geometry-based distance metric, PQ w/ GMAD, shows higher accuracy over the statistics based one proposed for PQ, PQ w/ ECAD. These results demonstrate benefits of our geometry-based distance metric.

4.4 Evaluation

In this section we evaluate our method for approximate nearest neighbor search and compare its results to the state-of-the-art techniques.

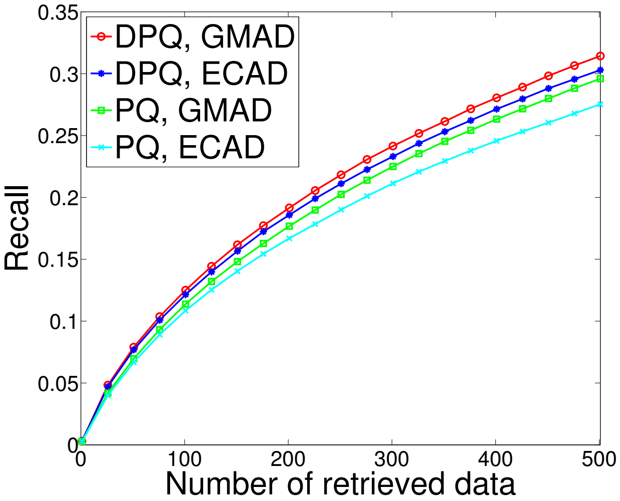


Figure 4.6: This figure shows the recall curves with respect to the number of retrieved data. Our geometry based distance metric, Ours w/ GMAD, improves our statistics based metric, Ours w/ ECAD, and the PQ method, PQ w/ ECAD. The results are obtained with GIST-1M-960D dataset and the number of ground truth K=100. We use 64 bits in this experiment and PQ uses 8 bits for each subspace to define 256 centers. Our method uses 7 bits for the center and 1 bit for the distance quantization.

	Bits		Num. of Subspaces			
	l_c	l_d	2	4	8	16
OPQ baseline	6	0	0.045	0.117	0.238	0.408
Ours+OPQ	6	2	0.101	0.230	0.408	0.584
OPQ baseline	7	0	0.061	0.144	0.276	0.459
Ours+OPQ	7	1	0.106	0.236	0.415	0.595

Table 4.1: This table shows mAPs with different bit budgets. mAPs are measured with the asymmetric distances in **GIST-1M-960D** dataset. 'OPQ baseline' is provided here to see how much our scheme improves the performance by using quantized distances. 8 bit l_c for OPQ is used in other tests.

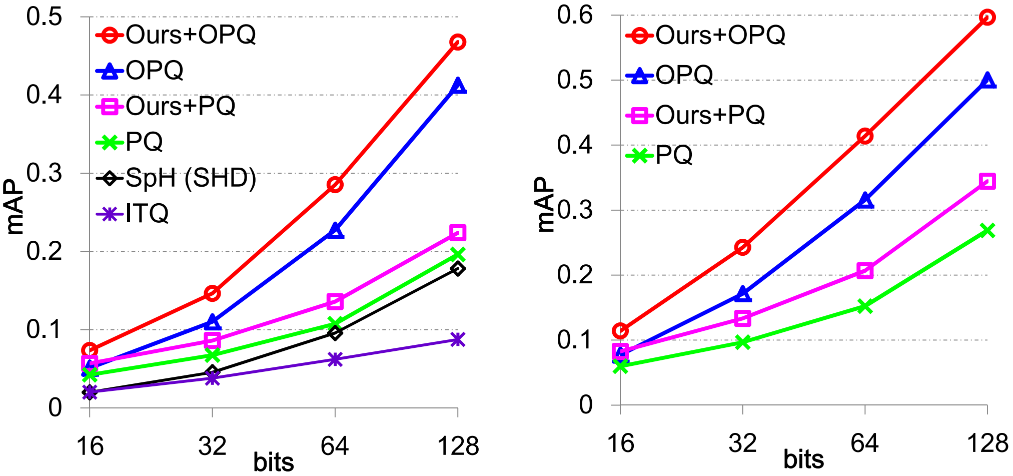
4.4.1 Protocol

We evaluate our method on the following public benchmarks:

- **GIST-1M-960D**: One million 960D GIST descriptors that are also used in [23, 24].
- **GIST-1M-384D**: One million 384D GIST descriptors, a subset of Tiny Images [8].
- **BoW-1M-1024D**: One million 1024D Bag-of-visual-Words (BoW) descriptors, which are subsets of the ImageNet database [57].

For all the experiments, we use 1000 queries that do not have any overlap with the data points. We compute $K = 1000$ nearest neighbors for each query point. Also we compute the ground truth for each query by performing the linear scan. The performance of different methods is measured by the mean Average Precision (mAP). To verify benefits of our method we compare the following methods:

- **PQ:** Product Quantization [23]



(a) Symmetric distance (b) Asymmetric distance

Figure 4.7: Comparisons on **GIST-1M-960D**. The left and right graphs show the mAP curves with symmetric and asymmetric distances, respectively.

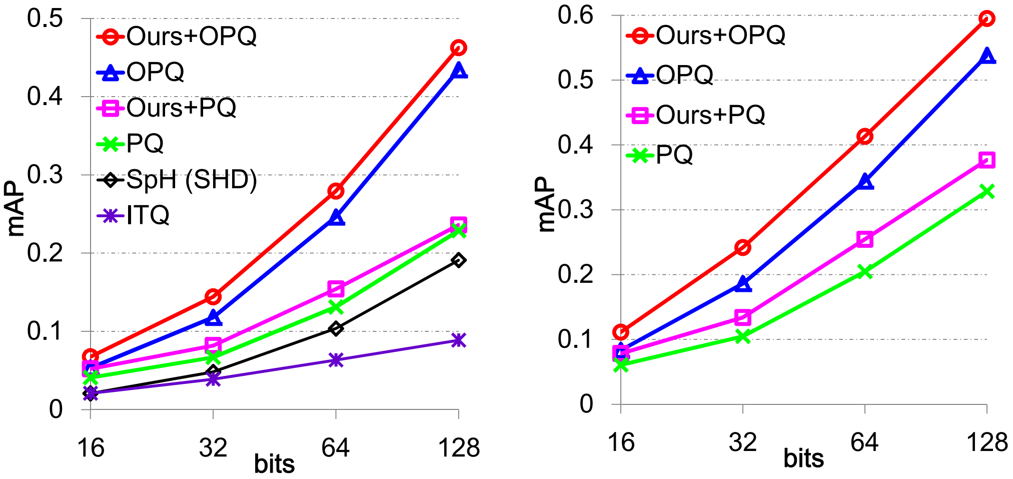
- **OPQ:** Optimized Product Quantization [24]. We use the non-parametric version, because it shows better accuracy than the parametric version.
- **Ours+PQ:** Our method using the same subspaces and clustering method as used for **PQ**, and the geometry based distance metrics.
- **Ours+OPQ:** Our method using the same subspaces, projection matrix, and clustering method, as used for **OPQ**, and the geometry-based distance metric.
- **ITQ**: Iterative Quantization [19]
- **SpH:** Spherical Hashing [63] with Spherical Hamming Distance (SHD)

For all the methods, $100K$ data points randomly sampled from the dataset are used in the training stage, and we allow 100 iterations in the k-means clustering. We assign 8 bits for each subspace as suggested in [23,24]. **PQ** and **OPQ** then have 256 centers in each subspace, and the number of subspace M is $L/8$, where L is the given code-length. We also use public source codes for all the compared methods including **PQ** and **OPQ**. For **PQ** and **OPQ** we use symmetric and asymmetric distances (Eq. 4.1 and Eq. 4.2), which achieved the best accuracy according to their original papers [23,24]. On the other hand, the proposed binary encoding scheme and our geometry based distance metrics (Eq. 4.8 and Eq. 4.9) are used for **Ours+PQ** and **Ours+OPQ**.

In our method, we have parameter l_d (the number of bits) for the distance quantization. We observe that $l_d = 1$ or $l_d = 2$ gives reasonable performance across tested benchmarks. Experimental results with these parameter values are given in Tab. 4.1. Although the performances with $l_d = 1$ and $l_d = 2$ are both similarly good, we pick $l_d = 1$ for all the following tests. In other words, we use 128 centers in each subspace and 2 distance quantizations for each cluster.

4.4.2 Results

Fig. 4.7 and Fig. 4.8 show mAPs of nearest neighbor search for all the tested methods on **GIST-1M-960D** and **GIST-1M-384D** datasets, respectively. **Ours+OPQ** shows better results over all the



(a) Symmetric distance (b) Asymmetric distance

Figure 4.8: Comparisons on **GIST-1M-384D**. The left and right graphs show the mAP curves with symmetric distances and asymmetric distances, respectively.

tested methods across all the tested code lengths ranging from 16 bits to 128 bits. Moreover, our methods **Ours+OPQ** and **Ours+PQ** consistently improve both **PQ** and **OPQ** in all the tests, respectively. In addition, we show mAPs of some of well-known binary embedding techniques, iterative quantization (ITQ) [19] and spherical hashing (SpH) [63] with its distance metric, SHD, for symmetric distance cases. Since these results are lower than PQ, we do not show them in other tests. This improvement clearly demonstrates the merits of our proposed binary code encoding scheme and the new distance metrics.

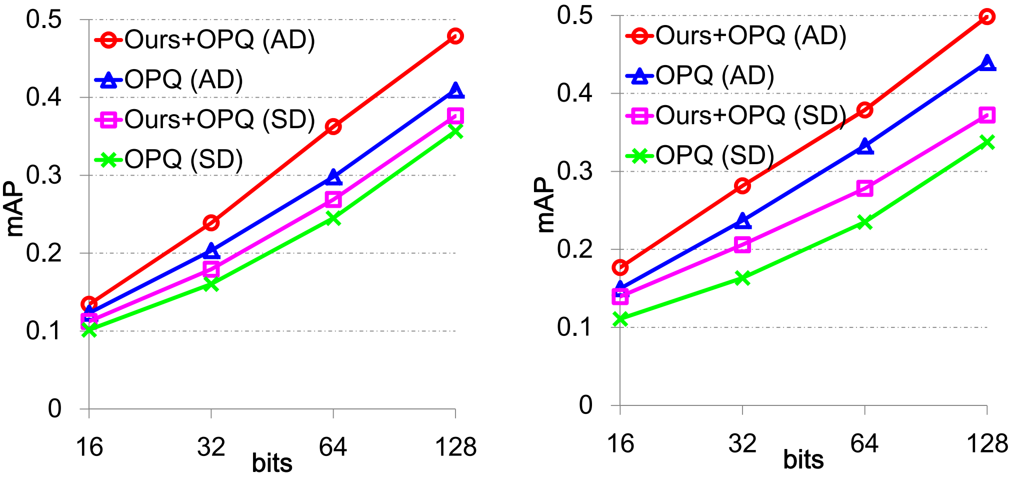
It also shows an interesting trend that in general the relative improvement of our method over the baseline **PQ** and **OPQ** is more significant for the higher dimensional cases, e.g. 960D vs 384D. This is encouraging since many powerful descriptors are of very high dimensions such as spatial pyramids, VLADs, and Fisher vectors. Furthermore, performance improvement for **Ours+OPQ** is in general larger than **Ours+PQ**, which indicates that better subspaces would provide more benefit for our method.

We also perform the same experiments with another popular global image descriptor Bag-of-Words (BoW) on **BoW-1M-1024D**, and its results are shown in Fig. 4.9 with SD and AD cases. Since BoW descriptors are also used with the vector normalization according to L_2 norm, we test the normalized data too. Our method robustly provides better performance compared to **OPQ**.

Finally, we measure the computational times with **GIST-1M-960D** dataset at 64 bits encoding. **Ours+PQ** takes 94.34s to encode one million data, 21ms for one million SD computations, and 819ms for one million AD computations, while **PQ** takes 187.45s, 21ms, and 829ms, respectively. Since our method uses less number of cluster centers, it has a faster encoding performance. Also, distance computation time of our methods are similar to that of PQ.

4.5 Conclusion

We have presented a novel compact code encoding scheme that quantizes distances of points from their corresponding cluster centers in each subspace. In addition, we have proposed two different distance metrics tailored for our encoding scheme: statistics and geometry based metrics. We have chosen the geometry based metrics, since it consistently show better accuracy over the statistics based one. We have tested our method against the-state-of-the-art techniques with three well known benchmark, and



(a) Original data

(b) Normalized data

Figure 4.9: Comparisons on **BoW-1M-1024D**. The left and right graphs show results for original and L_2 normalized data with symmetric and asymmetric distance metrics.

have demonstrated benefits of our method over them.

Many interesting research directions lie ahead. Our current encoding scheme uses a fixed bit length for quantizing distances from all the clusters. A clear improvement would be to use a variable bit length for different clusters depending on quantization distortions of them. The key challenge is to design an effective optimization for deciding the bit distributions between encoding clusters and quantizing distances. We leave it as future work.

Chapter 5. Accurate Shortlist Computation in Inverted File

5.1 Overview

Continuously growing image databases enable many promising computer vision applications including web-scale image retrieval. However, traditional data structures and search mechanisms do not provide sufficient scalability in terms of both the size and dimensionality of data. Thus, recent research directions are shifting to develop scalable indexing and search methods mostly based on inverted file structures and compact data representations [23].

In this chapter, we propose a novel searching mechanism for retrieving the inverted file structures. Specifically, we derive a distance estimator designed for high-dimensional spaces and propose a shortlist retrieving scheme according to the estimated distance.

5.2 Background

In this section, we describe a baseline procedure of indexing and search using an inverted file structures with a PQ-like encoding method.

Let us first define notations. L_i is i^{th} inverted list ($i = 1...k$) and $c_i \in \mathbb{R}^D$ is i^{th} center of a coarse quantizer corresponding to L_i, where D is the dimensionality of the image descriptors. $d(x, y)$ is the Euclidean distance between x and y.

We also define a coarse quantizer $q(x)$ as the index of the closet center from x:

$$q(x) = \arg\min_{i} d(c_i, x)$$

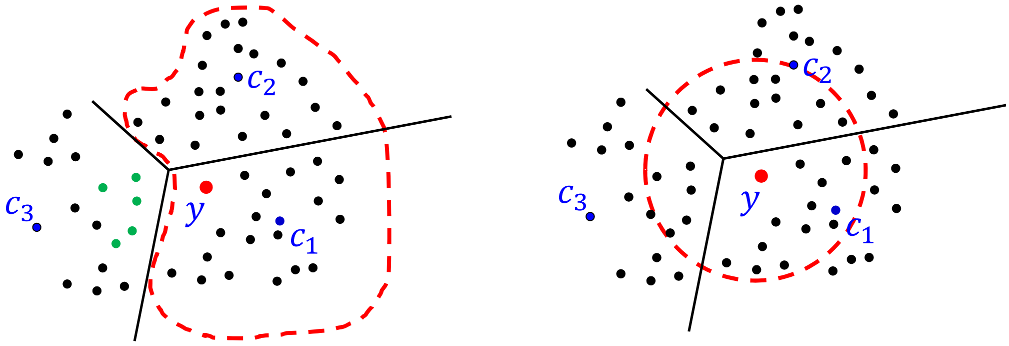
, and the closet coarse quantizer center $c(x)$ as following:

$$c(x) = \arg\min_{c_i} d(c_i, x).$$

When storing an image to the inverted file with its image descriptor x, we firstly quantize x to determine a inverted list where the image will be appended. We then append the image index to the determined inverted list $L_{q(x)}$ with a compact code of x computed by PQ-like encoding method.

Recently many compact code encoding methods have been actively studied, since they provide both a high compression ratio and fast distance estimation. There are two different categories in the encoding methods: hashing and Product Quantization (PQ) based methods. We will rather focus on the PQ based approaches since the proposed research is more related to PQ than hashing.

PQ [23] decomposes the given data space into lower-dimensional subspaces and quantizes each subspace separately using k-means clustering. The compact code is then computed as a concatenation of cluster indices encoded in subspaces. Optimized PQ [24] and Cartesian k-means [47] have been also proposed to optimize PQ by minimizing quantization distortions with respect to the space decomposition and cluster centers. Lately, DPQ [64] has been proposed to distribute memory budget to encode both cluster indices and distance from data to corresponding centers. The inverted file structure has been



(a) (b)

Figure 5.1: These two figures illustrate examples of shortlists for given query y. The shortlists consist of points inside the red-dotted shape. The left figure shows a limitation of the shortlist construction method described in Sec. 5.2. The green points are close to the query y but they are missed in the shortlist since c_3 is farther than c_1 and c_2 from y. In the other hand, the shortlist in the right figure is more accurately constructed. Our goal is to develop a method to construct accurate shortlist as the right figure.

applied to the PQ framework to avoid expensive exhaustive distance estimations [23]. Specifically, in order to prevent linear scan on whole database coarse quantization is performed with given a query to compute candidate search results called shortlist which is a small fraction of the whole data.

5.3 Our Approach

5.3.1 Motivation

Given a shortlist size T, items for the shortlist are determined with the coarse quantizer as described in Sec. 5.2. An assumption under the procedure is that the distance between a data point x and a query y is estimated by the distance between the coarse quantizer center $c(x)$ and y as following:

$$\tilde{d}(y, x) = d(y, c(x)). \tag{5.1}$$

However, this conventional shortlist construction scheme has a limitation that some close points to the query can be missed in the shortlist (Fig. 5.2(a)). This is mainly because the distance estimation by Eq. 5.1 is inaccurate and causes high quantization error. In this research, our goal is to propose a method to construct a shotlist more accurately as illustrated in Fig. 5.2(b).

Both search accuracy and time is depending on the quality of a shortlist. Given fixed size T the search accuracy is higher if we have a better shotlist. It also implies that we can decrease the shotlist size for the same accuracy and thus the search time can be accelerated by reducing the number of distance estimations.

5.3.2 Distance Estimator for High-Dimensional Space

As the dimensionality goes higher, two random vectors are more likely to be orthogonal [64, 65]. We start to develop our distance estimator with the orthogonality between two random vectors in high-dimensional spaces. Given a query y and a data point x, we assume that $y - c(x)$ and $x - c(x)$ are orthogonal. Under this assumption, we formulate our squared distance estimator between y and x as following:

$$\hat{d}(y, x)^2 = d(y, c(x))^2 + d(x, c(x))^2. \tag{5.2}$$

We propose to use our distance estimator (Eq. 5.2) instead of conventional one (Eq. 5.1) to construct more accurate shortlist.

5.3.3 Shortlist Construction

Precomputation

In order to construct shortlist according to our distance estimator, the distances from data to their corresponding center $d(x, c(x))$ are required. However, storing such distances takes additional memory overhead. Moreover, ordering the points according to the estimated distance also requires additional computations. To overcome both memory and computational cost problems, we propose an efficient shortlist construction scheme.

We first sort data in each inverted list L_i according to the squared distances $d(x, c(x))^2$. We compute R_0 and R_{max} as the minimum and maximum squared distance to the center respectively:

$$R_0 = \min d(x, c(x))^2 \qquad \text{and} \qquad R_{max} = \max d(x, c(x))^2$$

The range $[R_0, R_{max}]$ is uniformly divided into fixed sized (ΔR) intervals. We denote the boundary values R_i as following:

$$R_i = R_0 + i\Delta R$$

We define a lookup table W to quickly refer the number of data points in L_i whose squared distance to the center $d(x, c_i)^2$ is less than R_j as following:

$$W(i, j) = \text{num}(\ \{x \mid d(x, c_i)^2 < R_j, x \in L_i\ \}\ )$$

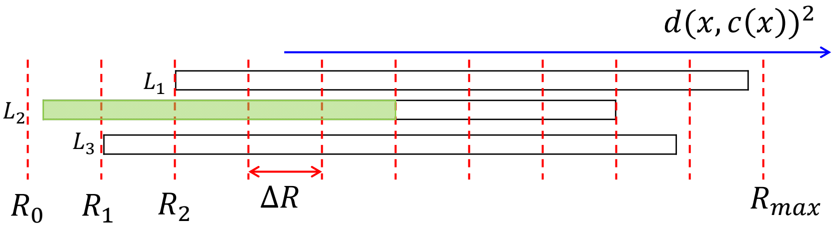
Fig. 5.2 provides an example of the above procedure to construct a lookup table W and refer a value in the W.

Runtime Algorithm

The precomputed lookup table W is based on $d(x, c(x))^2$ but we need also consider $d(y, c(x))^2$ which is depending on the query y and computable in the runtime. In order to compute the number of data points $w(i, t)$ in L_i whose estimated distance (by Eq. 5.2) to the query is less than a value t, we refer the table W with shifted index as following:

$$\begin{aligned} w(i, t) &= \text{num}(\ \{x \mid d(x, c_i)^2 + d(y, c_i)^2 < t, x \in L_i\ \}) \\ &= \text{num}(\ \{x \mid d(x, c_i)^2 < t - d(y, c_i)^2, x \in L_i\ \}) \\ &= W(i, \left\lceil \frac{t - d(y, c_i)^2 - R_0}{\Delta R} \right\rceil) \end{aligned}$$

When query y is given, we firstly compute and store $d(y, c_i)^2$ for all the c_i. Since each row in the table W is increasing order, the column-wise sum is also increasing order. We use binary search to find appropriate threshold of estimated distance t that meets a given shortlist size T. The binary search for



(a)

L_1	0	0	10	15	22	30	40	40	50	55
L_2	5	10	15	15	20	28	30	35	35	35
L_3	0	4	10	15	20	27	33	28	30	30

(b)

Figure 5.2: These two figures show an example of the procedure described in Sec. 5.3.3 (a) The data points in each list L_i is sorted according to the squared distance to the center $d(x, c_i)^2$, and range of the squared distances $[R_0, R_{max}]$ is uniformly divided with fixed interval length ΔR. (b) A lookup table W is defined to $W(i, j)$ be the number of data points in L_i whose squared distance to the center $d(x, c_i)^2$ is less than R_j. For instance, the number of data points within the green box in (a) is $W(2, 5) = 20$ in the lookup table.

t is performed within the range $[\min d(y, c_i)^2 + R_0, \max d(y, c_i)^2 + R_{max}]$, and stopped when the lower bound that satisfies following:

$$T \leq \sum_{i=1}^{k} w(i, t)$$

The resulting shotlist is constructed by collecting points that has smaller estimated distance than the threshold found by the binary search.

Parameter Selection

Since we aim to avoid additional computational overhead to construct a shortlist, we suggest a parameter selection scheme based on the cost analysis. The computational cost C_{search} for the search based on the procedure described in Sec. 5.2 is formulated as following:

$$C = C_S + C_R$$

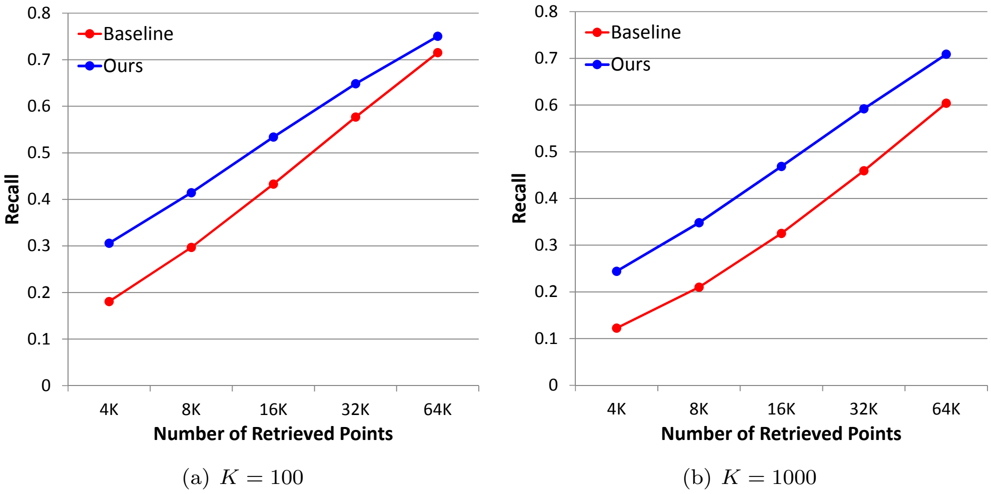
, where C_S and C_R are the costs for computing shortlist and for rerank the shortlist respectively. In order to identify the near coarse quantizer centers computing distances to the centers is required, so the first part of the cost equation is:

$$C_S = kD + k \log k$$

and the cost for the reranking phase is:

$$C_R = MT$$

, where M is an unit cost for one distance estimation. The prior method and our proposed method have same complexity on reranking phase. Our proposed shortlist construction method desribed in Sec. 5.3.3



(a) $K = 100$ (b) $K = 1000$

Figure 5.3: These two figure show the experimental results with different K which is number of defined true neighbors. x-axis and y-axis represent the shortlist size (T) and Recall@T respectively.

runs with following time complexity:

$$C_{S'} = kD + k \log \frac{R_{max} - R_0}{\Delta R}$$

In order to ensure $C_S = C_{S'}$, we can set the parameter ΔR to satisfy

$$\frac{R_{max} - R_0}{\Delta R} = k.$$

5.4 Evaluation

Dataset: We collect about 11 millions of Internet images from Flickr, and extract 4096 dimensional image descriptors by using a deep convolutional neural network. As a training set for the neural network, we used 1 million of images from ImageNet within 1000 categories.

Protocol: We use 1000 queries do not have any overlap with the data points. K nearest neighbors for each query points are computed and defined to be the ground truth via the exhaustive search. The accuracy of retrieved shortlist is measured by recall@T where T is the size of the shotlist. Recall@T means how many true neighbors are in the retrieved shortlist. We use the $k = 4096$ inverted lists with 4096 coarse quantizer centers.

Results: We compare our method (**Ours**) against the conventional method (**Baseline**) described in Sec. 5.2. The experimental results are reported in the Fig. 5.3. Our proposed method **Ours** consistently provides more accurate shortlist compared to the **Baseline** in all the tested configurations. **Ours** and **Baseline** take 6.5ms and 6.3ms to retrieve the shortlists in average respectively.

Chapter 6. Large-Scale Tag-Driven Feature Learning

6.1 Overview

The deep convolutional neural network trained on the ImageNet dataset [25] shows the state-of-the-art performance in the image classification task. The image features computed in the intermediate layers in the deep convolutional neural network can be used in many computer vision applications including visual search and tagging. Specifically, the vectors in the fully connected layers in the network is known as one of the best image features.

However, the neural network trained on the ImageNet dataset has a few disadvantages. First, most of images in the ImageNet dataset contain a single object and are strictly assigned to a single object label. This can bring training bias to the pre-defined fixed number of object categories. And the trained network is weak at images containing drawings, graphics, or illustrations since most images in the training set are real photos.

In this chapter, we propose a new scheme to train a neural network with large-scale image dataset that are weakly annotated with multiple text tags. Since the number of tags is usually very large (*i.e.* a few millions of keywords), it is not feasible to directly train a classification network with the tags. So we suggest a novel way to utilize such valuable tag information to train a neural network.

6.2 Tag-Driven Pseudo Classes

Suppose that we have images $I_1, I_2, ..., I_n \in \mathbb{I}$ and each image I_i has m_i tags $T^i = \{t^i_1, t^i_2, ..., t^i_{m_i}\}$. Let us define a set of all possible tags as $U = \{u_1, u_2, ..., u_M\}$. We represent tags of each image I_i to a M dimensional binary vector b^i as followings:

$$b^i_j = \begin{cases} 1 & \text{when } u_j \in T^i \\ 0 & \text{otherwise} \end{cases}$$

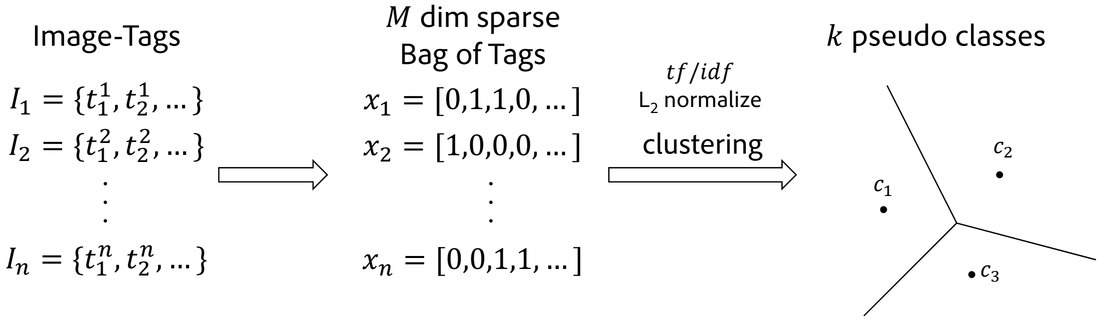


Figure 6.1: This figure shows our framework to define pseudo classes.

	10 tags		25 tags		50 tags		100 tags	
	P@10	R@10	P@25	R@25	P@50	R@50	P@100	R@100
ImageNet	0.530	0.161	0.403	0.295	0.297	0.424	0.196	0.552
Ours	0.591	0.181	0.454	0.333	0.337	0.480	0.220	0.616

Table 6.1: This table shows experimental results of image tagging task. 25 nearest neighbors are considered in the voting scheme that determines tags of given query.

The j^{th} entry of the vector b_i is 1 only when the image I_i has a tag u_j. We then apply normalization terms to all vectors b^i with consideration of tf (term frequency), *idf* (inverse document frequent), and their L_2 norms. The idf_i of each u_i is defined as followings:

$$idf_i = \log \frac{n}{\text{occurrence of } u_i}$$

We use the term frequency to normalize varying number of number of tags for each images, and we use the inverse document frequency to assign higher importance to more informative tags since less common tags are more informative. For example, 'Eiffel Tower' is more informative keyword than 'architecture'. We finally define a Bag of Tags (BoT) representation of each image as followings:

$$x_j^i = \begin{cases} \dfrac{idf_j}{m_i} & \text{when } u_j \in T^i \\ 0 & \text{otherwise} \end{cases}$$

We then normalize x^i according to their L_2 norm $\parallel x^i \parallel$, and perform k-means clustering to x^i vectors. In the clustering stage, we use an inverted representation for x^i since they are very sparse vectors. For example, the number of total tags M is usually very large (*e.g.* more than 100K) and the number of tags m_i of each image I_i is not larger than a hundred.

We call each cluster as *pseudo class*, and let $c_i \in \mathbb{R}^M$ for the centroid of the i^{th} class. The pseudo class is not exist in real, but weighted combination of tags. The overall procedure of computing pseudo classes is illustrated in Fig. 6.1. Also, we provide Fig. 6.2 that shows examples of clustering results that confirm the visual coherency within each pseudo class.

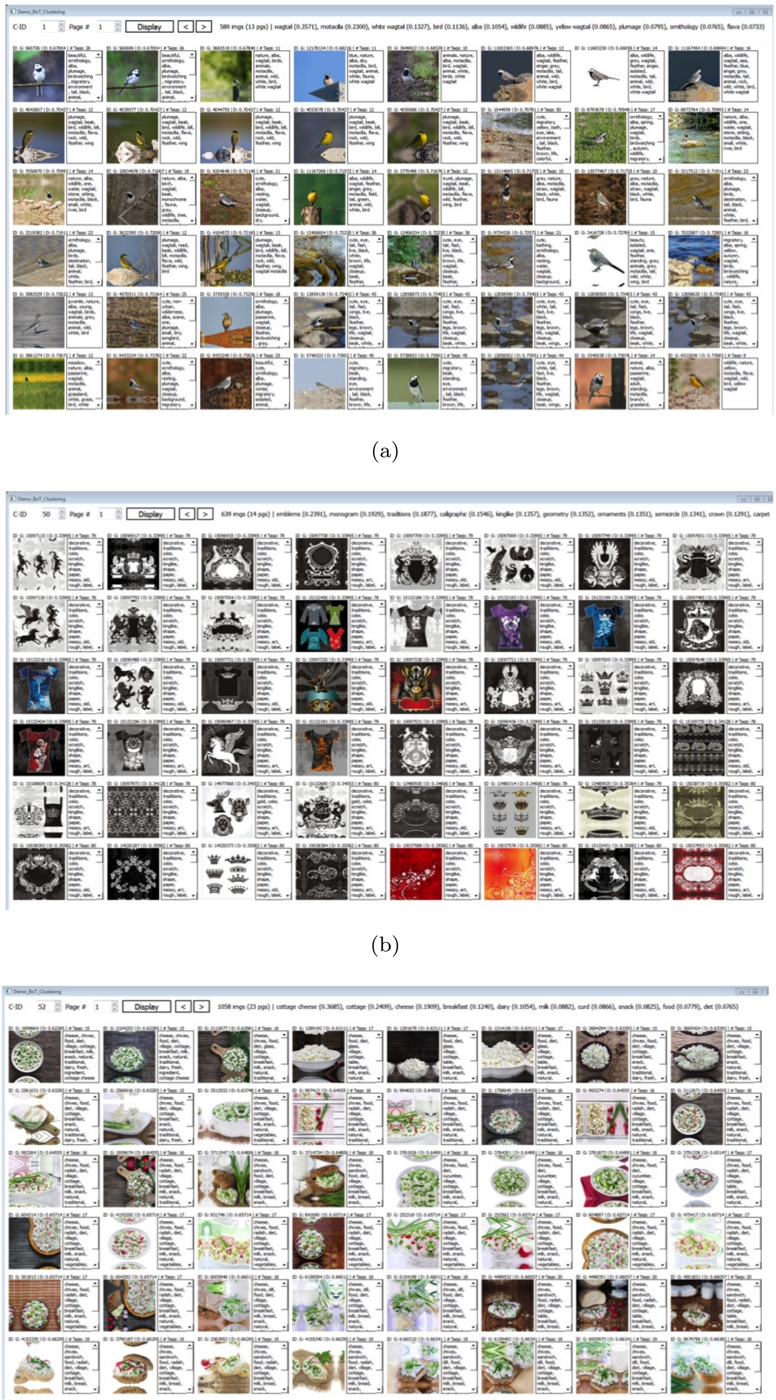
Once we have training images and their pseudo class indices, we train a classification neural network with image and label pairs.

6.3 Evaluation

We evaluate our proposed method against one of the state-of-the-art method **ImageNet** [25]. We use 12.5 million of tagged images as the dataset that do not have overlap with training set. Our method is trained with 3500 pseudo classes and 3.5 millions of images. And we use the last fully connected layers (fc7) of the trained networks for all tested tasks.

We quantitatively evaluate our method with the image tagging task. 10K randomly selected queries are performed, and performances are evaluated with precision and recall. To obtain the tags of given query, we use a simple k-NN voting scheme. The experimental results are provided in Table. 6.1

We also qualitatively evaluate our method with the image search task. Our method provided more visually relevant and consistent results compared to **ImageNet**. Some example results are provided in Fig. 6.3.



C-ID 1 Page # 1 580 imgs (13 pgs) | wagtail (0.3571), motacilla (0.2300), white wagtail (0.1327), bird (0.1136), alba (0.1054), wildlife (0.0885), yellow wagtail (0.0865), plumage (0.0795), ornithology (0.0765), flava (0.0733)

(a)

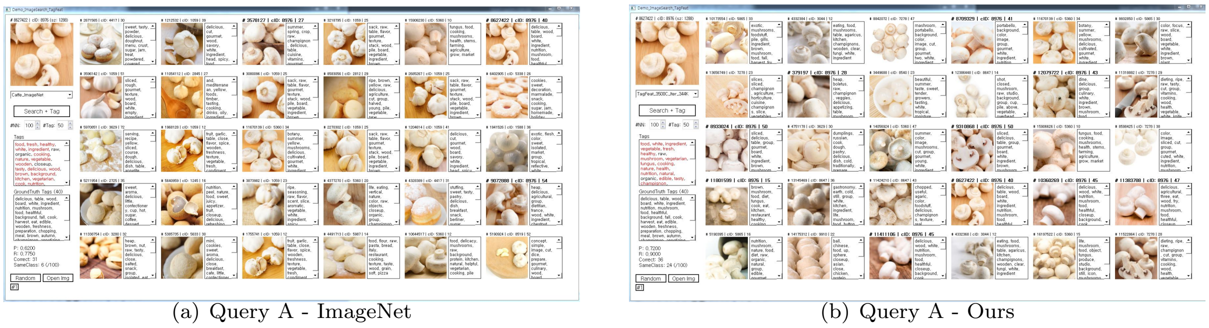
C-ID 50 Page # 1 639 imgs (14 pgs) | emblems (0.2391), monogram (0.1929), traditions (0.1877), calligraphic (0.1546), kinglike (0.1357), geometry (0.1352), ornaments (0.1351), semicircle (0.1341), crown (0.1291), carpet

(b)

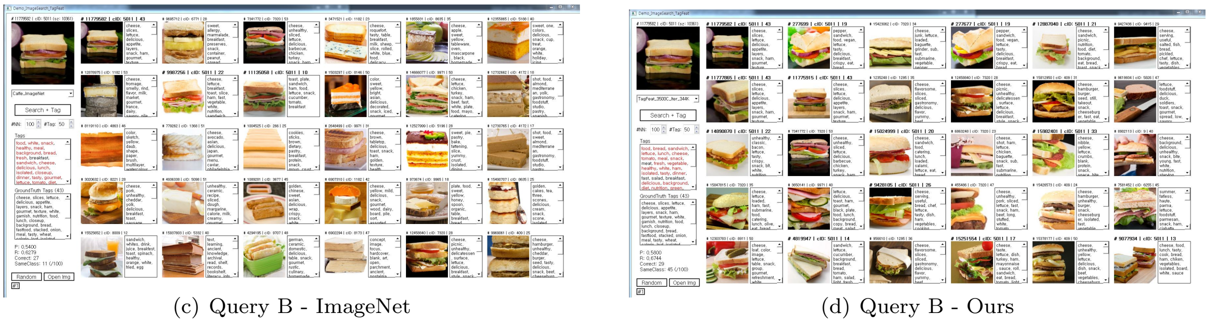
C-ID 52 Page # 1 1058 imgs (23 pgs) | cottage cheese (0.3685), cottage (0.2409), cheese (0.1909), breakfast (0.1240), dairy (0.1054), milk (0.0882), curd (0.0866), snack (0.0825), food (0.0779), diet (0.0765)

(c)

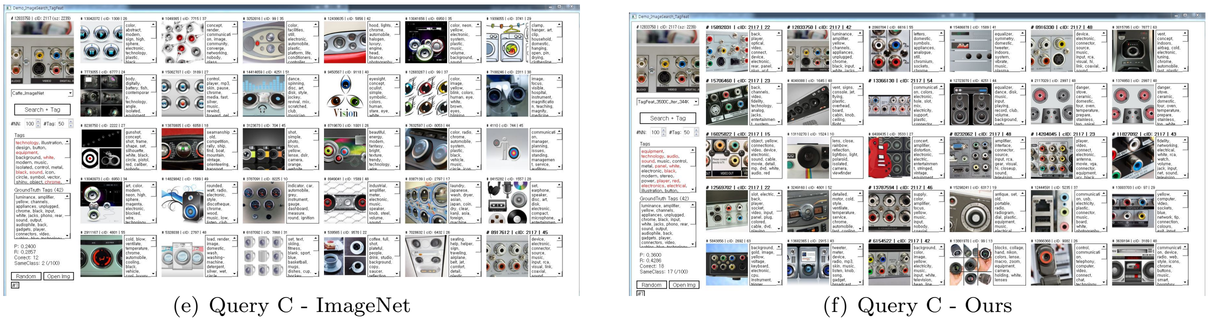
Figure 6.2: These figures show examples of three pseudo classes that confirm the visual coherency within each pseudo classes



(a) Query A - ImageNet (b) Query A - Ours



(c) Query B - ImageNet (d) Query B - Ours



(e) Query C - ImageNet (f) Query C - Ours

Figure 6.3: These figures show that three image search results of our method and the compared method. The leftmost picture of each sub-figure is the provided query image.

6.4 Regression Network

In the classification network that we discussed above, we assign the index of the closest pseudo class to each image as its label. However we can define a multi-dimensional label for each image and train a regression network based on such multi-dimensional labels. We can compute a multi-dimensional labels based on the distances from the Bag of Tags vector x^i of each image to the pseudo class centroid c_j:

$$s_j^i = f(\| x^i - c_j \|)$$

where $f(\cdot)$ is a numerical function that adjusts scales. One example of the function is $f(x) = e^{-\gamma x^2}$.

Chapter 7. Conclusion

This dissertation addresses scalable approximate nearest neighbor search problem for high-dimensional data. Traditional techniques such as kd-trees are not scalable to large-scale high-dimensional data. To overcome the difficulty, two compact representations of high-dimensional data and search scheme based on the compact codes are proposed in this dissertation: 1) Spherical Hashing (Chapter. 3) and 2) Distance Encoded Product Quantization (Chapter. 4). In the Spherical Hashing, a novel hypersphere-based hashing functions are proposed to map more spatially coherent data into a binary code compared to hyperplane-based methods. We also propose a new binary code distance function tailored for our hypersphere-based binary code encoding scheme, and an efficient iterative optimization process to achieve both balanced partitioning for each hashing function and independence between hashing functions. Furthermore, we generalize the Spherical Hashing to support various similarity measures define by kernel functions. We also propose a novel compact code encoding scheme that distributes the available bit budget to encode both the cluster index and the quantized distance between point and its cluster center in the Distance Encoded Product Quantization (DPQ). We also propose two different distance metrics tailored to the DPQ. All the proposed schemes are extensively evaluated against the state-of-the-art techniques with various large-scale benchmarks consisting of high-dimensional image descriptors. The proposed methods significantly outperform the state-of-the-art techniques.

References

[1] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, vol. 18, pp. 509–517, September 1975.

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

[3] A. Guttman, “R-trees: a dynamic index structure for spatial searching,” *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, Jun. 1984.

[4] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *VISAPP*, 2009, pp. 331–340.

[5] P. Indyk and R. Motwani, “Approximate nearest neighbors: toward removing the curse of dimensionality,” in *STOC*, 1998.

[6] M. S. Charikar, “Similarity estimation techniques from rounding algorithms,” in *STOC*, 2002.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *SoCG*, 2004.

[8] A. Torralba, R. Fergus, and Y. Weiss, “Small codes and large image databases for recognition,” in *CVPR*, 2008.

[9] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in *NIPS*, 2008.

[10] O. Chum, J. Philbin, and A. Zisserman, “Near duplicate image detection: min-hash and tf-idf weighting,” in *BMVC*, 2008.

[11] P. Jain, B. Kulis, and K. Grauman, “Fast image search for learned metrics,” in *CVPR*, 2008.

[12] M. Raginsky and S. Lazebnik, “Locality sensitive binary codes from shift-invariant kernels,” in *NIPS*, 2009.

[13] R. Salakhutdinov and G. Hinton, “Semantic hashing,” *International Journal of Approximate Reasoning*, 2009.

[14] J. Wang, S. Kumar, and S.-F. Chang, “Sequential projection learning for hashing with compact codes,” in *ICML*, 2010.

[15] L. Pauleve, H. Jégou, and L. Amsaleg, “Locality sensitive hashing: a comparison of hash function types and querying mechanisms,” *Pattern Recognition Letters*, 2010.

[16] J. Wang, S. Kumar, and S.-F. Chang, “Semi-supervised hashing for scalable image retrieval,” in *CVPR*, 2010.

[17] R.-S. Lin, D. Ross, and J. Yangik, “Spec hashing: Similarity preserving algorithm for entropy-based coding,” in *CVPR*, 2010.

[18] M. Norouzi and D. J. Fleet, “Minimal loss hashing for compact binary codes,” in *ICML*, 2011.

[19] Y. Gong and S. Lazebnik, “Iterative quantization: a procrustean approach to learning binary codes,” in *CVPR*, 2011.

[20] J. He, R. Radhakrishnan, S.-F. Chang, and C. Bauer, “Compact hashing with joint optimization of search accuracy and time,” in *CVPR*, 2011.

[21] A. Joly and O. Buisson, “Random maximum margin hashing,” in *CVPR*, 2011.

[22] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, “Supervised hashing with kernels,” in *CVPR*, 2012.

[23] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE TPAMI*, 2011.

[24] T. Ge, K. He, Q. Ke, and J. Sun, “Optimized product quantization for approximate nearest neighbor search,” in *CVPR*, 2013.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *NIPS*, 2012.

[26] R. Datta, D. Joshi, J. Li, and J. Z. Wang, “Image retrieval: Ideas, influences, and trends of the new age,” *ACM Computing Survey*, vol. 40, no. 2, pp. 1–60, 2008.

[27] J. Sivic and A. Zisserman, “Video google: A text retrieval approach to object matching in videos,” in *ICCV*, 2003.

[28] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *IJCV*, vol. 60, no. 2, pp. 91–110, 2004.

[29] A. Oliva and A. Torralba, “Modeling the shape of the scene: a holistic representation of the spatial envelope,” *IJCV*, 2001.

[30] B. Kulis and K. Grauman, “Kernelized locality-sensitive hashing for scalable image search,” in *ICCV*, 2009.

[31] T. cker Chiueh, “Content-based image indexing,” in *In Proceedings of the 20th VLDB Conference*, 1994, pp. 582–593.

[32] C. Böhm, S. Berchtold, and D. A. Keim, “Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases,” *ACM Comput. Surv.*, vol. 33, no. 3, pp. 322–373, Sep. 2001.

[33] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín, “Searching in metric spaces,” *ACM Comput. Surv.*, vol. 33, no. 3, pp. 273–321, Sep. 2001.

[34] C. Silpa-Anan, R. Hartley, S. Machines, and A. Canberra, “Optimised kd-trees for fast image descriptor matching,” in *CVPR*, 2008.

[35] Y. Jia, J. Wang, G. Zeng, H. Zha, and X.-S. Hua, “Optimizing kd-trees for scalable visual descriptor indexing,” in *CVPR*, 2010.

[36] K. Kim, M. K. Hasan, J.-P. Heo, Y.-W. Tai, and S.-E. Yoon, "Probabilistic cost model for nearest neighbor search in image retrieval," *Computer Vision and Image Understanding (CVIU)*, 2012.

[37] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *CVPR*, 2006.

[38] A. Joly and O. Buisson, "A posteriori multi-probe locality sensitive hashing," in *Proceedings of the 16th ACM international conference on Multimedia*, ser. MM '08. New York, NY, USA: ACM, 2008, pp. 209–218.

[39] J. Bourgain, "On lipschitz embeddings of finite metric spaces in hilbert space," *Israel Journal of Mathematics*, vol. 52, no. 1, pp. 46–52, Mar. 1985.

[40] W. Johnson and J. Lindernstrauss, "Extensions of lipschitz mapping into hilbert space," *Contemporary Mathematics*, vol. 26, pp. 189–206, 1984.

[41] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *ICML*, 2011.

[42] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua, "Ldahash: Improved matching with smaller descriptors," *PAMI*, 2010.

[43] W. Kong and W.-J. Li, "Double-bit quantization for hashing," in *AAAI*, 2012.

[44] Y. Lee, J.-P. Heo, and S.-E. Yoon, "Quadra-embedding: Binary code embedding with low quantization error," in *ACCV*, 2012.

[45] W. Kong, W.-J. Li, and M. Guo, "Manhattan hashing for large-scale image retrieval," in *SIGIR*, 2012.

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

[47] M. Norouzi and D. J. Fleet, "Cartesian k-means," in *CVPR*, 2013.

[48] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large-scale image search," in *ECCV*, 2008.

[49] R. F. S. Filho, A. Traina, C. J. Traina., and C. Faloutsos, "Similarity search without tears: the omni-family of all-purpose access methods," in *Int'l Conf. on Data Engineering*, 2001.

[50] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "idistance: An adaptive b^+-tree based indexing method for nearest neighbor search," *ACM T. on Database Systems*, 2005.

[51] J. Venkateswaran, D. Lachwani, T. Kahveci, and C. Jermaine, "Reference-based indexing of sequence databases," in *VLDB*, 2006.

[52] A. M. Yaglom and I. M. Yaglom, *Challenging Mathematical Problems with Elementary Solutions.* New York: Dover Pub., Inc., 1987.

[53] L. Greengard, *The rapid evaluation of potential fields in particle systems.* Cambridge: MIT Press, 1988.

[54] K. Grauman and T. Darrell, "The pyramid match kernel: discriminative classification with sets of image features," in *ICCV*, 2005.

[55] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, “Local features and kernels for classification of texture and object categories: A comprehensive study,” *IJCV*, vol. 73, pp. 213–238, 2007.

[56] I. S. Dhillon, Y. Guan, and B. Kulis, “Kernel k-means: spectral clustering and normalized cuts,” in *KDD*, 2004.

[57] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *CVPR*, june 2009, pp. 248 –255.

[58] H. Jégou, M. Douze, C. Schmid, and P. Pérez, “Aggregating local descriptors into a compact image representation,” in *CVPR*, 2010, pp. 3304 –3311.

[59] A. Gordo and F. Perronnin, “Asymmetric distances for binary embeddings,” in *CVPR*, 2011.

[60] F. Perronnin, J. Sánchez, and Y. Liu, “Large-scale image categorization with explicit data embedding,” in *CVPR*, 2010.

[61] A. Banerjee and J. Ghosh, “On scaling up balanced clustering algorithms,” in *SIAM Int. Conf. on Data Mining*, 2002.

[62] J. Hopcroft, “High dimensional data in course notes of mathematical foundations for the information age,” 2010.

[63] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, “Spherical hashing,” in *CVPR*, 2012.

[64] J.-P. Heo, Z. Lin, and S.-E. Yoon, “Distance encoded product quantization,” in *CVPR*, 2014.

[65] T. Cai, J. Fan, and T. Jiang, “Distributions of angles in random packing on spheres,” *JMLR*, 2013.

Summary

Compact Representation of High-Dimensional Data for Large-Scale Image Search

근사적 근접점 검색 (approximate nearest neighbor search) 은 계산기하학, 데이터마이닝, 컴퓨터 비전 등 다양한 전산학의 분야에서 오랫동안 집중적으로 연구되어 왔다. 이 문제는 주어진 데이터가 고차원이거나 그 규모가 방대할 때에 그 복잡도가 크게 증가한다. 하지만 실제 어플리케이션들에서서는 이러한 고차원 대규모 데이터에 대한 효율적인 근접점 검사 기법이 중요한 실정이다. 전통적으로 잘 알려진 계층 구조 기반의 방법들 (hierarchical methods) 은 저차원 데이터에 대해서 효율적인 해답이 될 수 있는 반면, 데이터의 차원이 높아짐에 따라 그 효율성이 급감하여 선형 탐색 (linear scan) 보다 비효율적일 경우도 있다. 그래서 최근에는 데이터의 간략한 표현법 (compact data representation) 에 대한 연구가 집중적으로 이루어지고 있는데, 그 이유는 데이터를 간략화함으로 높은 압축률과 빠른 거리 (혹은 유사도) 계산 속도를 얻을 수 있고 나아가서 방대한 데이터베이스를 다룰 수 있기 때문이다.

본 박사학위논문에서는 두 가지의 데이터 간략화 기법 1) Spherical Hashing (SpH)과 2) Distance Encoded Product Quantization (DPQ) 을 제안한다. 기존의 방법들이 초평면 (hyperplane) 기반 해싱 함수들을 이용한 반면, SpH 에서는 새로운 초구 (hypersphere) 기반의 해싱 함수를 사용할 것을 제안한다. 나아가 초구 기반의 해싱 함수를 위해 고안된 이진 코드간의 거리 함수와, 각 해싱 함수의 공간의 균등한 분할과 해싱 함수들 사이의 독립성을 만족시키는 최적화 과정 역시도 제안한다. 또한 커널 함수 (kernel function) 로 정의된 다양한 유사도 척도를 사용할 수 있도록 SpH 를 일반화 한다. DPQ는 공간 양자화 (space quantization) 방법을 개선한 방법이다. 기존의 방법들이 데이터가 어떤 보로노이 셀 (Voronoi cell) 에 속하는지 그 인덱스만으로 데이터를 이진 코드화 했다면, DPQ에서는 클러스터의 인덱스 뿐만 아니라 그 클러스터의 중심으로부터 어느정도 거리로 떨어져 있는지의 정보 역시도 이진코드화 하여 저장한다. 나아가 DPQ 에 특화된 두 개의 이진코드 거리 함수를 제안한다. 고안한 방법들은 모두 다양한 고차원 벤치마크에서 최신 기법들과 비교하였고, 최신 기법들 대비 나은 성능을 보임을 확인하였다.

더 나아가 본 박사학위논문에서 현재 진행 중인 다음의 두 연구, 역파일 (inverted file) 에서 후보군 선출 기법, 2) 이미지 태그 (image tag) 에 기반한 이미지 특징점 (image feature) 학습법, 을 소개한다. 먼저 첫 번째 연구에서는 고차원에서 벡터들 끼리의 수직 성질에 기반을 둔 거리 추정 함수를 정의하고, 이 함수에 기반하여 거리 기반 인덱싱 기법과 검색 결과 후보군 선출법을 제안하고 평가한다. 두 번째 연구에서는 대규모 태깅된 이미지 (large-scale tagged images) 를 이용하여 이미지 특징점 추출 기법을 학습하는 방법을 제안한다. 구체적으로는 허종 (pseudo class) 를 정의하고 각각의 이미지를 하나의 허종에 배정한 후, 뉴럴 네트워크 (neural network) 를 학습한다.

Curriculum Vitae

Name : Jae-Pil Heo (허재필, 許載弼)
Date of Birth : December 14, 1986
Birthplace : Jinju, Gyeongsangnam-do, Korea (경상남도 진주시)
E-mail : jaepilheo@gmail.com

Educations

2002. 3. – 2004. 2.	Gyeongnam Science High School (경남과학고등학교)
2004. 3. – 2008. 2.	Computer Science at KAIST, Korea (B.S.)
2008. 2. – 2010. 8.	Computer Science at KAIST, Korea (M.S.)
2010. 9. – 2015. 2.	Computer Science at KAIST, Korea (Ph.D.)

Career

2013. 12. – 2014. 03	Research Intern at Adobe, San Jose, California, USA
2014. 07. – 2014. 10	Research Intern at Adobe, San Jose, California, USA

Publications

1. Distance-Encoded Product Quantization
   Jae-Pil Heo, Zhe Lin, and Sung-eui Yoon
   IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014
2. VLSH: Voronoi-based Locality Sensitive Hashing
   Tieu Lin Loi, **Jae-Pil Heo**, Junghwan Lee, and Sung-Eui Yoon
   IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013
3. Spherical Hashing
   Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-eui Yoon
   IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012
4. Quadra-Embedding: Binary Code Embedding with Low Quantization Error
   Youngwoon Lee, **Jae-Pil Heo**, and Sung-eui Yoon
   Asian Conference on Computer Vision (ACCV), 2012
   Extended Version: Computer Vision and Image Understanding (CVIU), 2014
5. Probabilistic Cost Model for Nearest Neighbor Search in Image Retrieval
   KunHo Kim, M. Hasan, **Jae-Pil Heo**, Yu-wing Tai, and Sung-eui Yoon
   Computer Vision and Image Understanding (CVIU), 2012
6. IRIW: Image Retrieval based Image Watermarking for Large-Scale Image Databases
   Jong Yun Jun, Kunho Kim, **Jae-Pil Heo**, and Sung-eui Yoon
   International Workshop on Digital-forensics and Watermarking (IWDW), 2011

7. VolCCD: Fast Continuous Collision Culling between Deforming Volume Meshes
   Min Tang, Dinesh Manocha, Sung-eui Yoon, Peng Du, **Jae-Pil Heo**, and Ruofeng Tong
   ACM Transactions on Graphics (ToG), 2011
8. Data Management for SSDs for Large-Scale Interactive Graphics Applications
   Behzad Sajadi, Shan Jiang, **Jae-Pil Heo**, Sung-Eui Yoon, and M. Gopi
   ACM Symposium on Interactive 3D Graphics and Games (I3D), 2011
9. FASTCD: Fracturing-Aware Stable Collision Detection
   Jae-Pil Heo, Joon-Kyung Seong, DukSu Kim, Miguel A. Otaduy, Jeong-Mo Hong, Min Tang, and Sung-Eui Yoon
   ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA), 2010
10. HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs
    DukSu Kim, **Jae-Pil Heo**, JaeHyuk Huh, John Kim, and Sung-Eui Yoon
    Computer Graphics Forum (Pacific Graphics), 2009

Activities

1. Conference/Journal Reviewer:
   SIGGRAPH Asia
   Computer Graphics Forum (CGF)
   Pacific Graphics
   Computer Vision and Image Understanding (CVIU)
   Neurocomputing