LARGE-SCALE IMAGE RETRIEVAL USING SIMILARITY PRESERVING BINARY CODES

Yunchao Gong

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2014

Approved by:

Svetlana Lazebnik

Alexander C. Berg

Dinesh Manocha

Jan-Michael Frahm

Marc Niethammer

Sanjiv Kumar

ABSTRACT

YUNCHAO GONG: LARGE-SCALE IMAGE RETRIEVAL USING SIMILARITY PRESERVING BINARY CODES.
(Under the direction of Svetlana Lazebnik.)

Image retrieval is a fundamental problem in computer vision, and has many applications. When the dataset size gets very large, retrieving images in Internet image collections becomes very challenging. The challenges come from storage, computation speed, and similarity representation. My thesis addresses learning compact similarity preserving binary codes, which represent each image by a short binary string, for fast retrieval in large image databases.

I will first present an approach called Iterative Quantization to convert high-dimensional vectors to compact binary codes, which works by learning a rotation to minimize the quantization error of mapping data to the vertices of a binary Hamming cube. This approach achieves state-of-the-art accuracy for preserving neighbors in the original feature space, as well as state-of-the-art semantic precision. Second, I will extend this approach to two different scenarios in large-scale recognition and retrieval problems. The first extension is aimed at high-dimensional histogram data, such as bag-of-words features or text documents. Such vectors are typically sparse and nonnegative. I develop an algorithm that explores the special structure of such data by mapping feature vectors to binary vertices in the positive orthant, which gives improved performance. The second extension is for Fisher Vectors, which are dense descriptors having tens of thousands

to millions of dimensions. I develop a novel method for converting such descriptors to compact similarity-preserving binary codes that exploits their natural matrix structure to reduce their dimensionality using compact bilinear projections instead of a single large projection matrix. This method achieves retrieval and classification accuracy comparable to that of the original descriptors and to the state-of-the-art Product Quantization approach while having orders of magnitude faster code generation time and smaller memory footprint.

Finally, I present two applications of using Internet images and tags/labels to learn binary codes with label supervision, and show improved retrieval accuracy on several large Internet image datasets. First, I will present an application that performs cross-modal retrieval in the Hamming space. Then I will present an application on using supervised binary classeme representations for large-scale image retrieval.

ACKNOWLEDGMENTS

First, I would like to express my sincere gratitude to my advisor, Professor Svetlana Lazebnik, for her great support of my Ph.D. research. She introduced me to the area of computer vision, shaped me as a serious researcher, taught me to always focus on challenging problems, and to always try to make an impact. In particular, from her I learned the benefits of critical thinking and striving for perfection. I have been very lucky to have Lana as my advisor, and this dissertation could not have been accomplished without her guidance. I am also grateful to my wonderful thesis committee members: Professor Dinesh Manocha, Professor Alexander C. Berg, Professor Jan-Michael Frahm, Professor Marc Niethammer, and Dr. Sanjiv Kumar.

During my doctoral studies, I had the good fortune to work with many world-class researchers in industry. I thank Dr. Qifa Ke and Dr. Michael Isard for hosting me as an intern at Microsoft Research Silicon Valley in 2011. Qifa not only served as my internship mentor but also is a great friend who has given me many insightful suggestions about my career. The internship with Dr. Sanjiv Kumar and Dr. Henry Rowley in 2012 was truly memorable and productive. I am grateful for the wonderful summer of living in New York City, the great research experiences, and the blueberries grown on the balcony of Google NY. Dr. Thomas Leung introduced me to the area of deep learning through my internship at Google Research Mountain View in 2013, and provided me with the

valuable opportunity to work on the world's largest brain-simulation system. I also thank Dr. Sergey Ioffe for serving as my mentor for my Google Ph.D. Fellowship, and for his many helpful suggestions.

Great work cannot be done without great collaborators. I am lucky to have worked with many smart colleagues. I especially thank Ruiqi Guo for his critical (and interesting!) comments about my work throughout my doctoral studies. I also thank many other collaborators and coauthors: Dr. Joseph Tighe, Dr. Sanjiv Kumar, Dr. Henry Rowley, Dr. Qifa Ke, Dr. Michael Isard, Dr. Albert Gordo, Dr. Florent Perronnin, Dr. Vishal Verma, Dr. Yangqing Jia, Dr. Thomas Leung, Dr. Sergey Ioffe, Dr. Alexander Toshev, Dr. Julia Hockenmaier, Felix X. Yu, Liwei Wang, and Micah Hodosh.

I shared every moment in the past five years with Chenyao Zhang. I truly thank her for sticking by my side, for her patience, and for her love. I believe that we will continue the journey together hand in hand.

TABLE OF CONTENTS

LIST OF FIGURES xi

LIST OF TABLES xiv

CHAPTER 1: Introduction 1

1.1 Overview of Contributions 6

CHAPTER 2: Related Works 10

2.1 Approximate Nearest Neighbor Search 10

2.1.1 Spatial Partitioning Tree and Locality Sensitive Hashing 11

2.1.2 Similarity Preserving Binary Codes 13

2.1.3 Product Quantization 19

2.2 Large-scale Image Retrieval and Recognition 20

2.2.1 Internet Computer Vision and Beyond 23

CHAPTER 3: Iterative Quantization for Learning Compact Binary Codes 26

3.1 The ITQ Formulation 27

3.1.1 Dimensionality Reduction 28

3.1.2 Binary Quantization 30

3.2 Evaluation of Unsupervised Code Learning 33

3.2.1 Datasets 33

3.2.2 Protocols and Baseline Methods 34

3.2.3 Results on CIFAR Dataset 36

3.2.4 Results on 580,000 Tiny Images 39
3.2.5 Evaluation of Hashing Performance 43
3.3 ITQ with a Kernel Embedding 47
3.3.1 Random Fourier Features 47
3.3.2 Results 48
3.4 Discussion 51
CHAPTER 4: Angular Quantization for Histogram Data 53
4.1 Angular Quantization-based Binary Codes 55
4.1.1 Data-independent Binary Codes 55
4.1.2 Learning Data-dependent Binary Codes 59
4.1.3 Optimization 63
4.1.4 Computation of cosine Similarity between Binary Codes 65
4.2 Experiments 65
4.2.1 Datasets and Protocols 65
4.2.2 Results on SUN and ImageNet 67
4.2.3 Results on 20 Newsgroups 68
4.2.4 Timing 73
4.3 Discussion 74
CHAPTER 5: Bilinear Hashing for Very High-dimensional Data 75
5.1 Bilinear Binary Codes 78
5.1.1 Learning Bilinear Binary Codes 80
5.1.2 Learning with Dimensionality Reduction 83

5.2 Experiments 86
5.2.1 Datasets and Features 86
5.2.2 Experimental Protocols 88
5.2.3 Baseline Methods 89
5.2.4 Code Generation Time and Storage 90
5.2.5 Retrieval on Holiday+Flickr1M 91
5.2.6 Retrieval on ILSVRC2010 with VLAD 95
5.2.7 Retrieval on ILSVRC2010 with LLC 96
5.2.8 Image Classification 97
5.3 Discussion 100
CHAPTER 6: Combining Semantic Embeddings and Binary Codes 103
6.1 Semantic Binary Codes for Weakly Tagged Data 104
6.1.1 Results on Tiny Images 106
6.1.2 Results on NUS-WIDE Dataset 108
6.2 Binary classeme 112
6.3 Discussion 118
CHAPTER 7: Hashing Revisited: Observations and Open Problems 120
7.1 Why Do Binary Codes Work? 121
7.2 Distance Function Matters 123
7.3 Neighborhood Definition Matters 130
7.4 Discussion 137

CHAPTER 8: Discussion and Future Directions 138

8.1 Summary of Contributions 138

8.2 Future Directions 139

BIBLOGRAPHY 142

LIST OF FIGURES

1.1 Visual search for shopping 2

1.2 Illustration of visual similarity 3

1.3 Mapping images to binary codes 4

2.1 Toy example of kd-tree and LSH 11

3.1 Toy example of ITQ 27

3.2 ITQ optimization 32

3.3 Main results for ITQ 38

3.4 NN search results on CIFAR dataset for ITQ 40

3.5 Semantic retrieval results on CIFAR dataset for ITQ 41

3.6 NN search results on Tiny images for ITQ 42

3.7 Overview of NN results on Tiny images for ITQ 43

3.8 Qualitative results for ITQ 44

3.9 NN search hashing results for ITQ 45

3.10 Semantic hashing results for ITQ 46

3.11 Kernel ITQ results 49

3.12 Kernel ITQ with different radius results 50

4.1 Quantization model in 3D 56

4.2 cosine of angle between binary vertices 59

4.3 Effect of rotation on Hamming weight 62

4.4 Results on SUN dataset 69

4.5 Results on ImageNet120K 70

4.6 Results on 20 Newsgroups 71

4.7 Effect of projection on Hamming weight 72

5.1 Memory of projections vs. dimensionality 76

5.2 Visualization of the distribution of the VLAD descriptor 84

5.3 Sample images from Holiday Dataset. 87

5.4 Results on Holiday dataset 93

5.5 Results on ILSVRC with VLAD feature 94

5.6 Results on ILSVRC with LLC feature 98

5.7 Sample image retrieval results 101

6.1 Logical flow of the PCA+ITQ and CCA+ITQ methods. 106

6.2 Semantic retrieval results on CIFAR 108

6.3 Sample image retrieval results on CIFAR 109

6.4 Sample image search results on NUS dataset 113

6.5 Data flow of the binary classeme idea 114

6.6 Qualitative results for text to image search 2 on NUS-WIDE 115

6.7 Sample image search results on ImageNet 119

7.1 Analysis of the distribution on Hamming cube 122

7.2 Nearest neighbor search results for different distance metrics 125

7.3 A comparison of different random hashing models 127

7.4 Comparison of different random hash models 128

7.5 Visualization of a toy dataset 130

7.6 Analysis of the distribution of the neighbors 133

7.7 Comparison of different methods on different neighborhood definitions 134

7.8 The distribution of neighbors for different queries 135

8.1 Sample results for finding Iconic images 140

LIST OF TABLES

4.1 Semantic retrieval results on SUN dataset 68

4.2 Semantic retrieval results on ImageNet120K dataset 68

4.3 Semantic retrieval results on 20 Newsgroups dataset 73

4.4 Timing for binary codes 73

5.1 Code generation time for linear and bilinear projections 90

5.2 Memory needed to store projections 90

5.3 Running time on ILSVRC 92

5.4 Image classification results on ILSVRC 99

6.1 Image retrieval results on NUS-WIDE 111

6.2 Image retrieval results on ILSVRC 116

6.3 Image classification results on ILSVRC 117

CHAPTER 1: Introduction

We interact with images every day. We use Google Image Search to find interesting images, and we upload our personal photos to Facebook, which contains billions of images contributed by its users. Mobile applications for image retrieval are available, including Google Goggles, a mobile-image-search tool that provides real-time image search results. Google Shopping also provides visually similar items (e.g., Figure 1.1) to improve the shopping experience. Clearly, Internet photo collections have become one of the most important parts of everyday life. However, how to organize such collections is still an open problem in computer vision research, and in order to address it, we must first address the problem of large-scale similarity search.

Today, image search and recognition research is being performed on ever-larger databases. For example, the ImageNet database (Deng et al., 2009) currently contains around 15M images and 20K categories. The main challenges for large-scale image retrieval are:

1. How to define similarity between images;

2. How to design compact representations for images, so we can store them;

3. How to design fast search schemes, so we can efficiently find similar images.

The first challenge, that of learning good visual similarity, is also one of the the most fundamental one in computer vision. Typically, we assume images are represented as a

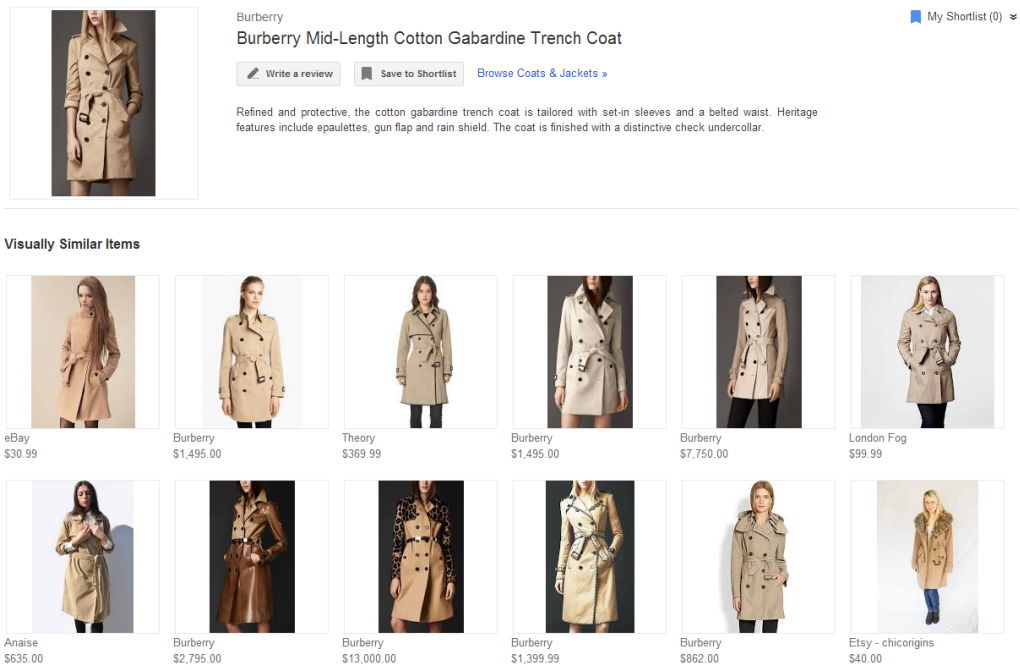


Figure 1.1: Image search for shopping applications – finding visually similar items. Image taken from Google Shopping.

feature vector, and directly compute the distance between these vectors, which is referred to as *feature similarity*. A good feature similarity should reflect *semantic similarity*: i.e. that images containing the same objects or scenes should have smaller distance, and images containing different objects or scenes should have larger distance, as shown in Figure 1.2. Defining good feature representations reflecting this semantic similarity is one of the major challenges of recognition in general, but it is not my primary focus. For the most part of this thesis, I will assume that the feature representation is given and fixed, and try to design binary codes that preserve both feature-space and semantic similarity in that representation.

The second and third challenges are the main foci of this thesis. To address these challenges, the vision community has recently devoted much attention to the problem

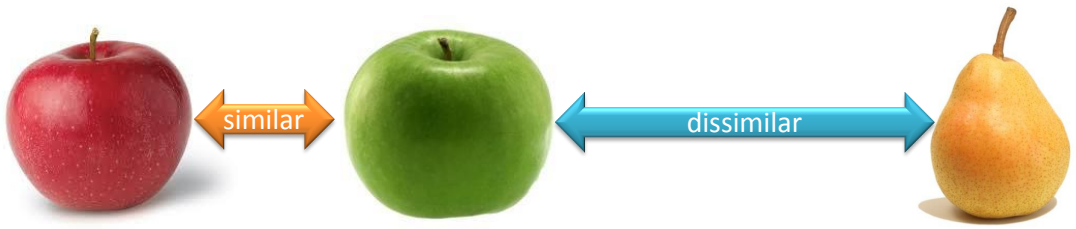
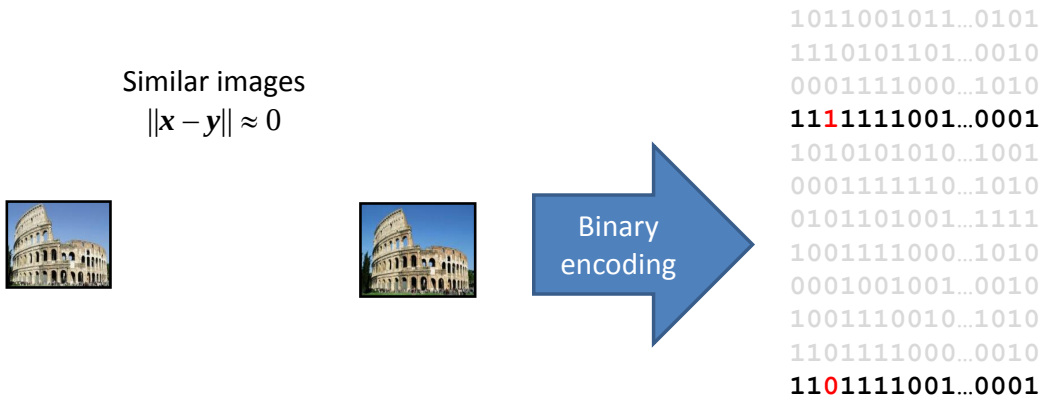


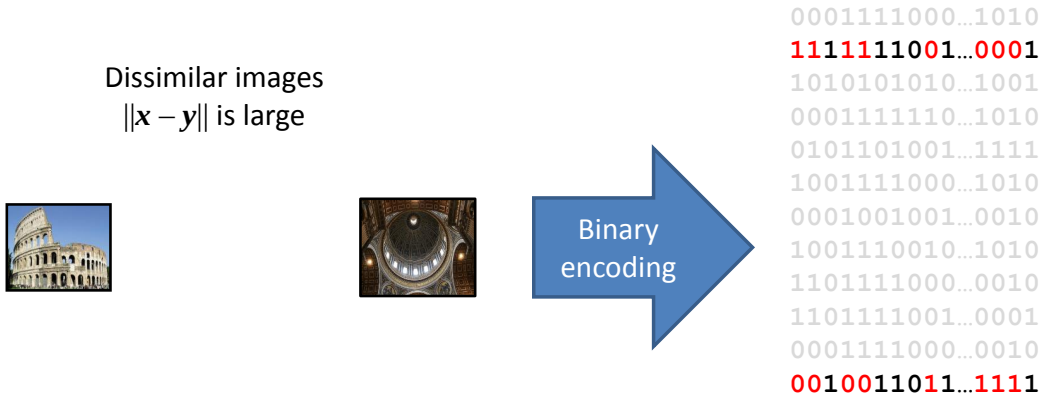
Figure 1.2: An illustration of visual similarity. Images containing same object should be more similar to images containing different objects.

of learning *similarity-preserving binary codes* for representing large-scale image collections (Torralba et al., 2008b; Raginsky and Lazebnik, 2009; Wang et al., 2010a,b; Weiss et al., 2008). Encoding high-dimensional image descriptors as compact binary strings can enable large efficiency gains in storage and computation speed for similarity search; moreover, these goals can be accomplished with much simpler data structures and algorithms than alternative large-scale indexing methods (Chum et al., 2009; Jégou et al., 2010; Kulis and Grauman, 2009; Nister and Stewenius, 2006). The basic idea of similarity-preserving binary codes is illustrated in Figure 1.3. We want to map similar images to similar binary codes (i.e., binary codes with a low Hamming distance), and dissimilar images to codes with a high Hamming distance. Attempting to do this means trying to transform data from a real space to a discrete Hamming space, a transformation that preserves the semantic and spatial relationship between data points in the original space.

The conversion of high-dimensional vectors to a discrete representation, which is referred to as *locality-sensitive hashing* (LSH) (Andoni and Indyk, 2008), has been studied by the computer science theory community. These methods, which are based on the use of random projections to perform spatial partitioning, mostly focus on establishing the



(a) Similar images have similar binary codes



(b) dissimilar images have dissimilar binary codes

Figure 1.3: Mapping images to similarity-preserving short binary codes, so distances between images are measured by fast Hamming distance. (a) Similar images $\boldsymbol{x}$ and $\boldsymbol{y}$ should have similar binary codes with small Hamming distance. (b) Dissimilar images $\boldsymbol{x}$ and $\boldsymbol{y}$ should have binary codes with large Hamming distance. (Figure credit: Svetlana Lazebnik)

underlying theoretical guarantees for using very large numbers of random projections to preserve the geometry of high-dimensional data. Not all LSH schemes yield binary embeddings of data, however. Torralba et al. (2008b) have introduced the binary coding problem to the vision community and have compared several methods based on boosting, restricted Boltzmann machines (Salakhutdinov and Hinton, 2009), and LSH. Such work opens a new avenue for LSH and its applications in computer vision, especially because it tries to answer the question of how to learn compact binary codes from real data, so that such codes can be used in real applications. Considerable attention has been devoted to this problem, subsequently (Weiss et al., 2008; Rahimi and Recht, 2007; Raginsky and Lazebnik, 2009; Wang et al., 2010a) as well as applications that show the promise of binary codes for vision problems (Frahm et al., 2010; Kuettel et al., 2012).

The goal of this thesis is to develop better binary coding algorithms for the conversion of high-dimensional data points to similarity-preserving binary codes. I will consider two scenarios to address these challenges. The first is *unsupervised* binary coding that does not use labeled training data but only learns binary codes that approximate distances (e.g., Euclidean distance or cosine similarity). I will propose three methods, in Chapter 3, Chapter 4, and Chapter 5, to address this problem. I will also consider a *supervised* setting that uses labeled data to learn semantically-consistent binary codes that renders semantically-similar images closer to each other in the Hamming space. I will tackle the latter problem in Chapter 6. A detailed outline of the contribution of each chapter is provided below.

1.1 Overview of Contributions

Many existing binary coding methods (Weiss et al., 2008; Wang et al., 2010a) start by computing a lower-dimensional projection of the data using Principal Component Analysis (PCA). As discussed in Weiss et al. (2008) and Wang et al. (2010a), the variance of the data in each PCA direction is different: higher-variance directions, in particular, carry much more information. Consequently, encoding each direction with the same number of bits is bound to produce poor performance. Spectral hashing (SH) (Weiss et al., 2008) and semi-supervised hashing (SSH) (Wang et al., 2010a) methods were intended to use spectral and nonorthogonal relaxations to address the problem of imbalanced variance. I also start with with PCA-projected data, but unlike these methods, I formulate the problem of learning a good binary code in terms of directly minimizing the quantization error of mapping this data to vertices of the binary hypercube. First, I show that simply applying a random orthogonal transformation to the PCA-projected data, as suggested by Jégou et al. (2010), already does a very good job of balancing the variance of different PCA directions and outperforms both SH (Weiss et al., 2008) and non-orthogonal relaxation (Wang et al., 2010a). Next, I propose an alternating minimization approach for refining the initial orthogonal transformation to reduce quantization error. This approach, called *iterative quantization* (ITQ), has connections to the orthogonal Procrustes problem (Schonemann, 1966) and to eigenvector discretization for multi-class spectral partitioning (Yu and Shi, 2003). I conduct experiments on several large image datasets and show that the proposed method achieves state-of-the-art performance. This approach

is described in Chapter 3.

My first extension to ITQ in Chapter 4 is motivated by the observation that ITQ works poorly on high-dimensional sparse histogram data, which is commonly used in many applications in image and document retrieval. The reason that ITQ does not work very well for such data is that it directly minimizes the Euclidean distance between original and binarized data. However, for histogram data, the values are usually nonnegative and sparse, and cosine similarity is more appropriate than Euclidean distance. I propose a novel angular quantization technique, in which the angle between two vectors is used as a similarity measure, to learn binary codes from non-negative data. This proposed technique works by quantizing each data point to the vertex of the positive orthant of the binary hypercube with which it has the smallest angle. The number of these quantization centers or landmarks is exponential in the dimensionality of the data, yielding a low-distortion quantization of a point. Note that it would be computationally infeasible to perform traditional nearest-neighbor quantization, as in regular ITQ, with such a large number of centers. Moreover, even at run time, finding the nearest center for a given point would be prohibitively expensive. Instead, I present a very efficient method to find the nearest landmark for a point, i.e., the vertex of the binary hypercube with which it has the smallest angle. In addition, because the basic form of my quantization method does not take data distribution into account, I further propose a learning algorithm that linearly transforms the data before quantization to reduce the angular distortion. I experimentally show that this algorithm significantly outperforms other state-of-the-art binary coding methods on both visual and textual data where histograms are used as

underlying representation.

My second extension of ITQ in Chapter 5 is motivated by recent the success of the *Fisher vector* (FV) (Perronnin and Dance, 2007; Perronnin et al., 2010b) and *vector of locally aggregated descriptor* (VLAD) (Jégou et al., 2010) – very powerful visual descriptors that have achieved state-of-the-art performance for visual retrieval and recognition. However, the dimensionality of FV and VLAD is extremely high and the feature is dense, which makes it very hard to perform learning and retrieval. The high dimensionality also makes mapping it to binary codes very inefficient. There are two major problems: first, the projection matrix will be extremely large, and may not even fit in memory; second, the projection speed will become extremely slow due to the multiplication with the huge projection matrix. To solve these two problems, I propose a bilinear formulation that uses two small projections to reconstruct the huge projection matrix. This approach addresses both the storage and projection speed problems. My experiments on large-scale image retrieval and classification datasets (Jégou et al., 2008; Deng et al., 2009) have demonstrated that my approach offers comparable performance to the original dense FV or VLAD, with a much more compact representation and faster retrieval speed.

In Chapter 6, I present two applications for using labels from Internet image collections to learn semantically consistent binary codes: one for using noisy tags from Internet images to learn semantic binary codes for web image retrieval, and the other for using clean class labels from ImageNet to learn binary *classeme* representations (Torresani et al., 2010) for novel class recognition. I show that my proposed binary coding methods can be effectively combined with semantic embedding methods, and that the learned bi-

nary codes work as well as the original real embedding for large-scale retrieval problems in Internet image collections.

Finally, in Chapter 7, I review and discuss current research on binary embeddings: why it works, how to better evaluate it, its limitations, and tricky issues. I hope that such self-reflection will provide insights that will benefit future research.

CHAPTER 2: Related Works

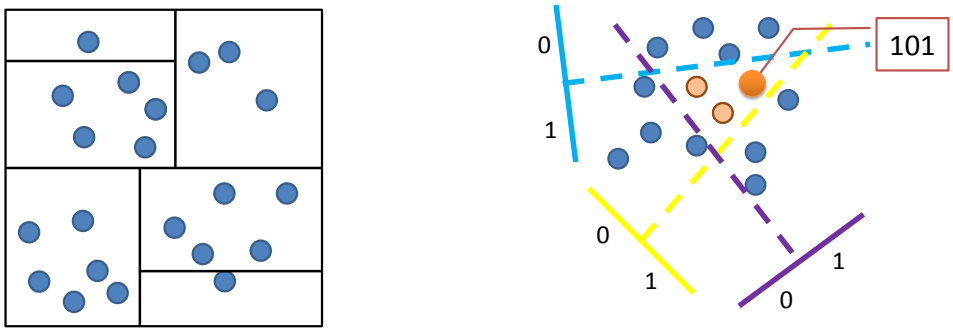
This chapter contains a survey of important works related to this thesis. In Section 2.1, I will present a survey on approximate nearest neighbor search and similarity preserving binary codes. In Section 2.2, I review relevant work in our key application areas of large-scale image classification and retrieval.

2.1 Approximate Nearest Neighbor Search

Nearest neighbor search (Shakhnarovich et al., 2006; Samet, 1990) is a fundamental operation underlying many computer vision approaches. Brute-force search, which means comparing a query point with every point in the database, becomes prohibitively expensive as the dataset size and the dimensionality of the features grow. To reduce search complexity, a number of algorithms and data structures have been proposed (Shakhnarovich et al., 2006). These methods can be broadly categorized into two classes:

- Spatial partitioning methods such as trees (Bentley, 1975; Samet, 1990) and hash tables (Andoni and Indyk, 2008), which are aimed at enabling sublinear-time similarity search;

- Similarity preserving binary codes (Torralba et al., 2008b; Weiss et al., 2008), which are typically aimed at making brute-force search more efficient.



(a) kd-tree (b) LSH

Figure 2.1: A toy example of kd-tree and LSH scheme. (a) kd-tree partitions the space by recursively partitioning the coordinates. (b) LSH partitions the space by random projections. (Figure (b) is from Professor Rob Fergus's slides.)

2.1.1 Spatial Partitioning Tree and Locality Sensitive Hashing

Tree- and hashing-based nearest neighbor search has been an active research area in theoretical computer science, machine learning, and computer vision. These methods partition the feature space into a large number of cells and try to achieve sub-linear searches by reducing the search space.

Tree-based spatial partitioning methods have been applied to nearest neighbor search problems and have achieved a certain degree of success. Among many successful methods, kd-tree (Bentley, 1975; Samet, 1990; Muja and Lowe, 2009) is probably one of the most widely used methods; it works by partitioning the coordinates recursively (see Figure 2.1(a)). During the construction stage, a tree is grown based on training data by splitting the median of selected dimension and stored in the memory. During the search stage, a query is passed to the tree and follows the spatial partition to a leaf node; next, a brute-force comparison between the query and the points in the leaf node is performed to find

neighbors. However, kd-trees suffer from the curse of dimensionality and cannot handle high-dimensional data (e.g., several hundred dimensions). A few other tree variants such as k-means trees (Nister and Stewenius, 2006) and random projection trees (Freund et al., 2007) have also been applied to nearest neighbor search in vision applications, but have achieved limited success for high-dimensional data.

Locality sensitive hashing (LSH) (Andoni and Indyk, 2008; Datar et al., 2004; Gionis et al., 1999; Chum and Matas, 2010) is considered to be an important breakthrough for fast nearest neighbor searches in high-dimensional space (e.g., more than hundreds of dimensions). One particular LSH scheme for approximating cosine similarity uses random hyperplanes to partition the space into many cells; each cell is known as a bucket. An illustration of this LSH scheme is shown in Figure 2.1 (b). During construction time, LSH only needs to generate several random hyperplanes and no training is performed. During indexing, points are mapped to hash keys as shown in Figure 2.1 (b) and points with the same key are stored in the same bucket. During the query stage, a point is first mapped to its key, which is used to locate the bucket. A brute-force comparison between the query and points in the corresponding bucket is then performed, which finds exact nearest neighbors. Many LSH variants have been proposed, for example LSH for spherical data (Charikar, 2002), and min-hash (Broder, 1997). However, LSH is a randomized algorithm and does not take any advantage of the data distribution.

2.1.2 Similarity Preserving Binary Codes

Unlike tree-based methods and LSH, similarity preserving binary codes (Torralba et al., 2008b; Weiss et al., 2008; Gong and Lazebnik, 2011b) do not necessarily offer sub-linear search complexity. However, they can reduce memory requirements and the running times of brute-force searches by significant, constant factors, and they do not involve building and maintaining complex data structures.

The similarity preserving binary code idea was first introduced by Salakhutdinov and Hinton (2009) for text retrieval, and then introduced to the vision community by Torralba et al. (2008b). Both of these researchers used a deep neural network architecture to learn binary codes and were able to show successful applications for text, image retrieval, and scene parsing. A number of subsequent works that are surveyed in the following pages focused on designing better models for learning binary codes. Most methods considered here consist of the following three steps:

1. Projection learning, or finding a linear or nonlinear projection of the data;

2. Binary thresholding, or quantizing continuous projected data to binary vectors;

3. Similarity computation, or finding distances between query and database points.

I will discuss each of these steps in related methods in the next sections.

Step 1: Projection Learning

The first step of computing binary codes usually involves finding an intermediate continuous embedding of the original data. This step corresponds to learning a linear or

nonlinear projection to reduce data dimensionality. One example is the angle-preserving LSH scheme by Charikar (2002); Andoni and Indyk (2008) that uses hyperplanes to preserve angles. Kulis and Grauman (2009); Kulis et al. (2009) generalized this LSH formulation to the setting in which the similarity is given by a "black-box" kernel function. Raginsky and Lazebnik (2009) used random Fourier features (Rahimi and Recht, 2007) to approximate the Gaussian kernel. All of these methods are based on random projections. With large numbers of random projections, these methods usually enjoy strong theoretical guarantees for approximating the distance between the original points; however, when the number of bits is small, they tend to be quite noisy. Therefore they might not be appropriate for applications with limited storage budgets.

Several recent works have focused on designing compact binary codes by learning projections from training data. Weiss et al. (2008) used the graph Laplacian as the underlying intermediate representation. They proposed several relaxations of the graph Laplacian decomposition step, and showed that PCA is a good choice of projection method. However, the relaxations made in Weiss et al. (2008) are very restricted, as they assume the data is sampled from uniform distribution, which does not hold for real-world data. Kulis and Darrell (2009) proposed learning projections with binary reconstructive embedding (BRE) that directly minimizes the Euclidean distance between binary codes and original data through coordinate descent. Subsequent works by Wang et al. (2010a,b) showed that PCA is a good projection method from an information-theoretical perspective, and proposed several ways to adjust PCA projections for better binary quantization (e.g., using non-orthogonal projection directions). Many later works

(Gordo et al., 2011; Jégou et al., 2011) also used PCA as an initial step for the projection step. Gordo et al. (2011) proposed to directly threshold PCA projections, and have obtained good performance with asymmetric distance computation (see below). Jégou et al. (2011) also applied PCA in their image retrieval pipeline for fast indexing. Beyond PCA, He et al. (2011) used a method similar to independent component analysis (ICA) for learning hash functions; they showed that such projections usually do not need to be coupled with an "adjustment" because ICA projections are learned to be independent. Strecha et al. (2010) used linear discriminant analysis (LDA) and Liu et al. (2012) used discriminative graph embedding to learn supervised hash codes that work better for semantic retrieval. In my work, I start with PCA projections. In Chapter 3 I will show how to adjust these projections for better preservation of similarities.

Step 2: Binary Thresholding

After the embedding, the next step is to binarize the data. For randomized embeddings, this can be done simply by appropriately thresholding the projected data (Andoni and Indyk, 2008; Kulis and Grauman, 2009; Raginsky and Lazebnik, 2009). Random projections followed by thresholding have strong theoretical guarantees (Andoni and Indyk, 2008). For PCA and related techniques, a more complex binarization scheme is necessary to alleviate the problem of unbalanced variance discussed in Section 1.1. In particular, as shown in Wang et al. (2010a), directly thresholding the PCA projections will lead to poor codes, since directions with larger variance will be quantized to have the same number of bits as the directions with smaller variance. Another approach, spectral

hashing (Weiss et al., 2008, 2012), allocated bits based on separable Laplacian eigenfunctions, but the allocation function was heuristic and involved very restricted assumptions about the data. The work by Liu et al. (2011) similarly used multiple bits per dimension to address the imbalanced variance problem. Norouzi and Fleet (2012); Norouzi et al. (2012a) proposed an elegant structured prediction framework to directly minimize the error of quantization through structured learning, which was solved by optimizing the upper bound of the structured loss. The methods Norouzi and Fleet (2012); Norouzi et al. (2012a) do not split the projection learning and quantization stage, but instead directly learn the hash functions, and have achieved superior performance. However, one limitation is that such approaches usually involve many free parameters that must be tuned.

In this thesis, I proposed approaches that directly try to minimize binary quantization error by learning a rotation of PCA-projected data and have achieved state-of-the-art performance. Some other approaches have also proposed other methods for thresholding. Heo et al. (2012) proposed to use hyperspheres instead of hyperplanes to partition the space. Some recent works (Kong and Li, 2012; Ge et al., 2013; Norouzi and Fleet, 2013) adopted the rotation learning idea of ITQ and further improved it.

Step 3: Distance Computation for Binary Codes

At retrieval time, given a new query image, it is necessary to compute the distances from that query to every image in the database. I will introduce three different distance metrics for binary codes in this section: : Hamming distance, cosine similarity, and

asymmetric distance.

Most methods directly compute the Hamming distance between the respective binary codes, which simply measures how many bits are different between two strings. Hamming distance can be computed very efficiently using XOR and POPCOUNT operation in CPU. For example, given two binary vectors $\boldsymbol{b}_1 = 1001$ and $\boldsymbol{b}_2 = 0011$, their XOR is computed as:

$$\begin{array}{rcccc} & 1 & 0 & 0 & 1 \\ \oplus & 0 & 0 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0, \end{array}$$

and then the internal CPU operation POPCOUNT is used to count the number of 1s in the resulting vector to obtain the Hamming distance. In practice, we can group 64 bits together to a unsigned 64-bit integer, and during distance computation we can directly perform operations on the integers.

The second method is *cosine similarity* between binary codes. The cosine of the angle θ between two binary vectors $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ is defined as:

$$\cos(\theta) = \frac{\boldsymbol{b}_1^T \boldsymbol{b}_2}{\|\boldsymbol{b}_1\|_2 \|\boldsymbol{b}_2\|_2}. \tag{2.1}$$

The dot-product $\boldsymbol{b}_1^T \boldsymbol{b}_2$ can be obtained by bitwise AND followed by POPCOUNT, and $\|\boldsymbol{b}_1\|_2$ and $\|\boldsymbol{b}_2\|_2$ can be obtained by popcount and lookup table to find the square root. Of course, if $\boldsymbol{b}_1$ is the query vector that needs to be compared to every database vector $\boldsymbol{b}_2$, then one can ignore $\|\boldsymbol{b}_1\|_2$. Therefore, even though the cosine similarity is marginally slower than Hamming distance, it is still very fast compared to computing similarity of

the original data vectors.

The third method is *asymmetric distance* (Dong et al., 2008; Gordo et al., 2011; Jégou et al., 2011), in which the database points are quantized but the query data point is not. Such an approach usually offers better performance than Hamming distance. However, the query data structure is also more complicated and the search speed is not as fast as that of Hamming distance. For a query $\boldsymbol{x} \in \mathbb{R}^c$ ($\boldsymbol{x}$ has been rotated or transformed by hash function without binarization) and database points $\boldsymbol{b} \in \{-1, +1\}^c$, the *lower-bounded asymmetric distance* is simply the L_2 distance between $\boldsymbol{x}$ and $\boldsymbol{b}$: $d_a(\boldsymbol{x}, \boldsymbol{b}) = \|\boldsymbol{x}\|_2^2 + c - 2\boldsymbol{x}^T\boldsymbol{b}$. Since $\|\boldsymbol{x}\|_2^2$ is on the query side and c is fixed, in practice, we only need to compute $\boldsymbol{x}^T\boldsymbol{b}$. We can do this by putting bits in groups of 8 and constructing a 1×256 lookup table to make the dot-product computation more efficient.

The three different functions introduced above will be used in following chapters. In particular, Hamming distance will be used in Chapter 3 and Chapter 6, cosine similarity will be used in Chapter 4, and asymmetric distance will be used in Chapter 5.

If nearest neighbor lookup by linear scan is not satisfactory, it is possible to treat the binary strings representing images as hash keys (Andoni and Indyk, 2008; Kulis and Grauman, 2009) and to apply bucket-based sub-linear search. Finally, Norouzi et al. (2012b) have also proposed a multi-indexing scheme to find exact k nearest neighbors in Hamming space in sub-linear time. This scheme uses buckets to find a superset of neighbors that contains exact kNNs, and then performs a local reranking to find exact k nearest neighbors.

2.1.3 Product Quantization

Aside from similarity preserving binary codes, other methods have been proposed for efficient large-scale image retrieval. One of these is product quantization (PQ) (Jégou et al., 2011), in which the feature space is decomposed into a Cartesian product of low-dimensional subspaces (or subsets of dimensions) and each subspace is quantized separately. During the training stage, each subspace is clustered into k centers (i.e., quantizers). The Cartesian product of all these centers forms a very dense partitioning of the whole space and provides very small quantization error. After the quantization learning stage, given a new query point, asymmetric distance is computed between the query and the database codes using a distance lookup table. PQ is very accurate for approximating distances between points. However, the PQ data structure is usually much more complicated than binary codes, and PQ itself does not yield a low-dimensional feature representation. In addition, it must be coupled with asymmetric distance in order to achieve good performance. Another limitation of PQ is that it needs the variance of each dimension to be balanced. Without balanced variances, PQ will produce poor results. Two recent works (Ge et al., 2013; Norouzi and Fleet, 2013) have shown that by combining ITQ and PQ (Jégou et al., 2011), even better performance can be achieved. However, when the data dimensionality gets much higher, learning and performing a rotation to the data becomes extremely hard. Simply storing the rotation matrix becomes infeasible since the matrix becomes huge. I will present an approach for efficiently learning rotations for PQ in Chapter 5. I particularly considers PQ here because it is one of the

most accurate baselines in the literature.

2.2 Large-scale Image Retrieval and Recognition

The main applications of similarity preserving binary codes in my thesis are large-scale image classification and retrieval. These in turn can serve as the bases for many other applications where binary codes can be useful, such as location recognition (Hays and Efros, 2008), scene parsing (Tighe and Lazebnik, 2010), object detection (Felzenszwalb et al., 2008), image segmentation (Kuettel et al., 2012), 3D reconstruction (Frahm et al., 2010), and many others.

Up until about a decade ago, most of the early research on visual recognition and retrieval (Sivic and Zisserman, 2003; Lazebnik et al., 2006; Fei-Fei and Perona, 2005) has focused on small-scale datasets, which usually contained only hundreds or thousands of images. For example, the Caltech101 (Fei-Fei et al., 2006) and Caltech256 (Griffin et al., 2008) datasets have been the most popular recognition benchmarks since 2004, and many successful recognition models have been developed based on these small-scale datasets. One of the most successful models is the bag of visual words (BoW) approach (Csurka et al., 2004; Sivic and Zisserman, 2003), in which the local features of an image are aggregated into a global high-dimensional visual descriptor. I will specifically develop binary coding methods for BoW type data in Chapter 4. Several kernel approaches have been proposed for training nonlinear support vector machine (SVM) classifiers on top of BoW representations. For example, histogram intersection kernel (Barla et al., 2003) and χ-square kernel have achieved success in recognition tasks.

Several large-scale datasets (Deng et al., 2009; Xiao et al., 2010; Torralba et al., 2008a) have recently become the new benchmarks for recognition and retrieval research. ImageNet (Deng et al., 2009) is arguably the most important web-scale image dataset; currently, it contains around 15M images and 20K categories. All the images in that database have been manually verified and labeled with WordNet (Miller, 1995) categories. The SUN dataset (Xiao et al., 2010) is the largest dataset for scene recognition,with 397 categories and more than 100,000 images. The 80M tiny-images dataset (Torralba et al., 2008a) is the largest publicly available dataset, though it suffers from poor image quality and noisy annotations.

Once one moves to large-scale datasets like ImageNet, it becomes infeasible to train nonlinear SVM classifiers, which have obtained state-of-the-art results on the smaller-scale datasets of a decade ago. To overcome this problem, researchers have been developing efficient kernel-mapping functions (Rahimi and Recht, 2007; Perronnin et al., 2011; Maji and Berg, 2009). These methods enable us to efficiently train linear SVM in the approximated kernel space, significantly improving the scalability of large-scale recognition and enabling us to handle millions or even billions of images. Rahimi and Recht (2007) developed a random-projection-based mapping for approximating a Gaussian kernel. Maji and Berg (2009) proposed a per-dimensional binning approach to approximate the histogram intersection kernel that showed promising performance for object-detection applications. Perronnin et al. (2010a) studied explicit embeddings for large-scale supervised learning with BoW features. Gong and Lazebnik (2011a) compared several data-independent and data-dependent feature-mapping algorithms and showed

that a data-independent mapping usually works comparably to (if not better than) a data-dependent one. Both the nonlinear embedding methods and the feature coding schemes result in representations that are extremely high-dimensional.

There are also many novel visual descriptors (Perronnin and Dance, 2007; Perronnin et al., 2010b; Yang et al., 2009; Wang et al., 2010c) that directly incorporate nonlinearity into the feature encoding scheme, which eliminates the need for nonlinear feature mapping. For example, Yang et al. (2009) proposed the use of sparse coding, and subsequent work (Wang et al., 2010c) has further simplified the sparse coding step into a weighted kNN coding. We will use the *locality-constrained linear coding* (LLC) features in Wang et al. (2010c) and develop efficient binary coding methods for these feature vectors in Chapter 5. *Fisher vectors* (Perronnin and Dance, 2007; Perronnin et al., 2010b) directly encode nonlinear information through a Gaussian mixture model and have achieved state-of-the-art performance for recognition problems. Jégou et al. (2010) proposed a simplified version of FV, called *vector of locally aggregated descriptor* (VLAD). It has achieved comparable performance to FV but is much easier to implement and faster to construct. Since Jégou et al. (2010), many works have further improved VLAD (Arandjelović and Zisserman, 2013; Tolias et al., 2013; Delhumeau et al., 2013; Jégou and Chum, 2012). In particular, Arandjelović and Zisserman (2013) showed that proper normalization, nonlinear transformation of SIFT descriptors, and codebook adaptation can considerably improve the performance of VLAD. In addition, VLAD has been applied to other domains such as video retrieval (Revaud et al., 2013). The dimensionality of VLAD is still high, however (e.g., hundreds of thousands), and directly using it in real applications is

difficult. Several recent works have experimented with methods for compressing VLAD. For example, Perronnin et al. (2011) have investigated binary coding methods including spectral hashing (Weiss et al., 2008) and LSH (Andoni and Indyk, 2008). Jégou et al. (2010) proposed a joint dimensionality reduction and coding method that first performs PCA to reduce dimensionality, and then performs product quantization (Jégou et al., 2011) to convert the data to the compressed domain. Subsequent works by Jégou and Chum (2012) proposed several simple improvements, such as whitened PCA, which further reduce the dimensionality and improve performance.

Throughout this thesis, I will perform retrieval and recognition using a variety of visual descriptors. In Chapter 3, I will start with the relatively low-dimensional GIST descriptor (Oliva and Torralba, 2001) which is the most commonly used feature for evaluating binary coding methods. Accordingly, I will develop specific binary coding methods for BoW type data in Chapter 4, and for very high dimensional descriptors (e.g. VLAD and LLC) in Chapter 5.

2.2.1 Internet Computer Vision and Beyond

My thesis also has connections to approaches that model Internet images and accompanying text. I use the idea of mapping images and text to the same latent space (Gong et al., 2013a). This process is usually referred to as *cross-modal embedding*. *Canonical correlation analysis* (Hardoon et al., 2004) has been widely applied to learning such mappings or embeddings.. I will use CCA in Chapter 6 and demonstrate applications of binary codes to modeling Internet images and text for cross-modal image retrieval in

noisily-tagged Internet image collections. Another popular way to obtain an intermediate embedding space for images is by mapping them to outputs of a bank of concepts or attribute classifiers (Rasiwasia et al., 2007; Wang et al., 2009b; Farhadi et al., 2009). This is related to the idea of *classeme* representation (Torresani et al., 2010). I will also demonstrate an application that converts classemes to binary codes in Chapter 6. There are many other applications as well for which cleanly labeled training data may be scarce (Guillaumin et al., 2010; Quattoni et al., 2007; Wang et al., 2009a). This is a weakly supervised setting (as opposed to the fully supervised setting (Deng et al., 2009; Perronnin et al., 2010b)), where the idea is to use a large amount of noisy data to improve the classification results on a small, clean training set. The methods I present in Chapter 6 are also useful for learning from weakly tagged data.

For completeness, let me conclude by mentioning a few other Internet vision tasks that are not directly considered in my thesis but can serve as important applications for similarity-preserving binary codes. Multilabel image annotation (Carneiro et al., 2007; Li and Wang, 2008; Monay and Gatica-Perez, 2004) is one of the applications that rely heavily on large databases. It differs from image classification in that it tries to assign multiple keywords to one image instead of a single keyword. This makes the problem considerably more challenging than single-label image classification, so a different way to solve it is needed. Recently, a number of publications have focused on simple data-driven schemes based on retrieving database images similar to a query, and transferring the annotations from those images (Chua et al., 2009; Guillaumin et al., 2009; Makadia et al., 2008; Verma and Jawahar, 2012; Wang et al., 2008). Some discriminative ranking

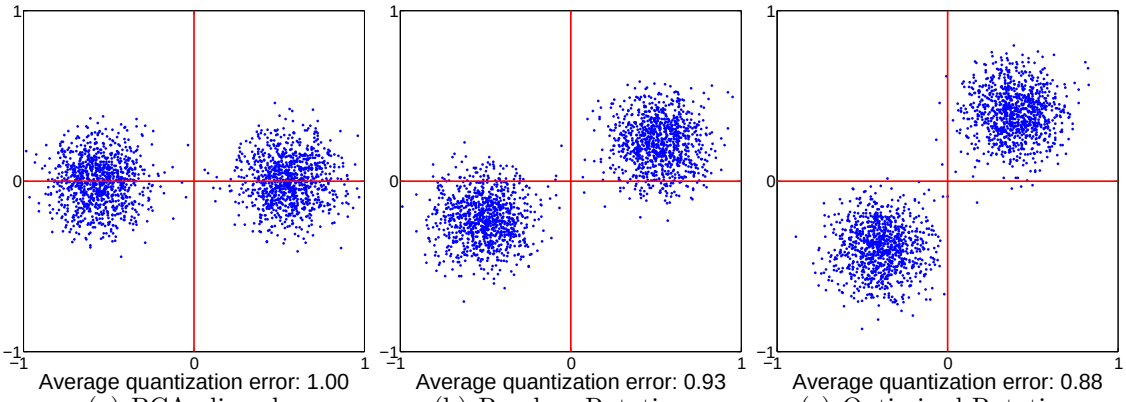
methods have also achieved success in annotation tasks involving Internet-scale datasets. Perhaps the largest-scale discriminative image-annotation system in the literature is the *WSABIE* (web scale annotation by image embedding) system proposed by Weston et al. (2011). It uses a stochastic gradient descent to optimize a ranking objective function and has been evaluated on datasets with 10M training examples. Because most of the annotation methods use nearest neighbor as an important component, binary codes can be used to improve their efficiency.

Finally, works in geometric vision use Internet images to reconstruct landmarks and popular tourist sites (Agarwal et al., 2011; Frahm et al., 2010). Among these, the work by Frahm et al. (2010) applied binary codes as an important component of its system.

CHAPTER 3: Iterative Quantization for Learning Compact Binary Codes

This Chapter details my Iterative Quantization (ITQ) algorithm, which will also serve as the foundation for the methods of Chapters 4 and 5. ITQ was mainly inspired by the work by Weiss et al. (2008) and Wang et al. (2010a), which both use principal component analysis (PCA) to reduce the dimensionality of the data prior to binary coding. However, since the variance of the data in each PCA direction is different (and in particular, higher-variance directions carry much more information), encoding each direction with the same number of bits is bound to produce poor performance.

In this chapter, I start with PCA-projected data and formulate the problem of learning a good binary code in terms of directly minimizing the quantization error of mapping this data to vertices of the binary hypercube. First, I show that simply applying a random orthogonal transformation to the PCA-projected data, as suggested by Jégou et al. (2010), already does a very good job of balancing the variance of different PCA directions. Next, I propose an alternating minimization approach (Section 3.1), dubbed Iterative Quantization (ITQ), for refining the initial orthogonal transformation to reduce quantization error. Figure 3.1 illustrates the idea behind this method. In my experiments (Section 3.2) it outperforms the methods of Raginsky and Lazebnik (2009); Wang et al. (2010a); Weiss et al. (2008). In addition, this method also works very well for hashing-based search. Finally, I will extend the ITQ approach to a kernel space (Section 3.3), and show how to incorporate nonlinearity in the learning process. This chapter will only



(a) PCA aligned. (b) Random Rotation. (c) Optimized Rotation.

Figure 3.1: Toy illustration of my ITQ method (see Section 3.1 for details). The basic binary encoding scheme is to quantize each data point to the closest vertex of the binary cube, $(\pm 1, \pm 1)$ (this is equivalent to quantizing points according to their quadrant). (a) The x and y axes correspond to the PCA directions of the data. Note that quantization assigns points in the same cluster to different vertices. (b) Randomly rotated data – the variance is more balanced and the quantization error is lower. (c) Optimized rotation found by ITQ – quantization error is lowest, and the partitioning respects the cluster structure.

concern the *unsupervised* setting, that does not use labeled data to learn binary codes. In Chapter 6, I will show applications that use label information to learn binary codes. This work was originally published in Gong and Lazebnik (2011b) and Gong et al. (2013c).

3.1 The ITQ Formulation

The proposed method first applies linear dimensionality reduction to the data, and then performs binary quantization in the resulting space. For the first step, discussed in Section 3.1.1, I follow the maximum variance formulation of Wang et al. (2010a); Weiss et al. (2008), which yields PCA projections. The major novelty of my method is in the second step (Section 3.1.2), where I try to preserve the locality structure of the projected data by rotating it so as to minimize the discretization error.

Let me first define notations. We have a set of n data points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, that form the rows of the data matrix $X \in \mathbb{R}^{n \times d}$. The goal is to learn a binary code matrix $B \in \{-1, 1\}^{n \times c}$, where c denotes the code length. Each row of B is denoted as $\boldsymbol{b}_i$. For each bit $k = 1, \ldots, c$, the binary encoding function is usually defined by $h_k(\boldsymbol{x}) = \text{sgn}(\boldsymbol{x}\boldsymbol{w}_k)$, where $\boldsymbol{w}_k$ is a column vector of hyperplane coefficients and

$$\text{sgn}(v) = \begin{cases} 1, & \text{if } v \geq 0; \\ -1, & \text{otherwise.} \end{cases}$$

For a matrix or a vector, $\text{sgn}(\cdot)$ will denote the result of element-wise application of the above function. Thus, we can write the entire encoding process as:

$$B = \text{sgn}(XW), \tag{3.1}$$

where $W \in \mathbb{R}^{d \times c}$ is the matrix with columns $\boldsymbol{w}_k$. We assume that the points are zero-centered, i.e., $\sum_{i=1}^{n} \boldsymbol{x}_i = 0$.

3.1.1 Dimensionality Reduction

Following the formulation of Wang et al. (2010a); Weiss et al. (2008), we want to produce an efficient code in which the variance of each bit is maximized and the bits are

pairwise uncorrelated. We can do this by maximizing the following objective function:

$$\begin{aligned} \mathcal{I}(W) &= \sum_k \mathrm{var}(h_k(\boldsymbol{x})) = \sum_k \mathrm{var}(\mathrm{sgn}(\boldsymbol{x}\boldsymbol{w}_k))\,, \\ & \frac{1}{n} B^T B = I\,. \end{aligned}$$

As shown in Wang et al. (2010a), the variance is maximized by encoding functions that produce exactly balanced bits, i.e., when $h_k(\boldsymbol{x}) = 1$ for exactly half of the data points and -1 for the other half. However, the requirement of exact balancedness makes the above objective function intractable. Adopting the same signed magnitude relaxation as in Wang et al. (2010a), we get the following continuous objective function:

$$\begin{aligned} \widetilde{\mathcal{I}}(W) &= \sum_k \mathbb{E}(\|\boldsymbol{x}\boldsymbol{w}_k\|_2^2) = \frac{1}{n} \sum_k \boldsymbol{w}_k^T X^T X \boldsymbol{w}_k \\ &= \frac{1}{n} \mathrm{tr}(W^T X^T X W)\,, \quad W^T W = I\,. \end{aligned} \tag{3.2}$$

The constraint $W^T W = I$ requires the hashing hyperplanes to be orthogonal to each other, which is a relaxed version of the requirement that code bits be pairwise decorrelated. This objective function is exactly the same as that of Principal Component Analysis (PCA). For a code of c bits, we obtain W by taking the top c eigenvectors of the data covariance matrix $X^T X$.

3.1.2 Binary Quantization

Let $\boldsymbol{v} \in \mathbb{R}^c$ be a vector in the projected space. It is easy to show (see below) that $\operatorname{sgn}(\boldsymbol{v})$ is the vertex of the hypercube $\{-1, 1\}^c$ closest to $\boldsymbol{v}$ in terms of Euclidean distance. The smaller the quantization loss $\|\operatorname{sgn}(\boldsymbol{v}) - \boldsymbol{v}\|^2$, the better the resulting binary code will preserve the original locality structure of the data. Now, going back to eq. (3.2), it is clear that if W is an optimal solution, then so is $\widetilde{W} = WR$ for any orthogonal $c \times c$ matrix R. Therefore, we are free to orthogonally transform the projected data $V = XW$ in such a way as to minimize the quantization loss

$$\mathcal{Q}(B, R) = \|B - VR\|_F^2 \,, \tag{3.3}$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

The idea of rotating the data to minimize quantization loss can be found in Jégou et al. (2010). However, the approach of Jégou et al. (2010) is based not on binary codes, but on product quantization with asymmetric distance computation (ADC). Unlike in my formulation, direct minimization of quantization loss for ADC is impractical, so Jégou et al. (2010) instead suggest solving an easier problem, that of finding a rotation (or, more precisely, an orthogonal transformation) to balance the variance of the different dimensions of the data. In practice, they find that a random rotation works well for this purpose. Based on this observation, a natural baseline for my method is given by initializing R to a random orthogonal matrix.

Beginning with the random initialization of R, I adopt a k-means-like procedure that

I call ITQ to find a local minimum of the quantization loss (3.3). In each iteration, each data point is first assigned to the nearest vertex of the binary hypercube, and then R is updated to minimize the quantization loss given this assignment. These two alternating steps are described in detail below.

Fix R and update B. Expanding (3.3), we have

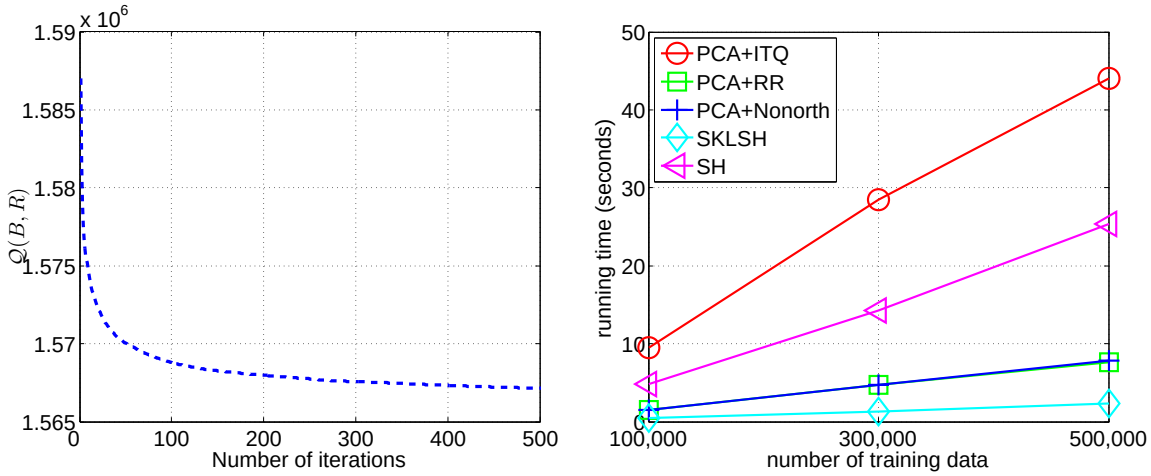
$$\begin{aligned} \mathcal{Q}(B,R) &= \|B\|_F^2 + \|V\|_F^2 - 2\,\mathrm{tr}(BR^TV^T) \\ &= nc + \|V\|_F^2 - 2\,\mathrm{tr}(BR^TV^T)\,. \end{aligned} \tag{3.4}$$

Because the projected data matrix $V = XW$ is fixed, minimizing (3.4) is equivalent to maximizing

$$\mathrm{tr}(BR^TV^T) = \sum_{i=1}^{n}\sum_{j=1}^{c} B_{ij}\widetilde{V}_{ij}\,,$$

where $\widetilde{V}_{ij}$ denote the elements of $\widetilde{V} = VR$. To maximize this expression with respect to B, I need to have $B_{ij} = 1$ whenever $\widetilde{V}_{ij} \geq 0$ and -1 otherwise. In other words, $B = \mathrm{sgn}(VR)$ as claimed in the beginning of this section.

Note that scaling the original data X by a constant factor changes the additive and multiplicative constants in (3.4), but does not affect the optimal value of B or R. Thus, while my method requires the data to be zero-centered, it does not care at all about the scaling. In other words, the quantization formulation (3.3) makes sense regardless of whether the average magnitude of the feature vectors is compatible with the radius of



(a) Quantization error for 32-bit.

(b) Training time for 32-bit code.

Figure 3.2: The behavior of ITQ quantization error and training time for a 32-bit code.

the binary cube.

Fix B and update R. For a fixed B, the objective function (3.3) corresponds to the classic Orthogonal Procrustes problem (Schonemann, 1966), in which one tries to find a rotation to align one point set with another. In my case, the two point sets are given by the projected data V and the target binary code matrix B. For a fixed B, (3.3) is minimized as follows: first compute the SVD of the $c \times c$ matrix $B^T V$ as $S\Omega\hat{S}^T = B^T V$ and then let $R = \hat{S}S^T$.

We alternate between updates to B and R for several iterations to find a locally optimal solution. The typical behavior of the error (3.3) is shown in Figure 3.2 (a). In practice, I have found that it is not necessary to iterate until convergence to achieve good performance, and I use 50 iterations for all experiments. Figure 3.2 (b) shows the training time for 32-bit code. I have found all the methods scale linearly with the number of images. Although my method takes a slightly longer time, it is still very practical.

My ITQ framework has been inspired by the approach of Yu and Shi (2003) for

discretizing relaxed solutions to multi-class spectral clustering, which is based on finding an orthogonal transformation of the continuous eigenvectors to bring them as close as possible to a discrete solution. One important difference between Yu and Shi (2003) and my approach is that Yu and Shi (2003) allows discretization only to the c orthogonal hypercube vertices with exactly one positive entry, while I use all the 2^c vertices as targets. This enables us to learn efficient codes that preserve the locality structure of the data.

3.2 Evaluation of Unsupervised Code Learning

3.2.1 Datasets

I evaluate my method on two subsets of the Tiny Images dataset (Torralba et al., 2008a). Both of these subsets come from Fergus et al. (2009). The first subset is a version of the CIFAR dataset (Krizhevsky, 2009), and it consists of 64,800 images that have been manually grouped into 11 ground-truth classes (airplane, automobile, bird, boat, cat, deer, dog, frog, horse, ship and truck). The second, larger subset consists of 580,000 Tiny Images. Apart from the CIFAR images, which are included among the 580,000 images, all the other images lack manually supplied ground truth labels, but they come associated with one of 388 Internet search keywords. In this section, I will use the CIFAR ground-truth labels to evaluate the semantic consistency of my codes.

The original Tiny Images are 32×32 pixels. I represent them with grayscale GIST descriptors (Oliva and Torralba, 2001) computed at 8 orientations and 4 different scales,

resulting in 320-dimensional feature vectors. Because my method (as well as many state-of-the-art methods) cannot use more bits than the original dimension of the data, the evaluation is limited to code sizes up to 256 bits.

3.2.2 Protocols and Baseline Methods

I follow two evaluation protocols widely used in recent papers (Raginsky and Lazebnik, 2009; Wang et al., 2010a; Weiss et al., 2008). The first one is to evaluate performance of nearest neighbor search using Euclidean neighbors as ground truth. As in Raginsky and Lazebnik (2009), a nominal threshold of the average distance to the 50th nearest neighbor is used to determine whether a database point returned for a given query is considered a true positive. Then, based on the Euclidean ground truth, I compute the recall-precision curve and the mean average precision (mAP), or the area under the recall precision curve. In particular, given the predefined ground truth, we rank the database items based on the learned codes, and compute recall and precision. To summarize the curves in a compact form, we directly report the mean average precision (mAP), which measures the area under precision recall curve. We first define average precision for one query,

$$\text{AveP} = \int_0^1 p(r)dr \approx \sum_{k=1}^{n} P(k)\Delta r(k) \tag{3.5}$$

where $P(k)$ and $r(k)$ are the values of the precision and recall. The mAP for Q different queries are computed as $\text{MAP} = \left(\sum_{q=1}^{Q} \text{AveP(q)}\right)/Q$. Second, I evaluate the semantic consistency of codes produced by different methods by using class labels as ground truth.

For this case, I report the averaged semantic precision of the top 500 ranked images for each query as in (Wang et al., 2010b). This measure is reporting the percentage of images having the same class label to the query within the 500 top ranked images. For all experiments, I randomly select 1000 points to serve as test queries. The remaining images form the training set on which the code parameters are learned, as well as the database against which the queries are performed. All the experiments reported in this chapter are averaged over five random training/test partitions.

I compare my ITQ method to three baseline methods that follow the basic hashing scheme $H(X) = \text{sgn}(X\widetilde{W})$, where the projection matrix $\widetilde{W}$ is defined in different ways:

1. **LSH:** $\widetilde{W}$ is a Gaussian random matrix (Andoni and Indyk, 2008). Note that in theory, this scheme has locality preserving guarantees only for unit-norm vectors. While I do not normalize the data to unit norm, I have found that it still works well as long as the data is zero-centered.
2. **PCA-Direct:** $\widetilde{W}$ is simply the matrix of the top c PCA directions. This baseline is included to show what happens when we do not rotate the PCA-projected data prior to quantization.
3. **PCA-RR:** $\widetilde{W} = WR$, where W is the matrix of PCA directions and R is a random orthogonal matrix. This is the initialization of ITQ, as described in Section 3.1.2.

I also compare ITQ to three state-of-the-art methods using code provided by the authors:

1. **SH** (Weiss et al., 2008): Spectral Hashing. This method is based on quantizing the values of analytical eigenfunctions computed along PCA directions of the data.

2. **SKLSH** (Raginsky and Lazebnik, 2009): This method is based on the random features mapping for approximating shift-invariant kernels (Rahimi and Recht, 2007). In Raginsky and Lazebnik (2009), this method is reported to outperform SH for code sizes larger than 64 bits. I use a Gaussian kernel with bandwidth set to the average distance to the 50th nearest neighbor as in Raginsky and Lazebnik (2009).
3. **PCA-Nonorth** (Wang et al., 2010a): Non-orthogonal relaxation of PCA. This method is reported in Wang et al. (2010a) to outperform SH. Note that instead of using semi-supervised PCA as in Wang et al. (2010a), the evaluation of this section uses standard unsupervised PCA since I assume there is no class label information available.

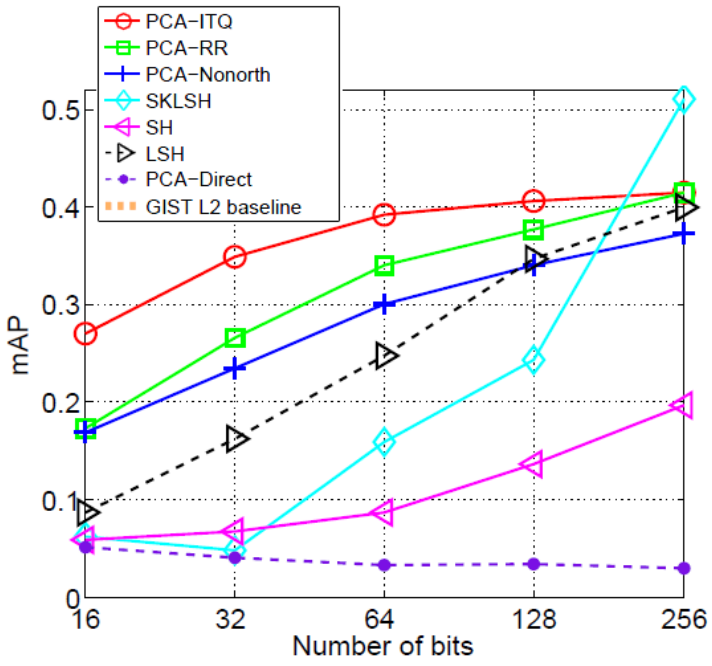
Note that of all the six methods above, LSH and SKLSH are the only ones that rely on randomized *data-independent* linear projections. All the other methods, including my PCA-RR and PCA-ITQ, use PCA (or a non-orthogonal relaxation of PCA) as an intermediate dimensionality reduction step.

3.2.3 Results on CIFAR Dataset

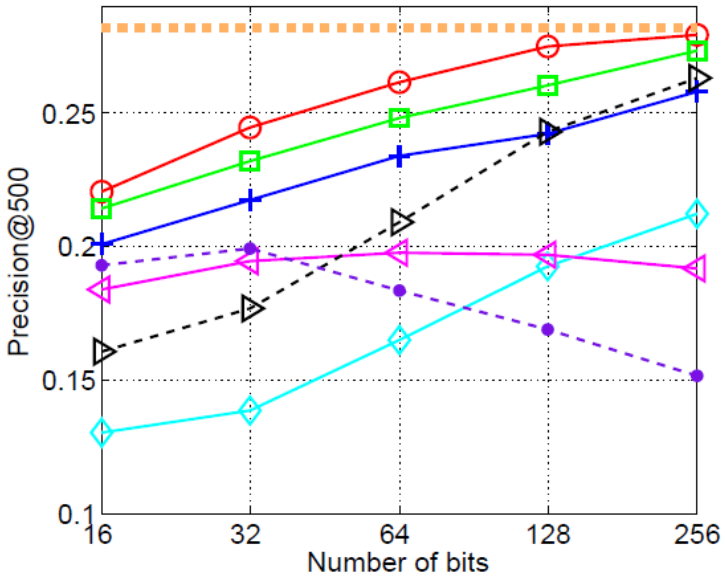
Figure 3.3(a) compares all the methods based on their mean average precision for Euclidean neighbor ground truth. Perhaps surprisingly, the natural baseline for my method, PCA-RR, already outperforms everything except PCA-ITQ for most code sizes. The only exception is SKLSH, which has a strongly upward trajectory and gets the best performance for a code size of 256. This behavior may be due to the theoretical convergence guarantee of SKLSH that when enough bits are assigned, Hamming distance

between binary codes approximates distance in the kernel space with high quality. LSH, which is data-independent just like SKLSH, also improves as the code size increases, and it almost reaches the performance level of PCA-RR at 256 bits. As for my proposed PCA-ITQ method, it consistently performs better than PCA-RR, although the advantage becomes smaller as the code size increases. Thus, adapting to the data distribution seems especially important when the code size is small. In particular, doing the ITQ refinement for a 64-bit code raises its performance almost to the level of the 256-bit PCA-RR code.

Figure 3.3(b) evaluates the semantic consistency of the codes using class labels as ground truth. For each method, it shows retrieval precision for the top 500 returned images as a function of code size. As in Figure 3.3(a), PCA-RR and PCA-ITQ outperform all the other methods, and PCA-ITQ has a small but consistent advantage over PCA-RR. There are some interesting differences in the performance of the other methods, however. Unlike in Figure 3.3(a), PCA-Direct works relatively well for the smallest code sizes (32 and 64 bits), while SKLSH works surprisingly poorly. This may be due to the fact that unlike most of the other methods, SKLSH does not rely on PCA. My results seem to indicate that PCA really helps to preserve semantic consistency for the smallest code sizes. Even at 256 bits, while SKLSH had by far the best performance for Euclidean neighbor retrieval, it lags behind most of the other methods in terms of class label precision. This underscores the fact that the best Euclidean neighbors are not necessarily the most semantically consistent, and that it is important to apply dimensionality reduction to the data in order to capture its class structure. Another observation worth making is that the two methods lacking a solid theoretical basis, namely PCA-Direct and SH, can



(a) Euclidean ground truth.



(b) Class label ground truth.

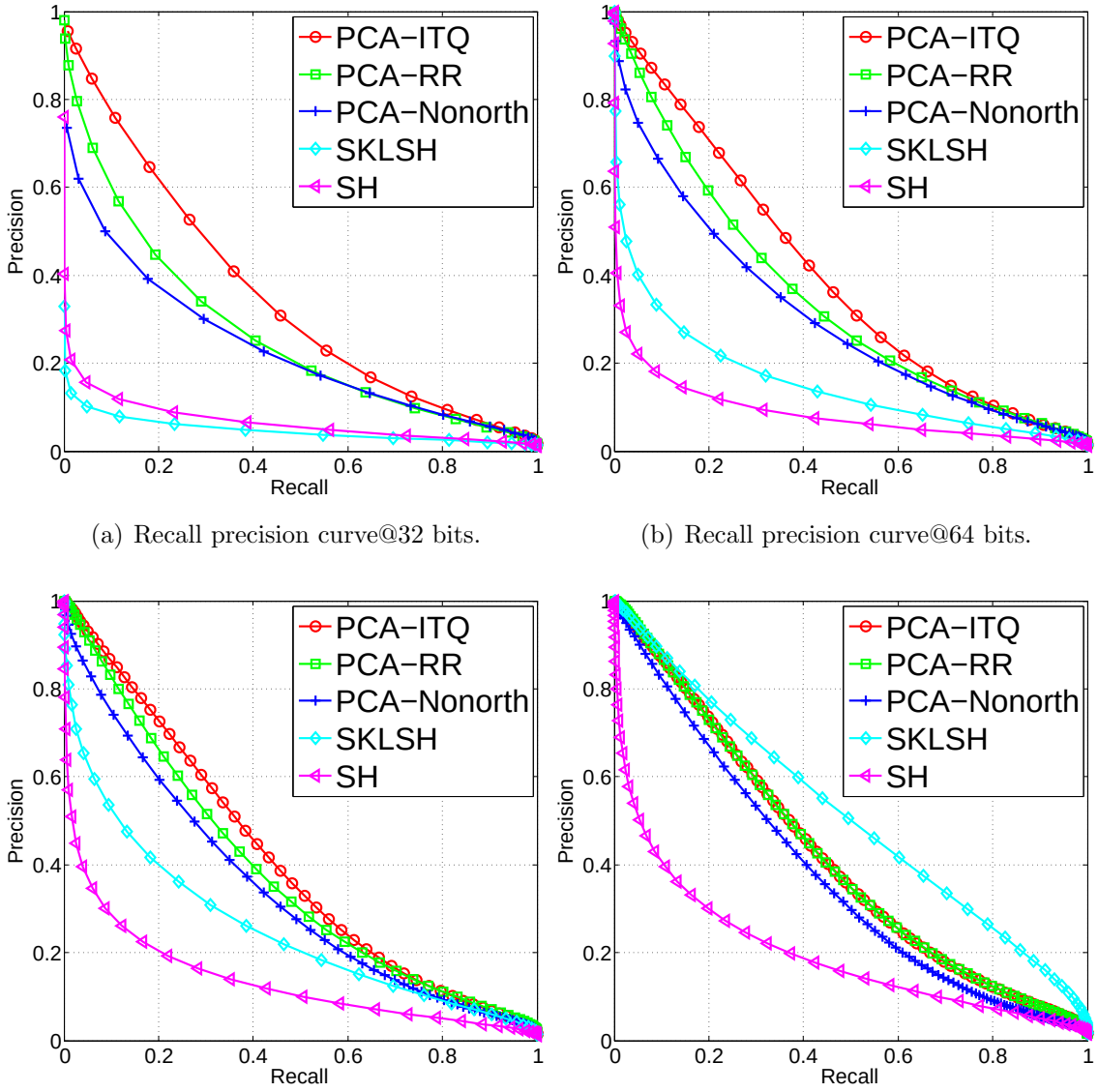
Figure 3.3: Comparative evaluation on CIFAR dataset. (a) Performance is measured by mean average precision (mAP) for retrieval using top 50 Euclidean neighbors of each query point as true positives. Refer to Figure 3.4 for the complete recall-precision curves for the state-of-the-art methods. (b) Performance is measured by the averaged precision of top p ranked images for each query where ground truth is defined by semantic class labels. Refer to Figure 3.5 for the complete class label precision curves for the state-of-the-art methods.

actually *decrease* in performance as the number of bits increases.

Figures 3.4 and 3.5 show complete recall-precision and class label precision curves corresponding to the summary numbers in Figures 3.3(a,b). To avoid clutter, these curves (and all the subsequent results reported in this chapter) omit the two baseline methods LSH and PCA-Direct. The complete curves confirm the trends seen in Figures 3.3 (a,b). What becomes especially apparent from looking at Figure 3.4(d) is that the data-dependent methods (PCA-Nonorth, PCA-RR, PCA-ITQ) seem to hit a ceiling of performance as code size increases, while the data-independent SKLSH method does not have a similar limitation (in fact, in the limit of infinitely many bits, SKLSH is guaranteed to yield exact Euclidean neighbors). Once again, the message seems to be that adapting binary codes to the data can give the biggest gain for small code sizes.

3.2.4 Results on 580,000 Tiny Images

Figure 3.6 and Figure 3.7 show precision-recall curves and mAP for Euclidean neighbor retrieval on the 580,000 Tiny Images. As explained in Section 3.2.1, there are no ground truth class labels for this dataset, so it is not possible to evaluate class label precision. The relative ordering of the different methods is largely consistent with results on CIFAR, with PCA-ITQ getting an even bigger performance advantage at small code sizes. Moreover, comparing Figure 3.6(d) with Figure 3.3(a), we can see that at 256 bits, SKLSH, PCA-Nonorth, PCA-RR, and PCA-ITQ converge to a higher level of mAP performance than on the smaller CIFAR dataset. This may be because the larger dataset samples the feature space more densely, making it easier to find good image matches.



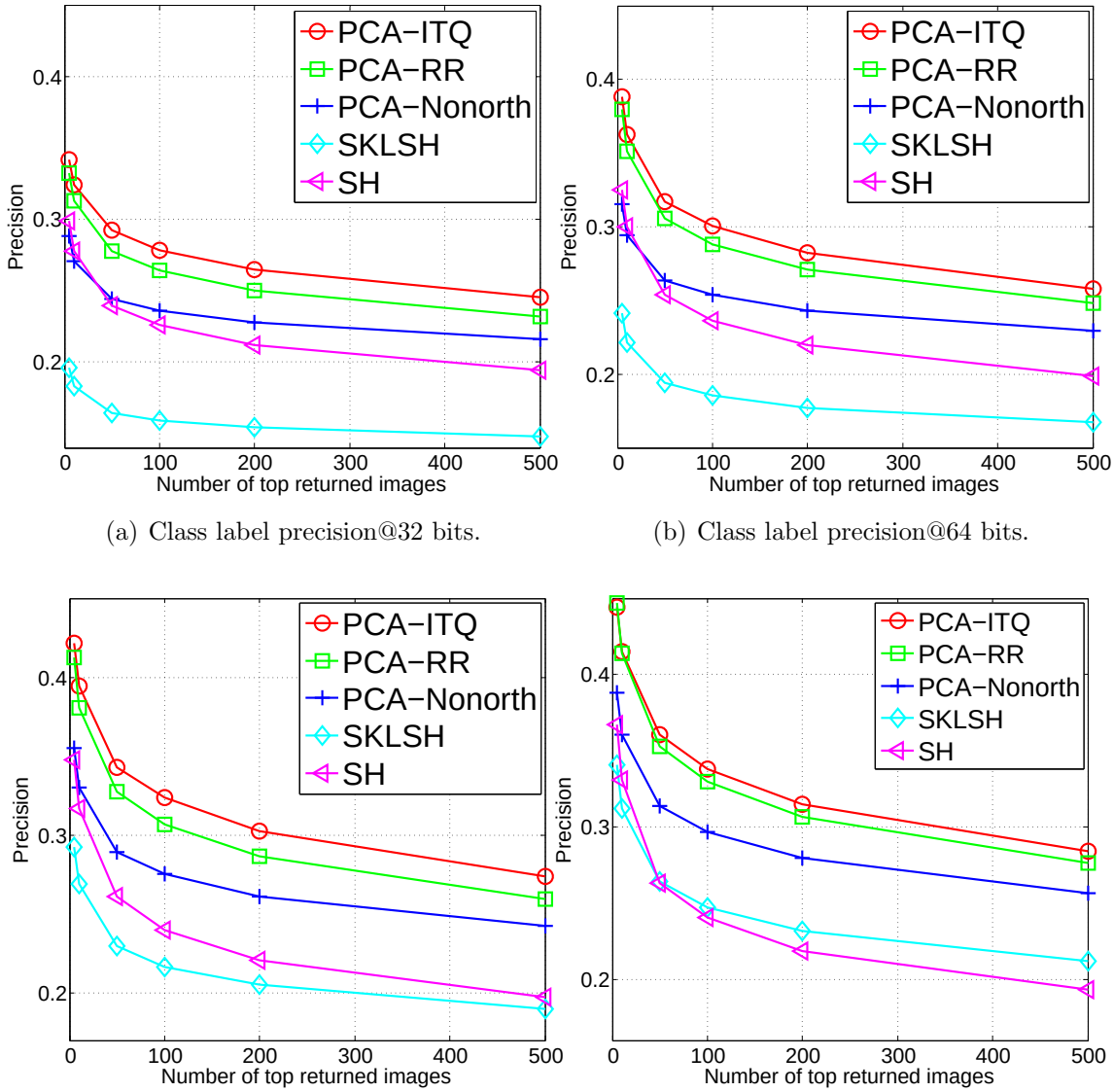
(a) Recall precision curve@32 bits.

(b) Recall precision curve@64 bits.

(c) Recall precision curve@128 bits.

(d) Recall precision curve@256 bits.

Figure 3.4: Comparison with state-of-the-art methods on CIFAR dataset using Euclidean neighbors as ground truth. Refer to Figure 3.3(a) for a summary of the mean average precision of these curves as a function of code size.



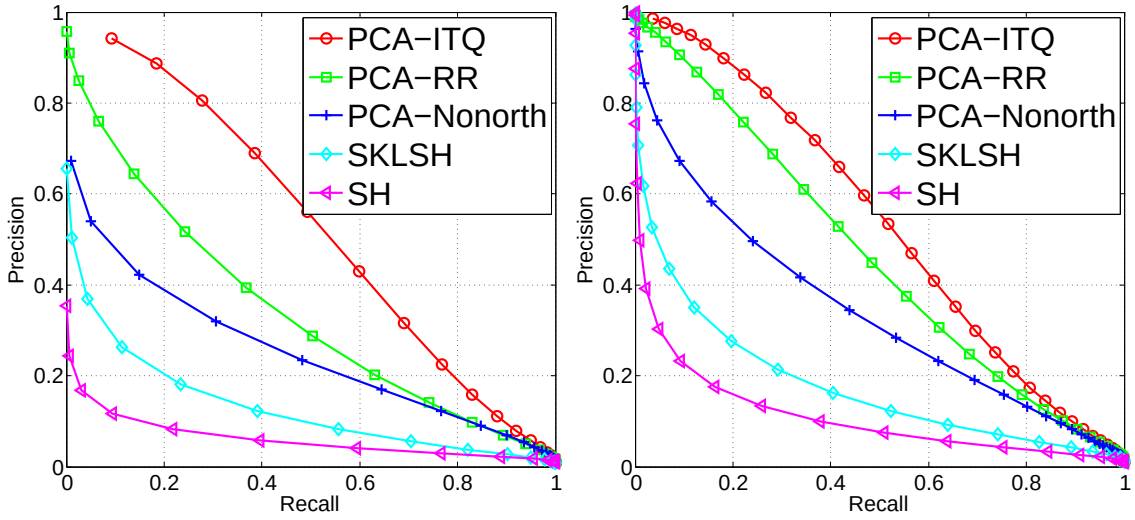
(a) Class label precision@32 bits.

(b) Class label precision@64 bits.

(c) Class label precision@128 bits.

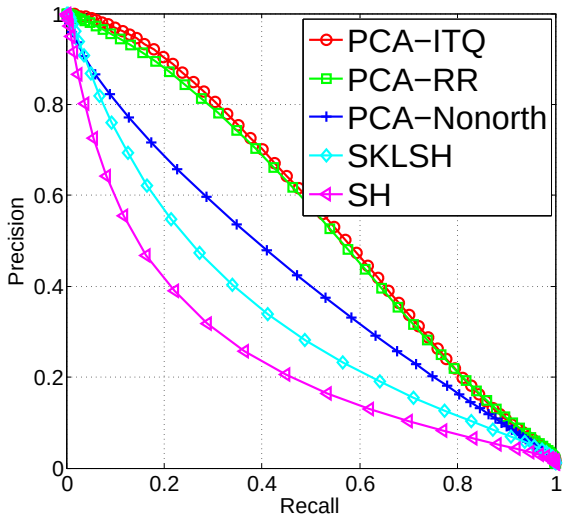
(d) Class label precision@256 bits.

Figure 3.5: Comparison with state-of-the-art methods on CIFAR dataset using semantic labels as ground truth. Figure 3.3(b) shows the summary plot of average precision as a function of code size.

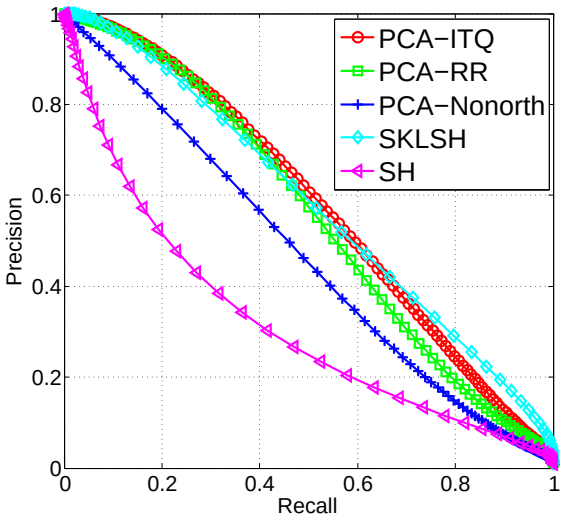


(a) Recall precision curve@32 bits.

(b) Recall precision curve@64 bits.



(c) Recall precision curve@128 bits.



(d) Recall precision curve@256 bits.

Figure 3.6: Results on the 580,000 Tiny Image subset. Ground truth is defined by Euclidean neighbors.

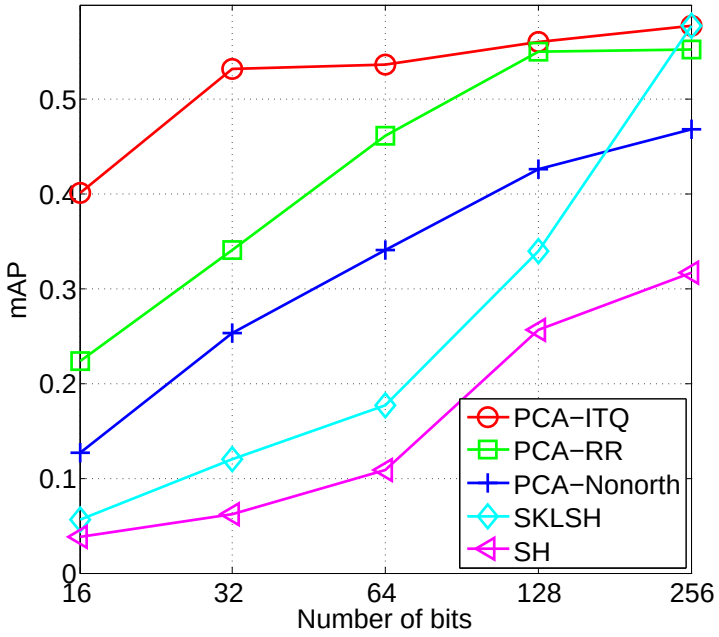


Figure 3.7: mAP on the 580,000 Tiny Image subset. Ground truth is defined by Euclidean neighbors.

Finally, Figure 3.8 shows image retrieval examples.

3.2.5 Evaluation of Hashing Performance

To fully realize the potential of binary codes for large-scale datasets, I would like to be able to use them for hashing or indexing as opposed to exhaustive search. For this, we would need a very small code (32 bits or less) to yield reasonably high precision and recall among retrieved points that lie within a Hamming distance of 0 to 2 from the query. Figure 3.9 shows the recall and precision of 8, 16 and 32-bit codes at Hamming radii $r = 0$, 1, and 2 for several methods on CIFAR and 580,000 Tiny Images with Euclidean neighbor ground truth. PCA-ITQ is almost always the best in terms of both recall and precision. Furthermore, comparing Figure 3.9(a-c) with (d-f), we can find that the recall of PCA-ITQ improves substantially when the dataset size is increased by an

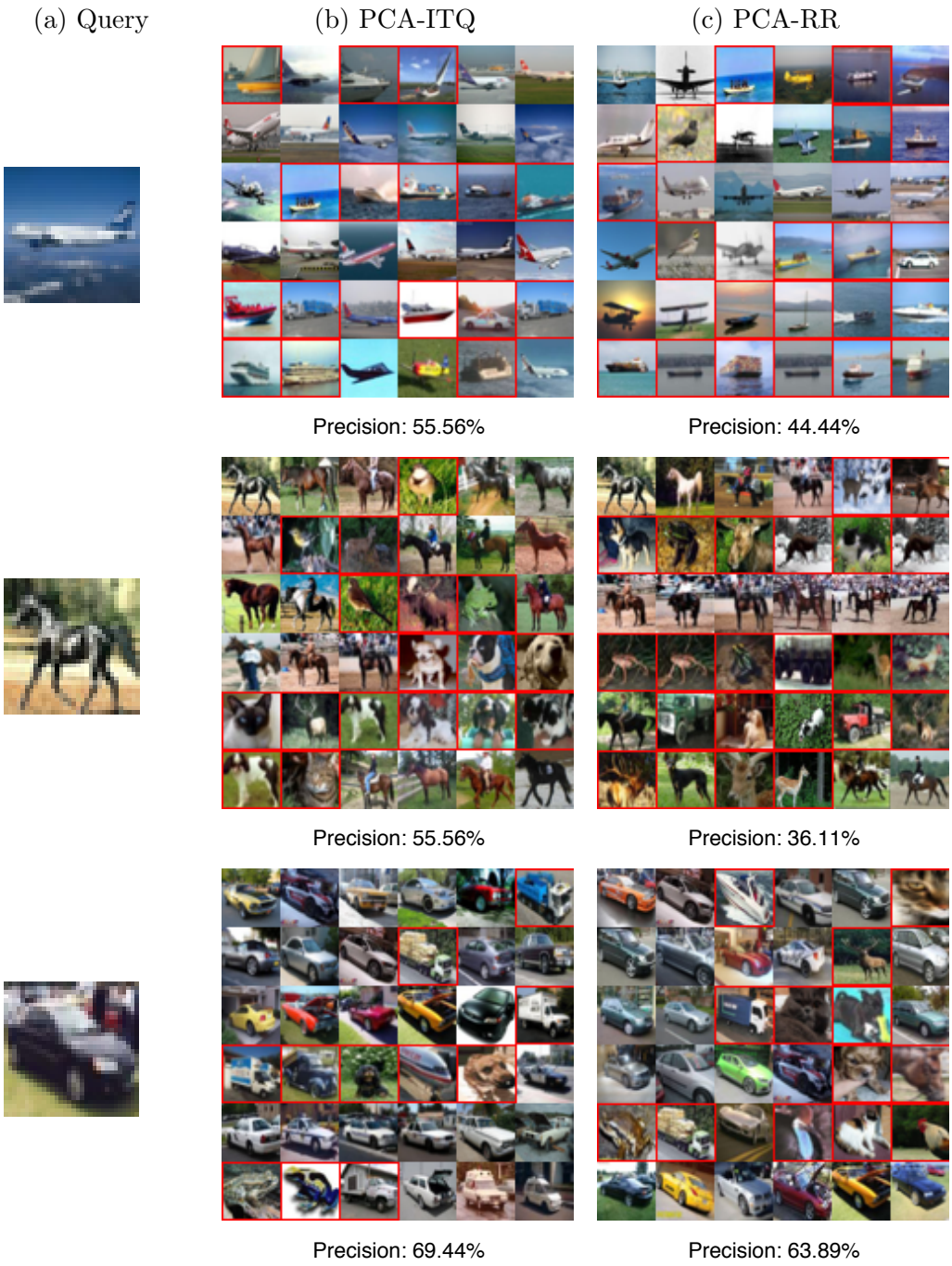
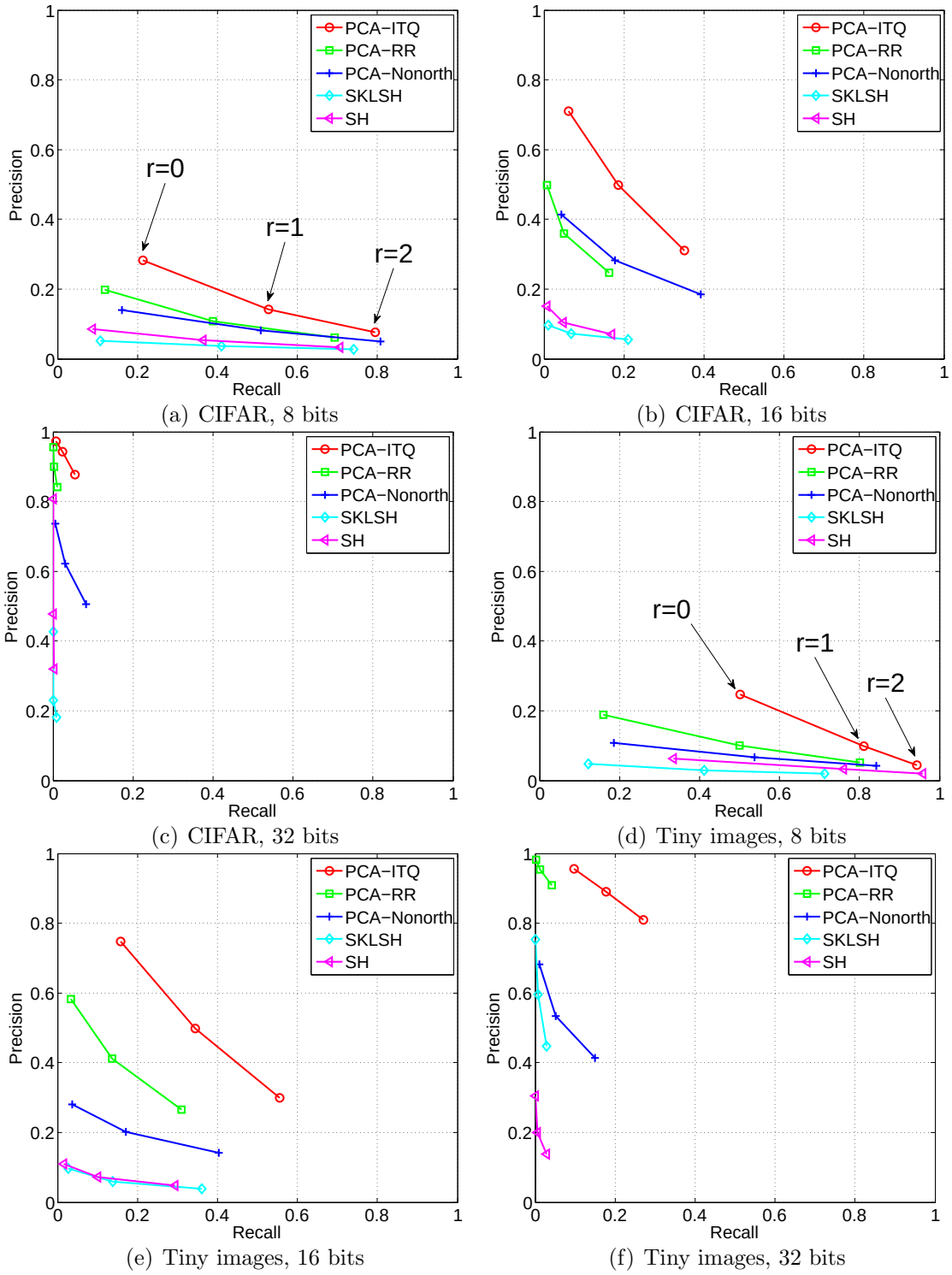


Figure 3.8: Image search (32-bit binary code) results on CIFAR. Red border means false positive. (a) is the query image, (b) and (c) shows the retrieved nearest neighbor images using binary codes produced by different methods.



(a) CIFAR, 8 bits

(b) CIFAR, 16 bits

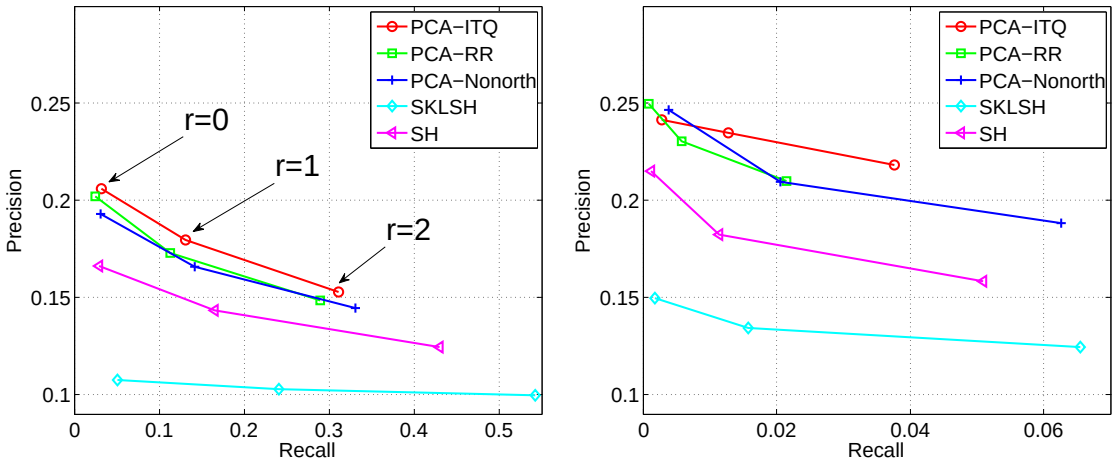
(c) CIFAR, 32 bits

(d) Tiny images, 8 bits

(e) Tiny images, 16 bits

(f) Tiny images, 32 bits

Figure 3.9: Hashing performance for different Hamming radii r with Euclidean neighbor ground truth. (a-c) CIFAR dataset; (d-f) 580,000 Tiny Images dataset.



(a) CIFAR, 8 bits (b) CIFAR, 16 bits

Figure 3.10: Hashing performance on the CIFAR dataset with class label ground truth. r is Hamming radius. Note the progressively decreasing recall from (a) to (c), and the different vertical scale in (c).

order of magnitude.

Figure 3.10 reports hashing performance in terms of class label retrieval. In this case, ITQ has good precision for 8 and 16 bits, but not for 32 bits. At first glance, this seems to contradict the result of Figure 3.3(b) where ITQ has the highest class label precision for the top 500 retrieved images for every code size. To explain this, note that some query points have many matches in a small Hamming radius (corresponding to dense areas of the feature space) and some queries have very few. The precision for a very small Hamming ball is significantly reduced by the queries with many matches since the matches usually do not all belong to the same class. Note that the same finding does not hold for Euclidean neighbor precision since the queries having many matches in a small Hamming ball also have many Euclidean neighbors. In any case, the semantic label recall of all the methods is dismally low, indicating that a lot more research needs to be done before similarity-preserving binary codes become truly useful for hashing.

3.3 ITQ with a Kernel Embedding

3.3.1 Random Fourier Features

A big limitation of PCA is that it can only capture linear structure in the data. In order to introduce nonlinearity into the embedding process, I can use kernel PCA (KPCA) (Scholkopf et al., 1997). Finding the KPCA embedding for n feature vectors involves computing the $n \times n$ kernel matrix and performing eigendecomposition on it. However, for large-scale image databases, these operations are prohibitively expensive, so I have to resort to approximation schemes.

In this section, I am particularly interested in the Gaussian kernel, whose radius can be used to control the neighborhood size for nearest-neighbor search. To approximate the Gaussian kernel $K(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/(2\sigma^2))$, I can use the explicit *random Fourier feature* (RFF) mapping (Rahimi and Recht, 2007). For a data point $\boldsymbol{x}$, each coordinate of this mapping is given by

$$\Phi_{\boldsymbol{w},b}(\boldsymbol{x}) = \sqrt{2}\cos(\boldsymbol{x}\boldsymbol{w} + b)\,,$$

where the random projection vector $\boldsymbol{w}$ is drawn from Normal$(0, \frac{1}{\sigma^2}I)$ and b is drawn from Unif$[0, 2\pi]$. A D-dimensional embedding is given by

$$\Phi^D(\boldsymbol{x}) = [\Phi_{\boldsymbol{w}_1,b_1}(\boldsymbol{x}), \Phi_{\boldsymbol{w}_2,b_2}(\boldsymbol{x}), \ldots, \Phi_{\boldsymbol{w}_D,b_D}(\boldsymbol{x})].$$

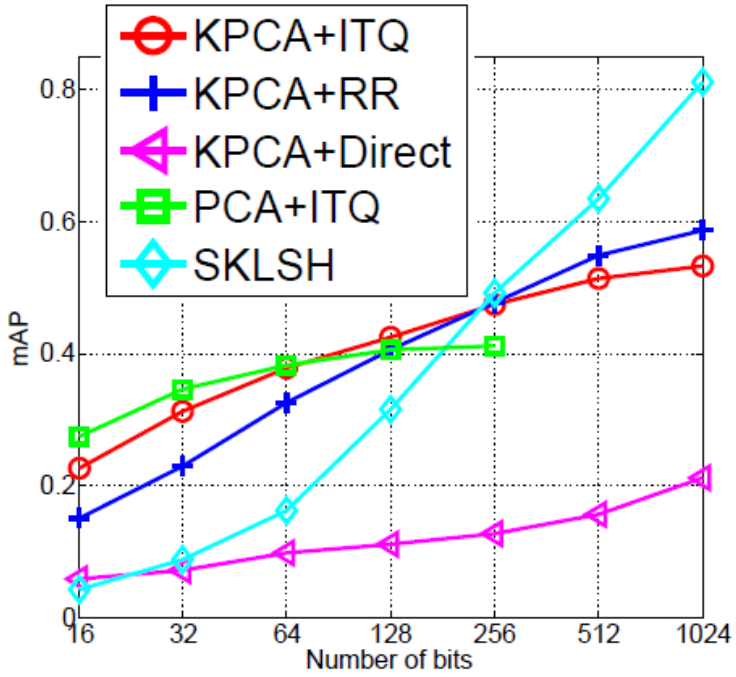
The inner product of the mapped data approximates the Gaussian kernel as $K(\boldsymbol{x}, \boldsymbol{y}) \approx$

$\Phi^D(\boldsymbol{x})\Phi^D(\boldsymbol{y})^T$. When D goes to infinity, the mapping becomes exact. In my experiments, I use $D = 3,000$. After the random Fourier mapping, I simply perform linear PCA on top of $\Phi^D(X)$ to obtain an approximate KPCA embedding for the data. Given the points in my dataset, I first transform them using RFF, then perform KPCA to reduce the dimensionality, and finally binarize the data in the same way as before.

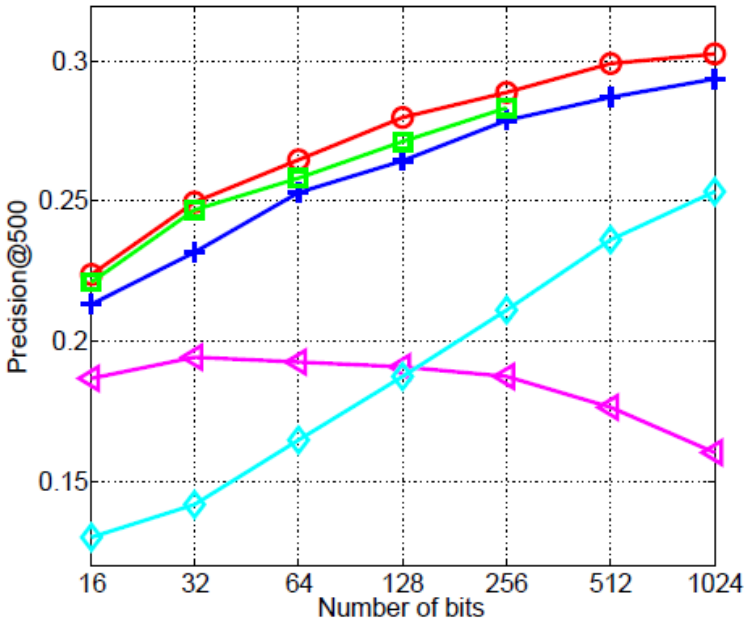
Note that while I solely focus on approximation of the Gaussian kernel, there also exist explicit mappings for other popular kernels (Maji et al., 2008; Perronnin et al., 2010a).

3.3.2 Results

I first compare KPCA with ITQ to other baseline methods, including linear PCA with ITQ. Results for Euclidean neighbor retrieval are reported in Figure 3.11 (a). One of the advantages of RFF is that it allows us to learn binary codes whose dimension is higher than that of the original data (320 in the experiments so far). Thus, I report results up to 1,024 bits for the kernel methods, and for PCA-ITQ, I report results up to 256 bits. For RFF, I set the radius of the Gaussian kernel to the average distance to the 50th nearest neighbor. From Figure 3.11(a), in terms of Euclidean neighbor retrieval, KPCA-ITQ starts to have an advantage over PCA-ITQ beginning with 128 bits (though I have found that it is possible to tune the radius of the Gaussian kernel to match the performance of PCA-ITQ for shorter codes), and KPCA-RR seems to work the best for the longest code sizes. Figure 3.11 (b) reports performance in terms of class label retrieval. For this case, KPCA-ITQ is consistently better than PCA-ITQ or KPCA-RR.



(a) Euclidean ground truth.



(b) Class label ground truth.

Figure 3.11: Comparative evaluation of kernel ITQ on CIFAR dataset. (a) Performance is measured by mean average precision (mAP) for Euclidean neighbor retrieval. (b) Performance is measured by the averaged precision of top 500 ranked images for each query where ground truth is defined by semantic class labels.

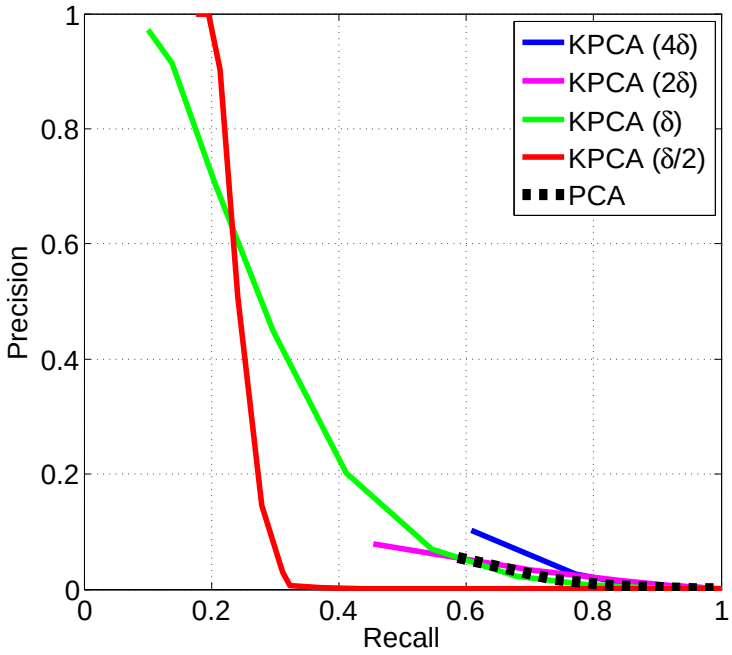


Figure 3.12: Performance of kernel ITQ (32 bits) with a ground truth radius δ equal to half the distance to the 50th nearest neighbor. The numbers in parentheses show the value of the kernel width σ used in the RFF mapping.

It is interesting to compare KPCA-RR/ITQ and SKLSH (Raginsky and Lazebnik, 2009), since the latter is also based on RFF. To obtain a c-bit code, SKLSH first computes a c-dimensional RFF embedding of the data and then quantizes each dimension after adding a random threshold. By contrast, my KPCA-based methods start with a 3,000-dimensional RFF embedding and then use PCA to reduce it to top c dimensions. Based on the results of Figure 3.11, this process makes an especially big difference for class label precision.

Finally, I want to demonstrate the effect of the radius σ of the Gaussian kernel in the RFF mapping. Intuitively, by tuning this parameter, it can control how the binary codes approximate the neighbors: a smaller radius can give us better approximation for very close neighbors, while a larger one can give us codes that better reflect the global

structure of the data. To demonstrate this, I use a ground-truth radius δ defined as *half* the distance to the 50th nearest neighbor. This results in only very close points (near duplicates) being defined as ground truth neighbors. Figure 3.12 shows recall and precision of KPCA-ITQ for $\sigma = [\delta/2,\ \delta,\ 2\delta,\ 4\delta]$. The best performance is obtained for δ and $\delta/2$, which match the desired neighborhood size the most closely. By contrast, KPCA-ITQ with a larger radius, or PCA-ITQ, which does not have a radius parameter, work very poorly in this regime.

Let me summarize the benefits of using the RFF embedding in combination with ITQ. First, it allows us to use more bits than original feature dimensions to get better code accuracy. Second, it significantly improves class label precision, especially when combined with CCA. Third, it has a tunable radius parameter that can be changed to obtain much better performance on tasks such as near-duplicate image retrieval.

3.4 Discussion

This chapter presents the "Iterative Quantization" work that proposes to learn a rotation to reduce quantization error, so as to learn better binary codes. This chapter makes several useful findings. First, I show that the performance of PCA-based binary coding schemes can be greatly improved by simply rotating the PCA-projected data. Even a random rotation already works better than more elaborate schemes like non-orthogonal relaxation (Wang et al., 2010a). Second, I demonstrate an iterative quantization method for refining this rotation that fits the problem very naturally, judging from the significant improvement it confers in practice. Unlike many other papers, I evaluate the performance

of my method *both* in terms of preserving Euclidean neighbors in the feature space and in terms of retrieving semantically similar images. This evaluation reveals that methods that do very well on the first task, like SKLSH, can actually do quite poorly on the second one, thus underscoring the importance of looking at both criteria. The proposed ITQ approach serves as a basis of my subsequent works. I will present extensions to this approach in Chapter 4 and Chapter 5.

CHAPTER 4: Angular Quantization for Histogram Data

In many vision and text-related applications, it is common to represent data as a Bag of Words (BoW) (Salton and McGill, 1986; Csurka et al., 2004; Sivic and Zisserman, 2003), or a vector of counts or frequencies, which contains only non-negative entries. Furthermore, cosine of the angle between vectors is typically used as a similarity measure for such data. This chapter presents an extension to the ITQ approach presented in Chapter 3 by exploring the special data distribution of histograms.

A popular binary coding method for cosine similarity is based on Locality Sensitive Hashing (LSH) (Charikar, 2002; Andoni and Indyk, 2008), but it does not take advantage of the non-negative nature of histogram data. As I will show in the experiments, the accuracy of LSH is limited for most real-world data. Min-wise Hashing is another popular method which is designed for non-negative data (Broder, 1997; Li and Konig, 2011; Li et al., 2011; Shrivastava and Li, 2012). However, it is appropriate only for Jaccard distance (defined as the size of the intersection divided by the size of the union of the sample sets) and it also does not result in binary codes. Special clustering algorithms have been developed for data sampled on the unit hypersphere, but they also do not lead to binary codes (Banerjee et al., 2005). To the best of my knowledge, this chapter describes the first work that specifically learns binary codes for non-negative data with cosine similarity.

The proposed technique, called Angular Quantization-based Binary Coding (AQBC),

works by quantizing each data point to the vertex of the positive orthant of the binary hypercube with which it has the smallest angle. This is very similar to ITQ, but the difference is the definition of landmark points. For ITQ, we define landmark points in $\{-1, +1\}^d$ while in this approach, we define landmark points in $\{0, +1\}^d$. Such a simple change, will make binary thresholding significantly harder, as we can no longer simply take the sign of data to obtain its binary codes as we did in ITQ. Furthermore, the number of these quantization centers or landmarks is exponential in the dimensionality of the data. While this yields a low-distortion quantization of a point, it would be computationally infeasible to perform traditional nearest-neighbor quantization as in (Banerjee et al., 2005) with such a large number of centers. Moreover, even at query time, finding the nearest center for a given point would be prohibitively expensive. Instead, I present in Section 4.1.1 a very efficient method to find the nearest landmark for a point, i.e., the vertex of the binary hypercube with which it has the smallest angle. Since the basic form of my quantization method does not take data distribution into account, I further propose in Sections 4.1.2 and 4.1.3 a learning algorithm that linearly transforms the data before quantization to reduce the angular distortion. The experimental evaluation is presented in Section 4.2, followed by a discussion in Section 4.3. This work was originally published in Gong et al. (2012).

4.1 Angular Quantization-based Binary Codes

4.1.1 Data-independent Binary Codes

We assume we are given a database containing n d-dimensional points $\{\boldsymbol{x}_i\}_{i=1}^n$ as defined in Chapter 3. I first address the problem of computing a d-bit binary code of an input vector $\boldsymbol{x}_i$. A c-bit code for $c < d$ will be described later in Sec. 4.1.2. For angle-preserving quantization, I define a set of quantization centers or landmarks by projecting the vertices of the binary hypercube $\{0,1\}^d$ onto the unit hypersphere. This construction results in $2^d - 1$ landmark points for d-dimensional data. Note that the vertex with all 0's is excluded as its norm is 0, which is not permissible in eq. (4.1). An illustration of the proposed quantization model is given in Figure 4.1. Given a point $\boldsymbol{x}$ on the hypersphere, one first finds its nearest[1] landmark $\boldsymbol{v}_i$, and the binary encoding for $\boldsymbol{x}_i$ is simply given by the binary vertex $\boldsymbol{b}_i$ corresponding to $\boldsymbol{v}_i$. Since in terms of angle from a point, both $\boldsymbol{b}_i$ and $\boldsymbol{v}_i$ are equivalent, I will use the term landmark for either $\boldsymbol{b}_i$ or $\boldsymbol{v}_i$ depending on the context.

One of the main characteristics of the proposed model is that the number of landmarks grows exponentially with d, and for many practical applications d can easily be in thousands or even more. On the one hand, having a huge number of landmarks is preferred as it can provide a fine-grained, low-distortion quantization of the input data, but on the other hand, it poses the formidable computational challenge of efficiently finding the nearest landmark (and hence the binary encoding) for an arbitrary input point. Note

[1]In terms of angle or Euclidean distance, which are equivalent for unit-norm data.

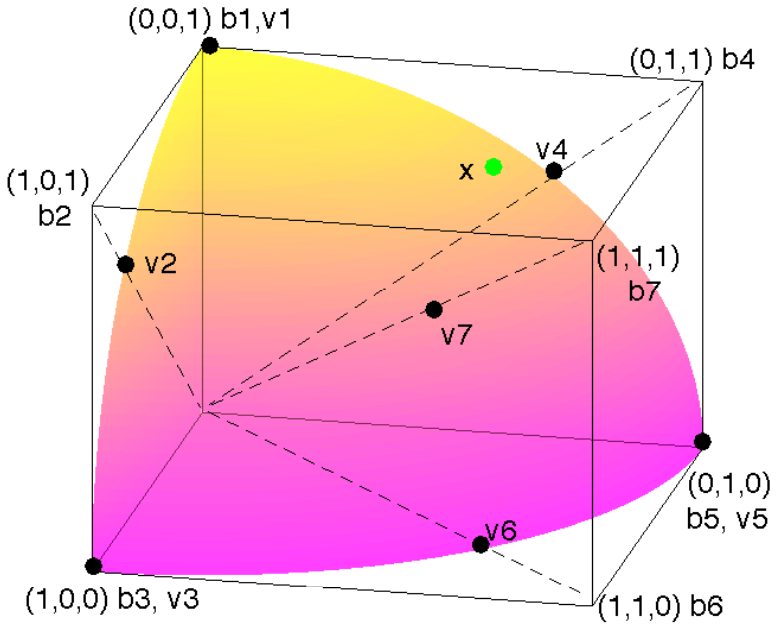


Figure 4.1: An illustration of my quantization model in 3D. Here $\boldsymbol{b}_i$ is a vertex of the unit cube and $\boldsymbol{v}_i$ is its projection on the unit sphere. Points $\boldsymbol{v}_i$ are used as the landmarks for quantization. To find the binary code of a given data point $\boldsymbol{x}$, I first find its nearest landmark point $\boldsymbol{v}_i$ on the sphere, and the correponding $\boldsymbol{b}_i$ gives its binary code ($\boldsymbol{v}_4$ and $\boldsymbol{b}_4$ in this case).

that performing brute-force nearest-neighbor search might even be slower than nearest-neighbor retrieval from the original database! To obtain an efficient solution, I propose to take advantage of the special structure of our set of landmarks, which are given by the projections of binary vectors onto the unit hypercube. The nearest landmark of a point $\boldsymbol{x}$, or the binary vertex having the smallest angle with $\boldsymbol{x}$, is given by

$$\hat{\boldsymbol{b}} = \arg\max_{\boldsymbol{b}} \frac{\boldsymbol{b}^T \boldsymbol{x}}{\|\boldsymbol{b}\|_2} \qquad \text{s. t.} \quad \boldsymbol{b} \in \{0,1\}^d. \tag{4.1}$$

This is an integer programming problem but its global maximum can be found very efficiently as I show in the lemma below. The corresponding algorithm is presented in Algorithm 1.

Algorithm 1: Finding the nearest binary landmark for a point on the unit hypersphere.

Input: point $\boldsymbol{x}$ on the unit hypersphere.
Output: $\hat{\boldsymbol{b}}$, binary vector having the smallest angle with $\boldsymbol{x}$.

1. Sort the entries of $\boldsymbol{x}$ in descending order as $x_{(1)}, \ldots, x_{(d)}$.
2. **for** $k = 1, \ldots, d$
3. **if** $\boldsymbol{x}_{(k)} = 0$ **break**.
4. Form binary vector $\boldsymbol{b}_k$ whose elements are 1 for the k largest positions in $\boldsymbol{x}$, 0 otherwise.
5. Compute $\psi(\boldsymbol{x}, k) = (\boldsymbol{x}^T \boldsymbol{b}_k)/\|\boldsymbol{b}_k\|_2 = \left(\sum_{j=1}^{k} x_{(j)}\right)/\sqrt{k}$.
6. end **for**
7. Return $\boldsymbol{b}_k$ corresponding to $m = \arg\max_k \psi(\boldsymbol{x}, k)$.

Lemma 1 *The globally optimal solution of the integer programming problem in eq. (4.1) can be computed in $O(d \log d)$ time. Further, for a sparse vector with s non-zero entries, it can be computed in $O(s \log s)$ time.*

Proof: Since $\boldsymbol{b}$ is a d-dimensional binary vector, its norm $\|\boldsymbol{b}\|_2$ can have at most d different values, i.e., $\|\boldsymbol{b}\|_2 \in \{\sqrt{1}, \ldots, \sqrt{d}\}$. We can separately consider the optimal solution of eq. (4.1) for each value of the norm. Given $\|\boldsymbol{b}\|_2 = \sqrt{k}$ (i.e., $\boldsymbol{b}$ has k ones), eq. (4.1) is maximized by setting to one the entries of $\boldsymbol{b}$ corresponding to the largest k entries of $\boldsymbol{x}$. Since $\|\boldsymbol{b}\|_2$ can take on d distinct values, we need to evaluate eq. (4.1) at most d times, and find the k and the corresponding $\hat{\boldsymbol{b}}$ for which the objective function is maximized (see Algorithm 1 for a detailed description of the algorithm). To find the largest k entries of $\boldsymbol{x}$ for $k = 1, \ldots, d$, We need to sort all the entries of $\boldsymbol{x}$, which takes $O(d \log d)$ time, and checking the solutions for all k is linear in d. Further, if the vector $\boldsymbol{x}$ is sparse with only s non-zero elements, it is obvious that the maximum of eq. (4.1) is achieved when k varies from 1 to s. Hence, one needs to sort only the non-zero entries of $\boldsymbol{x}$, which takes $O(s \log s)$ time and checking all possible solutions is linear in s. □

Now I study the properties of the proposed quantization model. The following lemma

helps to characterize the angular resolution of the quantization landmarks.

Lemma 2 *Suppose* $\boldsymbol{b}$ *is an arbitrary binary vector with Hamming weight* $\|\boldsymbol{b}\|_1 = m$*, where* $\|\cdot\|_1$ *is the* L_1 *norm. Then for all binary vectors* $\boldsymbol{b}'$ *that lie at a Hamming radius* r *from* $\boldsymbol{b}$*, the cosine of the angle between* $\boldsymbol{b}$ *and* $\boldsymbol{b}'$ *is bounded by* $\left[\sqrt{\frac{m-r}{m}}, \sqrt{\frac{m}{m+r}}\right]$.

Proof: Since $\|\boldsymbol{b}\|_1 = m$, there are exactly m ones in $\boldsymbol{b}$ and the rest are zeros, and $\boldsymbol{b}'$ has exactly r bits different from $\boldsymbol{b}$. To find the lower bound on the cosine of the angle between $\boldsymbol{b}$ and $\boldsymbol{b}'$, we want to find a $\boldsymbol{b}'$ such that $\frac{\boldsymbol{b}^T\boldsymbol{b}'}{\sqrt{\|\boldsymbol{b}\|_1}\sqrt{\|\boldsymbol{b}'\|_1}}$ is maximized. It is easy to see that this will happen when $\boldsymbol{b}'$ has exactly $m-r$ ones in common positions with $\boldsymbol{b}$ and the remaining entries are zero, i.e., $\|\boldsymbol{b}'\|_1 = m-r$ and $\boldsymbol{b}^T\boldsymbol{b}' = m-r$. This gives the lower bound of $\sqrt{\frac{m-r}{m}}$. Similarly, the upper bound can be obtained when $\boldsymbol{b}'$ has all ones at the same locations as $\boldsymbol{b}$, and additional r ones, i.e., $\|\boldsymbol{b}'\|_1 = m+r$ and $\boldsymbol{b}^T\boldsymbol{b}' = m$. This yields the upper bound of $\sqrt{\frac{m}{m+r}}$. □

We can understand this result as follows. The Hamming weight m of each binary vertex corresponds to its position in space. When m is low, the point is closer to the boundary of the positive orthant and when m is high, it is closer to the center. The above lemma implies that for landmark points on the boundary, the Voronoi cells are relatively coarse, and cells become progressively denser as one moves towards the center. Thus the proposed set of landmarks non-uniformly tessellates the surface of the positive orthant of the hypersphere. Figure 4.2 shows the lower and upper bounds on angle for various m and r. It is clear that for relatively large m, the angle between different landmarks is very small, thus providing dense quantization even for large r. To get good performance,

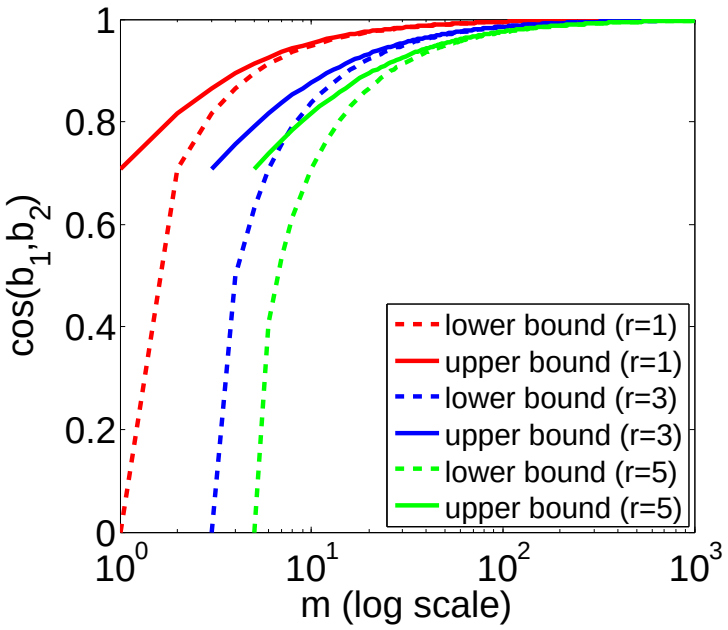


Figure 4.2: Bound on cosine of angle between a binary vertex $\boldsymbol{b}_1$ with Hamming weight m, and another vertex $\boldsymbol{b}_2$ at a Hamming distance r from $\boldsymbol{b}_1$. See Lemma 2 for details.

the distribution of the data should be such that a majority of the points fall closer to landmarks with higher m.

Algorithm 1 constitutes the core of my proposed angular quantization method, but it has several limitations: (i) it is data-independent, and thus cannot adapt to the data distribution to control the quantization error; (ii) it cannot control m which, based on my analysis, is critical for low quantization error; (iii) it can only produce a d-bit code for d-dimensional data, and thus cannot generate shorter codes. In the following section, I present a learning algorithm to address the above issues.

4.1.2 Learning Data-dependent Binary Codes

I start by addressing the first issue of how to adapt the method to the given data to minimize the quantization error. Similarly to the ITQ method described in Chapter

3, I would like to align the data to a pre-defined set of quantization landmarks using a rotation, because rotating the data does not change the angles – and, therefore, the similarities – between the data points. Later in this section, I will present an objective function and an optimization algorithm to accomplish this goal, but first, by way of motivation, I would like to illustrate how applying even a random rotation to a typical frequency/count vector can affect the Hamming weight m of its angular binary code.

Since we are mostly interested in modeling histogram data that is counts and frequencies, I propose to use Zipf's distribution to generate synthetic data for illustration purposes, as most real world histogram data follows this distribution (Manning and Schütze, 1999; Zipf, 1935). Suppose, for a data vector $\boldsymbol{x}$, the sorted entries $x_{(1)}, \ldots, x_{(d)}$ follow Zipf's law, i.e., $x_{(k)} \propto 1/k^q$, where k is the index of the entries sorted in descending order, and s is the power parameter that controls how quickly the entries decay. The effective m for $\boldsymbol{x}$ depends directly on the power q: the larger q is, the faster the entries of $\boldsymbol{x}$ decay, and the smaller m becomes. More germanely, for a fixed s, applying a random rotation R to $\boldsymbol{x}$ makes the distribution of the entries of the resulting vector $R^T\boldsymbol{x}$ more uniform and raises the effective m. Figure 4.3 (a) plots the sorted entries of $\boldsymbol{x}$ generated from Zipf's law with $s = 0.8$. Based on Algorithm 1, I compute the scaled cumulative sums $\psi(\boldsymbol{x}, k) = \sum_{j=1}^{k} \frac{x_{(j)}}{\sqrt{k}}$, which are shown in Figure 4.3 (b). Here the optimal $m = \arg\max_k \psi(\boldsymbol{x}, k)$ is relatively low ($m = 2$). In Figure 4.3 (c), I randomly rotate the data and show the sorted values of $R^T\boldsymbol{x}$, which become more uniform. Finally, in Figure 4.3 (d), I show $\psi(R^T\boldsymbol{x}, k)$. The Hamming weight m after this random rotation becomes much higher ($m = 25$). This effect is typical: the average of m over 1000 random

rotations for this example is 27.36. Thus, even randomly rotating the data tends to lead to finer Voronoi cells and reduced quantization error. Next, it is natural to ask whether we can *optimize* the rotation of the data to increase the cosine similarities between data points and their corresponding binary landmarks.

I seek a $d \times d$ orthogonal transformation R such that the sum of cosine similarities of each transformed data point $R^T \boldsymbol{x}_i$ and its corresponding binary landmark $\boldsymbol{b}_i$ is maximized. Note that after rotation, $R^T \boldsymbol{x}_i$ may contain negative values but this does not affect the quantization since the binarization technique described in Algorithm 1 effectively suppresses the negative values to 0. Let $B \in \{0,1\}^{d \times n}$ denote a matrix whose columns are given by the $\boldsymbol{b}_i$. Then the objective function for my optimization problem is given by

$$\mathcal{Q}(B, R) = \arg\max_{B,R} \sum_{i=1}^{n} \frac{\boldsymbol{b}_i^T}{\|\boldsymbol{b}_i\|_2} R^T \boldsymbol{x}_i \qquad \text{s. t.} \quad \boldsymbol{b}_i \in \{0,1\}^d, \quad R^T R = I_d, \tag{4.2}$$

where I_d denotes the $d \times d$ identity matrix.

The above objective function still yields a d-bit binary code for d-dimensional data, while in many real-world applications, a low-dimensional binary code may be preferable. To generate a c-bit code where $c < d$, I can learn a $d \times c$ projection matrix R with orthogonal columns by optimizing the following modified objective function:

$$\mathcal{Q}(B, R) = \arg\max_{B,R} \sum_{i=1}^{n} \frac{\boldsymbol{b}_i^T}{\|\boldsymbol{b}_i\|_2} \frac{R^T \boldsymbol{x}_i}{\|R^T \boldsymbol{x}_i\|_2} \qquad \text{s. t.} \quad \boldsymbol{b}_i \in \{0,1\}^c, \quad R^T R = I_c. \tag{4.3}$$

Note that to minimize the angle after a low-dimensional projection (as opposed to a rota-

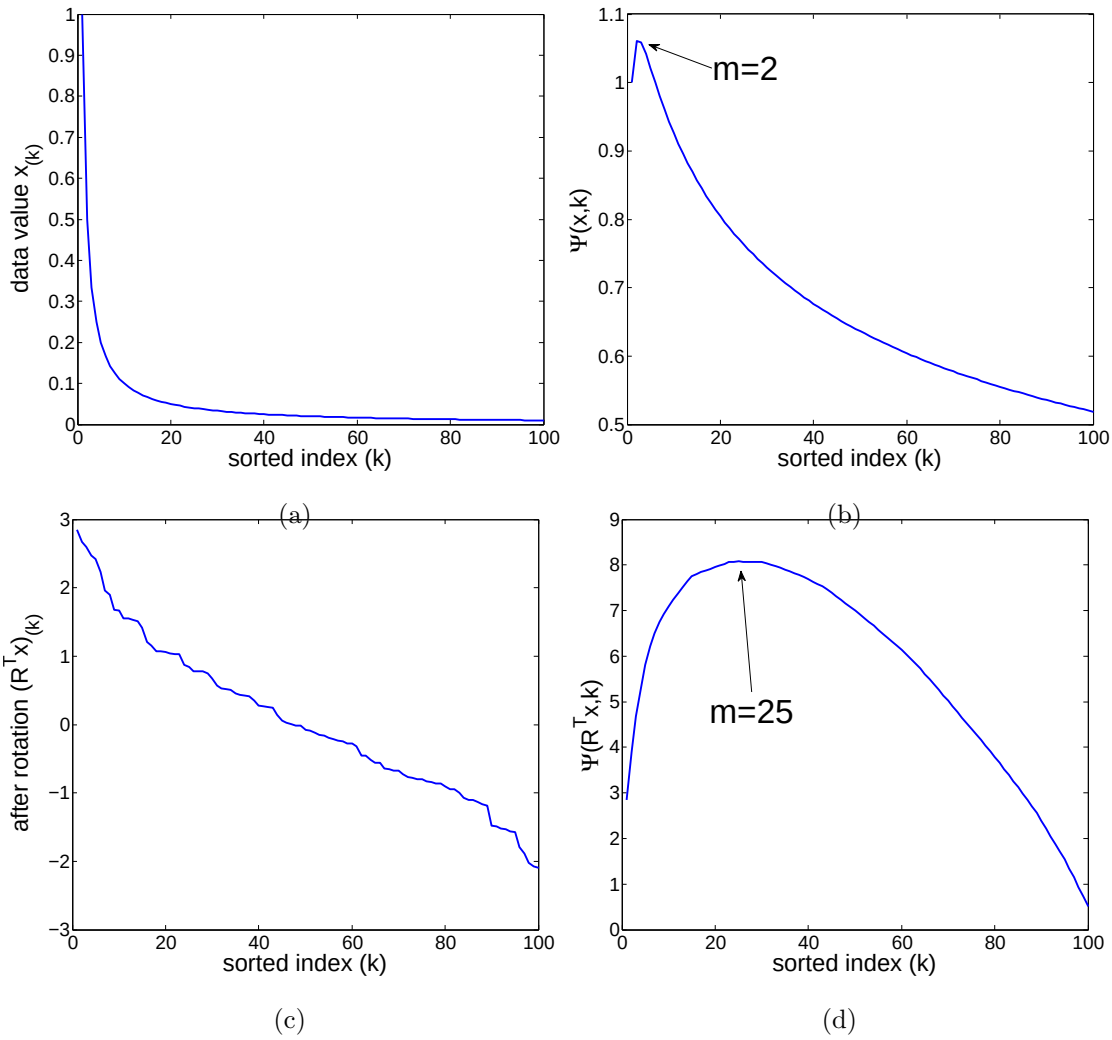


Figure 4.3: Effect of rotation on Hamming weight m of the landmark corresponding to a particular vector. (a) Sorted vector elements $x_{(k)}$ following Zipf's law with $q = 0.8$; (b) Scaled cumulative sum $\psi(\boldsymbol{x}, k)$; (c) Sorted vector elements after random rotation; (d) Scaled cumulative sum $\psi(R^T\boldsymbol{x}, k)$ for the rotated data. See text for discussion.

tion), the denominator of the objective function contains $\|R^T \boldsymbol{x}_i\|_2$ since after projection $\|R^T \boldsymbol{x}_i\|_2 \neq 1$. However, adding this new term to the denominator makes the optimization problem hard to solve. I propose to relax it by optimizing the linear correlation instead of the angle:

$$\mathcal{Q}(B, R) = \arg\max_{B,R} \sum_{i=1}^{n} \frac{\boldsymbol{b}_i^T}{\|\boldsymbol{b}_i\|_2} R^T \boldsymbol{x}_i \qquad \text{s. t.} \quad \boldsymbol{b}_i \in \{0,1\}^c, \quad R^T R = I_c. \tag{4.4}$$

This is similar to eq. (4.2) but the geometric interpretation is slightly different: I am now looking for a projection matrix R to map the d-dimensional data to a lower-dimensional space such that after the mapping, the data has high linear correlation with a set of landmark points lying on the lower-dimensional hypersphere. Section 4.2 will demonstrate that this relaxation works quite well in practice.

4.1.3 Optimization

The objective function in (4.4) can be written more compactly in a matrix form:

$$\mathcal{Q}(\widetilde{B}, R) = \arg\max_{\widetilde{B},R} \operatorname{Tr}(\widetilde{B}^T R^T X) \qquad \text{s. t.} \quad R^T R = I_c, \tag{4.5}$$

where $\operatorname{Tr}(\cdot)$ is the trace operator, $\widetilde{B}$ is the $c \times n$ matrix with columns given by $\boldsymbol{b}_i/\|\boldsymbol{b}_i\|_2$, and X is the $d \times n$ matrix with columns given by $\boldsymbol{x}_i$. This objective is nonconvex in $\widetilde{B}$ and X jointly. To obtain a local maximum, I use a simple alternating optimization procedure as follows.

(1) Fix R, update $\widetilde{B}$. For a fixed R, eq. (4.5) becomes separable in $\boldsymbol{x}_i$, and I can solve

for each $\boldsymbol{b}_i$ separately. Here, the individual sub-problem for each $\boldsymbol{x}_i$ can be written as

$$\hat{\boldsymbol{b}_i} = \arg\max_{\boldsymbol{b}_i} \frac{\boldsymbol{b}_i^T}{\|\boldsymbol{b}_i\|_2}(R^T\boldsymbol{x}_i). \tag{4.6}$$

Thus, given a point $\boldsymbol{y}_i = R^T\boldsymbol{x}_i$ in c-dimensional space, I want to find the vertex $\boldsymbol{b}_i$ on the c-dimensional hypercube having the smallest angle with $\boldsymbol{y}_i$. To do this, I use Algorithm 1 to find $\boldsymbol{b}_i$ for each $\boldsymbol{y}_i$, and then normalize each $\boldsymbol{b}_i$ back to the unit hypersphere: $\widetilde{\boldsymbol{b}}_i = \boldsymbol{b}_i/\|\boldsymbol{b}_i\|_2$. This yields each column of $\widetilde{B}$. Note that the $\widetilde{B}$ found in this way is the global optimum for this subproblem.

(2) Fix $\widetilde{B}$, update R. When $\widetilde{B}$ is fixed, I want to find

$$\hat{R} = \arg\max_R \mathrm{Tr}(\widetilde{B}^T R^T X) = \arg\max_R \mathrm{Tr}(R^T X \widetilde{B}^T) \quad \text{s. t.} \quad R^T R = I_c. \tag{4.7}$$

This is a well-known problem and its global optimum can be obtained by polar decomposition (Chen et al., 2011). Namely, I take the SVD of the $d \times c$ matrix $X\widetilde{B}^T$ as $X\widetilde{B}^T = USV^T$, let U_c be the first c singular vectors of U, and finally obtain $R = U_c V^T$.

The above formulation involves solving two sub-problems in an alternating fashion. The first subproblem is an integer program, and the second one has non-convex orthogonal constraints. However, in each iteration the global optimum can be obtained for each sub-problem as discussed above. So, each step of the alternating method is guaranteed to increase the objective function. In practice, one needs only a few iterations (less than five) for the method to converge. The optimization procedure is initialized by first generating a random binary matrix by making each element 0 or 1 with probability

$\frac{1}{2}$, and then normalizing each column to unit norm. Note that the optimization is also computationally efficient. The first subproblem takes $O(nc \log c)$ time while the second one takes $O(dc^2)$. This is linear in data dimension d, which enables us to handle very high-dimensional feature vectors.

4.1.4 Computation of cosine Similarity between Binary Codes

We use the cosine similarity between binary codes introduced in Section 2.1.2 as the similarity measure. The cosine similarity between two binary codes can also be very efficiently computed using CPU efficient operations.

4.2 Experiments

4.2.1 Datasets and Protocols

To test the effectiveness of the proposed Angular Quantization-based Binary Codes (AQBC) method, I have conducted experiments on two image datasets and one text dataset. The first image dataset is **SUN**, which contains 142,169 natural scene images (Xiao et al., 2010). Each image is represented by a 1000-dimensional bag of visual words (BoW) feature vector computed on top of dense SIFT descriptors. The BoW vectors are power-normalized by taking the square root of each entry, which has been shown to improve performance for recognition tasks (Perronnin et al., 2011).

The second dataset contains 122,530 images from **ImageNet** (Deng et al., 2009), each represented by a 5000-dimensional vector of locality-constrained linear coding (LLC) features (Wang et al., 2010c), which are improved versions of BoW features. Dense SIFT

is also used as the local descriptor in this case.

The third dataset is **20 Newsgroups**,[2] which contains 18,846 text documents and 26,214 words. Tf-idf weighting is used for each text document BoW vector. The feature vectors for all three datasets are sparse, non-negative, and normalized to unit L_2 norm. Due to this, Euclidean distance directly corresponds to the cosine similarity as $\text{dist}^2 = 2 - 2\,\text{sim}$. Therefore, in the following, I will talk about similarity and distance interchangeably.

To perform evaluation on each dataset, I randomly sample and fix 2000 points as queries, and use the remaining points as the "database" against which the similarity searches are run. For each query, I define the ground truth neighbors as all the points within the radius determined by the average distance to the 50th nearest neighbor in the dataset, and plot precision-recall curves of database points ordered by decreasing similarity of their binary codes with the query. This methodology is similar to our protocol in Chapter 3, and to that of other recent works (Raginsky and Lazebnik, 2009; Weiss et al., 2008). Since my AQBC method is unsupervised, I compare with several state-of-the-art *unsupervised* binary coding methods: Locality Sensitive Hashing (LSH) (Charikar, 2002), Spectral Hashing (Weiss et al., 2008), ITQ (Chapter 3), Shift-invariant Kernel LSH (SKLSH) (Raginsky and Lazebnik, 2009), and Spherical Hashing (SPH) (Heo et al., 2012). Although these methods are designed to work with the Euclidean distance, they can be directly applied here since all the vectors have unit norm. As in Chapter 3, we also evaluate the semantic precision for binary codes in this section. For this evaluation,

[2]http://people.csail.mit.edu/jrennie/20Newsgroups

we use ITQ as the main baseline. We follow the experimental protocols mentioned above, and use precision@50 to evaluate different methods. This metric reports the percentage of images having the same class label to each query image within its 50 nearest neighbors.

4.2.2 Results on SUN and ImageNet

Precision-recall curves for Euclidean neighbor retrieval on the SUN dataset are shown in Figure 4.4. For all the code lengths (from 64 to 1000 bits), my method (AQBC) performs better than other state-of-the-art methods. For a relatively large number of bits, SKLSH works much better than other methods, while still being worse than mine. It is interesting to verify how much we gain by using the learned data-dependent quantization instead of the data-independent naive version (Sec. 4.1.1). Since the naive version can only learn a d-bit code (1000 bits in this case), its performance (AQBC naive) is shown only in Figure 4.4 (c). The performance is much worse than that of the learned codes, which clearly shows that adapting quantization to the data distribution is important in practice. Figure 4.5 shows results for Euclidean neighbor retrieval on ImageNet. On this dataset, the strongest competing method is ITQ. For a relatively low number of bits (e.g., 64), AQBC and ITQ are comparable, but AQBC has a more clear advantage as the number of bits increases. This is because for fewer bits, the Hamming weight (m) of the binary codes tends to be small resulting in larger distortion error as discussed in Sec. 4.1.1. I also found the SPH (Heo et al., 2012) method works well for relatively dense data, while it does not work very well for high-dimensional sparse data.

Next, I evaluate the semantic retrieval precision on both datasets, and report the

code size	16	32	64	128	256	512	1024
ITQ	2.64	3.30	4.22	5.28	5.64	5.91	6.22
AQBC	2.22	2.95	3.58	4.36	5.00	5.97	5.46

Table 4.1: Semantic retrieval results (precision@50) on SUN dataset.

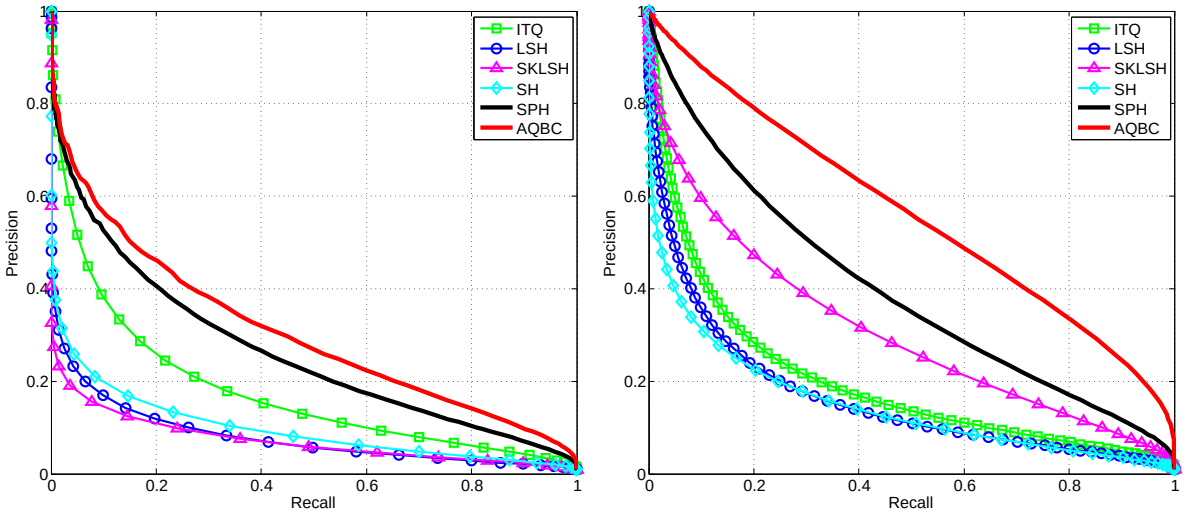
code size	16	32	64	128	256	512	1024
ITQ	6.37	8.33	11.09	12.95	14.09	14.84	15.15
AQBC	5.78	7.44	9.61	11.68	13.12	13.86	14.43

Table 4.2: Semantic retrieval results (precision@50) on ImageNet120K dataset.

results in Table 4.1 and Table 4.2. We can find the proposed AQBC method is usually slightly worse than ITQ. This suggests that using cosine similarity might not have a very significant advantage in terms of semantic retrieval on these datasets. The best distance preserving approach might not be able to lead to the best semantic retrieval performance, and it depends on specific applications.

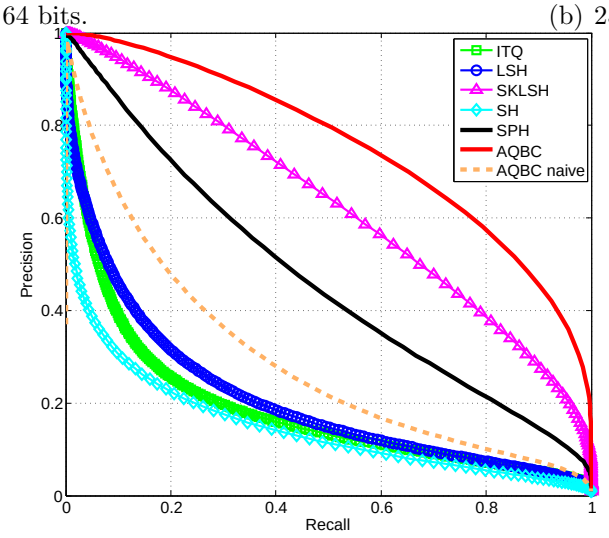
4.2.3 Results on 20 Newsgroups

Euclidean neighbor retrieval results on the text features (Figure 4.6) are consistent with those on the image features. Because the text features are the sparsest and have the highest dimensionality, I would like to verify whether learning the projection R helps in choosing landmarks with larger m as conjectured in Sec. 4.1.2. The average empirical distribution over sorted vector elements for this data is shown in Figure 4.7 (a) and the scaled cumulative sum in Figure 4.7 (b). It is clear that vector elements have a rapidly decaying distribution, and the quantization leads to codes with low m implying higher quantization error. Figure 4.7 (c) shows the distribution of entries of vector $R^T\boldsymbol{x}$, which



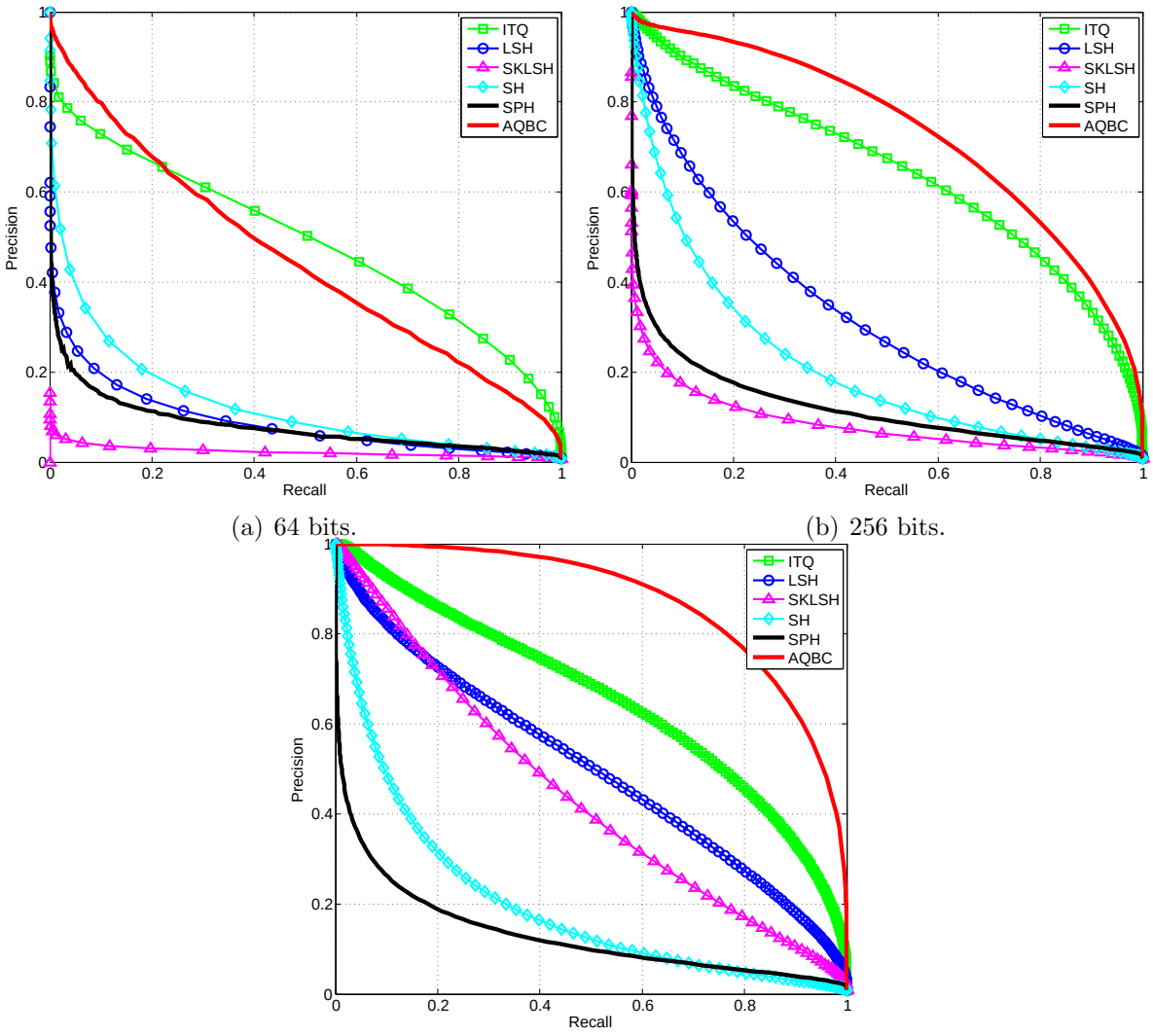
(a) 64 bits.

(b) 256 bits.



(c) 1000 bits.

Figure 4.4: Precision-recall curves for different methods on the SUN dataset.

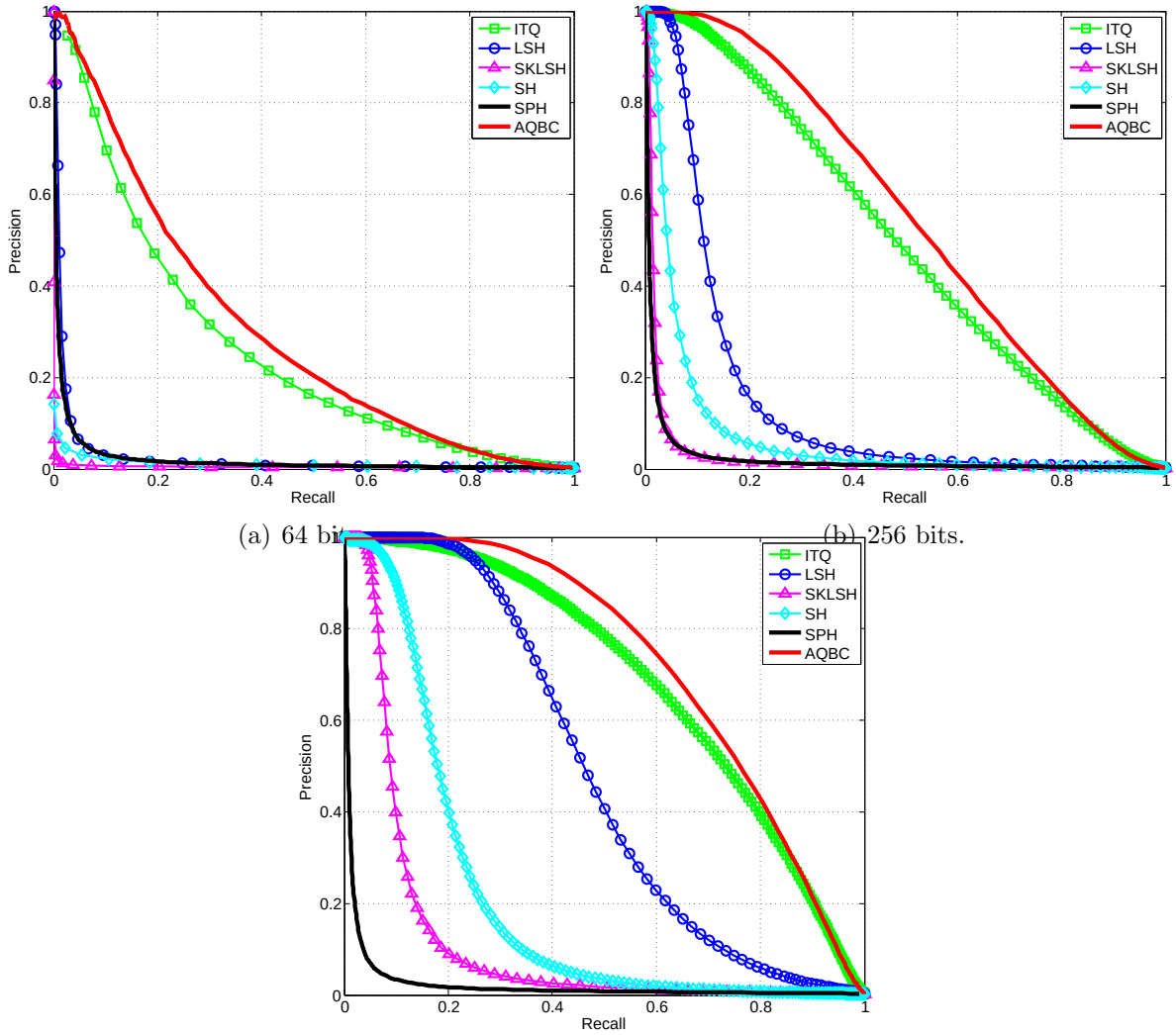


(a) 64 bits.

(b) 256 bits.

(c) 1024 bits.

Figure 4.5: Precision-recall curves for different methods on the ImageNet120K dataset.



(a) 64 bi

(b) 256 bits.

(c) 1024 bits.

Figure 4.6: Precision-recall curves for different methods on the 20 Newsgroups dataset.

decays more slowly than the original distribution in Figure 4.7 (a). Figure 4.7 (d) shows the scaled cumulative sum for the projected vectors, indicating a much higher m.

I also evaluate semantic retrieval results on this dataset, and results are reported in Table 4.3. Unlike in Section 4.3.2, the proposed AQBC method works better than ITQ on this dataset for code sizes of up to 256 bits. Once again, this underscores the dataset-dependent nature of the effectiveness of cosine similarity for semantic retrieval.

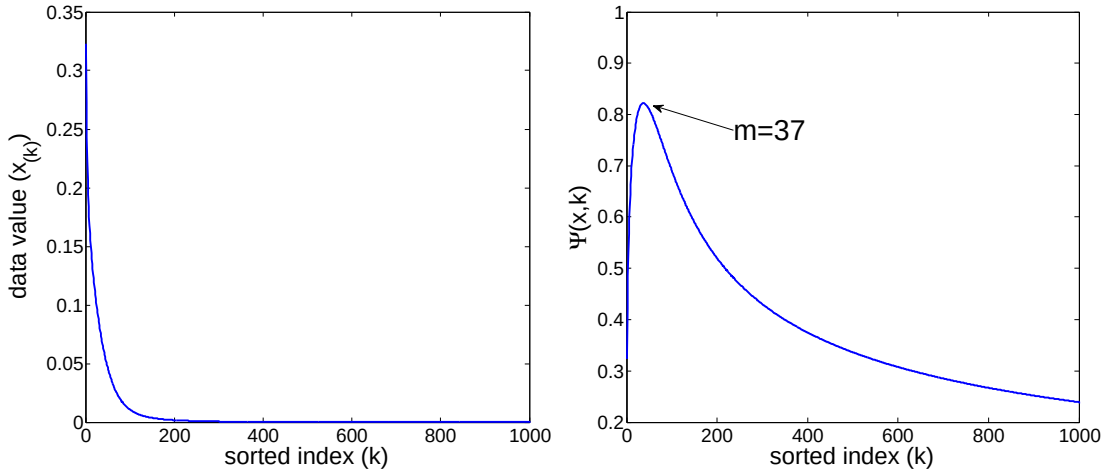


Figure 4.7: Effect of projection on Hamming weight m for 20 Newsgroups data. (a) Distribution of sorted vector entries, (b) scaled cumulative function, (c) distribution over vector elements after learned projection, (d) scaled cumulative function for the projected data. For (a, b) I show only top 1000 entries for better visualization. For (c, d), I project the data to 1000 dimensions.

code size	16	32	64	128	256	512	1024
ITQ	48.20	56.08	59.22	57.46	57.28	58.03	58.55
AQBC	54.92	60.37	59.84	59.89	58.14	58.00	58.52

Table 4.3: Semantic retrieval results (precision@50) on 20 Newsgroups dataset.

	(a) Code generation time						(b) Retrieval time	
code size	SH	LSH	ITQ	SKLSH	SPH	AQBC	Hamming	cosine
64 bits	2.20	0.14	0.14	0.33	0.21	$0.14 + 0.09 = 0.23$	2.4	3.4
512 bits	40.38	3.66	3.66	5.81	3.94	$3.66 + 0.55 = 4.21$	15.8	20.4

Table 4.4: Timing results. (a) Average binary code generation time per query (milliseconds) on 5000-dimensional LLC features. For the proposed AQBC method, the first number is projection time and the second is quantization time. (b) Average time per query, i.e., exhaustive similarity computation against the 120K ImageNet images. Computation of Euclidean distance on this dataset takes 11580 ms.

4.2.4 Timing

Table 5.1 compares the binary code generation time and retrieval speed for different methods. All results are obtained on a workstation with 64GB RAM and 4-core 3.4GHz CPU. My method involves linear projection and quantization using Algorithm 1, while ITQ and LSH only involve linear projections and thresholding. SPH involves Euclidean distance computation and thresholding. SH and SKLSH involve linear projection, nonlinear mapping, and thresholding. The results show that the quantization step (Algorithm 1) of my method is fast, adding very little to the coding time. The coding speed of my method is comparable to that of LSH, ITQ, SPH, and SKLSH. As shown in Table 5.1(b), computation of cosine similarity is slightly slower than that of Hamming distance, but both are orders of magnitude faster than Euclidean distance.

4.3 Discussion

In this chapeter, I have introduced a novel AQBC method for generating binary codes for non-negative frequency/count data. Such data appears very frequently in computer vision and text retrieval applications. This approach extends the ITQ approach proposed in the previous chapter, and is specifically designed for exploring the nonnegative distribution of the histogram data. Retrieval results on high-dimensional image and text datasets have demonstrated that the proposed codes accurately approximate neighbors in the original feature space according to cosine similarity.

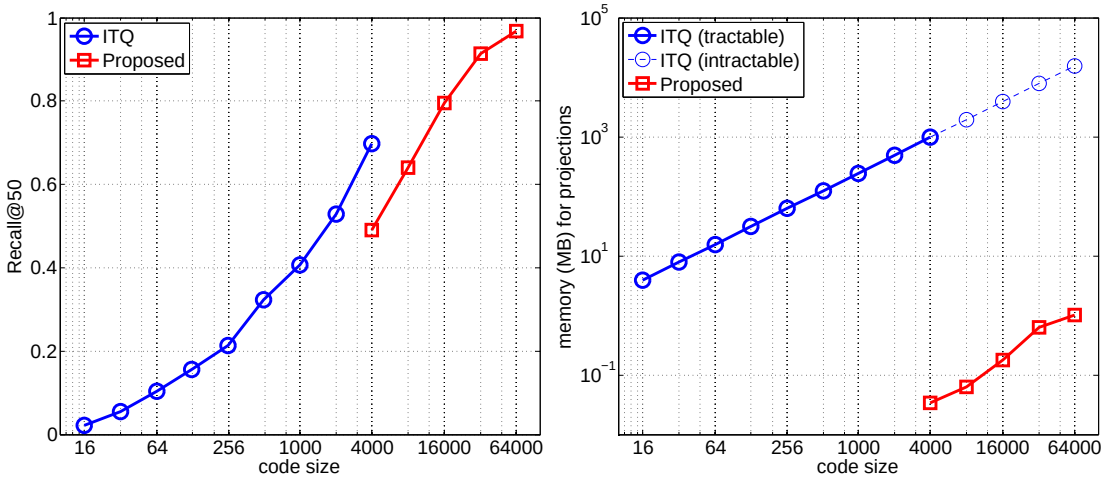
However, I have also found one potential limitation of this approach: its semantic retrieval accuracy varies on different datasets. On two image datasets, AQBC works slightly worse than ITQ, and on the 20 Newsgroups text dataset, it works considerably better than ITQ. This suggests that for different datasets, different similarity measure might have a very big impact to semantic retrieval performance. It remains an open problem to investigate what kind of metric is best for different kinds of data and features.

The methods I have presented in Chapters 3 and 4 can at most handle data with tens of thousands of dimensions. However, in the vision community, for certain applications, we need to use descriptors with hundreds of thousands or even millions of dimensions (Perronnin et al., 2010b,a). In the next chapter, I will present an algorithm that explores bilinear projections to handle such data efficiently.

CHAPTER 5: Bilinear Hashing for Very High-dimensional Data

To perform well on datasets such as ImageNet, extremely high-dimensional descriptors are required, such as the Fisher Vector (FV) (Perronnin and Dance, 2007; Perronnin et al., 2010b, 2011; Sanchez and Perronnin, 2011), Vector of Locally Aggregated Descriptors (VLAD) (Jégou et al., 2010), or Locally Linear Codes (LLC) (Wang et al., 2010c). This chapter focuses on learning high-dimensional binary codes that represent such descriptors with high fidelity. Including the two methods presented in previous chapters, there has been lots of work on learning compact binary codes (Gong and Lazebnik, 2011b; He et al., 2011; Heo et al., 2012; Kulis et al., 2009; Kulis and Grauman, 2009; Liu et al., 2011; Norouzi and Fleet, 2012; Norouzi et al., 2012a; Raginsky and Lazebnik, 2009; Torralba et al., 2008b; Wang et al., 2010a; Weiss et al., 2008), but most existing methods can only handle relatively low-dimensional descriptors such as GIST (Oliva and Torralba, 2001), which are not sufficient for state-of-the-art applications.

In this chapter, I focus on descriptors having tens or hundreds of thousands of dimensions and seek to convert them to long binary strings. This motivation seems to be counter-intuitive, as in the previous chapters, I have focused on learning extremely short codes. However, as shown in Figure 5.1 (a), for relatively high-dimensional descriptor, short codes are far from satisfactory performance (similar findings were reported in Perronnin et al. (2010b); Sanchez and Perronnin (2011)). As discussed in Chapter 2, a common step of many binary coding methods is performing a linear projection of the



(a) NN search recall. (b) Storage of projections.

Figure 5.1: Example on 50K Flickr images with 64,000 dimensional VLAD descriptor. (a) recall of 10NN at top 50 retrieved neighbors using binary codes with different code size. Short codes produced by ITQ do not work well, and cannot scale to higher dimensional codes. The proposed method can easily produce long binary codes that works as well as the original descriptor. (b) comparison of model storage (log scale). The storage rotation matrix by ITQ can easily become intractable while the proposed approach maintains very small model storage.

data (e.g., PCA or a Gaussian random projection). However, due to the high dimensionality of both the input feature vector and the resulting binary string, the storage and computation requirements for even a random projection matrix become extreme. For example, a $64,000 \times 64,000$ random projection matrix takes roughly 16GB of memory and the projection step takes more than one second. As shown in Figure 5.1 (b), when the dimensionality goes higher, the storage of the projection matrix can easily become intractable. In other words, all the RAM is wasted storing the model parameters, and there are no space left for data. Methods that require the projection matrix to be learned, such as ITQ (Chapeter 3) and AQBC (Chapter 4), become even more infeasible, since their training time scales cubically in the number of bits.

There are a few works on learning compact representations for high-dimensional de-

scriptors such as FV. For example, Perronnin et al. (2011) have investigated a few basic methods, including simply thresholding the descriptor, LSH and SH. A more powerful method is PQ (Section 2.1.3), which has produced state-of-the-art results for compressing FV for large-scale image classification (Sanchez and Perronnin, 2011). As discussed in Jégou et al. (2010), a random rotation is usually performed prior to PQ as a preprocessing step. I will show in my experiments that, for high-dimensional descriptors whose variance is highly imbalanced, the good performance of PQ heavily relies on this random rotation, which is very expensive for high-dimensional data.

In Section 5.1, I first present a method for generating binary codes for high-dimensional descriptors via a random bilinear projection. This method takes advantage of the natural two-dimensional structure of descriptors such as FV, VLAD, and LLC and is inspired by bilinear models used for other applications (Pirsiavash et al., 2009; Schönemann, 1968; Ye et al., 2004). My bilinear formulation uses two small projection matrices to implicitly represent a bigger projection matrix. Next, I propose a method for efficiently learning the projection matrices from data. Section 5.2 demonstrates the promise of my method, dubbed *bilinear projection-based binary codes (BPBC)*, through experiments on two large-scale datasets. For most scenarios I consider, BPBC produces little or no degradation in performance compared the original continuous descriptors; furthermore, it matches the accuracy of PQ codes, while having much lower running time and storage requirements for code generation.

This work was originally published in (Gong et al., 2013b)

5.1 Bilinear Binary Codes

Most high-dimensional descriptors have a natural matrix or tensor structure. For example, the matrix representation of a Fisher Vector has dimensionality $k \times 2l$, where k is the visual vocabulary size and l is the dimensionality of the local image features (the most common choice is SIFT with l=128). An LLC descriptor with s spatial bins can be represented as a $k \times s$ matrix. A HOG descriptor is a two-dimensional grid of histograms, and this structure has been exploited for object detection (Pirsiavash et al., 2009).

Let $\boldsymbol{x} \in \mathbb{R}^d$ denote our descriptor vector, and assume we can reorganize it as a $d_1 \times d_2$ matrix, with $d = d_1 d_2$:

$$\boldsymbol{x} \in \mathbb{R}^{d_1 d_2 \times 1} \quad \mapsto \quad X \in \mathbb{R}^{d_1 \times d_2}. \tag{5.1}$$

As discussed above, the dimensions of visual descriptors adhere to some physical interpretation, and the reorganization of the vectors into matrices must be based on these interpretations (I have also tried to randomly reorganize the vectors into matrices, but this produced inferior performance). I also assume that each vector $\boldsymbol{x} \in \mathbb{R}^d$ is zero-centered and has unit norm, as L_2 normalization is widely used to preprocess visual features and can usually improve performance (Perronnin et al., 2010b). In other words, I assume my data is sampled on the high-dimensional hyper-sphere. I first discuss a randomized method to obtain d-bit bilinear binary codes and then explain how to learn data-dependent codes in Section 5.1.1. The method for learning reduced-dimension codes will be discussed in Section 5.1.2.

To convert a descriptor $\boldsymbol{x} \in \mathbb{R}^d$ to a d-dimensional binary string, I first consider the framework of Chapter 3 that applies a random rotation $R \in \mathbb{R}^{d \times d}$ to $\boldsymbol{x}$:

$$H(\boldsymbol{x}) = \operatorname{sgn}(R^T \boldsymbol{x}). \tag{5.2}$$

However, as discussed above, this operation is very expensive when d is large. Instead, since $\boldsymbol{x}$ can be represented as a matrix $X \in \mathbb{R}^{d_1 \times d_2}$, to encode it efficiently, I propose a bilinear formulation using two random orthogonal matrices $R_1 \in \mathbb{R}^{d_1 \times d_1}$ and $R_2 \in \mathbb{R}^{d_2 \times d_2}$:

$$H(X) = \operatorname{vec}\Big(\operatorname{sgn}(R_1^T X R_2)\Big), \tag{5.3}$$

where $\operatorname{vec}(\cdot)$ vectorizes its argument by concatenating its columns.

It is easy to show that applying a bilinear rotation to $X \in \mathbb{R}^{d_1 \times d_2}$ is equivalent to applying a rotation matrix of $d_1 d_2$ dimensions to $\operatorname{vec}(X)$. This rotation is given by $\hat{R} = R_2 \otimes R_1$, where $\otimes$ denotes the Kronecker product:

$$\operatorname{vec}(R_1^T X R_2) = (R_2^T \otimes R_1^T) \operatorname{vec}(X) = \hat{R}^T \operatorname{vec}(X)$$

follows from the properties of the Kronecker product (Laub, 2004). Another basic property of the Kronecker product is that if R_1 and R_2 are orthogonal matrices, then $R_2 \otimes R_1$ is orthogonal as well (Laub, 2004). Thus, a bilinear rotation is simply a special case of a full rotation, such that the full rotation matrix $\hat{R}$ can be *reconstructed* from two smaller matrices R_1 and R_2. While the degree of freedom of the bilinear rotation is more

restricted than that of a full rotation, the projection matrices are much smaller, and the projection speed is much faster. In terms of time complexity, performing a full rotation on $\boldsymbol{x}$ takes $O((d_1 d_2)^2)$ time, while the approach is $O(d_1^2 d_2 + d_1 d_2^2)$. In terms of space for projections, full rotation takes $O((d_1 d_2)^2)$, and the approach only takes $O(d_1^2 + d_2^2)$. For example, as will be shown in Section 5.2.4, for a 64,000-dimensional vector, a full rotation will take roughly 16GB of RAM, while the bilinear rotations only take 1MB of RAM. The projection time for a full rotation is more than 1 second, vs. only 3 ms for bilinear rotations.

5.1.1 Learning Bilinear Binary Codes

In this section, I present a method to learn the rotations R_1 and R_2 inspired by the two-sided Procrustes analysis (Schönemann, 1968) and the methods of Chapter 3 and Chapter 4. In Section 5.2.2, I will extend this method to learning of lower-dimensional projection matrices.

Following Chapter 3 and Chapter 4, we want to find a rotation $\hat{R}$ such that the angle θ_i between a rotated feature vector $\hat{R}^T \boldsymbol{x}_i = \text{vec}(R_1^T X_i R_2)$ and its binary encoding (geometrically, the nearest vertex of the binary hypercube), $\text{sgn}(\hat{R}^T \boldsymbol{x}) = \text{vec}(\text{sgn}(R_1^T X_i R_2))$,

is minimized. Given N training points, we want to maximize the cosine of angle

$$\begin{aligned}
&\sum_{i=1}^{N} \cos(\theta_i) \\
&= \sum_{i=1}^{N} \left(\frac{\operatorname{sgn}(\hat{R}^T \boldsymbol{x}_i)^T}{\sqrt{d}} (\hat{R}^T \boldsymbol{x}_i) \right) \qquad (5.4) \\
&= \sum_{i=1}^{N} \left(\frac{\operatorname{vec}(\operatorname{sgn}(R_1^T X_i R_2))^T}{\sqrt{d}} \operatorname{vec}(R_1^T X_i R_2) \right) \\
&= \frac{1}{\sqrt{d}} \sum_{i=1}^{N} \left(\operatorname{vec}(B_i)^T \operatorname{vec}(R_1^T X_i R_2) \right) \\
&= \frac{1}{\sqrt{d}} \sum_{i=1}^{N} \operatorname{tr}(B_i R_2^T X_i^T R_1), \qquad (5.5)
\end{aligned}$$

where $B_i = \operatorname{sgn}(R_1^T X_i R_2)$. Notice that (5.4) involves the large projection matrix $\hat{R} \in \mathbb{R}^{d \times d}$, direct optimization of which is challenging. However, after reformulation into bilinear form (5.5), the expression only involves the two small matrices R_1 and R_2. Letting $\mathcal{B} = \{B_1, \ldots, B_N\}$, my objective function is as follows:

$$\begin{aligned}
\mathcal{Q}(\mathcal{B}, R_1, R_2) &= \max_{\mathcal{B}, R_1, R_2} \sum_{i=1}^{N} \operatorname{tr}(B_i R_2^T X_i^T R_1) \qquad (5.6) \\
\text{s.t.} \quad B_i &\in \{-1, +1\}^{d_1 \times d_2}, \quad R_1^T R_1 = I, \quad R_2^T R_2 = I.
\end{aligned}$$

This optimization problem can be solved by block coordinate ascent by alternating between the different variables $\{B_1, \ldots, B_N\}$, R_1, and R_2. The update steps for each variable are described below, assuming the others are fixed.

(S1) Update B_i. It is easy to check that the above program is independent for each B_i

when R_1 and R_2 are fixed. Thus we can solve for each B_i separately. I have

$$\mathcal{Q}(B_i) = \mathrm{tr}(B_i R_2^T X_i^T R_1) = \sum_{k=1}^{d_1} \sum_{l=1}^{d_2} B_i^{kl} \widetilde{V}_i^{lk}, \tag{5.7}$$

where $\widetilde{V}_i^{lk}$ denote the elements of $\widetilde{V}_i = R_2^T X_i^T R_1$. $\mathcal{Q}(B_i)$ is maximized by $B_i = \mathrm{sgn}(\widetilde{V}_i^T)$. Each B_i is updated in this way independently.

(S2) Update R_1. Expanding (5.6) with R_2 and B_i fixed, we have the following:

$$\begin{aligned} \mathcal{Q}(R_1) &= \sum\nolimits_{i=1}^{N} \mathrm{tr}(B_i R_2^T X_i^T R_1) \\ &= \mathrm{tr}\Big(\sum\nolimits_{i=1}^{N} (B_i R_2^T X_i^T) R_1\Big) = \mathrm{tr}(D_1 R_1)\,, \end{aligned} \tag{5.8}$$

where $D_1 = \sum_{i=1}^{N} (B_i R_2^T X_i^T)$. The above expression is maximized with the help of polar decomposition:

$$R_1 = V_1 U_1^T\,, \tag{5.9}$$

where $D_1 = U_1 S_1 V_1^T$ is the SVD of D_1.

(S3) Update R_2:

$$\begin{aligned} \mathcal{Q}(R_2) &= \sum\nolimits_{i=1}^{N} \mathrm{tr}(B_i R_2^T X_i^T R_1) \\ &= \sum\nolimits_{i=1}^{N} \mathrm{tr}(R_2^T X_i^T R_1 B_i) \\ &= \mathrm{tr}\Big(R_2^T \sum\nolimits_{i=1}^{N} (X_i^T R_1 B_i)\Big) = \mathrm{tr}(R_2^T D_2)\,, \end{aligned} \tag{5.10}$$

where $D_2 = \sum_{i=1}^{N} (X_i^T R_1 B_i)$. Analogously to the update rule for R_1, the update rule for

R_2 is:

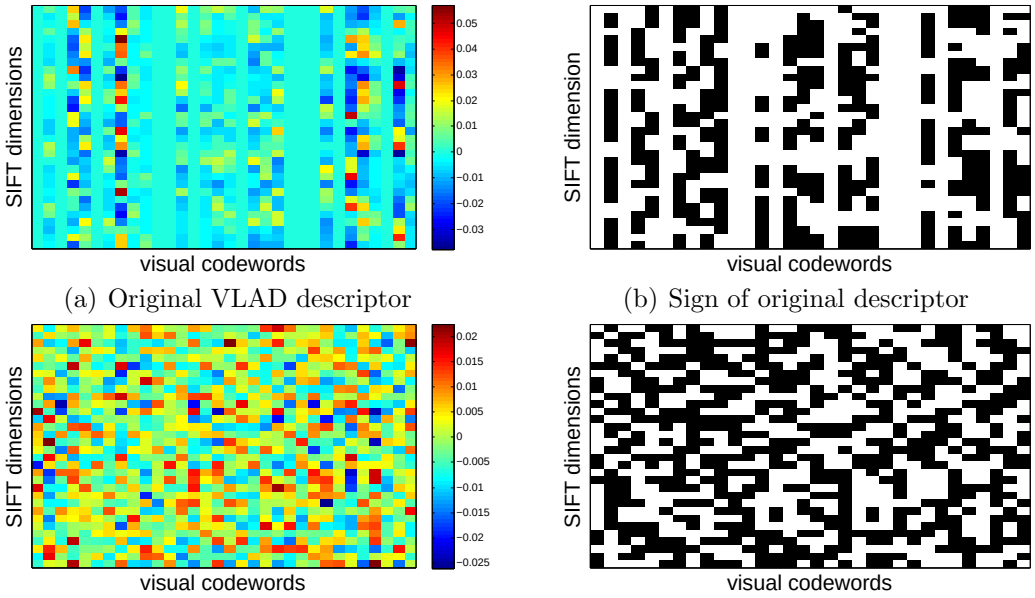
$$R_2 = U_2 V_2^T \,, \tag{5.11}$$

where $D_2 = U_2 S_2 V_2^T$ is the SVD of D_2.

We cycle between these updates for several iterations to obtain a local maximum. The convergence of the above program is guaranteed in a finite number of iterations as the globally optimal solution of each step is exactly obtained, each step is guaranteed not to decrease the objective function value, and the objective is bounded from above. In my implementation, I initialize R_1 and R_2 by random rotations, and use 3 iterations. I have not found significant improvement of performance by using more iterations. The time complexity of this program is $O(N(d_1^3 + d_2^3))$ where d_1 and d_2 are typically fairly small (e.g., $d_1 = 128,\ d_2 = 500$). Figure 5.2 visualizes the structure of a VLAD descriptor and the corresponding binary code before and after a learned bilinear rotation.

5.1.2 Learning with Dimensionality Reduction

The optimization method of the previous section is used to produce a d-dimensional binary code from a d-dimensional feature vector. However, in many applications we want be able to moderately reduce the dimensionality so as to reduce storage and processing time, but still preserve enough discriminative power. This can be done by a simple modification of (5.4). To produce a code of size $c = c_1 \times c_2$, where $c_1 < d_1$ and $c_2 < d_2$, I now look for projection matrices $R_1 \in \mathbb{R}^{d_1 \times c_1}$, $R_2 \in \mathbb{R}^{d_2 \times c_2}$ such that $R_1^T R_1 = I$ and $R_2^T R_2 = I$. Each B_i is now a $c_1 \times c_2$ binary variable. Consider the cosine of the angle



(a) Original VLAD descriptor

(b) Sign of original descriptor

(c) Bilinearly rotated VLAD

(d) Bilinearly rotated binary codes

Figure 5.2: Visualization of the distribution of the VLAD descriptor. I only show the top 32 SIFT dimensions and top 32 visual codewords. For (a) and (b), we can clearly find some codewords have higher weights than others, and directly taking the sign will drop the codewords with lower weights. In (c) and (d), the weights are much more uniform after rotation, and after thresholding, the bit selection are much more informative.

between a lower-dimensional projected vector $\hat{R}^T \boldsymbol{x}_i$ and its binary encoding, given by $\text{sgn}(\hat{R}^T \boldsymbol{x})$:

$$\cos(\theta_i) = \frac{\text{sgn}(\hat{R}^T \boldsymbol{x}_i)^T}{\sqrt{c}} \frac{\hat{R}^T \boldsymbol{x}_i}{\|\hat{R}^T \boldsymbol{x}_i\|_2},$$

where $\hat{R} \in \mathbb{R}^{d_1 d_2 \times c_1 c_2}$ and $\hat{R}^T \hat{R} = I$. This formulation differs from that of (5.4) in that it contains $\|\hat{R}^T \boldsymbol{x}_i\|_2$ in the denominator, which makes the optimization hard as discussed in Chapter 4. Instead, I follow Chapter 4 to define a relaxed objective function based on the sum of linear correlations

$$\mathcal{Q}(\mathcal{B}, R_1, R_2) = \sum\nolimits_{i=1}^{N} \left(\frac{\text{sgn}(\hat{R}^T \boldsymbol{x}_i)^T}{\sqrt{c}} (\hat{R}^T \boldsymbol{x}_i) \right).$$

The optimization framework for this objective is similar to that of Section 5.1.1: rewrite $\hat{R}$ as $\hat{R} = R_2 \otimes R_1$, and efficiently solve for R_1 and R_2. For the three alternating optimization steps, step (S1) remains the same. For steps (S2) and (S3), I compute the SVD of $D_k,\ k = 1, 2$ as $D_k = U_k S_k V_k^T$, and let R_k be

$$R_1 = \hat{V}_1 U_1^T, \qquad R_2 = \hat{U}_2 V_2^T, \tag{5.12}$$

where $\hat{V}_1$ is the top c_1 singular vectors of V_1 and $\hat{U}_2$ is the top c_2 singular vectors of U_2. To initialize the optimization, I generate c_k random orthogonal directions.

In this work, I adopt the lower-bounded asymmetric distance proposed in (Dong et al., 2008) to measure the distance between the query and binary codes. This metric has been

introduced in Section 2.1.1, which can be computed efficiently using lookup tables.

5.2 Experiments

5.2.1 Datasets and Features

I test my proposed approach, bilinear projection-based binary codes (**BPBC**), on two widely used large-scale image datasets. The first one is the INRIA Holiday dataset with 1M Flickr images (referred to as **Holiday+Flickr1M**) (Jégou et al., 2008). There are 1419 images in the Holiday dataset corresponding to 500 different scene instances, and each instance has three images on average. There is a set of 500 query images, and the remaining 919 images together with 1 million Flickr images are used as the database. I use the SIFT features of interest points provided by Jégou et al. (2010), and cluster them to 500 k-means centers. Then I represent each image by a $128 \times 500 = 64,000$ dimensional VLAD feature vector. The vectors are power normalized (element wise square root) and L_2-normalized as in (Perronnin et al., 2010b). Some sample images from Holiday dataset are shown in Figure 5.3.

The second dataset is the **ILSVRC2010** subset of ImageNet (Deng et al., 2009), which contains 1.2M images and 1000 classes. On this dataset, I use the publicly available SIFT features, which are densely extracted from the images at three different scales. I cluster the features into 200 centers, and then aggregate them into dense VLAD feature vectors of $25,600$ dimensions. These vectors are also power- and L_2-normalized. In addition, to evaluate my method on diverse feature types, I compute LLC features (Wang

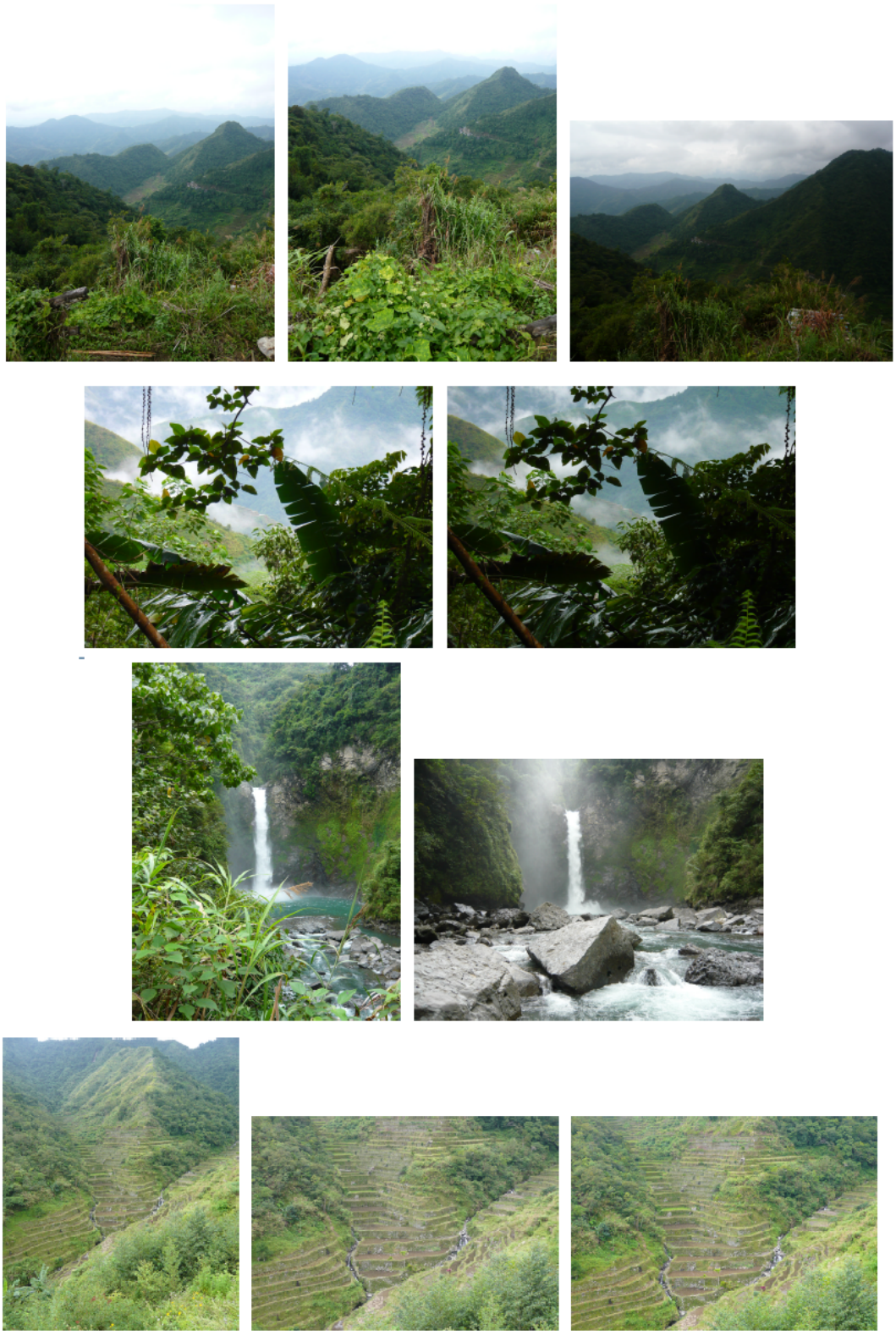


Figure 5.3: Sample images from Holiday dataset.

et al., 2010c) on this dataset using a 5,000-dimensional visual codebook and a three-level spatial pyramid. This results in a 105,000-dimensional feature vector that can be naturally reorganized into a $5,000 \times 21$ matrix. The statistics of VLAD and LLC are very different from each other – the former are dense and have both positive and negative values, while the latter are nonnegative and sparse. As a preprocessing step, I zero-center the LLC descriptors and L_2-normalize them (I have found no need to zero-center VLAD, since they are already approximately zero-centered).

5.2.2 Experimental Protocols

To learn binary codes using the methods of Sections 5.1.1 and 5.1.2, I randomly sample 20,000 images from each dataset as training set. To compare different methods, I sample a set of images as queries, and use the rest as the database against which the queries are run. For nearest neighbor search experiments, I define ground truth neighbors as the top 10 nearest neighbors for each query based on original descriptors, and report the recall of 10NN for different numbers of retrieved data points based on binary codes, which follows the methodology of Jégou et al. (2010, 2008).

In addition, I evaluate the accuracy of different methods on semantic retrieval and recognition tasks. For Holiday+Flickr1M, I report instance-level image retrieval performance, and use the standard mean average precision (mAP) to evaluate retrieval accuracy. For ILSVRC2010, I report precision@k as in Chapter 3, which evaluates the precision of top k retrieved images in terms of semantic class labels. Finally, I perform image classification experiments on the ILSVRC2010 dataset.

In this chapter, I will use both Hamming distance (symmetric distance, SD) and the asymmetric distance (ASD, see Section 2.1.1) for evaluation. This is mainly because PQ heavily relies on ASD, and to ensure a fair comparison, we also use both metrics for binary codes in this chapter.

5.2.3 Baseline Methods

My main baseline is PQ, which is the state-of-the-art approach for encoding high-dimensional descriptors. PQ groups the data dimensions in batches of size s and quantizes each group with k codebook centers. In my experiments, I use $s = 8$ and $k = 256$ following Jégou et al. (2010). In practice, a random rotation is usually applied to the data prior to PQ in order to balance the variance of different dimensions. This will be referred to as RR+PQ in the following. At query time, PQ uses asymmetric distance to compare an uncompressed query point to quantized database points. Namely, the distances between the the code centers and corresponding dimensions of the query are first computed and stored in a lookup table. Then the distances between the query and database points are computed by table lookup and summation.

Besides PQ, I also consider LSH and the $\alpha = 0$ binarization scheme proposed in (Perronnin et al., 2011), which simply takes the sign of data (0 still maps to 0). Methods like ITQ are not appropriate as baselines here because they scale poorly to the high-dimensional data used in my experiments (as demonstrated in Figure 5.1).

Feature dim.	LSH	PQ	RR+PQ	BPBC
128×10	0.12	2.8	2.92	**0.08**
128×100	9.35	26.5	35.85	**0.54**
128×200	29.14	47.3	76.44	**0.86**
128×500	186.22	122.3	308.52	**3.06**
128×1000	–	269.5	–	**9.53**

Table 5.1: Code generation time (ms) for LSH, PQ, and my bilinear method. The VLAD feature dimension is $l \times k$.

Feature dim.	LSH	PQ	RR+PQ	BPBC
128×10	6.25	1.25	7.50	**0.063**
128×100	625	12.5	637	**0.10**
128×200	2500	25.0	2525	**0.22**
128×500	15625	62.5	15687	**1.02**
128×1000	62500	125	62625	**3.88**

Table 5.2: Memory (MB) needed to store the projections (or codebooks), assuming each element is a float. The VLAD feature dimension is $l \times k$.

5.2.4 Code Generation Time and Storage

First, I evaluate the scalability of my method compared to LSH, PQ, and RR+PQ for converting d-dimensional vectors to d-bit binary strings. All running times are evaluated on a machine with 24GB RAM and 6-core 2.6GHz CPU. Table 5.1 reports code generation time for different VLAD sizes, and Table 5.2 reports the memory requirements for storing the projection matrix. For relatively low-dimensional data, LSH can still efficiently perform the projection step, while for higher-dimensional data (128×1000 descriptor size), the $d \times d$ projection matrix used in LSH and RR+PQ takes 61GB, so it cannot be stored in the RAM of most machines and it is extremely hard to perform operations on it. My bilinear formulation is orders of magnitude faster in terms of pro-

jection time, and the projection matrices are very small. PQ without random rotation is much more scalable, but my method still has the lowest time and memory overhead (plus, as my subsequent experiments will show, PQ without random rotation can fail on some datasets).

5.2.5 Retrieval on Holiday+Flickr1M

I report the results on the Holiday+Flickr1M dataset with 64,000-dimensional VLAD. I use the predefined 500 Holiday queries, which are guaranteed to have close neighbors (image from the same instance) in the database. For 64,000-dimensional features, evaluating RR+PQ is prohibitively expensive, so instead I try to combine the bilinear rotation with PQ (denoted as BR+PQ). For BPBC with dimensionality reduction (Section 5.1.2), I use bilinear projections $R_1 \in \mathbb{R}^{500\times400}, R_2 \in \mathbb{R}^{128\times80}$. This reduces the dimensionality in half.

Figure 5.4 (a) shows the recall of 10NN with different numbers of retrieved images. Surprisingly, PQ without rotation fails on this dataset; BR+PQ is slightly better, but is still disappointing. This is due to many Flickr images being relatively blank (e.g., sky and sea images) and having few interest points, which leads to VLAD with entries that are mostly zero, which in turn results in significantly imbalanced distribution which hampers PQ quantization. Bilinear rotation appears to be insufficient to fully balance the variance in this case, and performing the full random rotation is too expensive.[1]

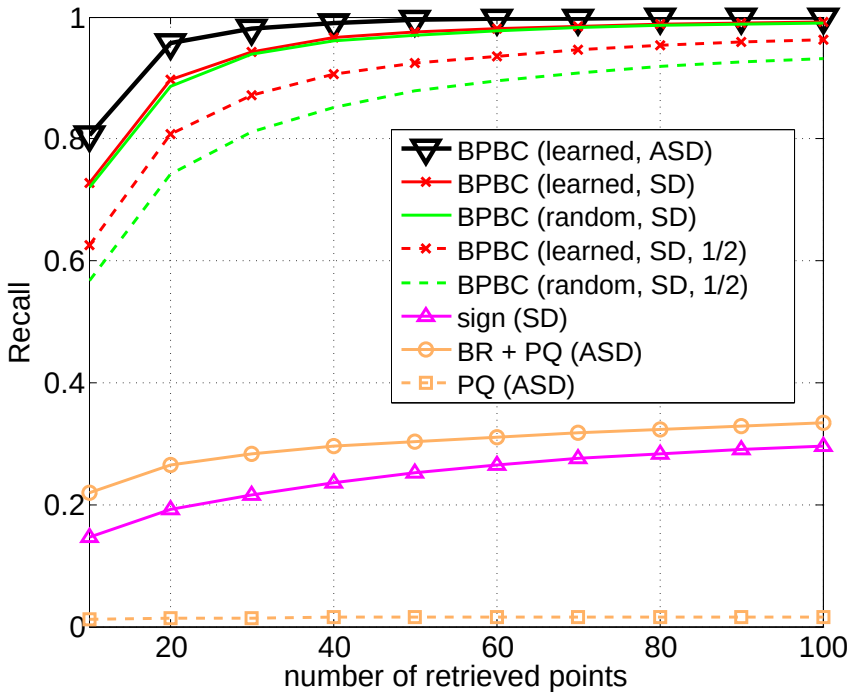
[1] Jégou et al. (2010) report relatively strong performance for RR+PQ on Holiday+Flickr1M, but they use lower-dimensional VLAD ($d = 2,048$ and $d = 8,192$) followed by PCA compression to 32-128 dimensions. These parameter settings are motivated by the goal of (Jégou et al., 2010) to produce extremely compact image codes. By contrast, my goal is to produce higher-dimensional codes that do

bits	SD (binary)	ASD (binary)	ASD (PQ)	Eucl. (est.)
12,800	0.33	4.48	4.59	∼120
25,600	0.60	11.29	11.28	∼241

Table 5.3: Retrieval time per query (seconds) on ILSVRC2010 dataset with 1.2M images and different code sizes. This is the time to perform exhaustive computation of distances from the query to all the 1.2M images in the database. For Euclidean distance, the original descriptors do not fit in RAM, so the timing is extrapolated from a smaller number of distance computations. The actual timing is likely to be higher due to file I/O.

However, as will be shown next, BR+PQ works similarly to RR+PQ on ILSVRC2010, all of whose images are object-centric and have much more balanced descriptors to begin with. On the other hand, all versions of BPBC show good performance. For a code size of $d/2$, learned rotation works much better than random, while for the full-dimensional BPBC, learned and random rotations perform similarly. Asymmetric distance (ASD) further improves the recall over symmetric distance (SD).

Next, Figure 5.4 (b) reports instance-level image retrieval accuracy measured by mean average precision (mAP), or the area under the recall-precision curve. Both learned and random BPBC work very well, and can produce comparable results to the original descriptor. PQ without rotation works poorly, and BR+PQ is more reasonable, but still worse than BPBC. This is consistent with the results of Figure 5.4.

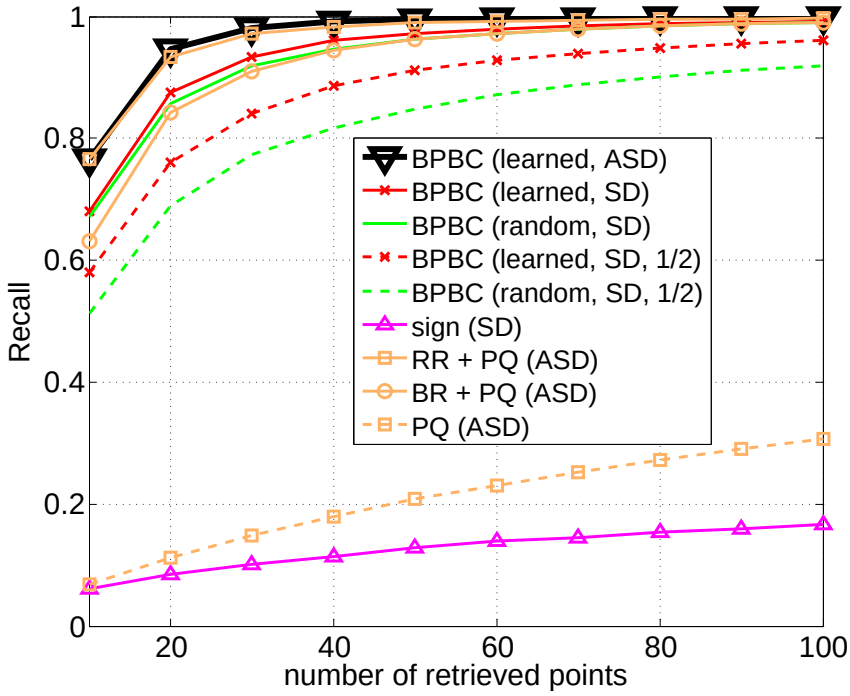


(a) Recall of 10NN.

Method	Rate	mAP
VLAD (float)	1	**39.0**
$\alpha = 0$ (SD)	32	25.6
PQ (with bilinear rotation, ASD)	32	24.0
PQ (w/o rotation, ASD)	32	2.3
BPBC (learned, ASD)	32	**40.1**
BPBC (learned, SD)	32	**40.1**
BPBC (random, SD)	32	**40.3**
BPBC (learned, SD, 1/2)	64	38.8
BPBC (random, SD, 1/2)	64	38.6

(b) Instance image retrieval results.

Figure 5.4: Results on the Holiday+FLickr1M dataset with 64,000-dimensional VLAD. (a) Recall of 10NN. "*Random*" refers to the method of eq. (5.3) with two random matricesand "*learned*" refers to the methods of Sections 5.1.1 and 5.1.2. "SD" denotes symmetric (Hamming) distance, and "ASD" denotes asymmetric distance. "1/2" refers to reducing the code dimensionality to $d/2$ with the method of Section 5.1.2. (b) Instance-level retrieval results (mAP). "Rate" is the factor by which storage is reduced compared to the original descriptors.



(a) Recall for 10NN.

Method	Rate	P@10	P@50
VLAD (float)	1	**17.73**	**7.29**
$\alpha = 0$ (SD)	32	13.15	3.87
PQ (with full rotation, ASD)	32	**18.06**	**7.41**
PQ (with bilinear rotation, ASD)	32	16.98	6.96
PQ (w/o rotation, ASD)	32	11.32	3.14
BPBC (learned, ASD)	32	**18.01**	**7.49**
BPBC (learned, SD)	32	**18.07**	**7.42**
BPBC (random, SD)	32	**18.17**	**7.60**
BPBC (learned, SD, 1/2)	64	17.80	7.25
BPBC (random, SD, 1/2)	64	16.85	6.78

(b) Categorical image retrieval results.

Figure 5.5: Results on ILSVRC dataset with 25,600-dimensional VLAD. (a) Recall of 10NN. "*Random*" refers to the method of eq. (5.3) with two random matrices and "*learned*" refers to the methods of Sections 5.1.1 and 5.1.2. "SD" denotes symmetric (Hamming) distance, and "AS with D" denotes asymmetric distance. "1/2" refers to reducing the code dimensionality to $d/2$ with the method of Section 5.1.2. (b) Semantic precision for different number of retrieved images.

5.2.6 Retrieval on ILSVRC2010 with VLAD

As discussed in Section 5.2.1, my VLAD descriptors for the ILSVRC2010 dataset have dimensionality 25,600. Random rotation for this descriptor size is still feasible, so I am able to evaluate RR+PQ. I randomly sample 1,000 query images and use the rest as the database. For BPBC with dimensionality reduction, I construct bilinear projections $R_1 \in \mathbb{R}^{200\times160}, R_2 \in \mathbb{R}^{128\times80}$, which reduces the dimensionality in half. Figure 5.5 (a) displays the recall of 10NN with increasing number of retrieved points for both descriptor sizes. The basic PQ method on this dataset works much better than on Holiday+Flickr1M (in particular, unlike in Figure 5.4, it is now better than $\alpha = 0$). This is because the images in ILSVRC2010 are textured and contain prominent objects, which leads to VLAD with fairly balanced variance. Furthermore, RR+PQ is feasible for the VLAD dimensionality I use on ILSVRC2010. We can see from Figure 5.5 (a) that the improvement from PQ to RR+PQ is remarkable. In fact, RR+PQ now becomes comparable to full-dimensional BPBC with asymmetric distance[2]. Recall from Tables 5.1 and 5.2, however, that BPBC has a much lower memory footprint and a much faster code generation time than RR+PQ. The relative performance of the other BPBC variants is the same as in Section 5.2.5.

Next, I evaluate the semantic retrieval accuracy on this dataset. Figure 5.5 (b) shows

not lose discriminative power. Indeed, by raising the dimensionality of the code, I are able to improve the retrieval accuracy in absolute terms: the mAP for my BPBC setup (Figure 5.4 (b)) is about 0.4 vs. about 0.2 for the PQ setup of (Jégou et al., 2010).

[2]In their original work Jégou et al. (2010) have claimed that a random rotation is not needed prior to PQ. By contrast, I have found for high-dimensional data like VLAD, this operation is very important to guarantee good performance. This is because the conclusions of Jégou et al. (2010) are mainly drawn from low-dimensional data like SIFT and GIST, whose variance is roughly balanced.

the average precision for top k images retrieved. I can observe that RR+PQ and most versions of BPBC have comparable precision to the original uncompressed features. In particular, using ASD as opposed to SD does not give any gains in semantic precision for my method. This finding points to a potential advantage of my method over PQ for applications where semantic precision is the main focus: ASD is required for PQ but not for my method, and I have seen from Table 5.3 that ASD computation is more than ten times slower than SD.

Table 5.3 compares the retrieval speed for symmetric (Hamming) vs. asymmetric distance computation for two code sizes. As one would expect, directly performing Hamming distance using XOR and popcount gives extremely fast retrieval speed. Unlike BPBC codes, PQ codes cannot be compared using Hamming distance. The speed of ASD for PQ vs. My method is comparable, and much slower than Hamming distance. However, note that for binary codes with ASD, one can first use Hamming distance to find a short list and then do re-ranking with ASD, which will be much faster than exhaustive ASD computation.

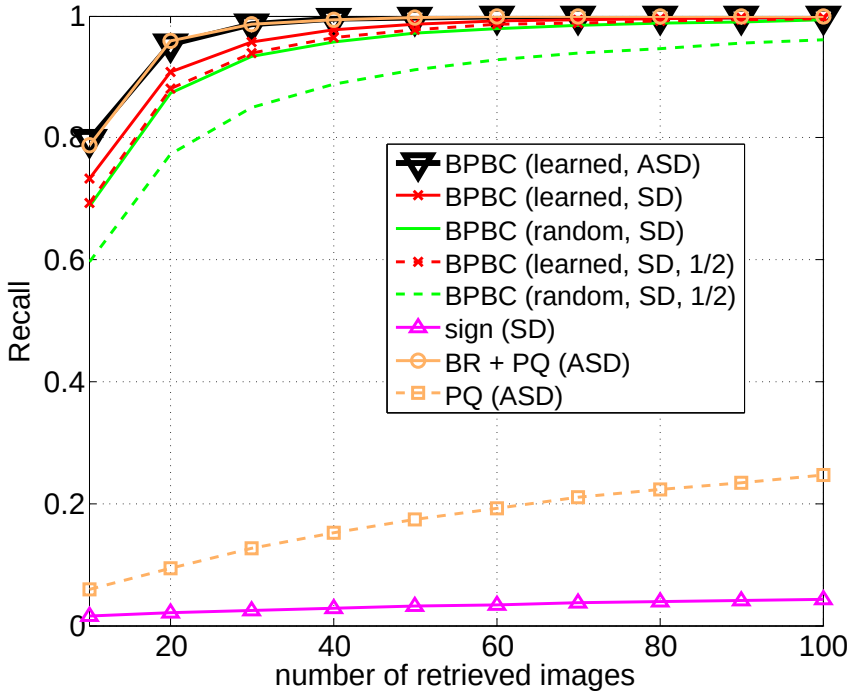
5.2.7 Retrieval on ILSVRC2010 with LLC

To demonstrate that the BPBC method is applicable to other high-dimensional descriptors besides VLAD, I also report retrieval results on the ILSVRC2010 dataset with LLC features. As discussed in Section 5.2.1, the features have dimensionality $5000 \times 21 = 105000$. To reduce dimensionality, I use bilinear projections $R_1 \in \mathbb{R}^{5000 \times 2500}$, $R_2 \in \mathbb{R}^{21 \times 21}$, which yields a code size of 52,500.

Figure 5.6 (a) reports the recall for 10NN on this descriptor. Most of the trends are similar to those of Section 5.2.6. Full-dimensional BPBC with ASD once again has the best performance, together with BR+PQ (because LLC is my highest-dimensional descriptor yet, evaluating RR+PQ is once again infeasible). Learned rotation works significantly better than the random one. Also, it is interesting to mention that for LLC feature, one side of the matrix has only 21 dimensions. I have found further reducing the dimensionality of this side makes the learning unstable, while keeping it as a full rotation leads to more stable results. Next, Table 5.6 (b) reports the semantic precision analogously to Table 5.5 (b). As in Table 5.5, BR+PQ and different versions of BPBC work similarly. By comparing Table 5.5 and 5.6 (b), we can see that LLC features have higher absolute precision than VLAD, which confirms that even extremely high-dimensional features and codes, far from overfitting, are necessary to obtain better performance on very large-scale many-category datasets. Figure 5.7 shows some sample image retrieval results.

5.2.8 Image Classification

Finally, I demonstrate the effectiveness of my proposed codes for image classification with LIBLINEAR SVM (Fan et al., 2008). Due to the computational overhead of SVM training and parameter tuning on original descriptors, I randomly sample 100 classes from the ILSVRC2010 dataset. I use 25,600-dimensional VLAD and five random splits of the dataset into 50% for training, 25% for validation, and 25% for testing. Following Sanchez and Perronnin (2011), I sample 200 points per negative class to speed up the training



(a) Recall of 10NN.

Method	Rate	P@10	P@50
VLAD (float)	1	**21.50**	**10.06**
sign (SD)	32	10.99	2.58
PQ (with bilinear rotation, ASD)	32	**21.41**	**10.11**
PQ (w/o rotation, ASD)	32	12.67	4.20
BPBC (learned, ASD)	32	**21.78**	**10.11**
BPBC (learned, SD)	32	**21.54**	**10.11**
BPBC (random, SD)	32	**21.73**	**10.35**
BPBC (learned, SD, 1/2)	64	21.22	9.96
BPBC (random, SD, 1/2)	64	21.27	9.88
BPBC (learned, SD, 1/5)	160	21.57	10.09
BPBC (random, SD, 1/5)	160	20.93	9.54

(b) Categorical image retrieval results.

Figure 5.6: Retrieval results on ILSVRC dataset with 105,000-dimensional LLC features. (a) Recall for 10NN. "*Random*" refers to the method of eq. (5.3) with two random matrices and "*learned*" refers to the methods of Sections 5.1.1 and 5.1.2. "SD" denotes symmetric (Hamming) distance, and "ASD" denotes asymmetric distance. "1/2" refers to reducing the code dimensionality to $d/2$ with the method of Section 5.1.2. (b) Categorical image retrieval precision.

Method	Rate	Classification accuracy
VLAD (float)	1	**44.87**±0.30
$\alpha = 0$	32	41.10±0.34
PQ	32	44.05±0.33
RR+PQ	32	**44.64**±0.13
BPBC (learned)	32	**44.34**±0.21
BPBC (random)	32	**44.27**±0.19
BPBC (learned, 1/2)	64	43.06±0.20
BPBC (random, 1/2)	64	41.28±0.20

Table 5.4: Image classification results on 100 classes randomly sampled classes from the ILSVRC2010 dataset. The set of classes is fixed but results are averaged over five different training/validation/test splits. The visual feature is 25,600-dimensional VLAD. "Rate" is the factor by which storage is reduced.

process, which does not sacrifice accuracy too much. For different methods, I validate the SVM hyperparameter C on a grid of $[2 \times 10^{-5}, 2 \times 10^{2}]$ with order-of-magnitude increments. Like Sanchez and Perronnin (2011), I train the SVM on PQ decoded vectors or my binary codes (for 100 classes, all the training data can fit in memory) and test on un-quantized vectors (after rotation or projection). In particular, as described in Sanchez and Perronnin (2011), the training data is compressed using PQ and decoded using the same codebook during LIBLINEAR optimization, and the test data is not compressed. Thus, SVM training time is comparable for all the methods with the same dimensionality.

Classification results are reported in Table 5.4. The full-dimensional BPBC incurs very little loss of accuracy over the original features, and the learned and random rotations work comparably. The performance of reduced-dimension BPBC degrades more noticeably, and the learned projection works better than the random one. RR+PQ outperforms the best version of BPBC by about 0.3%, but it is not clear whether the

difference is statistically significant. Interestingly, PQ without RR still produces reasonable classification results, while its performance on all other tasks was severely degraded. Evidently, the SVM training can absorb the quantization error, whereas nearest-neighbor search without a supervised learning stage cannot. Similar reasoning applies to $\alpha = 0$ binarization, whose performance is also fairly good for this task.

An interesting question is as follows: instead of starting with d-dimensional data and reducing the dimensionality to $d/2$ through binary coding, can we obtain the same accuracy if we start with $d/2$-dimensional features and maintain this dimensionality in the coding step? To answer this question, I perform classification on 12,800-dimensional VLAD for the same 100 classes and obtain an average accuracy of 43.08%. This is comparable to the accuracy of my 12,800-dimensional binary descriptor learned from the 25,600-dimensional VLAD. On the other hand, when I learn a 12,800-dimensional code starting with a 12,800-dimensional VLAD, I get an accuracy of 41.63%. Thus, starting with the highest possible dimensionality of the original features appears to be important for learning binary codes with the most discriminative power.

5.3 Discussion

This chapter has presented a novel bilinear formulation for learning binary codes that exploits the natural two-dimensional structure of many existing descriptors. This work differs from the ITQ and AQBC methods described in previous chapters in that it targets learning very high-dimensional binary codes. This is because for applications where very high-dimensional VLAD descriptors are used, very short codes lead to poor performance.

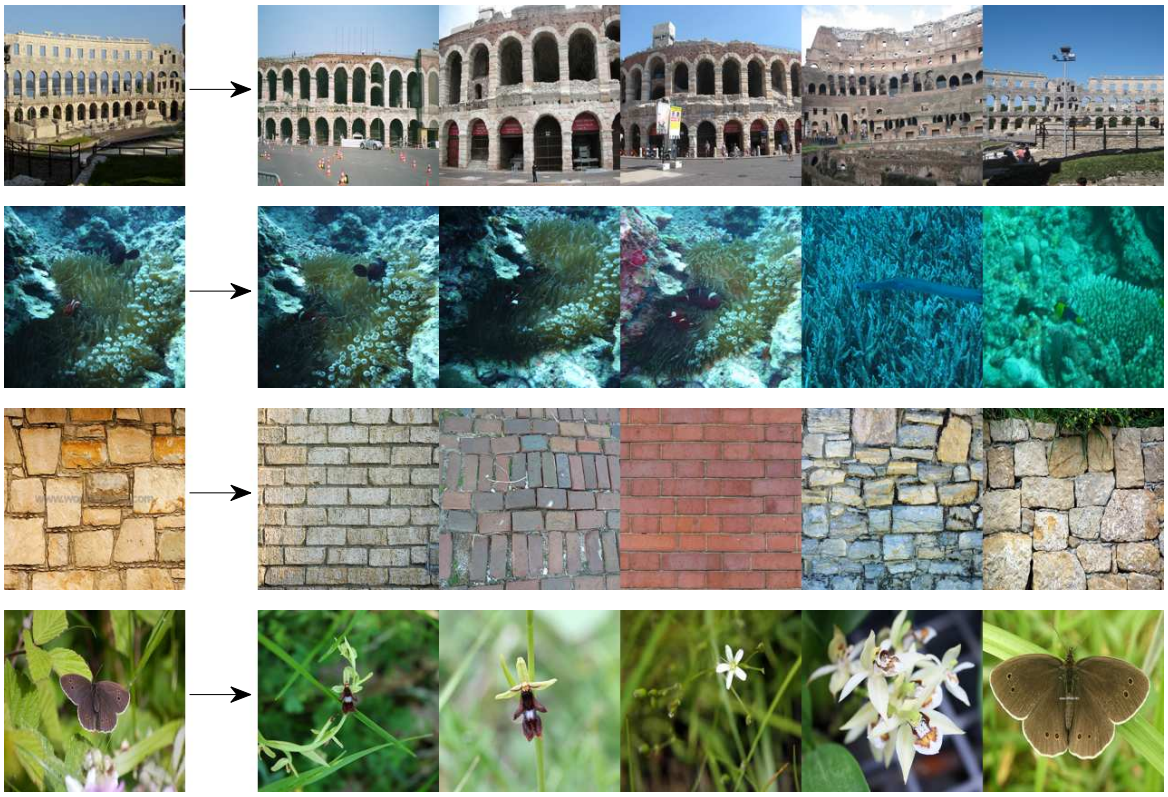


Figure 5.7: Sample image queries on Holiday+Flickr1M and ILSVRC2010 dataset using proposed BPBC approach.

To my knowledge, this is the first work that provides an efficient learning framework for generating a high-fidelity binary representation suitable for retrieval and recognition with extremely high-dimensional descriptors.

This work opens several interesting avenues for future research. As recent progress in recognition shows that using descriptors with *millions* of dimensions can lead to even better performance (Lin et al., 2011; Sanchez and Perronnin, 2011), it will be interesting to apply the BPBC approach on such data. It will also be interesting to design efficient indexing structures (e.g., min-hash (Chum et al., 2009)) on top of my high-dimensional binary codes to enable more efficient retrieval of such descriptors, as well as to explore

faster ways to train classifiers. As has already been shown in this chapter, the proposed bilinear projections can also be combined with PQ to improve its robustness and efficiency.

Chapters 3, 4, and 5 have already presented my three novel approaches for learning similarity preserving binary codes. In the next chapter, I am going to present applications for using binary codes to speed up large-scale image retrieval applications in Internet image collections, as well as for learning semantic binary embeddings for image retrieval.

CHAPTER 6: Combining Semantic Embeddings and Binary Codes

Chapters 3-5 have focused on learning binary codes from data in an unsupervised way, without using labeled data for learning the similarity between images. So far, this thesis has not addressed whether it is possible to use labeled data to obtain binary codes that better reflect *semantic* similarity between images – images belonging to the same object category having similar binary codes, and images belonging to different object categories having different binary codes (recall Figure 1.2). In this chapter, I present two applications that learn semantically consistent binary codes. The underlying idea is simple: I first use a supervised dimensionality reduction method to learn a semantically consistent low-dimensional embedding, and then apply binary coding methods on top of the embedding to obtain binary codes. I show that this approach can lead to very good accuracy for retrieval in Internet image collections. In particular, the binary codes can achieve accuracy as high as that of the original real-valued semantic embedding.

The first application I present in Section 6.2 is modeling Internet images with noisy tags, such as those on Flickr. Some previous works (Wang et al., 2009b,a; Quattoni et al., 2007; Gong et al., 2013a) have investigated using such data to improve recognition and retrieval. I adopt the canonical correlation analysis (CCA) (Hotelling, 1936) to learn the semantic embedding, and then apply ITQ to learn binary codes on top of the embedding. I found CCA works very well for modeling images and tags, and can lead to significant performance improvement over unsupervised baselines. In addition, the binary codes

obtained by ITQ work as well as the continuous CCA embedding, but with much smaller memory footprint and faster retrieval speed.

The second application that I present in Section 6.3 is using ITQ to learn binary classeme features (Torresani et al., 2010). The idea of classemes is to train large numbers of visual classifiers and then use their outputs on a new test image as a high-level feature for novel category recognition. My idea is to first learn the classeme representation, and then apply ITQ to binarize the embeddings. This chapter is based on my previous work (Gong et al., 2013c).

6.1 Semantic Binary Codes for Weakly Tagged Data

Weakly tagged data are abundant on the Internet. For example, there are billions of noisy tagged images on Flickr, and they provide a great source for learning visual representations. In this section, I show how to learn binary codes in a supervised setting using CCA, which has proven to be an effective tool for extracting a common latent space from two views (Foster et al., 2010) and is robust to noise (Blaschko and Lampert, 2008). CCA learns a joint latent space for images and tags where their correlation is maximized. We can use this latent space to perform image-to-image and cross-modal retrieval (i.e., using text to retrieve images or images to retrieve text).

I assume that each training image descriptor $\boldsymbol{x}_i \in \mathbb{R}^d$ has associated with it a label vector $\boldsymbol{y}_i \in \{0,1\}^t$, where t is the total number of labels (search keywords, tags) available, and a given entry of $\boldsymbol{y}_i$ is 1 if the image is associated with the corresponding label and 0 otherwise. Note that the labels do not have to be mutually exclusive and may be noisy.

The label vectors form the rows of a label matrix $Y \in \{0,1\}^{n \times t}$. The goal of CCA is to find projection directions $\boldsymbol{w}_k$ and $\boldsymbol{u}_k$ for feature and label vectors to maximize the correlation between the projected data $X\boldsymbol{w}_k$ and $Y\boldsymbol{u}_k$:

$$\mathcal{C}(\boldsymbol{w}_k, \boldsymbol{u}_k) = \boldsymbol{w}_k^T X^T Y \boldsymbol{u}_k$$

$$s.t. \quad \boldsymbol{w}_k^T X^T X \boldsymbol{w}_k = 1,\ \boldsymbol{u}_k^T Y^T Y \boldsymbol{u}_k = 1\,.$$

Maximizing the above objective function involves solving the following generalized eigenvalue problem to get $\boldsymbol{w}_k$ and $\boldsymbol{u}_k$:

$$\begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_1 \\ \boldsymbol{u}_1 \end{pmatrix} = \lambda \begin{pmatrix} S_{11} & 0 \\ 0 & S_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_1 \\ \boldsymbol{u}_1 \end{pmatrix}, \tag{6.1}$$

in which $S_{11} = X^T X$, $S_{12} = X^T Y$, $S_{22} = Y^T Y$, $S_{21} = Y^T X$. The leading generalized eigenvectors of (6.1) then give us a sequence of directions $\boldsymbol{w}_k$ and $\boldsymbol{u}_k$ that span the solution space. For a c-bit code, I form projections $\widehat{W} \in \mathbb{R}^{d \times c}$ for images and $\widehat{U} \in \mathbb{R}^{t \times c}$ for tags:

$$\widehat{W} = [\lambda_1^p \boldsymbol{w}_1, \ldots, \lambda_c^p \boldsymbol{w}_c] \tag{6.2}$$

$$\widehat{U} = [\lambda_1^p \boldsymbol{u}_1, \ldots, \lambda_c^p \boldsymbol{u}_c], \tag{6.3}$$

where λ_i^p is the pth power of the corresponding eigenvalues (Chapelle et al., 2002). Then I can obtain the embeddings for images as $V = X\widehat{W}$, and for tags as $T = Y\widehat{U}$. Finally,

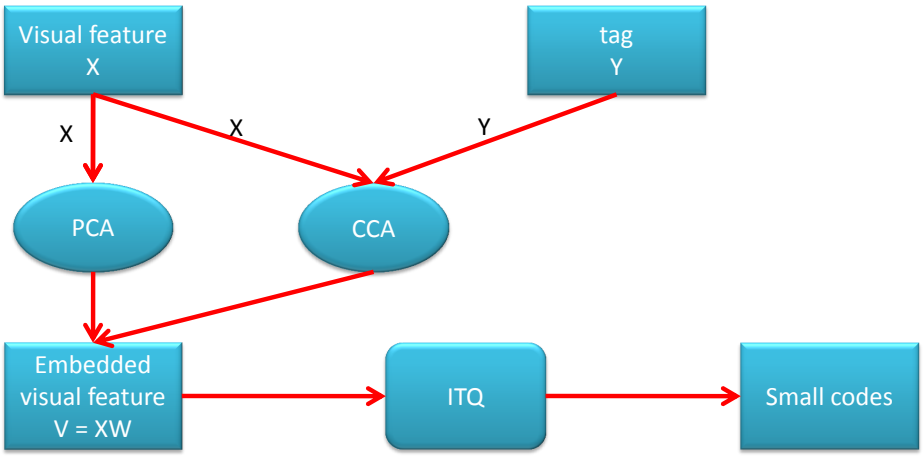


Figure 6.1: Logical flow of the PCA+ITQ and CCA+ITQ methods. Projecting the data with PCA corresponds to the unsupervised binary coding method ITQ described in Chapter 3, and projecting it with CCA corresponds to the supervised method described in this chapter.

I use the ITQ method from Chapter 3 to rotate the data in the latent space to obtain binary codes for images and the tags. A logical flow of the proposed method is shown in Figure 6.1.

6.1.1 Results on Tiny Images

I first evaluate the proposed combination of CCA with ITQ on the CIFAR and Tiny images datasets described in Section 3.2.1. Recall from Section 3.2.1 that the CIFAR dataset comes with manually verified keywords, while the 580,000 Tiny Images subset comes with noisy keywords that have not been verified by humans. These two different kinds of annotation allow us to explore the power of the CCA embedding given both "clean" and "noisy" supervisory information. For the "clean" scenario, I use a setup analogous to that of Section 3.2.3: namely, I set aside 1000 query images from the

CIFAR dataset and use the remaining CIFAR images as the training set and the database against which the queries are run. The labels in the training set are used to train the CCA embedding as described above. For the query images, the class labels are used only for benchmarking. For the "noisy" scenario, I learn the CCA embedding from all the Tiny Images that are disjoint from the CIFAR dataset using their unverified search keywords as the supervisory information. Then I apply the resulting embedding to the CIFAR dataset, split it into query images and reference database as in the clean scenario, and use the "clean" ground-truth labels for benchmarking.

As a baseline, I use the semi-supervised approach of Wang et al. (2010a), in which the label information of the n data points is used to construct an $n \times n$ matrix S that modulates the data covariance matrix. I set $S_{ij} = 1$ if two data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ have the same label, and 0 otherwise. Then I find the projection matrix W by taking the eigendecomposition of $X^T S X$. Note that Wang et al. (2010a), which assumes that few labeled images are available, regularizes $X^T S X$ by adding to it a small multiple of the covariance matrix $X^T X$. In my case, I have found this regularization to be unnecessary. I then take the data-dependent embedding W and perform ITQ refinement. I call the resulting method SSH-ITQ. Note that in Wang et al. (2010a), the semi-supervised embedding is combined with nonorthogonal relaxation (SSH-Nonorth), however, just as in Section 3.2, I have found that SSH-ITQ works better than SSH-Nonorth, so I only reproduce the SSH-ITQ results here.

Figure 6.2 shows the averaged precision at top 500 retrieved images for the "clean" and "noisy" versions of the CCA and SSH embeddings. For reference, I also include the

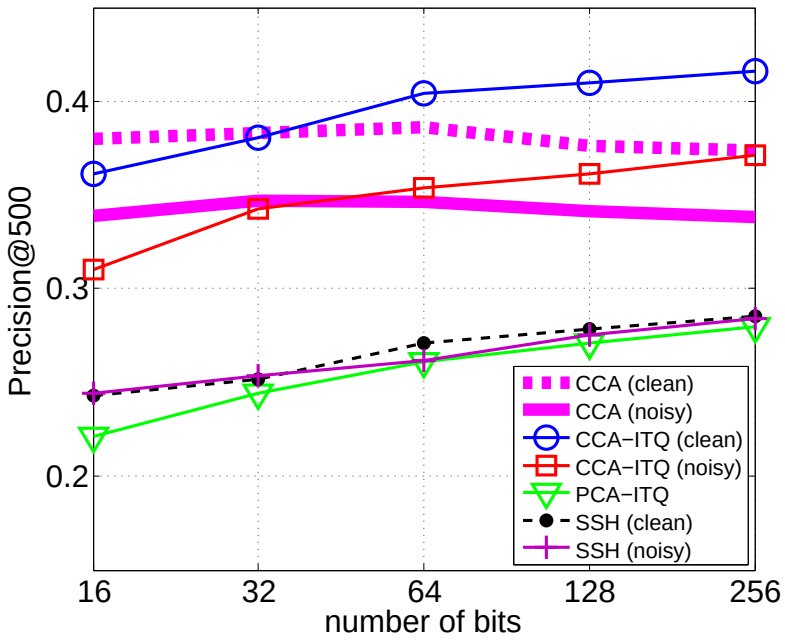


Figure 6.2: Average precision for top 500 retrieved images for supervised data embeddings based on clean and noisy labels.

performance of the unsupervised PCA embedding. We can see that CCA-ITQ with clean labels achieves the highest performance, while CCA-ITQ with noisy labels still gives a big improvement over the unsupervised PCA-ITQ. On the other hand, SSH produces a very small improvement over PCA, and there is almost no difference in the power of the SSH embeddings learned from clean and noisy data. Finally, Figure 6.3 shows the results of my methods on some sample queries. We can clearly see that when labels are incorporated, the results are much more semantically consistent.

6.1.2 Results on NUS-WIDE Dataset

Next, I test my method on the NUS-WIDE dataset (Chua et al., 2009), which is the largest publicly available multilabel dataset with ground truth annotations. This dataset

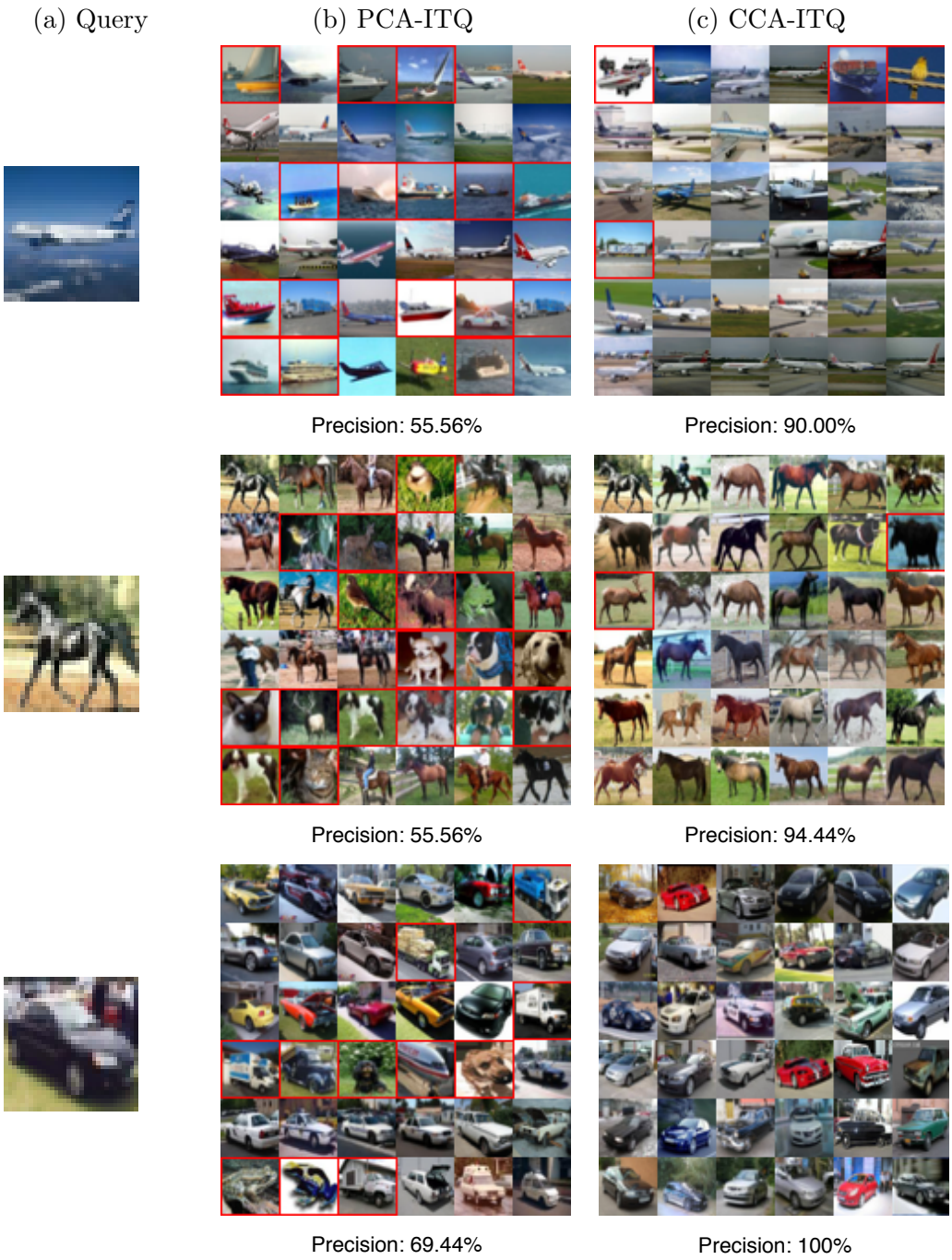


Figure 6.3: Image search (32-bit binary code) results on CIFAR. Red border means false positive. CCA leads to consistently better performance than PCA.

was collected at the National University of Singapore. It also originates from Flickr, and contains 269,648 images. The dataset is manually annotated with 81 ground truth concept labels, e.g., animal, snow, dog, reflection, city, storm, fog, etc. One important difference between NUS-WIDE and other datasets is that NUS-wide images may be associated with multiple ground truth labels. For the tags, I use the list of 1,000 words provided by Chua et al. (2009); on average, each image has 5.78 tags and 1.86 ground truth annotations. Each ground truth concept is also in the tag dictionary.

I extract the following features to represent the images:

- **GIST** (Oliva and Torralba, 2001): I resize each image to 200×200 and use three different scales $[8, 8, 4]$ to filter each RGB channel, resulting in 960-dimensional (320×3) GIST feature vectors.

- **SIFT**: I extract six different texture features based on two different patch sampling schemes: dense sampling and Harris corner detection. For each local patch, I extract **SIFT** (Lowe, 2004), **CSIFT** (van de Sande et al., 2011), and **RGBSIFT** (van de Sande et al., 2011). For each feature, I form a codebook of size 1,000 using k-means clustering and build a two-level spatial pyramid (Lazebnik et al., 2006), resulting in a 5000-dimensional vector. I will refer to these six features as D-SIFT, D-CSIFT, D-RGBSIFT, H-SIFT, H-CSIFT, and H-RGBSIFT.

- **HOG** (Dalal and Triggs, 2005): To represent texture and edge information on a larger scale, I use 2×2 overlapping HOG as described in (Xiao et al., 2010). I quantize the HOG features to a codebook of size 1,000 and use the same spatial

Method	Image to Image	Tag to Image
Original Feature (4500D)	31.36	–
CCA (512D)	43.63	42.54
CCA-ITQ (32 bits)	35.81	35.06
CCA-ITQ (64 bits)	39.28	38.21
CCA-ITQ (128 bits)	41.13	40.05
CCA-ITQ (256 bits)	42.30	41.55
CCA-ITQ (512 bits)	43.41	42.06

Table 6.1: Image retrieval results (average class label precision at top 20 returned images) on NUS-WIDE.

pyramid scheme as above, once again resulting in 5,000-dimensional feature vectors.

- **Color**: I use a joint RGB color histogram of 8 bins per dimension, for a 512-dimensional feature. Then I perform PCA to each feature, and reduce their dimensionality to 500, and finally concatenate all the features into a 4500 dimensional feature vector as our global feature.

Since the NUS dataset has both images and tags, I perform two tasks: image-to-image search, and tag-to-image search. For image-to-image search, I still follow the standard image retrieval evaluation, that report Precision@k, which is the percentage of correct images within top k retrieved images for every query. For tag-to-image retrieval, I simply use the tags to retrieve nearest image data points in the common CCA space (resp. binary space obtained by CCA-RR or CCA-ITQ), and also use the same metric to evaluate different methods. I randomly split the dataset into 219,648 training and 50,000 test images. As before, I learn the joint embedding using the training images, and test retrieval accuracy on testing dataset. In the test set, I randomly sample and fix

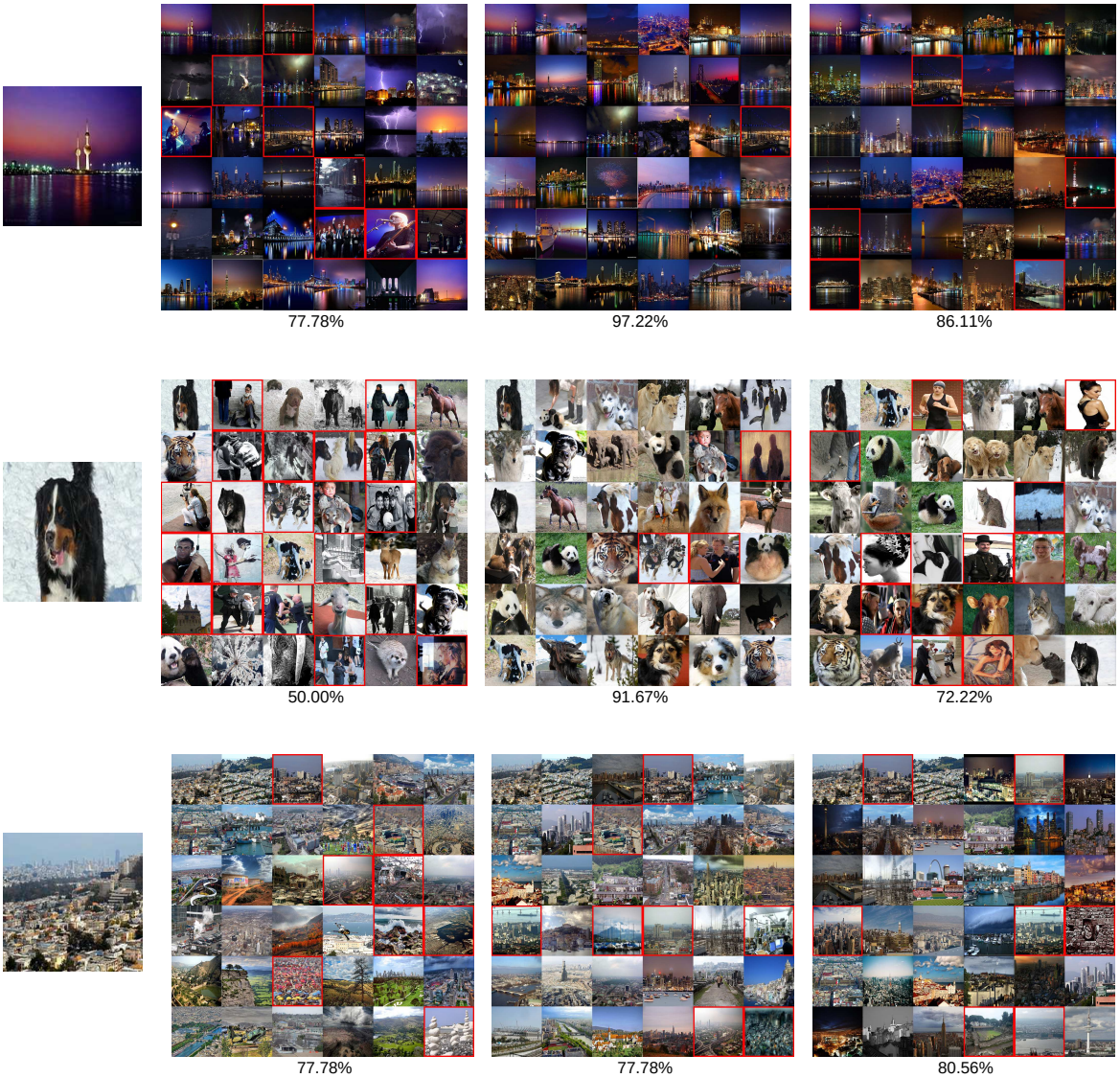
1,000 images as the queries, 1,000 images as the validation set, and retrieve the remaining images.

I report the results in Table 6.1, comparing retrieval results on original features, on CCA embedded space, and on the binary codes learned from CCA+ITQ. From the results, we can find that by performing CCA, we can obtain substantially better retrieval results than original feature. After CCA, performing ITQ to convert the data to binary codes can still obtain decent retrieval results. For a 64-bit binary code, the performance is slightly worse than that of the original embedding, but when we use a 256-bit code, the performance becomes comparable. I show qualitative image retrieval results in Figure 6.4, and tag-to-image retrieval results in Figures 6.6. It is interesting to mention that by using different keywords, we are able to find images that reflect the fine-grained meaning of the multiple keywords, as shown in Figure 6.6.

6.2 Binary classeme

Finally, I present an application of ITQ to learning binary classeme features (Bergamo et al., 2011; Rasiwasia et al., 2007; Torresani et al., 2010; Wang et al., 2009b). The idea of classeme is to train large numbers of visual classifiers and then use their outputs on a new test image as a high-level feature for novel category recognition. I first train a set of classemes from pre-defined categories, and then apply ITQ to binarize these classemes. A flowchart of the proposed binary classeme idea is shown in Figure 6.5.

For the experiments of this section, I use the ILSVRC2010 subset of ImageNet containing 1.2 million images from 1000 categories. I represent images using FV descriptors,



(a) Query. (b) Visual feature. (c) CCA. (d) CCA+ITQ.

Figure 6.4: Sample image search results on NUS-WIDE for compare Visual features, CCA, and CCA+ITQ. Red border means false positive.

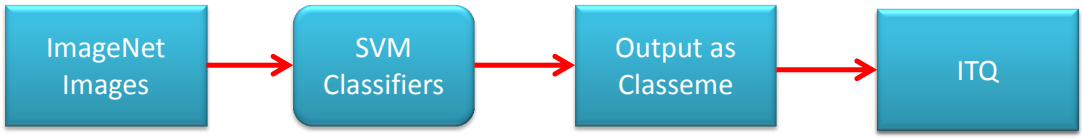


Figure 6.5: Data flow of the binary classeme idea. We first train classifiers from ImageNet images, and then pass the classifier output to ITQ.

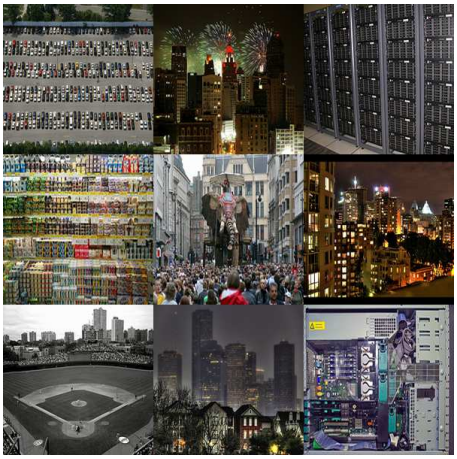
which are computed as follows. Images are resized to have an area of 100,000 pixels (if larger). SIFT (Lowe, 2004) and color descriptors (Perronnin et al., 2010b) are extracted from 24×24 patches every 6 pixels at 5 scales (by resizing the image by a factor $\sqrt{2}$ between two scales). The dimensionality of both SIFT and color descriptors is reduced to 64 by PCA. For each descriptor type, a Gaussian Mixture Model with 16 components is trained. Given an image, a 2048-dimensional FV is computed for SIFT and color by taking the gradient with respect to the mean and standard deviation parameters. The FVs are power- and L2-normalized as suggested in (Perronnin et al., 2010b). The final descriptor is the concatenation of the SIFT and color FV's, and is 4096-dimensional.

To learn the classemes, I randomly pick 950 classes from ILSVRC2010 and train LIBLINEAR SVM classifiers (Fan et al., 2008) on them. I use all the positive training data for each class, and randomly sample the same amount of negative training data from the remaining classes. The regularization parameter C of LIBLINEAR SVM is set to 2. Then the output scores of the 950 classifiers are used as classeme features to recognize the remaining 50 categories. To convert classemes to binary codes, I simply run ITQ (or any other binarization method) on top of the classifier outputs without any additional dimensionality reduction.

(a) river

(b) river reflection

(c) reflection city



(d) city

(e) city building

(f) city night

Figure 6.6: Example tag-to-image search results on NUS-WIDE.

Method	Dimensionality	FV Precision@50
Fisher Vector	4096×32 bits	33.40
classeme	950×32 bits	**39.24**
classeme-Threshold	950×1 bits	31.54
classeme-ITQ	950×1 bits	**38.98**
classeme-RR	950×1 bits	37.06
classeme-SH	950×1 bits	22.66
classeme-LSH	950×1 bits	36.27
classeme-SKLSH	950×1 bits	33.35

Table 6.2: Image retrieval results (average class label precision at top 50 returned images) on 50 classes from ILSVRC2010.

For my first experiment, I want to see how well binary classeme work as a representation for image retrieval. There are a total of 68,295 images in our 50 "testing" categories. I set aside 1,000 of those as "queries" and find their nearest neighbors among the remaining 67,295 images. Table 6.2 compares the average class label precision of the top 50 returned images for a number of coding methods. First of all, we can see that the continuous 950-dimensional classeme give better retrieval performance than the low-level FV features. Binarizing the classeme by directly thresholding them drops the performance significantly. However, binarizing them with ITQ results in almost no degradation (I have randomly sampled five other 950/50 class splits, and found that classeme-ITQ is consistently about 0.5% worse than the uncompressed classeme). For comparison, the table includes a number of other binary coding methods, all of which are worse than ITQ. Qualitative image retrieval results are shown in Figure 6.7.

Next, I would like to use classeme for novel category recognition, as suggested in Berg-

Method	Percent of training set 1%	5%	10%	50%	100%
Fisher Vector	35.55	52.21	**57.11**	**66.21**	**69.16**
classeme	38.54	51.49	56.18	64.31	66.77
classeme-Threshold	34.17	47.93	51.16	56.75	58.72
classeme-ITQ	**40.58**	**53.32**	56.62	61.12	62.68
classeme-RR	37.33	50.39	54.06	58.83	60.55

Table 6.3: Image classification results (%) on 50 classes from ILSVRC2010 using classeme trained on Fisher Vectors. Standard deviations are around 1%.

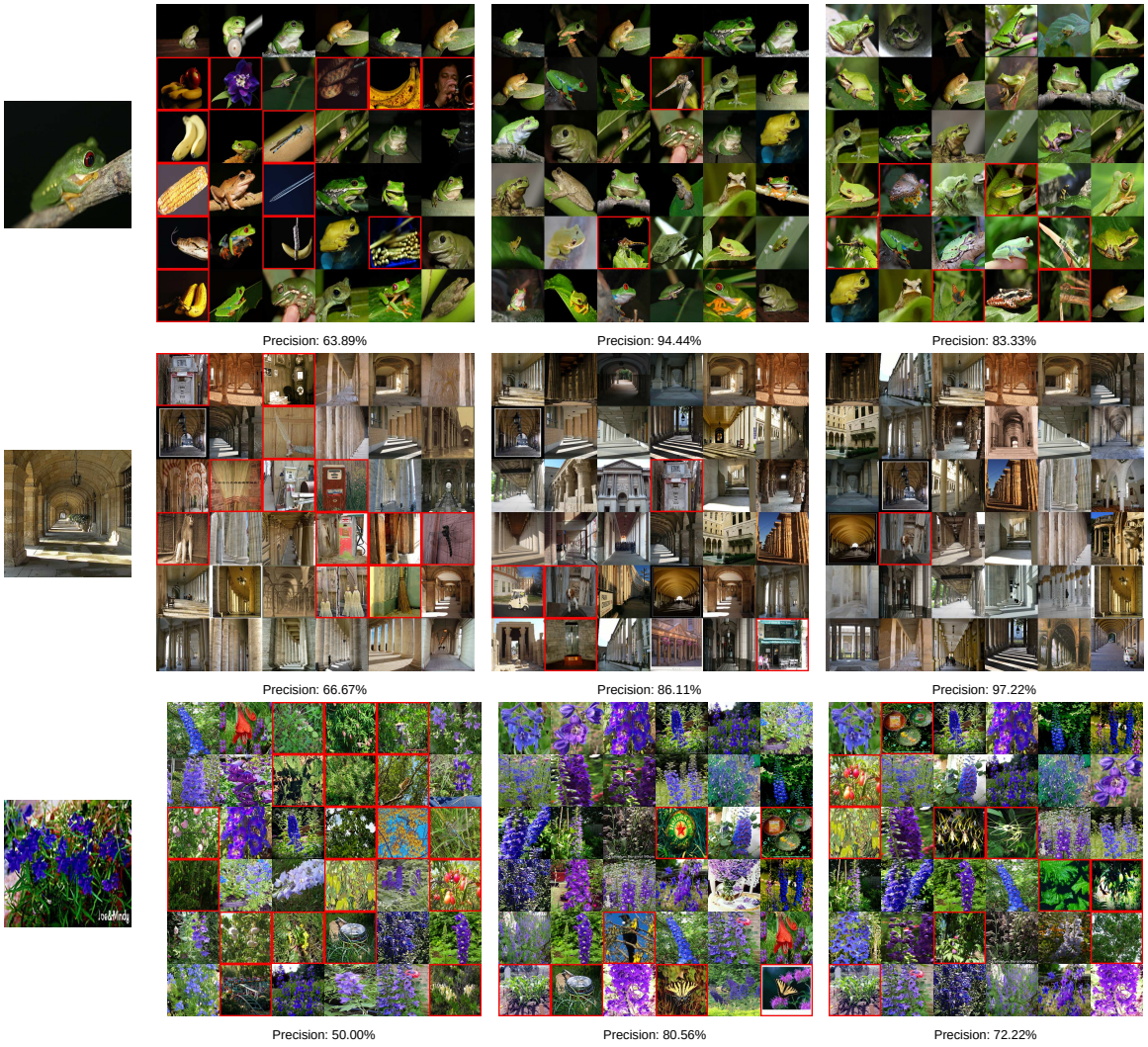
amo et al. (2011); Torresani et al. (2010). The motivating scenario is as follows: suppose we have already learned visual models for a number of categories using a large amount of training data, and then we are given a much smaller amount of training data for a never before seen category. Can we leverage our previous knowledge to quickly and efficiently learn a model for the new category?

In my experimental setup, I use the classifiers for the 950 categories as "previous knowledge." For the remaining 50 categories, I set aside 80% of the data for training, 10% for validation, and 10% for testing. I randomly sample different amounts of training data from the 80% training set for five trials, and train LIBLINEAR SVMs on the binary classeme features. The regularization parameter of linear SVM is tuned on the validation set using the grid of $[0.0002,\ 0.002,\ 0.02,\ 0.2,\ 2,\ 20]$. Table 6.3 shows the resulting classification accuracies. As before, the classeme-ITQ representation outperforms all the other binary embeddings. More interestingly, classeme-ITQ works better than continuous classeme or Fisher vectors for small training set size, where the higher-dimensional features may be overfitting. For larger training sizes, overfitting is no longer an issue,

and the original Fisher vectors work the best. Thus I conclude that ITQ-based binary classeme may be well suited for recognition of novel categories when a very small amount of training data is available.

6.3 Discussion

This chapter has presented two applications that combine semantic embeddings and binary codes. From the results, we can find that reasonably short binary codes (e.g. 256 bits) work as well as the continuous embedding for image retrieval. This confirms that a low-dimensional Hamming space has strong enough fidelity for capturing complex visual similarity between natural images. In the future, it will be more interesting to go beyond modeling images and tags, and study mapping images and sentences.



(a) Query. (b) Fisher Vector. (c) classeme. (d) classeme+ITQ.

Figure 6.7: Sample image search results on ImageNet (50 testing categories). I qualitatively compare Fisher Vector, classeme, and binary classeme (950 bits) obtained by ITQ. Red border means false positive. Note that in some of the examples, classeme-ITQ has lower precision than the continuous classeme.

CHAPTER 7: Hashing Revisited: Observations and Open Problems

Before I conclude the thesis, it will be interesting and important to review several methodological problems. Many papers on similarity preserving binary codes have been published. Almost all of them claim they have achieved state-of-the-art performance. However, I have found that the evaluation of different methods might have interesting and tricky issues. For example, for certain methods (Heo et al., 2012), it is not clear why the performance is significantly improved by using hyperspheres instead of hyperplanes as hash functions. In my own work described in Section 3.2.3, I have observed that the performance of ITQ hits a ceiling after 128 bits, however, I did not explain why that was the case. After a few years of research, I have taken notice of such issues, and I would like to take the opportunity to review them here. More importantly, I hope the discussion and experiments contained in this chapter will help make this line of research more systematic in the future.

The first question I am going to answer in Section 7.1 is "why can very short binary codes represent the similarity relationships among millions of images?" For example, a 64-bit code only has 64 levels in Hamming space, how can only 64 different distance levels distinguish so many points? The second question I am going to investigate in Section 7.2 is "which hashing algorithm is the best?" This question naturally arises when people want to apply hashing algorithms to their problem, however, every new paper shows better performance than before. I will show that it is important to properly define the

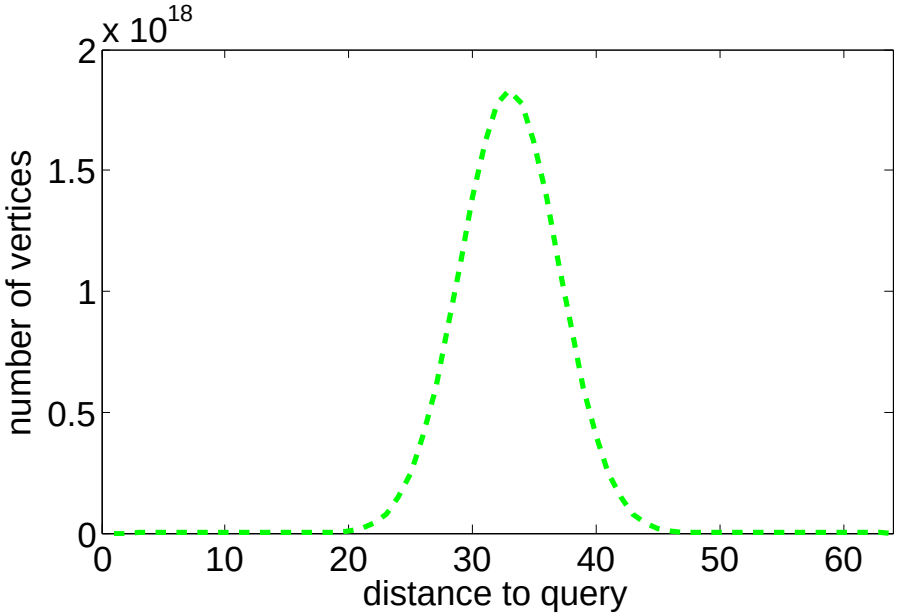
ground truth distance metric in order to correctly compare different methods. Finally in Section 7.3, I will also show that ground truth neighborhood definition also makes a very big difference in the evaluation of different methods. In particular, defining ground truth neighbors based on a fixed radius (ϵNN) or a fixed number of neighbors (kNN) will make a very big difference in terms of the relative performance of different methods, and needs to be investigated in the future.

7.1 Why Do Binary Codes Work?

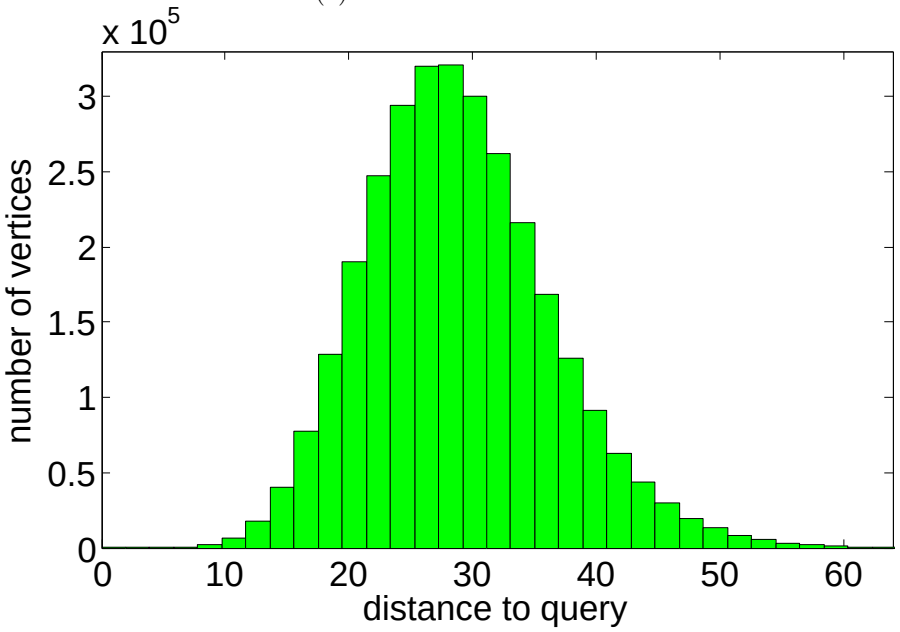
One might be curious why binary codes work well for image retrieval, especially when the number of data points in the dataset is large. If we use 64 bits, each query will only have 64 different distances (from 0 to 64) to all the other database points. How can only 64 distance levels distinguish so many points? This section is dedicated to giving an explanation to this question.

First of all, assume we have a c-bit binary code, corresponding to 2^c binary vertices serving as landmark points for quantization. The number of landmark points grows exponentially in the number of bits, and it should be sufficient for providing accurate quantization of the data points (Chapter 4). Assuming a uniform distribution of the data out of n total data points, on average $\frac{n}{2^c}$ points will be mapped to each vertex. Now, let's compute the number of binary vertices having Hamming radius r to a given query vertex. For every vertex, there are C_c^r other binary vertices having Hamming distance r to it. Thus, there are

$$\frac{n}{2^c} C_c^r = \frac{n}{2^c} \frac{c!}{r!(c-r)!} \tag{7.1}$$



(a) Theoretical distribution



(b) Empirical distribution on CIFAR dataset

Figure 7.1: (a) Analysis of the number of binary vertices having Hamming distance r to any binary vertex on 64-dimensional binary cube. (b) Empirical distribution of the number of data points having Hamming distance r to any data point on 64-dimensional binary cube. Result is generated using ITQ.

data points having Hamming distance r to every binarized query point. This means the number of points within a small Hamming radius to a query is very small, and it exponentially increases when r grows to $c/2$, and then decreases again when r grows from $c/2$ to c, which is shown in Figure 7.1 (a). This means the binary codes actually provide very fine-grained partitioning for near neighbors, and most of the points in the database will be lying far away from a query. To confirm this analysis, I show the empirical distance distribution on the CIFAR dataset in Figure 7.1 (b) using a 64-bit binary code. We can find the empirical distance distribution is similar to that of Figure 7.1 (a).

7.2 Distance Function Matters

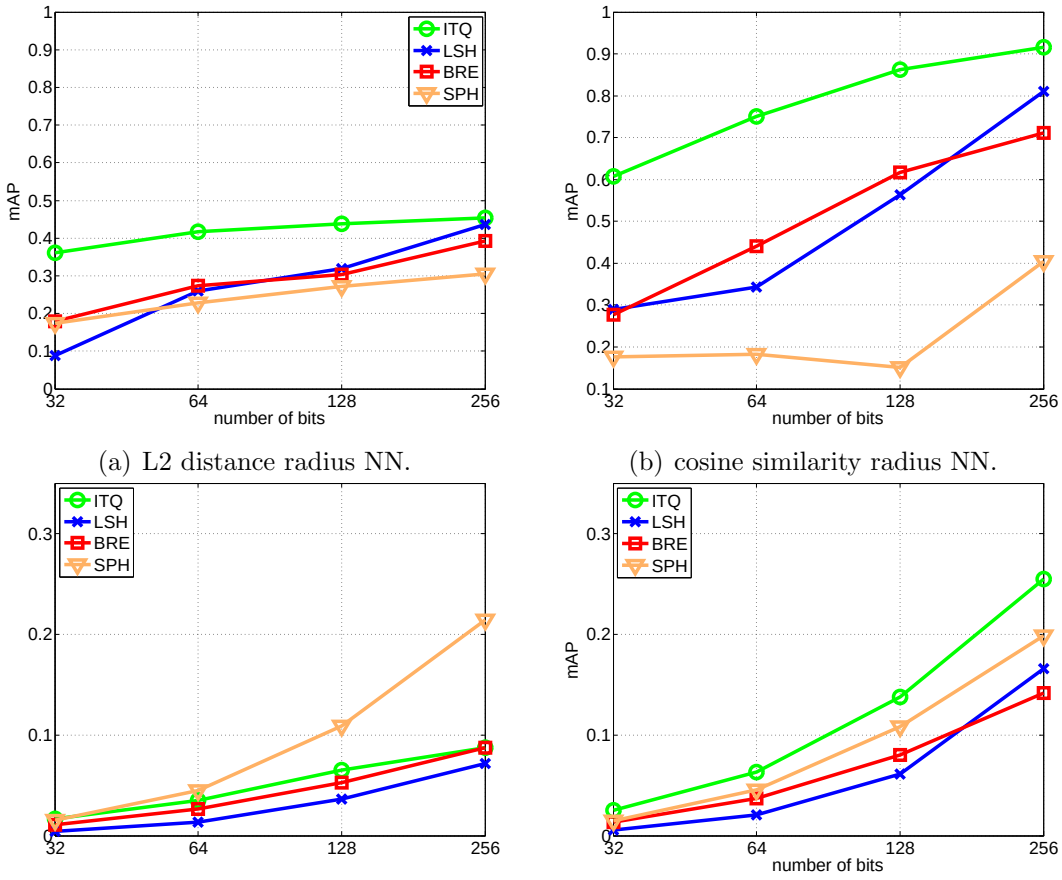
This section is mostly inspired by the experimental findings in Section 3.2.3, that the performance of ITQ hits a ceiling after 128 bits. At the time of performing that work, I did not have an understanding of why adding more bits does not improve performance anymore. This motivated me to investigate the relationship between different hashing methods and the way I measure ground truth distance in the original feature space. Most existing methods are evaluated based on Euclidean distance. However, I will show in this section that evaluation using the Euclidean metric might be misleading and might cause several subtle problems.

Let us use two ways to define the ground truth distance in the original feature space: Euclidean distance and cosine similarity. Note that using the cosine similarity is equivalent as normalizing the descriptors to unit L2 norm and then using the Euclidean distance. This normalization is a standard practice in recognition applications and usually

improves sematic accuracy. Let us also define ground truth neighbors in two ways: using a radius equal to the average distance to the 50th nearest neighbor, and using 100 nearest neighbors for each query.

To evaluate the different ground truth distance functions and neighborhood definitions, I perform nearest neighbor search experiments on the GIST1M dataset, which is from Jégou et al. (2011). This dataset contains 1 million images randomly downloaded from Flickr. The images are represented by a 960-dimensional GIST descriptors, and the descriptors are zero-centered and normalized. As baselines, we use LSH and ITQ, as well as binary reconstructive embedding (BRE) (Kulis and Darrell, 2009) and spherical hashing (SPH) (Heo et al., 2012). The linear version of BRE produces binary codes by thresholding linearly projected data, just like ITQ. SPH uses hyperspheres instead of hyperplanes to partition the space: points within a all are mapped to 1, and points outside are mapped to zero.

Most existing binary coding methods claim they are approximating L2 distance, so we would naturally expect them to work the best for L2 distance. The results are surprising: it seems all methods produce better results on the ground truth defined by cosine similarity instead of the L2 distance. Shouldn't all methods work the best for whatever they argue they are approximating? Based on the objective functions of ITQ, BRE, and SPH, they are indeed trying to preserve L2 distances. Then why do they all produce better results on the cosine neighbors instead of L2 neighbors? Our second observation is that SPH works the best for kNN on L2 distance, while ITQ works the best for $k$nn on cosine similarity. Why do different methods perform differently for different



(a) L2 distance radius NN.

(b) cosine similarity radius NN.

(c) L2 distance kNN.

(d) cosine similarity kNN.

Figure 7.2: Nearest neighbor search results for different distance metrics. (a) Results for different methods for L2 distance based radius NN. (b) Results for different methods for cosine similarity based radius NN. (c) Results for different methods for L2 distance based kNN. (d) Results for different methods for cosine similarity based kNN.

distance functions? Third, For ITQ evaluated using L2 distance, after 128 bits, it hits a ceiling, but continues to improve for cosine. What is the reason for this behavior?

To answer these questions, let's first compare and analyze three theoretical models for LSH. In particular, we are interested in the hash function they define, and the distances they are approximating. Here I define $\sigma(\cdot)$ as the hash function that produces binary codes for a vector, and define $h(\cdot)$ as the projection function.

Angle-LSH: The method proposed by Charikar (2002) that approximates angle:

$$\begin{aligned} \sigma(\boldsymbol{x}) &= \text{sgn}(h(\boldsymbol{x})), \\ h(\boldsymbol{x}) &= \boldsymbol{x}R, \end{aligned} \tag{7.2}$$

where R is drawn from random Gaussian.

L2-LSH: The method by Dong et al. (2008), which preserves the L2 distance:

$$\begin{aligned} \sigma_i(\boldsymbol{x}) &= \lfloor h_i(\boldsymbol{x}) \rfloor \mod 2, \\ h_i(\boldsymbol{x}) &= \frac{\boldsymbol{x}R_i + b_i}{T}, \quad i = 1, 2, \ldots, m \end{aligned} \tag{7.3}$$

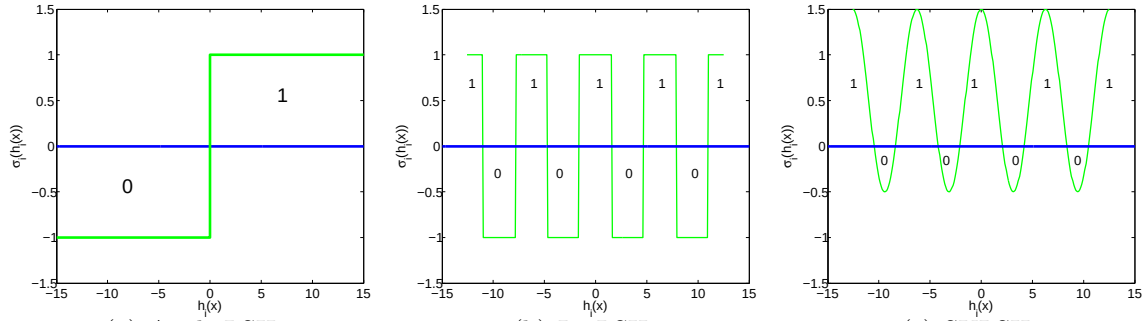
where $R_i \in R^d$ is a random projection vector drawn from Gaussian $N(0, I)$, and $b_i \in R$ is a scalar drawn from uniform distribution $[0, T)$.

SKLSH: The method by Raginsky and Lazebnik (2009) which preserves shift-invariant kernel values defined on top of L2 distance:

$$\begin{aligned} \sigma_i(\boldsymbol{x}) &= \text{sgn}(\cos(h_i(\boldsymbol{x})) + t_i) \\ h_i(\boldsymbol{x}) &= \frac{\boldsymbol{x}R_i + b_i}{T}, \quad i = 1, 2, \ldots, m \end{aligned} \tag{7.4}$$

where R_i is randomly drawn from a Gaussian $N(0, I)$, b_i is drawn from uniform $[0, 2T\pi]$, and t_i from uniform $[-1, 1]$.

The above formulations share many similarities. Both L2-LSH and SKLSH have very



(a) Angle LSH. (b) L2 LSH. (c) SKLSH.

Figure 7.3: A comparison of different random hashing models. (a) Angle-LSH, which partitions one projection direction into two parts. (b) L2-LSH, which uses mod to split one projection direction into many parts. (c) SKLSH, which uses sign of cosine to split one projection direction into many parts.

similar projection steps, that map the data by a random projection R_i with a shift b_i, and then scale the projected data by a scalar T, where T is considered as the window size in L2-LSH or bandwidth in SKLSH. The major difference is how to generate binary strings from the projected data. Angle-LSH simply takes the sign of the data, and splits the data into two parts. L2-LSH uses mod to generate bits, while SKLSH uses sign of $\cos(\cdot)$. The latter two are indeed intuitively very similar, as they are both trying to produce multiple cuts on data projected onto single directions. The only difference is SKLSH has a shift in y-axis which produces different sizes for 0 and 1. Figure 7.3 illustrates the cuts produced by all the methods on individual projection directions. With a single cut, Angle-LSH can only preserve the angle, but not the magnitude of the data. This is because single cut LSH only partitions the data on the hypersphere. For multi-cut LSH, it partitions each direction into several parts, which handles magnitude naturally. SPH enjoys similar idea to multi-cut in that it uses hyperspheres to partition the space into multiple regions. However, a single cut method is used in most binary coding formulations including ITQ and BRE, and this formulation can only preserve angle, and the magnitude

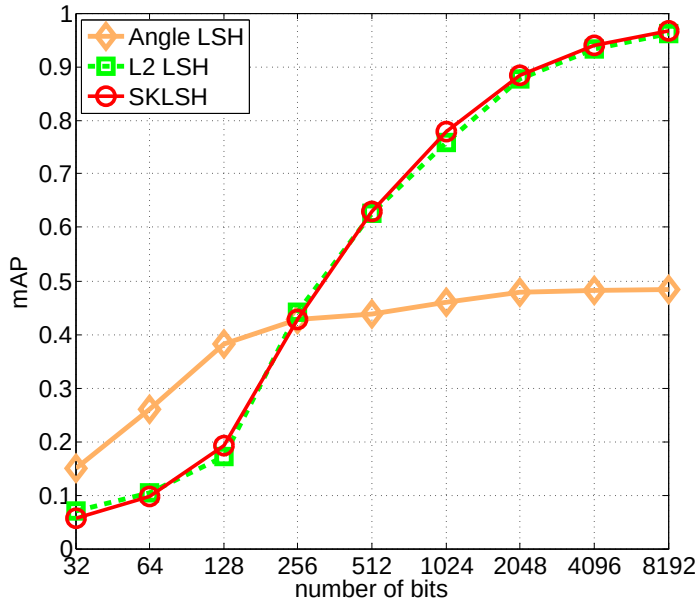


Figure 7.4: Performance comparison of different random hash models on GIST1M. We define ground truth neighbors using L2 distance and ϵNN.

of the data is completely lost. A real world experiment on GIST1M (GIST descriptors are not normalized) comparing these three hash models are shown in Figure 7.4. We can find angle-LSH is not able to capture L2 distance, and its performance saturates after 256 bits. Other methods work better and better when using more bits because they are approximating the L2 distance.

Next, I would like to analyze my ITQ method in detail. ITQ maps data from an arbitrary distribution to $\{-1, +1\}^c$, which is a set of points on the hypersphere (i.e., a set of points with constant norm). Such a mapping will never be able to approximate the original distance well. For ITQ, the quantization error is directly formulated as minimizing the Euclidean distance between a set of rotated original vectors and their quantized binary vectors:

$$\|\boldsymbol{b}_i - \boldsymbol{x}_i R\|_2^2, \tag{7.5}$$

where $\boldsymbol{b}_i = \text{sgn}(\boldsymbol{x}_i R)$. Expanding the above expression we have:

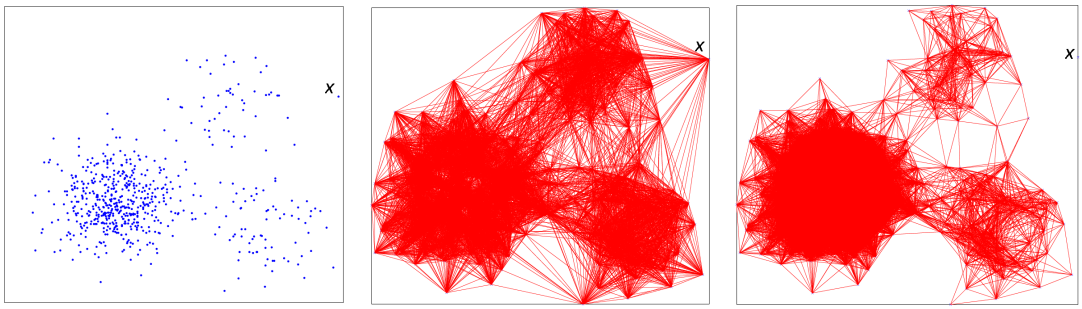
$$\|\boldsymbol{x}R - \boldsymbol{b}\|_2^2 \;\; = \;\; m_1^2 + 1 - 2\boldsymbol{x}^T R^T \boldsymbol{b} = m_1^2 + 1 - 2m_1 \cos(\theta), \tag{7.6}$$

where θ is the angle between $\hat{\boldsymbol{x}}R$ and $\boldsymbol{b}$, and m_1 is the magnitude of $\boldsymbol{x}$. It is clear that rotating the data by R can only reduce the angle between $\boldsymbol{x}$ and $\boldsymbol{b}$, and increase $\cos(\theta)$. Suppose that after the rotation, the rotated data become exactly binary valued, which means there is no error in the angle part ($\theta = 0$ and $\cos(\theta) = 1$). We will still get a nonnegative error term:

$$m_1^2 + 1 - 2m_1 = (m_1 - 1)^2 \geq 0. \tag{7.7}$$

This means that binarization by taking the sign of projected data is not able to preserve the magnitude of the data. The above analysis clearly shows why the performance of ITQ saturates after 128 bits. We choose ITQ as an example, due to its simplicity. For other more complicated methods, analogous arguments also hold. The SPH approach is harder to discuss as its hash function is not supported by a strong theoretical model. However, by using hyperspheres with different centers , it might be able to capture the magnitude of the data.

To conclude, many existing methods may be paying too much attention to how to better optimize their respective loss functions, while the correct form of the hash function is largely ignored. In particular, the $\mod(\cdot)$ function in L2-LSH or the $\text{sgn}(\cos(\cdot))$ function in SKLSH are the key for preserving L2 distance. However, in most recogni-



(a) Toy data. (b) KNN (c) ϵNN

Figure 7.5: (a) Toy dataset with three Gaussians. (b) The resulting 50NN graph. Notice the rightmost point x, it is far away from other data, but is still connected to many points. (c) ϵNN graph (ϵ is equal to average distance to 50NN). Notice that x is not connected to any other point.

tion applications that I have examined, cosine similarity (equivalently, normalizing the descriptors) leads to the highest semantic accuracy. Therefore, binary coding methods that preserve the magnitude of the data or L2 distance seem to have limited utility from the viewpoint of applications in computer vision. In general, when considering the appropriate definition of feature space distance, one needs to take the ultimate recognition application into account.

7.3 Neighborhood Definition Matters

Given a fixed distance function, next I study the impact of different neighborhood definitions:

ϵ **Nearest Neighbor**: for each query $\boldsymbol{x}$, its neighbors are points having distance smaller than a distance threshold ϵ.

k **Nearest Neighbor**: for each query point $\boldsymbol{x}$, it's neighbors are the k points nearest to it. This definition does not consider whether the distance is small or not.

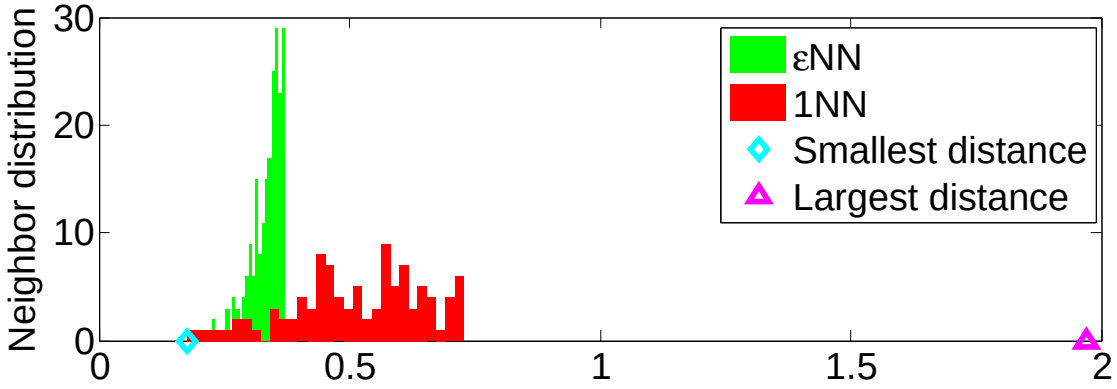
In my previous experiments and in most works by others, different methods are evaluated using ϵ nearest neighbor. The evaluation on k nearest neighbor is lacking. I will first visualize the distribution of these two different neighborhood structures in Figure 7.5. I show a toy example of three Gaussians, and their corresponding kNN and ϵNN graphs in Figure 7.5 (a), (b), and (c), respectively. We can see that for ϵNN, close points have more edges and faraway points have no neighbors. For kNN, since each point is forced to find its nearest k neighbors, the absolute distances do not matter too much. Queries lying far away from the data point cloud will still be connected to their k closest neighbors (e.g. point x in the figures).

Next, I show the statistics on the GIST1M dataset in Figure 7.6. I first show the histogram of distances for two different neighborhood definitions in Figure 7.6 (a). We can clearly find that for kNN, even for the first neighbors of each query, the distances already become quite big. This means algorithms that are very good at preserving small distances will not necessarily be good at preserving kNN. In addition, I show the distance distribution histograms for different k in Figure 7.6 (b). This figure further confirms that even 1NN of some points might have quite big distances that are comparable to random NN. I also show the number of neighbors for each query in Figure 7.6 (c), from which we can find for ϵNN, some queries might get lots of neighbors, while for others, they might not have any neighbors within the given radius. With these statistics in mind, next, I compare different algorithms on GIST1M dataset using these two neighborhood definitions as ground truth in Figure 7.7. For kNN, I use k=1 and k=100. For ϵNN, I define ϵ as the average distance to 50th nearest neighbors, and also show results for $\epsilon/3$.

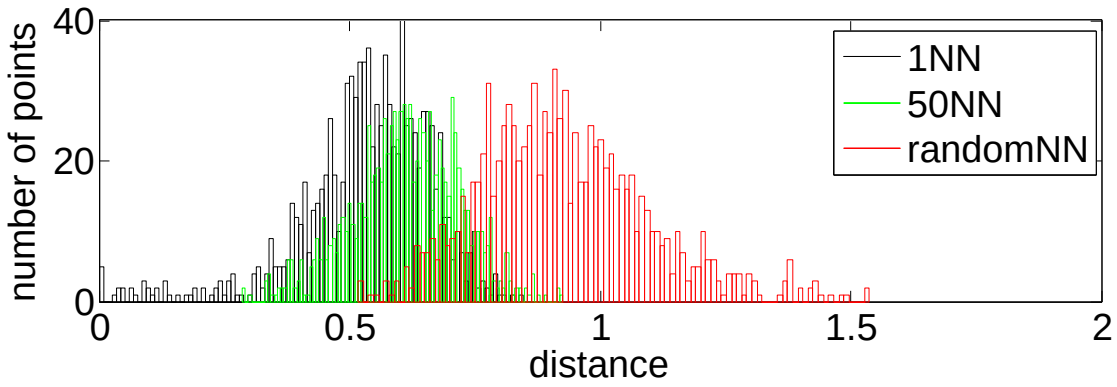
For all cases, I use cosine similarity.

From these results, we can have the following main observations: (1) ITQ works extremely well for ϵNN, while for other distance metrics, its advantage is not that significant; (2) SPH works quite well for kNN, but poorly for ϵNN; (3) overall, the performance of different methods on kNN is much worse than on ϵNN. To explain these observations, I analyze the statistics of the neighbors in Figure 7.8. First of all, in Figure 7.8 (a), we can find that for ϵNN, some queries are lying in a dense area of the data distribution, so they have lots of neighbors, while some are lying in a sparse area so they do not have neighbors. Parts (b) and (c) of the figure show: the number of Hamming neighbors retrieved for each query within a Hamming radius of 2 (red), how many of them are correctly retrieved as ϵNN (green), and how many of them are correctly retrieved as 100NN (blue) for ITQ and SPH. The neighbors retrieved by ITQ in Figure 7.8 (b) follow the general pattern of the data distribution: queries in dense area get lots of neighbors retrieved, and most of them are ground truth; queries in sparse area get no neighbors. By contrast, SPH in Figure 7.8 (c) produces a more balanced pattern, even for points in a sparse area. This suggests the underlying partition geometry of different methods: ITQ tends to map points in the same dense area to the same bucket (it avoids partitioning dense areas); while SPH tends to equally partition the space.

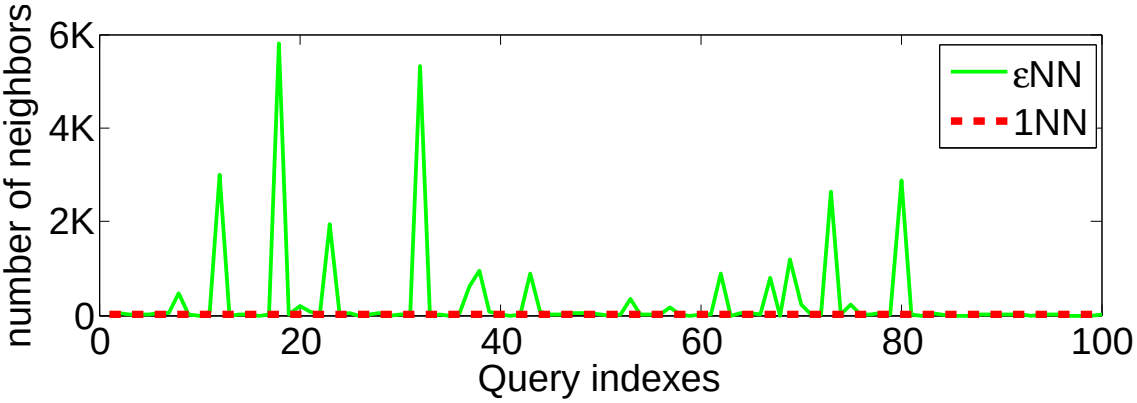
The above visualization helps to explain our previous observations. ITQ better preserves neighbors in dense areas with very small distances. Since most neighbors retrieved for ϵNN are in the dense area, ITQ will produce better results, which explains the first observation. However, when we define ground truth using kNN, the situation changes.



(a) Histogram of distance

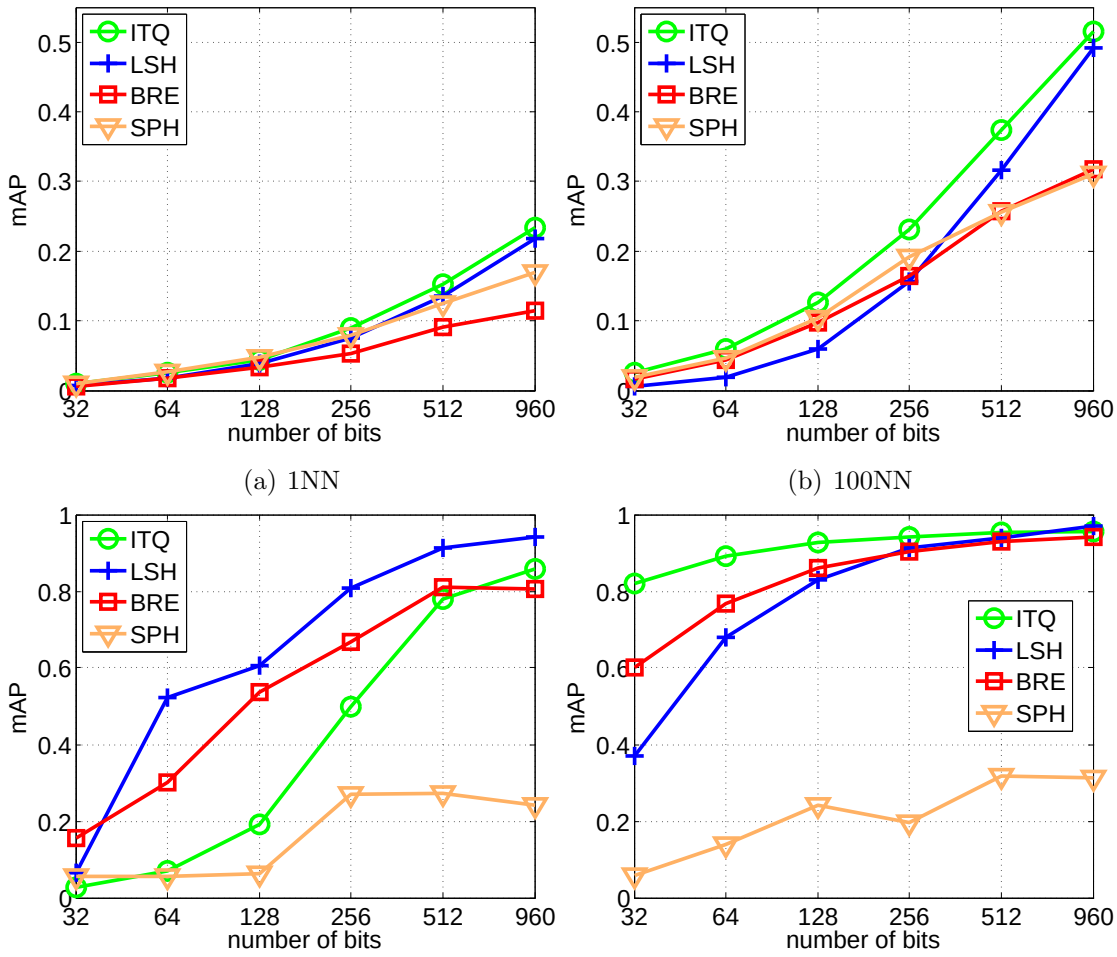


(b) Histogram of distance distributions



(c) # of neighbors per query

Figure 7.6: (a) The histogram of distances of different neighbors. (b) Histogram of distances for k nearest neighbors with different k. RandomNN is the distance between query to random points in the database. (c) The number of neighbors for each query point. ϵ is defined by average distance to 50th nearest neighbors.

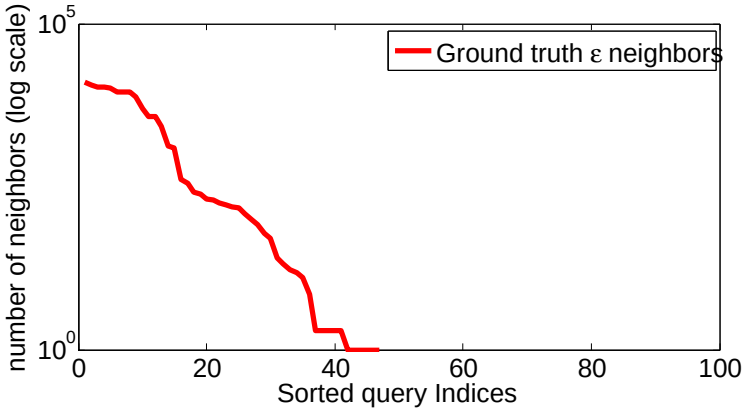


(a) 1NN (b) 100NN

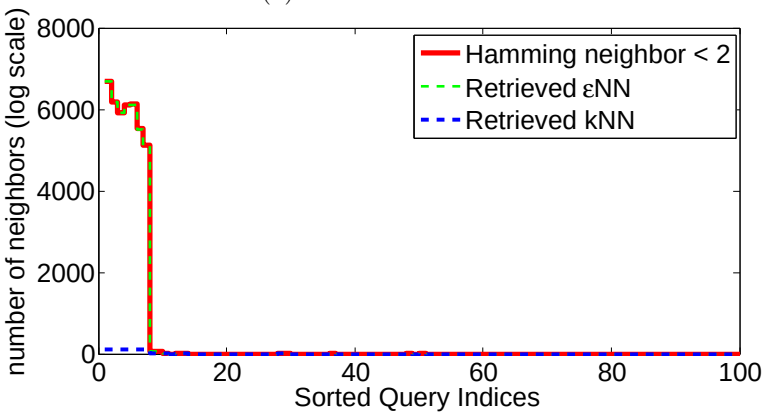
(c) $\epsilon/3$ (d) ϵ

Figure 7.7: Comparison of different methods based on different neighbor definition on GIST1M. (a, b) Ground truth is defined by 1NN and 100NN. (c, d) Ground truth is defined by points within $\epsilon/3$ or ϵ to queries.

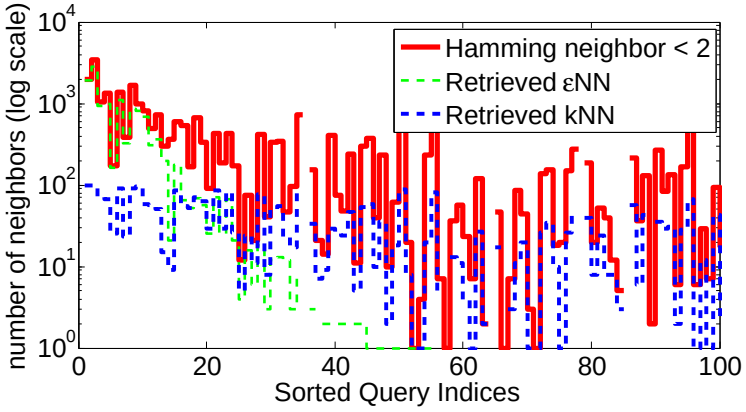
Since ITQ tends to map all points in dense areas together, there is not too much discrimination between them, which mixes the target close neighbors with many other points, so the top kNN returned become a choice between many points. Since SPH tends to partition the dense areas more, it might provide better discrimination for kNN retrieval. This explains the second observation. For the third observation, based on the theory of Dong et al. (2008), LSH-based methods are much better at preserving close neighbors (in



(a) Ground truth ϵ NN.



(b) ITQ



(c) SPH

Figure 7.8: The distribution of neighbors for different queries (y-axis in log scale). The curves are sorted based on the ground truth density of neighbors for each query. (a) The distribution of ground truth neighbors. (b) For ITQ, we can see it retrieves most neighbors for points lying in the dense area, while for points in the sparse area, it does not produce any neighbors. (c) For SPH, we can find the neighbors retrieved for different queries are quite balanced.

terms of distance), while they do worse for neighbors with larger distances. As discussed earlier, kNN contains lots of neighbors that are far away from each query. This explains why the performance of all methods on ϵNN is much better than on kNN.

The main insight from this section is the definition of ground truth neighbors makes a huge difference for the performance of different methods. In particular, no existing method dominates on all measures. I argue that for a particular application with particular neighbors, we should apply specific algorithms that optimize for that. Most existing methods are approximating small distances (corresponding to ϵNN), and among these, ITQ works the best. For kNN search, it is not clear which method is always the best. SPH tends to favor kNN, but it lacks strong theoretical guarantees. The current performance on kNN neighbors is still not satisfactory, and how to improve it is an open problem.

Finally, it is important to note that during our previous evaluations in Chapter 3, Chapter 4, Chapter 5, and Chapter 6, we have also evaluated different methods using semantic neighbors (neighbors belonging to the same class). However, I have not addressed this neighborhood definition in this chapter, since semantic neighbors heavily rely on the specific application and specific dataset. Although the performance of different methods according to semantic neighbor retrieval is hard to characterize analytically, it is ultimately crucial to evaluate different methods using this metric in real applications.

7.4 Discussion

This chapter revisits the problem of learning similarity preserving binary codes. I started with the question of "why only 64 distance levels for 64-bit code can distinguish millions of points?", and provided a justification using synthetic and real data. I then revealed two potential problems in the evaluation of different binary coding algorithms. In particular, I showed that the definitions of ground truth distance and neighborhood structure in the feature space can greatly affect the absolute and relative performance of different methods. This suggests that most existing works including my own are not very complete. No method will always work better than others for all metrics. I believe this line of research is still preliminary, and the problems I have raised need more investigation in the future.

CHAPTER 8: Discussion and Future Directions

8.1 Summary of Contributions

The main contributions of this thesis are as follows:

- Iterative Quantization (ITQ), presented in Chapter 3, is a state-of-the-art approach for learning binary codes for high-dimensional descriptors. It is simple, parameter-free, and very powerful. A few recent works (Norouzi et al., 2012a; Heo et al., 2012) use ITQ as a strong baseline, and some recent very successful methods (Ge et al., 2013; Norouzi and Fleet, 2013) are built on top of ITQ. In addition, several well-known papers, such as the ECCV2012 best paper (Kuettel et al., 2012), also rely on ITQ as a fast binary coding scheme for the handling of huge datasets.

- The angular quantization approach of Chapter 4 (AQBC) extends ITQ to high-dimensional sparse histogram data, which helps to further improve performance for bag-of-words features. It also provides new insights into the problem of learning binary codes for data not uniformly sampled on the hypersphere.

- The bilinear projection approach of Chapter 5 (BPBC) is the first one to address the scalability problem of encoding very high-dimensional dense descriptors. It achieves retrieval and classification performance performance almost as good as that of the original dense descriptor while being much more compact.

- The applications of Chapter 6 show how to learn semantic binary codes from noisy tagged Internet images or large amounts of manually labeled images. I also report promising image retrieval results using very compact binary codes on Internet-scale datasets.

- In Chapter 7, I review and reconsider the hashing problem and current research, and make several observations on methodological issues relating to evaluation.

8.2 Future Directions

To conclude the thesis, I would like to discuss my ideas on promising future research directions. For algorithmic development, I believe there is space for developing better fast indexing and coding methods for large-scale image retrieval. For example, several recent works (Ge et al., 2013; Norouzi and Fleet, 2013) combine the idea of ITQ and PQ to achieve state-of-the-art performance for indexing large image collections. It will be interesting to investigate whether other, potentially more flexible transformation models instead of rotation can be used (e.g., without orthogonality) to optimize quantization error. It will also be interesting to incorporate special structural constraints into the learning process, for example learning sparse projections with L1 regularization (Tibshirani, 1994), or finding group sparsity structure for groups of variables. Such constraints might further improve the performance of hashing and also provide additional efficiency gains. It may be worthwhile to go back to the original motivation of learning binary codes by a deep auto-encoder, as suggested in Salakhutdinov and Hinton (2009). Due to the recent great success of deep convolutional neural networks (CNN) (Krizhevsky

Figure 8.1: Sample results for finding Iconic images for "love" concept.

et al., 2012), it is promising to combine binary codes with deep CNN. For example, we might want to combine the supervised Hamming distance metric loss (Norouzi et al., 2012a) with the CNN architecture in Krizhevsky et al. (2012) to learn binary codes from large amounts of labeled or weakly labeled images. Another idea is to learn unsupervised binary codes from large amounts of unlabeled Internet data by using very deep auto-encoders with convolutional layers.

Next, let me discuss applications to which binary codes can be applied in the future. First, one interesting avenue for future research is to apply binary coding methods to large-scale multi-label image annotation (Barnard and Forsyth, 2001; Makadia et al., 2008; Guillaumin et al., 2009), in particular because the state-of-the-art performance for image annotation is achieved by nonparametric nearest-neighbor methods. To my knowledge, to date nobody has considered addressing this problem by applying binary codes as the underlying fast indexing method. Another interesting future application is mining Internet image collections. For example, finding "iconic images" of a visual concept (Berg and Berg, 2009; Raguram and Lazebnik, 2008) might require the mining of very large amounts of Internet images, involving operations such as clustering and

nearest-neighbor search. Using binary codes as an underlying representation can not only speed up the mining significantly but also allow us to handle much larger datasets. Some preliminary results (Figure 8.1) suggest that we can indeed find very interesting iconic clusters from millions of Internet images.

BIBLOGRAPHY

Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., and Szeliski, R. (2011). Building rome in a day. *Commun. ACM*, 54(10):105–112.

Andoni, A. and Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communication of the ACM*.

Arandjelović, R. and Zisserman, A. (2013). All about VLAD. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Banerjee, A., Dhillon, I. S., Ghosh, J., and Sra, S. (2005). Clustering on the unit hypersphere using von Mises-Fisher distributions. *Journal of Machine Learning Research*.

Barla, A., Odone, F., and Verri, A. (2003). Histogram intersection kernel for image classification. In *ICIP (3)*, pages 513–516.

Barnard, K. and Forsyth, D. (2001). Learning the semantics of words and pictures. In *International Conference on Computer Vision*.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517.

Berg, T. L. and Berg, A. C. (2009). Finding Iconic Images. In *IEEE International Conference on Computer Vision and Pattern Recognition*.

Bergamo, A., Torresani, L., and Fitzgibbon, A. (2011). Picodes: Learning a compact code for novel-category recognition. *Neural Information Processing Systems*.

Blaschko, M. B. and Lampert, C. H. (2008). Correlational spectral clustering. *IEEE Conference on Computer Vision and Pattern Recognition*.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCES97*, pages 21–29. IEEE Computer Society.

Carneiro, G., Chan, A., Moreno, P., and Vasconcelos, N. (2007). Supervised learning of semantic classes for image annotation and retrieval. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Chapelle, O., Weston, J., and Schölkopf, B. (2002). Cluster kernels for semi-supervised learning. In *Advances in Neural Information Processing Systems 15*, pages 585–592.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. *ACM Symposium on the Theory of Computing*.

Chen, X., Bai, B., Qi, Y., Lin, Q., and Carbonell, J. (2011). Sparse latent semantic analysis. *Siam Data Mining Conference*.

Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z., and Zheng, Y.-T. (2009). NUS-WIDE: A real-world web image database from National University of Singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece.

Chum, O. and Matas, J. (2010). Large scale discovery of spatilly related images. *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Chum, O., Perdoch, M., and Matas, J. (2009). Geometric min-hashing: Finding a (thick) needle in a haystack. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Csurka, G., Dance, C. R., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *In IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM.

Delhumeau, J., Gosselin, P.-H., Jégou, H., and Pérez, P. (2013). Revisiting the VLAD image representation. In *ACM Multimedia*, Barcelona, Spain.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*.

Dong, W., Charikar, M., and Li, K. (2008). Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. *SIGIR*.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research*.

Farhadi, A., Endres, I., Hoiem, D., and Forsyth, D. (2009). Describing objects by their attributes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1778–1785. IEEE.

Fei-Fei, L., Fergus, R., and Perona, P. (2006). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *Computer Vision and Image Understanding*.

Fei-Fei, L. and Perona, P. (2005). A bayesian hierarchical model for learning natural scene categories. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Felzenszwalb, P., McAllester, D., and Ramanan, D. (2008). A Discriminatively Trained, Multiscale, Deformable Part Model. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Fergus, R., Torralba, A., and Weiss, Y. (2009). Semi-supervised learning in gigantic image collections. *Neural Information Processing Systems*.

Foster, D. P., Johnson, R., Kakade, S. M., and Zhang, T. (2010). Multi-view dimensionality reduction via canonical correlation analysis. *Tech Report. Rutgers University*.

Frahm, J.-M., Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S., and Pollefeys, M. (2010). Building Rome on a cloudless day. In *European Conference on Computer Vision*.

Freund, Y., Dasgupta, S., Kabra, M., and Verma, N. (2007). Learning the structure of manifolds using random projections. *Neural Information Processing Systems*.

Ge, T., He, K., Ke, Q., and Sun, J. (2013). Optimized product quantization for approximate nearest neighbor search. *IEEE Conference on Computer Vision and Pattern Recognition*.

Gionis, A., Indyk, P., Motwani, R., et al. (1999). Similarity search in high dimensions via hashing. *VLDB*, 99:518–529.

Gong, Y., Ke, Q., Isard, M., and Lazebnik, S. (2013a). A multi-view embedding space for modeling internet images, tags, and their semantics. *International Journal of Computer Vision*.

Gong, Y., Kumar, S., Rowley, H. A., and Lazebnik, S. (2013b). Learning binary codes for high-dimensional data using bilinear projections. *IEEE Conference on Computer Vision and Pattern Recognition*.

Gong, Y., Kumar, S., Verma, V., and Lazebnik, S. (2012). Angular quantization-based binary codes for fast similarity search. *Neural Information Processing Systems*.

Gong, Y. and Lazebnik, S. (2011a). Comparing Data-Dependent and Data-Independent Embeddings for Classification and Ranking of Internet Images. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Gong, Y. and Lazebnik, S. (2011b). Iterative quantization: A procrustean approach to learning binary codes. *IEEE Conference on Computer Vision and Pattern Recognition*.

Gong, Y., Lazebnik, S., Gordo, A., and Perronnin, F. (2013c). Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Gordo, A., Perronnin, F., Gong, Y., and Lazebnik, S. (2011). Asymmetric distances for binary embeddings. *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Griffin, G., Holub, A., and Perona, P. (2008). The caltech 256. In *Caltech Technical Report*.

Guillaumin, M., Mensink, T., Verbeek, J., and Schmid, C. (2009). TagProp: Discriminative metric learning in nearest neighbor models for image auto-annotation. *International Conference on Computer Vision*.

Guillaumin, M., Verbeek, J., and Schmid, C. (2010). Multimodal semi-supervised learning for image classification. *IEEE Conference on Computer Vision and Pattern Recognition*.

Hardoon, D., Szedmak, S., and Shawe-Taylor, J. (2004). Canonical correlation analysis; an overview with application to learning methods. *Neural Computation*, 16.

Hays, J. and Efros, A. A. (2008). Im2gps: estimating geographic information from a single image. In *in IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

He, J., Radhakrishnan, R., Chang, S.-F., and Bauer, C. (2011). Compact hashing with joint optimization of search accuracy and time. *IEEE Conference on Computer Vision and Pattern Recognition*.

Heo, J.-P., Lee, Y., He, J., Chang, S.-F., and Yoon, S.-E. (2012). Spherical hashing. *IEEE Conference on Computer Vision and Pattern Recognition*.

Hotelling, H. (1936). Relations between two sets of variables. *Biometrika*, 28:312–377.

Jégou, H. and Chum, O. (2012). Negative evidences and co-occurrences in image retrieval: the benefit of PCA and whitening. In *ECCV - European Conference on Computer Vision*, Firenze, Italy.

Jégou, H., Douze, M., and Schmid, C. (2008). Hamming embedding and weak geometric consistency for large-scale image search. *European Conference on Computer Vision*.

Jégou, H., Douze, M., and Schmid, C. (2011). Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128.

Jégou, H., Douze, M., Schmid, C., and Pérez, P. (2010). Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision & Pattern Recognition*, pages 3304–3311.

Kong, W. and Li, W.-J. (2012). Double-bit quantization for hashing. *AAAI Conference on Artificial Intelligence*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Tech Report. University of Toronto*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*.

Kuettel, D., Guillaumin, M., and Ferrari, V. (2012). Segmentation propagation in imagenet. *European Conference on Computer Vision*.

Kulis, B. and Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. In *in Proc. Neural Information Processing Systems*, pages 1042–1050.

Kulis, B. and Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *International Conference on Computer Vision*.

Kulis, B., Jain, P., and Grauman, K. (2009). Fast similarity search for learned metrics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2143–2157.

Laub, A. J., editor (2004). *Matrix Analysis for Scientists and Engineers*. SIAM.

Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *IEEE Conference on Computer Vision and Pattern Recognition*.

Li, J. and Wang, J. (2008). Real-time computerized annotation of pictures. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Li, P. and Konig, C. (2011). Theory and applications of b-bit minwise hashing. *Communications of the ACM*.

Li, P., Shrivastava, A., Moore, J., and Konig, C. (2011). Hashing algorithms for large-scale learning. *Neural Information Processing Systems*.

Lin, Y., Cao, L., Lv, F., Zhu, S., Yang, M., Cour, T., Yu, K., and Huang, T. (2011). Large-scale image classification: Fast feature extraction and SVM training. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Liu, W., Kumar, S., and Chang, S.-F. (2011). Hashing with graphs. *International Conference on Machine Learning*.

Liu, W., Wang, J., Ji, R., Jiang, Y., and Chang, S. (2012). Supervised hashing with kernels. *Proceedings of Computer Vision and Pattern Recognition*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.

Maji, S. and Berg, A. (2009). Max-margin additive classifiers for detection. *International Conference on Computer Vision.*

Maji, S., Berg, A., and Malik, J. (2008). Classification using intersection kernel support vector machines is efficient. *IEEE Conference on Computer Vision and Pattern Recognition*.

Makadia, A., Pavlovic, V., and Kumar, S. (2008). A new baseline for image annotation. In *European Conference on Computer Vision*.

Manning, C. D. and Schütze, H. (1999). Foundations of statistical natural language processing. *MIT Press*.

Miller, G. A. (1995). Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM*, 38:39–41.

Monay, F. and Gatica-Perez, D. (2004). PLSA-based image auto-annotation: Constraining the latent space. In *ACM Multimedia*.

Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press.

Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Norouzi, M. and Fleet, D. (2012). Minimal loss hashing for compact binary codes. *International Conference on Machine Learning*.

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

Norouzi, M., Fleet, D., and Salakhutdinov, R. (2012a). Hamming distance metric learning. *Neural Information Processing Systems*.

Norouzi, M., Punjani, A., and Fleet, D. J. (2012b). Fast search in hamming space with multi-index hashing. In *Computer Vision and Pattern Recognition (CVPR) 2012*. IEEE.

Oliva, A. and Torralba, A. (2001). Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*.

Perronnin, F. and Dance, C. R. (2007). Fisher kernels on visual vocabularies for image categorization. *IEEE Conference on Computer Vision and Pattern Recognition*.

Perronnin, F., Liu, Y., Sanchez, J., and Poirier, H. (2011). Large-scale image retrieval with compressed fisher vectors. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Perronnin, F., Sanchez, J., , and Liu, Y. (2010a). Large-scale image categorization with explicit data embedding. *IEEE Conference on Computer Vision and Pattern Recognition*.

Perronnin, F., Sanchez, J., and Mensink, T. (2010b). Improving the fisher kernel for large-scale image classification. *European Conference on Computer Vision*.

Pirsiavash, H., Ramanan, D., and Fowlkes, C. (2009). Bilinear classifiers for visual recognition. *Neural Information Processing Systems*.

Quattoni, A., Collins, M., and Darrell, T. (2007). Learning visual representations using images with captions. *IEEE Conference on Computer Vision and Pattern Recognition*.

Raginsky, M. and Lazebnik, S. (2009). Locality sensitive binary codes from sift-invariant kernels. *Neural Information Processing Systems*.

Raguram, R. and Lazebnik, S. (2008). Computing iconic summaries of general visual concepts. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. *Neural Information Processing Systems*.

Rasiwasia, N., Moreno, P., and Vasconcelos, N. (2007). Bridging the gap: Query by semantic example. *IEEE Transactions on Multimedia*.

Revaud, J., Douze, M., Schmid, C., and Jégou, H. (2013). Event retrieval in large video collections with circulant temporal encoding. In *CVPR - International Conference on Computer Vision and Pattern Recognition*, Portland, United States. IEEE.

Salakhutdinov, R. and Hinton, G. (2009). Semantic hashing.

Salton, G. and McGill, M. J. (1986). *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

Samet, H. (1990). Applications of spatial data structures.

Sanchez, J. and Perronnin, F. (2011). High-dimensional signature compression for large-scale image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Scholkopf, B., Smola, A. J., and Muller, K.-R. (1997). Kernel principal component analysis. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN-1997)*, pages 583–588, Berlin.

Schonemann, P. (1966). A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31.

Schönemann, P. (1968). On two-sided orthogonal Procrustes problems. *Psychometrika*.

Shakhnarovich, G., Darrell, T., and Indyk, P., editors (2006). *Nearest-Neighbors methods in Learning and Vision: Theory and Practice*. MIT Press.

Shrivastava, A. and Li, P. (2012). Fast near neighbor search in high-dimensional binary data. *European Conference on Machine Learning*.

Sivic, J. and Zisserman, A. (2003). Video Google: A text retrieval approach to object matching in videos. In *International Conference on Computer Vision*.

Strecha, C., Bronstein, A. M., Bronstein, M. M., and Fua, P. (2010). Ldahash: Improved matching with smaller descriptors. *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288.

Tighe, J. and Lazebnik, S. (2010). SuperParsing: Scalable Nonparametric Image Parsing with Superpixels. In *European Conference on Computer Vision*.

Tolias, G., Avrithis, Y., and Jégou, H. (2013). To aggregate or not to aggregate: selective match kernels for image search. In *International Conference on Computer Vision*, Sydney, Australia.

Torralba, A., Fergus, R., and Freenman, W. (2008a). 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Torralba, A., Fergus, R., and Weiss, Y. (2008b). Small codes and large image databases for recognition. *IEEE Conference on Computer Vision and Pattern Recognition*.

Torresani, L., Szummer, M., , and Fitzgibbon, A. (2010). Efficient object category recognition using classemes. *European Conference on Computer Vision*.

van de Sande, K. E. A., Gevers, T., and Snoek, C. G. M. (2011). Empowering visual categorization with the gpu. *IEEE Transactions on Multimedia*, 13(1):60–70.

Verma, Y. and Jawahar, C. V. (2012). Image annotation using metric learning in semantic neighbourhoods. In *European Conference on Computer Vision*.

Wang, G., Hoiem, D., and Forsyth, D. (2009a). Building text features for object image classification. *IEEE Conference on Computer Vision and Pattern Recognition*.

Wang, G., Hoiem, D., and Forsyth, D. (2009b). Learning image similarity from Flickr groups using stochastic intersection kernel machines. *International Conference on Computer Vision*.

Wang, J., Kumar, S., and Chang, S.-F. (2010a). Semi-supervised hashing for large-scale image retrieval. *IEEE Conference on Computer Vision and Pattern Recognition*.

Wang, J., Kumar, S., and Chang, S.-F. (2010b). Sequential projection learning for hashing with compact codes. *International Conference on Machine Learning*.

Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., and Gong, Y. (2010c). Locality-constrained linear coding for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3360–3367. IEEE.

Wang, X.-J., Zhang, L., Li, X., and Ma, W.-Y. (2008). Annotating images by mining image search results. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 30(11):1919–1932.

Weiss, Y., Fergus, R., and Torralba, A. (2008). Spectral hashing. *Neural Information Processing Systems*.

Weiss, Y., Fergus, R., and Torralba, A. (2012). Multidimensional spectral hashing. *European Conference on Computer Vision*.

Weston, J., Bengio, S., and Usunier, N. (2011). Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*.

Xiao, J., Haysy, J., Ehinger, K. A., Oliva, A., and Torralba, A. (2010). SUN Database: Large-scale Scene Recognition from Abbey to Zoo. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Yang, J., Yu, K., Gong, Y., and Huang, T. (2009). Linear spatial pyramid matching using sparse coding for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1794–1801. IEEE.

Ye, J., Janardan, R., and Li, Q. (2004). Two-dimensional linear discriminant analysis. *Neural Information Processing Systems*.

Yu, S. X. and Shi, J. (2003). Multiclass spectral clustering. *International Conference on Computer Vision*.

Zipf, G. K. (1935). The psychobiology of language. *Houghton-Mifflin*.