Learning hash codes for multimedia retrieval

Junjie Chen

Follow this and additional works at: https://repository.hkbu.edu.hk/etd_oa

Recommended Citation
https://repository.hkbu.edu.hk/etd_oa/684

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at HKBU Institutional Repository. It has been accepted for inclusion in Open Access Theses and Dissertations by an authorized administrator of HKBU Institutional Repository. For more information, please contact repository@hkbu.edu.hk.
DATE: August 28, 2019

STUDENT'S NAME: CHEN Junjie

THESIS TITLE: Learning Hash Codes for Multimedia Retrieval

This is to certify that the above student's thesis has been examined by the following panel members and has received full approval for acceptance in partial fulfilment of the requirements for the degree of Master of Philosophy.

Chairman: Dr Li Hung Wing
Associate Professor, Department of Chemistry, HKBU
(Designated by Dean of Faculty of Science)

Internal Members: Dr Chen Li
Associate Professor, Department of Computer Science, HKBU
(Designated by Head of Department of Computer Science)

Dr Cheung Kwok Wai
Associate Professor, Department of Computer Science, HKBU

External Members: Dr Chung Fu-lai Korris
Associate Professor
Department of Computing
The Hong Kong Polytechnic University

Issued by Graduate School, HKBU
Learning Hash Codes for Multimedia Retrieval

CHEN Junjie

A thesis submitted in fulfilment of the requirements
for the degree of
Master of Philosophy

Principal Supervisor:
Dr. CHEUNG Kwok Wai (Hong Kong Baptist University)

August 2019
DECLARATION

I hereby declare that this thesis represents my own work which has been done after registration for the degree of MPhil at Hong Kong Baptist University, and has not been previously included in a thesis or dissertation submitted to this or any other institution for a degree, diploma or other qualifications.

I have read the University’s current research ethics guidelines, and accept responsibility for the conduct of the procedures in accordance with the University’s Research Ethics Committee (REC). I have attempted to identify all the risks related to this research that may arise in conducting this research, obtained the relevant ethical and/or safety approval (where applicable), and acknowledged my obligations and the rights of the participants.

Signature: Junjie Chen

Date: August 2019
Abstract

The explosive growth of multimedia data in online media repositories and social networks has led to the high demand of fast and accurate services for large-scale multimedia retrieval. Hashing, due to its effectiveness in coding high-dimensional data into a low-dimensional binary space, has been considered to be effective for the retrieval application. Despite the progress that has been made recently, how to learn the optimal hashing models which can make the best trade-off between the retrieval efficiency and accuracy remains to be open research issues. This thesis research aims to develop hashing models which are effective for image and video retrieval. An unsupervised hashing model called APHash is first proposed to learn hash codes for images by exploiting the distribution of data. To reduce the underlying computational complexity, a methodology that makes use of an asymmetric similarity matrix is explored and found effective. In addition, the deep learning approach to learn hash codes for images is also studied. In particular, a novel deep model called DeepQuan which tries to incorporate product quantization methods into an unsupervised deep model for the learning. Other than adopting only the quadratic loss as the optimization objective like most of the related deep models, DeepQuan optimizes the data representations and their quantization codebooks to explores the clustering structure of the underlying data manifold where the introduction of a weighted triplet loss into the learning objective is found to be effective. Furthermore, the case with some labeled data available for the learning is also considered. To alleviate the high training cost (which is especially crucial given a large-scale database), another hashing model named Similarity Preserving Deep Asymmetric
Quantization (SPDAQ) is proposed for both image and video retrieval where the compact binary codes and quantization codebooks for all the items in the database can be explicitly learned in an efficient manner. All the aforementioned hashing methods proposed have been rigorously evaluated using benchmark datasets and found to outperform the related state-of-the-art methods.

**Keywords:** hashing, quantization, multimedia retrieval, image retrieval
Acknowledgements

Foremost, I would like to express my sincere gratitude to my super nice supervisor Prof. William Kwok-Wai Cheung for his patient guidance and useful advices during my research study. I am very grateful for his encouragement and help during the past two years, which is like sunlight bringing me hope and warmth in difficult times. I could not have imagined having a better supervisor for my M.Phil. study.

I would like to thank the rest of the panel members: Prof. Hung-Wing Li, Prof. Li Chen and Prof. Korris Fu-Lai Chung (from Hong Kong Polytechnic University).

My sincere thanks also go to Dr. Anran Wang for offering me a research assistant position at our university before formal admission, and Dr. Takanori Maehara for offering me a research internship opportunity at Discrete Optimization Unit, RIKEN Center for Advanced Intelligence Project.

I would like to thank my friendly roommates and colleagues: Dong Qian, Qi Tan and Dr. Yang Zhou for the lunches we had together, and for the encouragement we gave to each other and for all the fun we had together. I thank Dr. Lu Yu and Mario Coutino (from TU Delft) for giving me advices about life and research. I would also thank my colleagues: Hao Li, Shujin Ye, Kejing Yin, Lihong Song, Dan Peng, Colman Tse, and Dr. Jacky Chan.

Last but not least, I would like to thank my parents Tianchi Chen and Lizhen Chen, for supporting me spiritually throughout my life.
# Table of Contents

Declaration i

Abstract ii

Acknowledgements iv

Table of Contents v

List of Tables viii

List of Figures x

Chapter 1 Introduction 1
  1.1 Problem Definition .................................. 1
  1.2 Thesis Outline ................................... 4

Chapter 2 Related Work 7
  2.1 Nearest Neighbor Search ............................. 7
  2.2 Data-independent Hashing ............................ 8
  2.3 Data-dependent Hashing ............................... 9
    2.3.1 Deep Learning .................................. 10
    2.3.2 Supervised Hashing .............................. 10
    2.3.3 Unsupervised Hashing ........................... 17

Chapter 3 APHash: Anchor-based Probability Hashing for Image Retrieval 23
3.1 Motivation ........................................... 23
3.2 Methodology ........................................... 26
    3.2.1 Problem Formulation ......................... 26
    3.2.2 Optimization ................................ 29
    3.2.3 Hash Function Learning ...................... 30
3.3 Experimental Results ................................. 31
    3.3.1 Datasets and Experimental Setup ............. 31
    3.3.2 Comparison with State-of-the-art Methods ...... 37
    3.3.3 Model Analysis ................................. 39
3.4 Conclusions ........................................ 41

Chapter 4 DeepQuan: Learning Deep Unsupervised Binary Codes
    for Image Retrieval ................................ 42
    4.1 Motivation ........................................ 42
    4.2 Methodology ...................................... 45
        4.2.1 Model Formulation ......................... 45
        4.2.2 Approximate Nearest Neighbor Search ...... 49
        4.2.3 Alternating Optimization .................... 50
    4.3 Experimental Results .............................. 52
        4.3.1 Datasets and Experimental Setting .......... 52
        4.3.2 Performance Comparison .................... 55
        4.3.3 Empirical Analysis of DeepQuan ............. 55
        4.3.4 Parameters Analysis ......................... 61
    4.4 Conclusions ....................................... 62

Chapter 5 SPDAQ: Similarity Preserving Deep Asymmetric Quantization for Multimedia Retrieval ........................................ 63
    5.1 Motivation ........................................ 63
    5.2 Methodology ...................................... 64
        5.2.1 Notation .................................... 65
List of Tables

3.1 The learning algorithm pipeline for APHash .................................. 32
3.2 Mean Average Precision of Hamming Ranking for different numbers of bits on two datasets. .................................................. 35
3.3 Mean Average Precision (mAP) on different size of anchor set |C| on CIFAR-10. The best mAP is in bold while the second best one is underlined. ................................................................. 39
3.4 Mean Average Precision (mAP) on different size of anchor set |C| on Youtube Face. The best mAP is in bold while the second best one is underlined. ................................................................. 40
3.5 Mean Average Precision (mAP) obtained by using k-mean clustering. The best mAP is in bold while the second best one is underlined. . . 40
4.1 Mean Average Precision (%) for different numbers of bits on two datasets. The best MAPs are shown in bold. . . . . . . . . . . . 51
4.2 Mean Average Precision (%) (i.e. MAP@1000) for CIFAR-10-CNN dataset. The best MAPs are shown in bold. . . . . . . . . . . . . 56
4.3 We compare DeepQuan with two variant methods PQ and DAE-PQ to show the encoder can learn more discriminative compact features. Results are compared with MAP@All under relatively short bit setting. 56
4.4 We compare DeepQuan with two variants Quad and WTPs to show effectiveness of the proposed weighted triplet loss and inclusion of the reconstruction term. MAP@All and MAP@1000 are computed for MNIST and CIFAR-10-CNN respectively. . . . . . . . . . . . . . . . 56
4.5 Mean Average Precision (mAP) obtained by different combination of $M$ and $K$ on CIFAR-10-CNN. The best mAP is in bold while the second best one is underlined. ................................. 60

4.6 Mean Average Precision (mAP) obtained by different combination of $M$ and $K$ on MNIST. The best mAP is in bold while the second best one is underlined. ................................. 60

5.1 Configuration of CNN-F deep convolution network. ...................... 69

5.2 Image examples of datasets. The third column contains the associated semantic labels. ................................................................. 79

5.3 Mean Average Precision (%) for different numbers of bits on CIFAR-10 and NUS-WIDE-21 datasets. The best mAPs are shown in bold. DVSQ* denotes we run the codes provided by the authors to get the results. ................................................................. 81

5.4 Mean Average Precision (%) on NUS-WIDE-81 and MS-COCO datasets for different deep models. The best results are in bold. .......... 90

5.5 Mean Average Precision (%) for variants of the proposed SPDAQ model. The best results are in bold, while the second best ones are underlined. ................................................................. 91

5.6 video examples (consecutive frames) of datasets ......................... 92

5.7 Mean Average Precision (%) for different numbers of bits on UCF-101 and HMDB-51 datasets. The best mAPs are shown in bold. .... 95
List of Figures

2.1 The illustration framework of DPSH [59]. ........................................ 16

3.1 The framework of the proposed Anchor-based Probability Hashing (APHash) method. Distances are transformed into probabilities and two asymmetric probability matrices $P$ and $Q$ are constructed for original space and hash code space respectively. In the first step, hash codes are learned by minimizing KL-divergence between $P$ and $Q$. Then, hash functions are learned with the guidance of learned hash codes. ............................................................. 25

3.2 Example images of dataset CIFAR-10. ................................. 33

3.3 Performance evaluated with precision-recall curve (@8 bits and @64 bits respectively) and precision curve w.r.t. top returned samples (@8 bits) on CIFAR-10 Dataset. ......................................................... 36

3.4 Performance evaluated with precision-recall curve (@8 bits and @64 bits respectively) and precision curve w.r.t. top returned samples (@8 bits) on Youtube Faces Dataset. ......................................................... 38

3.5 The effect of choosing different numbers of nearest neighbors. A PHash achieves better performance on both datasets at around $k = 1000$. . . 39

4.1 The framework of the proposed DeepQuan model, which is built upon a deep autoencoder. All the layers are fully connected. A weighted triplet loss and a manifold preserving loss are employed to learn discriminative compact representations for better quantization. ........... 43
4.2 Computing the binary codes on MNIST dataset @ 32bit. (a) Average distance between codewords which is computed using Eq. (4.2.1) and Eq. (4.2.3) keeps decreasing, resulting in trivial solutions (i.e. all the codewords become the same) and thus poor generalization. It is obvious that Eq. (4.2.4) ends up with a much larger average distance, which leads to more discriminative structure. (b) Loss value decreases until convergence as the training proceeds. 46

4.3 Example images of dataset MINST. 54

4.4 Visualization of the first four subspaces learned with Quad-R on MNIST @32 bits, which is depicted with t-SNE. Best view in color. 57

4.5 Visualization of the first four subspaces learned with DeepQuan on MNIST @32 bits, which is depicted with t-SNE. Best view in color. 58

4.6 Parameters sensitivity analysis. 61

5.1 The framework of the proposed SPDAQ model which make use of a CNN network and composite quantization to learn unquantized and quantized embeddings for training set images and database images respectively. 66

5.2 Retrieval performance evaluated with Top-K precision curve (@8 bits and @24 bits respectively) on CIFAR-10 datasets. Best view in color. 82

5.3 Retrieval performance evaluated with Top-K precision curve (@8 bits and @24 bits respectively) on NUS-WIDE-21 datasets. Best view in color. 83

5.4 Training time evaluation of SPDAQ and other state-of-the-art deep models on large-scale dataset NUS-WIDE-21. Best view in color. 88

5.5 The mAP performance and training efficiency on different size of training set. X-axis is the number of images per class. Best view in color. 89
5.6 Training time comparison between SPDAQ and the state-of-the-art deep quantization model DTQ on video dataset UCF-101. Best view in color. ................................. 92
Chapter 1

Introduction

1.1 Problem Definition

The sheer volume of high dimensional multimedia data like images and videos in online media repositories and social networks makes the precise and yet efficient retrieval a challenging research problem which has attracted a lot of attention from machine learning and computer vision communities [105, 110, 104]. To develop systems for large-scale image and video retrieval, there is no need to return precisely the nearest neighbor where a high computational cost is often incurred. Very often, a system that can retrieve similar items which are approximately ranked based on the similarity to the query for the user to choose from is already good enough. Therefore, Approximated Nearest Neighbor (ANN) [4] approaches are often used, which can take both two often conflicting objectives of precision and efficiency into consideration.

Among the existing ANN methods, one of the most popular models is \textit{hashing}, which has been widely adopted due to its advantages of low storage cost and computational efficiency. In the following, we briefly discuss how hashing achieves these two advantages.
Hash Function and Hash Code  The key idea of hashing model is to learn a hash function:

\[ f : x \rightarrow \{0, 1\}^k, x \in \mathbb{R}^d, d \gg k \quad \text{or} \quad f : x \rightarrow \{+1, -1\}^k, x \in \mathbb{R}^d, d \gg k, \]

which maps an item \( x \) from a high-dimensional real-value space into a low-dimensional binary space. The obtained short binary code is called hash code. One key property of the hash codes is to preserve as far as possible the underlying similarity of the data items (e.g. either obtained from supervised information or defined in the data distribution) such that similar data items will be indexed with similar binary codes.

Towards the objective of low storage cost, the high-dimensional real-value feature vectors of the data items are stored as short binary codes as indexes for efficient retrieval with enormous computational savings [54].

Metric for Retrieval  The metric used for retrieval in the binary space is one critical component of hashing model, which should be computed in a fast way for efficient retrieval. Hamming distance and dictionary-related distance are two commonly used metrics. Different metrics are usually used in the retrieval process based on hash codes. Their definitions are given as follows:

**Definition:** Hamming distance \( d(b_1, b_2) \) is defined as the number of different bits between two binary vectors \( b_1 \) and \( b_2 \).

For example, the distance between two binary vectors \( b_1 = 01010101 \) and \( b_1 = 11110000 \) is 4 since the number of different bit is 4. For the ease of subsequent discussion of dictionary-related distance, we first assume that the binary vectors are one-hot.

**Definition:** Suppose two binary vectors \( b_1 \) and \( b_2 \) are one-hot indicator vectors, each with only one bit equal to 1 and others 0s. Also, assume that there exists one commonly-shared dictionary \( C \) in the form a matrix associated with these two binary
vectors. The dictionary-related distance is defined as \( d(b_1, b_2) = \|Cb_1 - Cb_2\|_2 \).

One can use the operation on bits such as XOR operation to compute Hamming distance efficiently. With Hamming distance, either performing efficient linear scans of the binary data to find the nearest neighbors or using data structures for finding approximate nearest neighbors even in a database of 100 million objects can currently be performed within a few seconds on a typical workstation as discussed in [10, 38, 54].

When using the dictionary-related distance as metric in retrieval, one way to accelerate the computation is by pre-computing a distance matrix of codewords. Suppose that the codebook \( C \in R^{K \times D} \) where \( K \) is the size of a dictionary and \( D \) is the size of each codeword in a dictionary. A pre-computed distance matrix is of size \( K \times K \) where each element in the matrix is the Euclidean distance between two codewords. Finally, the dictionary-related distance can be directly obtained by indexing the elements in the pre-computed distance matrix with the binary hash codes, e.g. \( b_1 \) and \( b_2 \), which are in fact one-hot indicator vectors. Such computation process is just slightly more costly than that of Hamming distance [42, 8]. If there are multiple (e.g. \( L \)) commonly-shared codebooks, \( L \) distance matrices will be pre-computed. Note that during the retrieval phrase, all these distance matrices only need to be pre-computed once.

In fact, one can further encode the \( K \)-dimensional one-hot binary code with \( \log_2 K \) bits of 0 and 1. Thus, we finally refer the encoded \( \log_2 K \)-bit binary codes as hash codes when discussing the dictionary-related distance.

In summary, with the used of the Hamming distance and dictionary-related distance as metrics for retrieval, one can achieve the objective of efficient retrieval.

Note that for the convenience of the discussion in sequel, we follow [110, 64] and refer those hashing models which make use of the dictionary-related distance as metric for retrieval as **quantization** models, while we still refer those using the
Hamming distance as *hashing* models.

When using hashing for retrieval, we are required to design a hash function or learn one from the given training data such that given a query item during the testing stage, similar items (either defined with supervised information or from data distribution) are retrieved from the database and should be ranked higher as compared with the dissimilar ones.

### 1.2 Thesis Outline

In this thesis, three methodologies are developed to learn hash functions and the hash codes effectively and efficiently in supervised and unsupervised settings. In particular, the thesis is organized as follows:

Chapter 2 provides a literature review on nearest neighbor search method and learning hash codes for multimedia retrieval.

In Chapter 3, an unsupervised image hashing method called *APHash* is proposed which learns hash codes by exploiting the distribution of the images in a dataset. In particular, *APHash* first randomly samples a small subset of images as anchors and learns the hash function based on this anchor set. The correlations in the original space between anchors and database items are then used to construct an asymmetric similarity matrix which can subsequently be formulated as probabilities. Also, the distance between the corresponding hash codes measured in the binary space can also be formulated as probabilities accordingly. *APHash* learns the hash codes and hash functions by minimizing the mismatch between the probability distributions of the data in the original space and the binary space.

In Chapter 4, a novel deep learning model named *DeepQuan* is proposed for image hashing in an unsupervised setting. Most of the existing deep unsupervised hashing methods make use of a quadratic constraint for minimizing the difference between the compact representations and the target binary codes, which inevitably
causes severe information loss. Instead, DeepQuan utilizes quantization methods to compute hash codes. It adopts first a deep autoencoder network, where the encoder is used to learn compact representations and the decoder is for manifold preservation. Product quantization is then adopted where the binary codes of the images are learned by minimizing the quantization error. To the best of our knowledge, it is the first work that incorporates quantization methods into an unsupervised deep learning framework for hash learning in an end-to-end manner. Furthermore, a weighted triplet loss is proposed to drive better quantization to be achieved. Our experimental results show that DeepQuan model outperforms the state-of-the-art unsupervised hashing methods for image retrieval tasks.

In Chapter 5, we further investigate the integration of quantization methods into the deep learning framework to learn hash codes but in a supervised setting where the multimedia items are labelled. Training such deep models given a large-scale database is challenging as a large amount of parameters are typically involved and the training process is highly time-consuming. So, most of the existing deep quantization methods often sample only a subset from the database for training, which may end up with unsatisfactory retrieval accuracy as a large portion of label information is discarded. We propose a novel deep model called Similarity Preserving Deep Asymmetric Quantization (SPDAQ) which can directly learn the compact binary codes and quantization codebooks for all the multimedia items in the database efficiently. To do that, SPDAQ makes use of a subset as well as the label information of all the database items so the subset items and the database items are mapped to two different but correlated distributions, where the label similarity can be well preserved. An efficient optimization algorithm is then proposed for the learning. Extensive experiments conducted on six widely-used benchmark datasets including four image datasets and two video datasets demonstrate the superiority of the proposed SPDAQ model.

In Chapter 6, we conclude this thesis research project by listing out the contribution. In addition, possible future research directions regarding hashing for
multimedia retrieval are discussed.
Chapter 2

Related Work

In this chapter, a literature review on recent development of nearest neighbor search method, as a closely related topic to hashing, will be presented first. After that, a literature review of hashing for large-scale multimedia retrieval will then be presented. The review for hashing is organized in accordance to two different hashing paradigms, namely data-independent hashing and data-dependent hashing. The latter is also called learning-to-hash which is the main focus of this thesis research.

2.1 Nearest Neighbor Search

There are two types of Nearest Neighbor Search (NNS) [2]: Exact Nearest Neighbor Search (Exact NNS) and Approximate Nearest Neighbor Search (Approximate NNS). The Exact NNS is defined as searching the exact nearest item $x$ from a given set $S$ for a given query $q$, such that $x = \arg \min_{x \in S} d(x, q)$ where $d(x, q)$ computes the distance between two items. For a massive dataset with high-dimensional data items, the computational cost is high, which hinder its applicability for practical use. Fortunately, exact nearest neighbor usually is not required in a retrieval task, in which case the approximate NNS can be used. In the approximate NNS, only the items within the distance at most $c$ times the distance between query $q$ and the closest item $p$ are expected. One example may be the $k$-nearest neighbors search which returns the $k$ nearest neighbors to query $q$. 
One of the critical problems in nearest neighbor search is the metric used to compute distance between each pair of items. The Euclidean distance \( d(x, q) = \|x - q\|_2 \) is widely studied. Other choices exist, such as cosine similarity, Hamming distance and so on.

To achieve efficient approximate nearest neighbor search, some indexing data structures have been proposed. The most famous one could be \( k \)-d tree [24] which is a binary tree constructed by recursive division with \( \log n \) depth. However, it has been turned out that the efficiency of \( k \)-d tree can only be observed in a low-dimensional space. When in a large-scale high-dimensional space, it takes higher computational cost even than that linear search. Quad-Tree [101] and Oct-Tree [41] are indexing structure similar to \( k \)-d tree. Another important tree structure should be R-tree [31] which can handle multidimensional data. And, it has been applied in some database management systems such as Oracle and MySQL.

Recently, some efforts have also been made to improve the searching of approximate nearest neighbors. [78, 11, 79, 4] proposed some error-constrained nearest neighbor search schema which for example, relaxes by \( \epsilon \)-error and achieves \((1 + \epsilon)\)-approximate nearest neighbor search. Time-constrained approximate nearest neighbor search trade off the query time spent and accuracy. For example, it terminates the search process after examining a fixed number of data items.

### 2.2 Data-independent Hashing

Besides the algorithms discussed above, hashing could be one of the approximate nearest neighbor search methods that receives much attention in recent years. Data-independent hashing methods like Locality-Sensitive Hashing (LSH) [38, 25] make use of hash functions which are pre-designed without considering the distribution of the input data. For example, in [38], a set of hash function family \( \mathcal{H} = \{h\} \) is adopted, given as

\[
h((x_1, x_2, \ldots, x_d)) = (a_1x_1 + \ldots + a_dx_d \mod P) \mod M
\]
where $P$ is a prime, $M$ is the hash table size and $a_1, \ldots, a_d \in \mathbb{Z}_P$. It has been proved that such a pre-designed hash function has the $(r_1, r_2, p_1, p_2)$-sensitive property[38]. The resulting hashing indexes can then be implemented using a special data structure called Ring-Cover Tree [38] for efficient approximate nearest neighbor search. In [10], a simple hash function based on the rounding technique was proposed as

$$h_r(x) = 1 \text{ if } r^T x \geq 0; \ 0 \text{ otherwise}$$

where $r$ is a hyperplane randomly generated from zero-mean multi-variate Gaussian $\mathcal{N}(0, I)$. In [55], a kernelized extension for LSH (KLSH) was proposed so that the metric can be measured in a non-linear space.

The key advantage of data-independent hashing is that it can simply be applied to the multimedia data regardless of the input data, and thus typically can be computed in a much faster way compared with the learning-to-hash approach. Despite these advantages, their retrieval performance in terms of accuracy is usually poor.

### 2.3 Data-dependent Hashing

Previous research has already shown that the data-dependent hashing [23, 112, 46, 93, 36, 27, 70, 91, 65, 14, 35, 57, 44, 67, 40] often outperforms the data-independent hashing. Thus in this thesis, the key focus is on data-dependent hashing methods which learn hash functions by exploiting the data distribution or making use of the semantic relations between data items. Data-dependent hashing can be further divided into *supervised* hashing and *unsupervised* hashing based on whether the supervision information is provided during the learning process. In addition, most of the state-of-the-art data-dependent hashing methods are based on deep learning. So before introducing supervised hashing methods, a brief review on deep learning is first provided.
2.3.1 Deep Learning

The recent years have witnessed the great achievement of deep learning in computer vision and machine learning communities. Some general models such as Variational Autocoder (VAE) [48], Generative Adversarial Network (GAN) [29], Graph Convolutional Network (GCN) [82], Convolutional Neural Network (CNN) [52] have been proposed and applied to a broad range of tasks. Since this thesis specifically focuses on image and video retrieval tasks, we describe only Convolutional Neural Network (CNN) which is often used in computer vision tasks especially related to images and videos.

Convolution Neural Network (CNN) has been proved to be an effective tool for many computer vision tasks such as object recognition [52, 87, 33, 96, 26, 88] and action recognition [95, 47], due to it effectiveness in directly learning semantic visual representation from raw-pixel images [111]. Therefore, some networks specifically designed for large-scale visual recognition like AlexNet [52], GoogLeNet [100], VGGNet [111], ResNet [33] are widely adopted as the backbone for a number of computer vision tasks such as image segmentation [15, 17]. These proposed deep networks all share similar components such as convolutional layer, fully-connected layer, drop-out layer [99], normalization layer [39] but vary in the ways of connection and the size of the convolutional kernel.

2.3.2 Supervised Hashing

Given the labels of the multimedia data (i.e., the supervision information) are available, supervised hashing methods learn hash codes and hash functions which are customized for the input data, and thus usually give better performance in retrieval tasks as compared with unsupervised hashing methods.

Different Forms of Supervision Information Supervised hashing utilizes semantic labels to mitigate the gap between the low-level raw-pixel images (or their features) and the high-level semantic hash codes. These semantic labels can be of dif-
different forms, including point-wise labels, pairwise labels and rank-based labels [59].

Given the semantic labels, corresponding training methods can be developed to learn the hash codes and the hash functions.

Point-wise labels mainly indicate the class information of items which is often represented by a one-hot vector. For example, Supervised Discrete Hashing (SDH) [91] learns hash codes with point-wise labels by making the hashing learning a classification problem. Pairwise labels usually indicate the similarity or dissimilarity between items, usually with the value of 1 to indicate that two items are similar and with 0 (or −1) to indicate that they are dissimilar. Many existing hashing methods are pairwise-label based models such as Kernelized Supervised Hashing (KSH) [70], Column Sampling Based Discrete Supervised Hashing (COSDISH) [46], Latent Factor Hashing (LFH) [112] and so on. Given pairwise labels of 1 and 0, hash codes and hash functions can be learned by minimizing the distance in the binary space between similar items and maximizing the distance between dissimilar items. The underlying objective is to minimize the mismatch between the ground-truth similarity and the learned metric in the binary space. Different from pairwise labels and point-wise labels, rank-based labels indicate more fine-grained ranking similarity. Usually, they are provided in triple or quadruple form. For example in triple form, relations of three different items \((x_a, x_p, x_n)\) are provided to indicate that the anchor sample \(x_a\) is more similar to the positive sample \(x_p\) than the negative sample \(x_n\). Thus, the objective of methods like Deep Neural Network Hashing (DNNH) [56] is to preserve these ordinal information based on a corresponding triplet loss.

**Conventional Supervised Hashing** We categorize the methods which are not using a deep convolutional network as the backbone as conventional supervised hashing methods, where two kinds of labels, pairwise labels and point-wise labels are usually used as supervision due to their convenience of adoption in matrix form. Representative hashing models like Supervised Hashing with Kernel (KSH) [70], Column Sampling-based Hashing (COSDISH) [46], Latent Factor Hashing (LFH) [112], Two-Step Hashing (TSH) [61], and Fast Supervised Hashing (FastH) [60] use
pairwise labels, while Supervised Discrete Hashing (SDH) [91] uses point-wise labels. In the following, the two methods KSH and SDH will be briefly described.

The Supervised Hashing with Kernel (KSH) model is derived from Kernelized Locality-Sensitive Hashing (KLSH) [55]. A hash function \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) to compute one bit for an input \( x \) can be defined as \( f(x) = \sum_{j=1}^{m} \kappa(x_j, x) a_j - b \), where \( x_1, \cdots, x_m \) are a subset of \( m \) items that are randomly sampled from the training set and \( \kappa(x_j, x) \) is a predefined kernel function. Through simple arithmetic operations, the new representation of item \( x \) can be obtained as \( k(x) = \left[ \kappa(x_1, x) - \mu_1, \cdots, \kappa(x_m, x) - \mu_m \right] \), where \( \mu_j = \sum_{i=1}^{n} \kappa(x_j, x_i) / n \) is the mean of the kernel function. Using this simple step, the new feature representation of \( x \) is transformed from the \( d \)-dimension space to the new \( m \)-dimension space while simultaneously taking the data distribution into account. Suppose that \( r \)-bit hash codes are needed to be learned. To find \( r \) coefficient parameter vectors \( a_1, \cdots, a_r \) for the computations of \( r \) hash functions as \( \{h_s(x) = \text{sgn}(a_s^T k(x))\}_{s=1}^{r} \), KSH proposes to maximally approximate the pairwise relations under the supervision of pair-wise labels \( S_{ij} \in \{+1, -1\} \), which can be formulated in matrix form as:

\[
Q = \frac{1}{r} H H^T - S^2
\]

where \( H \in \{+1, -1\}^{n \times r} \) denotes the hash code matrix of the training dataset \( \mathcal{X} \) of \( n \) items. The relation between the Hamming distance and the inner product can be derived as \( H(b_i, b_j) = \frac{1}{2}(r - b_i^T b_j) \) where \( b \in \{+1, -1\}^r \). Therefore, minimizing the objective function can be interpreted as maximizing the Hamming distance for \( S_{ij} = -1 \) while minimizing the Hamming distance for \( S_{ij} = 1 \). By optimizing the above objective function, the coefficient parameters vectors \( \{a_i\}_{i=1}^{r} \) can be obtained.

In the paper of KSH [70], a sequential projection algorithm was proposed to optimize the model.

Supervised Discrete Hashing (SDH) is another hashing method that makes use of point-wise labels to learn hash codes. Methods such as KSH [70] and Spectral Hashing [106] could allow relaxation to the binary constraint (i.e. \( b \in \{+1, -1\}^r \)) during training, and then the binary codes can be obtained with a post binarization
process (i.e., $b = \text{sgn}(\hat{b})$). This may result in a sub-optimal solution with large quantization errors. To maintain the binary constraint during the training in order to obtain better solutions, SDH [91] tries to learn the hash codes with optimization in a discrete way. Given that the point-wise labels $\{y_i\}_{i=1}^n$ are represented as a one-hot vector which indicates the class that item $i$ belongs to, the objective function is given as:

$$\sum_{i=1}^{n} ||y_i - W^T b_i||^2 + \lambda ||W||^2 + \nu \sum_{i=1}^{n} ||b_i - F(x_i)||^2$$

subject to $b_i \in \{+1, -1\}^r$

where $F(x)$ denotes a linear projection function from item $x$ to binary codes $b$ and $W$ is a parameter matrix. Note that the first term $||y_i - W^T b_i||^2$ is a classification loss so that the learned binary codes will preserve the class information by learning a discriminative projection to a specific class. The above objective function can be further rewritten in a matrix form

$$||Y - W^T B||^2 + \lambda ||W|| + \nu ||B - F(X)||^2$$

subject to $B \in \{-1, +1\}^{r \times n}$

An iterative process among $W, B, F(X)$ is proposed to optimize the objective function. During the optimization with respect to $B$, a discrete cyclic coordinate descent method is applied to maintain the binary constraint. More specifically, it sequentially optimizes one selected column $\tilde{B}_i$ in $B$ with relaxation into a real-value space at one time while fixing all other columns with the binary constraint. After $\tilde{B}_i$ is computed, a rounding function $\text{sgn}()$ is simply applied to get the binary codes $B_i = \text{sgn}(\tilde{B}_i)$.

**Supervised Deep Hashing and Quantization** Conventional supervised hashing methods [27, 70, 91, 65, 14, 35, 57, 44, 67, 40] learn the hashing function based on some pre-defined visual features, e.g., SIFT [74] and GIST [84] for images and IDT [102] for videos. It is obvious that the representations adopted, though effective to some extent, are by no means optimal. Recently, many deep learning based hashing models have been proposed for image retrieval, where compatible visual features
and binary hash codes can be jointly learned from raw pixel images under a unified deep learning framework. The extracted features from the raw pixel images are then more favorable for the hash code learning task, which has been shown to achieve better performance compared to conventional methods. DNNH [56] was the first model proposed to simultaneously learn hash codes and image representations based on a deep convolution network in an end-to-end training manner to make the hash codes and the representations more compatible. While DNNH trains the deep CNN with a triplet loss, Deep Pairwise Supervised Hashing (DPSH) [59] and Deep Hashing Network (DHN) [114] are two deep models which jointly learn visual features and hash codes by approximating pair-wise similarity. Note that many existing methods would relax the binary constraint on hash codes during the optimization. Instead, Deep Supervised Discrete Hashing (DSDH) [58] and Deep Discrete Supervised Hashing (DDSH) [43] try to preserve the discrete constraints to make the gradient descent based optimization tractable. DSH-GANs [86] learns to hash with a generative adversarial network [29], and Deep Asymmetric Pairwise Hashing [90] proposes to learn two different hashing functions with two deep networks.

While directly mapping features into the binary space may lose much fine-grained information, quantization models can alleviate this limitation by computing codebooks and then the binary codes for more accurate ranking, which requires slightly higher storage and computational cost. Quantization models were originally designed for unsupervised settings. To make the thesis more self-contained, three quantization models that are used in our proposed models will first be introduced here.

One fundamental quantization model is vector quantization (VQ) [30], which was proposed to quantize the feature space by maintaining one codebook. Formally, given a set of items \( S = \{x_i\}_{i=1}^n \) in the feature space, it is required to learn a codebook \( C \) of \( k \) codewords by minimizing the quantization error \( E \) defined as

\[
E = \sum_{i=1}^n \| x_i - Cb_i \| \quad \text{subject to } b_i \in \{0, 1\}^k \text{ and } \|b_i\|_0 = 1
\]

where \( b \) is one-hot vector. It is an NP-hard problem and usually solved by \( k- \)
mean clustering. However, the one-hot constraint on \( b \) limits the capacity of VQ model where at most \( k \) codewords can be obtained. To alleviate the model capacity problem, Product quantization (PQ) [42] and composite quantization (CQ) [113] construct several codebooks to quantize the feature space to allow an exponential number of code words to be supported, which generally leads to a much smaller quantization error. Concretely, PQ first decomposes a given feature vector \( x_i \) into \( M \) subvectors such that \( x_i = \text{concat}(x_{i1}, x_{i2}, \ldots, x_{iM}) \), where \( \text{concat}() \) is the concatenation operation. Subvector \( x_{im} \) will be related to the corresponding \( m^{th} \) subspace which is quantized by a codebook \( C_m \) consisting of \( k \) codewords. As in VQ, PQ learns \( M \) codebooks \( \{C_m\}_{m=1}^{M} \) by minimizing the quantization error

\[
\min_{C, b} \quad E = \sum_{i=1}^{n} \sum_{m=1}^{M} \|x_{im} - C_{m}b_{im}\|
\]

subject to \( b_{im} \in \{0, 1\}^k, \|b_{im}\|_0 = 1 \)

\( x_i = \text{concat}(x_{i1}, x_{i2}, \ldots, x_{iM}) \)

By different combination of \( k \) codewords from \( M \) codebooks, PQ can finally achieve exponential number \( k^M \) codewords for quantizing the original space so that more fine-grained information can be preserved. Instead of using concatenation for approximation, CQ approximates the original vector by summing up the codewords selected. Given a feature vector \( x_i \), the objective is formulated as

\[
\min_{C, b} \quad E = \sum_{i=1}^{n} \|x_i - \sum_{m=1}^{M} C_{m}b_{im}\|
\]

subject to \( b_{im} \in \{0, 1\}^k, \|b_{im}\|_0 = 1 \)

Same as PQ, CQ has totally exponential number \( k^M \) codewords by different combination of \( k \) codewords from \( M \) codebooks. Furthermore, Zhang et al. [113] empirically showed that CQ outperforms PQ.

Recently, many deep quantization models designed for image retrieval have been proposed, which incorporate quantization models into an end-to-end deep learning model so that both can be learned simultaneously. The first incorporation quantization into an unsupervised deep quantization model was proposed in [13]. To utilize the supervision information, Deep Quantization Network (DQN) [8] and Product Quantization Network (PQN) [110] were proposed to jointly learn product quanti-
zation codebooks and visual features in a supervised deep learning framework. Deep Triplet Quantization (DTQ) [64] adopts composite quantization and learns visual features by sampling triplets as training samples. Deep Visual Semantic Quantization (DVSQ) [7] makes use of the auxiliary text embeddings and labels to learn quantization codebooks in order to utilize the semantic information.

To illustrate how a deep learning based hashing model is formulated, we here briefly describe Deep Pairwise Supervised Hashing (DPSH). As shown in Figure 2.1, Deep Pairwise-Supervised Hashing (DPSH)[59] was proposed as an end-to-end framework containing three parts: (1) a deep CNN network for learning representations with raw-pixel images as input, (2) one hashing layer to learn binary codes by projecting high-dimensional features into a low-dimensional binary space, and (3) one pairwise loss function for training the whole deep network. The objective function is to maximize the distance between dissimilar pairs and at the same time minimize the distance between similar pairs, given as

\[
- \sum_{s_{ij} \in S} (s_{ij}\theta_{ij} - \log(1 + e^{\theta_{ij}})) + \eta \sum_{i=1}^{n} \|b_i - u_i\|_2^2
\]

subject to \( b_i \in \{+1, -1\}^r \), \( s_{ij} \in \{1, 0\} \)

where \( u_i \) denotes the learned real-value low-dimensional embedding obtained from a raw-pixel input image \( x_i \) through a deep convolutional network and \( \theta_{ij} = u_i^T u_i \).
which is forced to be in the binary space by minimizing the last quantization error term $\|b_i - u_i\|_2^2$. The whole model is fine-tuned in an end-to-end way by optimizing the proposed objective function with the standard gradient back propagation (BP) using the chain rule.

### 2.3.3 Unsupervised Hashing

Although supervised hashing can always achieve high accuracy, the performance of supervised hashing can be hindered by the availability of supervision information. This is especially challenging when deep models are considered because a large number of labels are typically needed for the training, which in fact is expensive and labor-intensive. Nowadays as the amount of data available online grows rapidly, it is much easier to obtain a large amount of unlabeled data than ever before. Thus, in contrast to supervised hashing, it is more applicable and significant to investigate unsupervised hashing for real-world applications. In the following, a literature review on two types of unsupervised hashing: *conventional* unsupervised hashing and *deep-learning* based unsupervised hashing are presented.

**Conventional Unsupervised Hashing**  Conventionally unsupervised hashing generally builds upon the extracted feature representations of items, where an assumption is made that data are distributed along a manifold and semantic similar items lie closely. In the following, I would like to roughly divide existing conventional unsupervised hashing methods into three categories including *Graph-based Hashing*, *Manifold-learning-based Hashing* and *Reconstruction-based Hashing*, which are usually seen in the literature.

**Graph-based hashing** is one of the most popular types of unsupervised methods to learn hash codes and hash functions, where a graph describing the underlying data distribution is first constructed. Each node in the graph represents a data item. The hash code for each node is obtained by approximately recovering the relations among items, which is generally done by solving a Laplacian Eigen-decomposition
problem. The whole pipeline is similar to another task called *Graph embedding* [6] which goes beyond the scope of this thesis.

To describe graph-based hashing method, Spectral Hashing (SH) [106] is presented as follows. Suppose that $y_i$ is a hash code to be obtained for item $i$. The Spectral Hashing is formulated as

$$
\sum_{ij} W_{ij} \|y_i - y_j\|^2
$$

subject to $y_i \in \{+1, -1\}^k$

$$
\sum_i y_i = 0
$$

$$
\frac{1}{n} \sum_i y_i y_i^T = I
$$

where the constraint $\sum_i y_i = 0$ is to balance the bit distribution and $\frac{1}{n} \sum_i y_i y_i^T = I$ encourages the bits to be uncorrelated. The objective above corresponds to the balanced graph partition problem which has been proved to be NP-hard. To make the problem easier to solve, spectral relaxation is applied and the binary constraint to hash codes $y_i \in \{+1, -1\}^k$ is dropped so that it could be done through solving an eigen-decomposition problem. The relaxed objective function is rewritten in a matrix form as

$$
Tr(Y^TLY)
$$

subject to $Y^T1 = 0$

$$
Y^TY = I
$$

where $L$ is a Laplacian matrix $L = D - W$ and $D$ is a diagonal $n \times n$ matrix $D(i, i) = \sum_j W(i, j)$. The solutions of $Y$ are $k$ eigenvectors related to first $k$ smallest eigenvalues excluding 0 which is associated with vector $1$. Then the hash codes are obtained using simple rounding (i.e. using a sign function $\text{sgn}(y_i)$).

However, the space complexity of $W$ is $O(n^2)$ which grows quadratically as the number of training items increases. The construction of the similarity matrix $W$ is an obstacle of directly applying Spectral Hashing to large-scale datasets. One way to address the problem is to adopt low-rank approximation with the Nystrom method.
Along this direction, Anchor Graph Hashing (AGH) [71] was proposed to approximate the similarity matrix $W$ by selecting a small set of $m$ items named anchors from the whole training set $X$. A similarity matrix $Z$ between the anchor set and the item $x$ from the whole training set can be computed to preserve the neighborhood structure. As in [68], the approximation of the similarity matrix $W$ can be obtained by $\tilde{W} = Z\Lambda^{-1/2}Z^T$ where $\Lambda = \text{diag}(Z^T 1)$ is the normalization factor. This approximation can be interpreted as one step of Markov transition. Instead of solving the eigen-decomposition of $\tilde{W}$ with a larger size of $n \times n$, it proposes to solve the eigenvalue system of $M = \Lambda^{-1/2}Z^T Z \Lambda^{-1/2}$ with a smaller size of $m \times m$. Through simple algebraic operations, one can obtain the solution as

$$Y = \sqrt{n}ZA^{-1/2}V\Sigma^{-1/2}$$

where the matrix $V$ and $\Sigma$ are the eigenvector matrix and eigenvalue matrix of $M$ respectively.

For the two aforementioned methods, the rounding step is applied as a post processing step which may cause large quantization errors, and thus suboptimal solutions. Discrete Graph Hashing (DGH) [69] proposed to address this problem by maintaining the binary constraint while learning hash codes $B$ through introducing an auxiliary real-value parameter matrix $Y$. Formally, the objective function is modified as

$$\text{Tr}(B^T WB) - \frac{\lambda}{2}\|B - Y\|_F^2$$

subject to $B \in \{+1, -1\}^{r \times n}, Y \in \mathbb{R}^{n \times r}, Y^T 1 = 0, YY^T = nI_r$.

By setting the weight parameter $\lambda$ to a large value, the real value matrix $Y$ will be forced to be closed to a binary value so that the algorithm will result in a nearly balanced and uncorrelated hash codes matrix $B$. An iterative process is used to minimize the objective function. To learn the hash codes matrix $B$, a gradient descent method is applied. For the optimization with respect to $Y$, an analytical solution can be derived.

Other graph-based hashing methods such as Scalable Graph Hashing (SGH) [44] and Graph PCA Hashing [115] follow almost the same concepts as above.
Manifold learning based hashing makes an assumption that the underlying distribution of data is along a manifold \( \mathcal{M} \) with the intrinsic \( d \) dimension embedded in the original \( D \) dimensional space, where \( d \ll D \). By the definition of manifold, \( \mathcal{M} \) is locally Euclidean, i.e. the neighborhood of each item is homeomorphic to \( \mathbb{R}^d \).

Based on the above observation, Locally Linear Hashing (LLH)[40] proposes that each item in the manifold is a linear combination of its locally near neighbors. The first step of LLH is to discover the locally linear structure of data. That is to find out the linear combination of locally near neighbors for each data item \( x_i \), formulated as

\[
\lambda \|s_i^T w_i\|_1 + \frac{1}{2} \|x_i - \sum_{j \in N_E(x_i)} w_{ij} x_j\|^2
\]

subject to \( w_i^T 1 = 1 \)

where \( w_i \) is the coefficient parameter of of linear combination of neighbor items of item \( x_i \). \( s_i \) is a predefined parameter vector and the \( L_1 \) norm enforces the sparsity of \( w_i \) to ensure that the locally ambient items of \( x_i \) are selected from the neighboring items \( N_E(x_i) \) measured by the Euclidean distance. Hash codes are then learned by reconstructing the coefficients matrix \( W \) in the Hamming space as

\[
\sum_i \|y_i - \sum_j w_{ij} y_j\|^2 = Tr(Y^T M Y)
\]

subject to \( y_i \in \{+1, -1\}^r \)

where \( M = (I_n - W)^T (I_n - W) \) is a sparse matrix. By minimizing the objective function, the learned hash codes \( Y \) are supposed to preserve the locally linear structure optimally.

Another example of manifold-learning-based hashing is called ordinal preserving hashing which tries to learn hash codes by exploiting the known ordinal relationships of items in a manifold. Examples include Ordinal Embedding Hashing (OEH) [67] and Ordinal Constraint Hashing (OCH)[65].

Reconstruction-based hashing methods refer to those where the hashing problem is formulated with the objective so that the original input can be con-
One typical reconstruction-based hashing method is to utilize the autoencoder as the basis model, such as Hashing with Binary Autoencoder [9] which uses an autoencoder for dimension reduction while at the same time reconstructing the original input. An auxiliary variable is introduced to keep the binary constraint during optimization leading to better solutions. Other methods like Hashing with Angular Reconstructive Embedding [35] also fall into this category since it basically reconstructs the angular information. Iterative Quantization (ITQ) [28] makes use of Principle component analysis (PCA) which is one effective dimension reduction algorithm that can maximally preserve the information for data reconstruction. PCA assumes that the noise is modeled by a Gaussian distribution so that it can be solved with Singular Value Decomposition (SVD). To obtain the hash codes from data, the simplest way is to first apply PCA reduction and then binarize the obtained low-dimensional embeddings. Instead, ITQ takes a further step by minimizing the quantization error with application of an orthogonal rotation matrix as follows

$$\|Y - R^TP\|^2_F$$

subject to $Y \in \{+1, -1\}^{r \times n}, RR^T = I$

where $P$ is the low-dimensional embedding matrix obtained by applying PCA to the training dataset $\mathcal{X}$. This problem is solved iteratively and an analytical solutions can be obtained. In this way, the data distribution obtained is not changed by the orthogonal rotation matrix but the quantization error is minimized as much as possible.

**Deep Unsupervised Hashing** Deep Hashing (DH) [21] is one of the first attempts which learns hash codes in an unsupervised way with a stack of fully connected layers to form multiple hierarchical nonlinear projections from input features to binary codes. Binary Deep Neural Network (UH-BDNN)[19] was developed based on reconstruction which simultaneously maintains the set of discrete constraints during training. Specifically, it introduces one auxiliary parameter vector $B$. The
objective function is formulated as

\[
\|X - W^{(m-1)}B\|_F^2 + \lambda_1 \sum_{l=1}^{m-1} \|W^{(l)}\|_F^2 + \lambda_2 \|H^{(m-1)} - B\|_F^2 \\
+ \lambda_3 R \left( \frac{1}{r} H^{(m-1)}(H^{(m-1)})^T - I \right) + \lambda_4 \|H^{(m-1)}1\|^2
\]

subject to \( B \in \{+1, -1\}^{r \times n} \)

where \( W^{(m-1)} \) denotes the parameter matrix of \((m-1)\)th fully connected layer of the deep neural network. The reconstruction is achieved at the last layer (i.e. \(m\)th layer) of the neural network.

The inputs of DH and UH-BDNN both are pre-computed features, while there are deep unsupervised methods incorporating feature learning part with raw-pixel images as input and hash codes learning part into one unified framework. For example, DeepBit [62] and Unsupervised Triplet Hashing (UTH) [37] learn hash functions with data argumentation by rotating the images. After that, the network is trained in a pair-wise or triplet way. Specifically, DeepBit formulates the objective function as

\[
\alpha \sum_{n=1}^{N} \| (b_n - 0.5) - \mathcal{F}(x_n; W) \|^2 + \beta \sum_{m=1}^{M} \| \mu_m - 0.5 \|^2 + \gamma \sum_{n=1}^{N} \sum_{\theta=-R}^{R} \| b_{n,\theta} - b_n \|^2
\]

where \( b_{n,\theta} \) denotes the binary codes learned from \(n\)th image rotated by \(\theta\) angle. \(\mathcal{F}(x_n)\) denotes the deep CNN applied to an input image \(x\) to return the low-dimensional embedding. DBD-MQ [20] obtains the hash codes by using a \(K\)-AutoEncoders network to minimize the reconstruction errors, which is achieved by minimizing the element-wise quantization error.

There is also recent interest on studying deep generative methods such as variational autoencoder (VAE) [48] and Generative Adversarial Network (GAN) [29]. Some methods based on those generative models have been already proposed for the hashing. For example, Progressive Generative Hashing (PGH) [76] uses GAN to generate vivid images with which to further improve the performance of the model in a supervised manner.
Chapter 3

APHash: Anchor-based Probability Hashing for Image Retrieval

3.1 Motivation

Unsupervised hashing generally learns hash codes by exploiting the underlying distribution of data items. Among the proposed unsupervised hashing methods proposed in the literature [40, 32, 28, 35, 71, 92, 106, 66, 44, 69, 67], graph-based hashing becomes one of the most popular ones. A graph to approximate the underlying distribution of data is firstly constructed. Then, learning the hash codes is related to solving the problem as Laplacian Eigen-decomposition with spectral relaxation [71] or in a discrete way [69]. Other methods incorporating quantization techniques [28, 32] and ordinal preservation [67, 66] have also been proposed to learn hash codes in an unsupervised way efficiently.

Instead of treating the unsupervised hashing as a Laplacian Eigenmap or ordinal preserving problem, a novel unsupervised method is proposed in this section to learn hash codes by exploiting the distribution of data items, referred as Anchor-based Probability Hashing (APHash). As shown in the proposed framework depicted in
Figure 3.1, distances between data items in the original space and the hash code space are respectively transformed into probability distributions $P$ and $Q$ that represent data similarities. If the derived hash codes correctly capture the similarities of the original data space, the mismatch between $P$ and $Q$ should be minimized. One of the direct ways to minimize the difference between two distributions is minimizing the KL-divergence between them as in t-SNE [77].

However, if we adopt symmetric strategy as in t-SNE and the other existing methods [70, 106, 71] and learn the hash function for both the query set and the database, assuming there are $n$ data items, $P$ and $Q$ will be represented by $n \times n$ probability matrices where the space complexity is $O(n^2)$. Thus, it is hard to directly employ this configuration for implementation, as the number of data items is typically large. To overcome this obstacle, we propose to first randomly select a set of $m$ ($m \ll n$) anchors and construct asymmetric probability matrices of size $m \times n$ to represent the probability distributions $P$ and $Q$. The space complexity now is reduced to $m \times n$ which is acceptable for implementation. As in [93, 81], we treat the query set and the database in an asymmetric way and learn the hash functions only with respect to the anchor set. We call the proposed method APHash. As our proposed APHash method can preserve the correlations between the anchors and the database items when learning the hash codes, we can obtain the hash codes for the queries with learned hash functions and directly use the hash codes of database items in the retrieval task. APHash adopts the widely-used two-step framework [61], which first learns hash codes and then hash functions. Extensive experiments are conducted on publicly available CIFAR-10 [51] and Youtube Faces [108] datasets and the results demonstrate that our proposed APHash method can achieve superior performance over the state-of-the-art hashing approaches in the image retrieval task.
Minimize the KL-divergence between $\mathcal{P}$ and $\mathcal{Q}$: Asymmetric probability matrix of the original space $\mathcal{P}$: Asymmetric probability matrix of the hash code space.

Hash Function $\mathcal{f}: \mathbb{R}^d \rightarrow \{0, 1\}^r$.

Figure 3.1: The framework of the proposed Anchor-based Probability Hashing (APHash) method. Distances are transformed into probabilities and two asymmetric probability matrices $\mathcal{P}$ and $\mathcal{Q}$ are constructed for original space and hash code space respectively. In the first step, hash codes are learned by minimizing KL-divergence between $\mathcal{P}$ and $\mathcal{Q}$. Then, hash functions are learned with the guidance of learned hash codes.
3.2 Methodology

3.2.1 Problem Formulation

**Distribution-preserving Loss** To capture the data structure in the original space, a novel method called Anchor-based Probability Hashing (APHash) is proposed in this section which aims to preserve the distribution of data items. Specifically, Euclidean distances between data items are transformed into probability distribution $\mathcal{P}$ which represent similarities in the original space. Similarly, Hamming distances in the hash code space are transformed into probability distribution $\mathcal{Q}$. The data distribution can be preserved by minimizing the mismatch between $\mathcal{P}$ and $\mathcal{Q}$. In our APHash method, KL-divergence is utilized to measure the difference between these two probability distributions $\mathcal{P}$ and $\mathcal{Q}$.

If we adopt symmetric strategy which treats query set and database equally, and learn identical hash function for both query set and database (i.e. the whole training set), $\mathcal{P}$ and $\mathcal{Q}$ will then be represented by probability matrices of $n \times n$ with given $n$ data items for training. However, there is a high complexity problem (space complexity equals to $\mathcal{O}(n^2)$) under this configuration, where $\mathcal{O}(n^2)$ grows quadratically with increasing of the number of data items. To address this problem, we first sample a set of $m$ items $C = \{c_i\}_{i=1}^m$ as anchor set from the database which contains totally $n$ data items $X = \{x_i\}_{i=1}^n$ and construct asymmetric probability matrices $\mathcal{P}$ and $\mathcal{Q}$ of size $m \times n$. Here we adopt strategy of randomly selecting a set of anchors for simplicity. Note that $m$ is much smaller than $n$ (i.e. $m \ll n$). In this way, APHash can make use of the correlations between anchors and all the data items from database to learn hash codes which preserve the data distribution. As in [93, 81], we adopt the asymmetric strategy and learn hash functions only for anchor set.

I first give a further explanation about the meaning of the word “asymmetric” used here:

- Note the similarity matrix for the original space $\mathcal{P}$ will then be normalized
row by row with respect to the selected anchor items, which obviously is no longer a symmetric similarity matrix.

- The selected anchor set and the given whole training set play different roles in the model. The hash codes learned for even the identity item in anchor set and given training set may not be the same. Also, APHash learns hash functions only with supervision of the hash codes of anchors set, while that of database items just serve as index for retrieval.

In the original space, to derive the probability distribution $P$, $p_{ji}$ is defined as the probability of assigning data item $x_j$ to anchor $c_i$, in other words, $p_{ji}$ indicates the similarity between data point $x_j$ and anchor $c_i$. $p_{ji}$ is represented as follows:

$$p_{ji} = \begin{cases} 1, & \text{if } d(c_i, x_j) \leq \theta \\ 0, & \text{if } d(c_i, x_j) > \theta \end{cases}$$ (3.2.1)

where $\theta$ is the threshold indicating the average distance between anchor $c_i$ and its $k$ nearest neighbors:

$$\theta = \frac{\sum_{j \in \mathcal{N}_k(c_i)} d(c_i, x_j)}{k}$$ (3.2.2)

where $d(c_i, x_j) = \|c_i - x_j\|_2^2$ denotes the Euclidean distance between anchor $c_i$ and database item $x_j$, and $\mathcal{N}_k(c_i)$ denotes the set of $k$ nearest neighbors of anchor $c_i$. To some extent, the threshold $\theta$ can help filter out those remote data points within these $k$ nearest neighbors. The probabilities is finally normalized such that the sum of probabilities with respect to anchor $i$ is 1, that is $\sum_{j=1}^n p_{ji} = 1$.

Similarly, a probability distribution $Q$ is defined in the hash code space using Hamming distances. Assume $r$-bit hash code matrices for the anchor set and the database are $H = \{h_i\}_{i=1}^m \in \{-1, +1\}^{r \times m}$ and $B = \{b_i\}_{i=1}^n \in \{-1, +1\}^{r \times n}$ respectively. $q_{ji}$ denotes the probability of of assigning data item $b_j$ to anchor $h_i$ in the hash code space. Inspired by t-SNE [77], a Student t-distribution is utilized with one degree of freedom to transform Hamming distances into probabilities, as shown in the following formula:

$$q_{ji} = \frac{(1 + g(h_i, b_j))^{-1}}{\sum_{t=1}^n (1 + g(h_i, b_t))^{-1}}$$ (3.2.3)
where \( g(h_i, b_j) \) denotes the Hamming distance between anchor \( h_i \) and database item \( b_j \) in the hash code space. Similar with the setting in the original space, APHash applies a normalization term to normalize the probabilities such that \( \sum_{j=1}^{n} q_{ji} = 1 \). With \( h_i \) and \( b_j \in \{+1, -1\}^r \) for any \( i \) and \( j \), the Hamming distance between hash codes of two instances can be calculated from their squared Euclidean distance as follows:

\[
g(h_i, b_j) = \frac{1}{4} \|h_i - b_j\|_2^2
\]  

(3.2.4)

thus \( q_{ji} \) can be further rewritten as follows:

\[
q_{ji} = \frac{(1 + \frac{1}{4} \|h_i - b_j\|_2^2)^{-1}}{\sum_{i=1}^{n}(1 + \frac{1}{4} \|h_i - b_i\|_2^2)^{-1}}
\]  

(3.2.5)

The objective of APHash is to learn optimal hash codes \( H \) and \( B \) of anchors and database items (separately) that can minimize the mismatch between probability distributions \( P \) and \( Q \). By making use of the correlations between anchors and database items, the distribution-preserving loss using KL-divergence to learn hash codes \( H \) and \( B \) is defined as follows:

\[
J_0 = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ji} \log \frac{p_{ji}}{q_{ji}}
\]  

(3.2.6)

where \( p_{ji} \) and \( q_{ji} \) are defined as Eq. (3.2.1) after normalization and E.q. (3.2.5) respectively. Minimizing the cost function \( J_0 \) means to make the distributions \( P \) and \( Q \) as consistent as possible, so that the data distribution in the training set and the correlations between anchors and database items can be well preserved in the learned hash codes \( H \) and \( B \).

However, with the binary constraint on hash codes \( H \) and \( B \), the problem becomes a combinatorial problem and hard to solve. To make the problem tractable, we follow the previous work \[106, 70\] and relax the binary constrained \( H \) and \( B \) to be real-valued, which are denoted as \( \hat{H} \) and \( \hat{B} \). Then, the formulation of \( q_{ji} \) will become:

\[
q_{ji} = \frac{(1 + \frac{1}{4} \|\hat{h}_i - \hat{b}_j\|_2^2)^{-1}}{\sum_{i=1}^{n}(1 + \frac{1}{4} \|\hat{h}_i - \hat{b}_i\|_2^2)^{-1}}
\]  

(3.2.7)
Quantization Loss  To minimize the quantization error between real-valued $\hat{H}$ and $\hat{B}$ and binary $H$ and $B$, we introduce a regularization term to force the entries of $\hat{H}$ and $\hat{B}$ to be near to +1 or -1, which is defined in the following formula:

$$J_1 = Q(\hat{H}) + Q(\hat{B})$$

$$= \frac{1}{Z_H}||\hat{H}||_2^2 + \frac{1}{Z_B}||\hat{B}||_2^2$$

(3.2.8)

where $1_H$ and $1_B$ are the matrices of the same dimensionality as $H$ and $B$ with all entries being 1. $Z_H = r \times m$ and $Z_B = r \times n$ are normalizing factors to eliminate the effect of the training set size and the hash code length. We choose the L2-norm regularizer for the simplicity of its optimization.

Overall Formulation  We formulate the cost function of our APHash method as:

$$J = J_0 + \lambda J_1$$

$$= \min_{\hat{H}, \hat{B}} \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ji} \log \frac{p_{ji}}{q_{ji}}$$

$$+ \lambda \left( \frac{1}{Z_H}||\hat{H}||_2^2 + \frac{1}{Z_B}||\hat{B}||_2^2 \right)$$

(3.2.9)

where $\lambda$ is a parameter to balance the effects of the distribution-preserving loss and the quantization loss. The asymmetric property of APHash provides an opportunity to learn more informative hash codes. As demonstrated in [81], the asymmetric structure tends to lead to a better retrieval performance, especially under the configuration of a short hash code length.

3.2.2 Optimization

We propose to utilize gradient descent based optimization technique to solve this unconstrained optimization problem and learn locally optimal $\hat{B}$ and $\hat{H}$. We calculate the gradients with respect to $\hat{B}$ and $\hat{H}$ respectively. The derivative with respect to $\hat{h}_i$ is as follows:

$$\frac{\partial J}{\partial \hat{h}_i} = \sum_{j=1}^{n} \left( 1 + \frac{1}{4}||\hat{h}_i - \hat{b}_j||_2^{-2} \right) \times (p_{ji} - q_{ji})$$

$$\times (\hat{h}_i - \hat{b}_j) + \frac{\lambda}{Z_H}(|\hat{h}_i| - 1) \odot sign(\hat{h}_i)$$

(3.2.10)
Similarly, the derivative w.r.t $\hat{b}_j$ is:

$$
\frac{\partial J}{\partial \hat{b}_j} = -\sum_{j=1}^{n} \left( 1 + \frac{1}{4} \| \hat{h}_i - \hat{b}_j \|_2^2 \right)^{-1} \times (p_{j|i} - q_{j|i}) \times (\hat{h}_i - \hat{b}_j) + \frac{\lambda}{Z_B}(|\hat{b}_j - 1|) \odot \text{sign}(\hat{b}_j)
$$

(3.2.11)

where $\odot$ denotes entry-wise multiplication, $\mathbf{1}$ is a column vector with all entries being 1. The optimization process will be performed in an alternative way. Stochastic Gradient Descent is applied to optimize $\hat{H}$ with Eq. (3.2.10) while fixing $\hat{B}$, and update $\hat{B}$ with Eq. (3.2.11) while fixing $\hat{H}$. These two step will be iteratively performed until convergence or reaching the predefined maximum steps. The corresponding binary hash code matrices can be derived via the sign function, i.e. $H = \text{sign}(\hat{H})$ and $B = \text{sign}(\hat{B})$.

### 3.2.3 Hash Function Learning

As discussed before, we treat query and database in an asymmetric way and learn hash function with anchor set. In the retrieval phrase, we can make use of the correlations between anchors and database items then compute hash codes for queries with hash functions while directly using hash codes for database items. Therefore, we utilize the original data matrix $C$ and the binary matrix $H$ of the anchors as the guidance to learn the linear hash functions due to its simplicity. The objective function is as follows:

$$
L = \min_{W} \| H - W^T C \|_2^2 + \alpha \| W \|_2^2
$$

(3.2.12)

where $W \in \mathbb{R}^{r \times d}$ is the projection matrix to be learned, $\alpha$ is a hyper parameter to weight the L2 norm imposed on $W$. Finally, we obtain the analytical solution of $W$:

$$
W = (CC^T + \alpha I_{d \times d})^{-1} CH^T
$$

(3.2.13)

where $I_{d \times d}$ is an identity matrix of dimensions $d \times d$.

The overall pipeline of hash code learning and hash function learning of our APHash method is briefly described in Algorithm. 1.
3.3 Experimental Results

To evaluate the effectiveness of our proposed APHash method, we conduct experiments on two publicly available benchmark datasets: CIFAR-10 [51] and Youtube Faces [108]. The details of the experiments and the results are described in the following subsections.

3.3.1 Datasets and Experimental Setup

CIFAR-10 Dataset  This dataset consists of 60,000 color images from 10 classes (6,000 images per class), and each image has the spatial resolution of $32 \times 32$. Following the standard input [67, 70], each image is represented by a 512-dimensional GIST feature [84]. We randomly select 1000 images for testing with the rest 59000 images for training. Since CIFAR-10 dataset is fully annotated, we evaluate retrieval performance using the ground-truth semantic labels.

Youtube Faces Dataset  This dataset consists of face data involved with 1,595 persons. The number of frames for each person ranges from 48 to 6070. On average, each person contains about 181 frames. We first randomly choose 30 individuals with at least 1600 images. Each face image is represented by a 1,770-dimensional LBP feature vector [3], which is a standard setting. Specifically, we randomly sample 100 images from each individual to form the testing set of 3000 images. We further randomly select 1500 images from each individual to form a training set of 45,000 images. For the evaluation of image retrieval performance, two images of the same person are considered as a similar pair.
Algorithm 1 Learning algorithm for APHash

Input:

$X$: training data;
$m$: the size of anchor set;
$\text{iter}$: the maximum number of iterations;
$\lambda$, $\alpha$: the tradeoff hyperparameters of Eq. (3.2.9) and Eq. (3.2.12);
$k$: the number of nearest neighbors

Output:

$W$: the projection matrix of the hash function;
$H, B$: the hash code matrices of the anchor set and the whole training set

1: Initialize the real-valued matrices $\hat{B}$ and $\hat{H}$.
2: Randomly select $m$ anchors and compute probability distribution $\mathcal{P}$ with Eq. (3.2.1).
3: for $i = 1 \rightarrow \text{iter}$
   (Re)compute probability distribution $\mathcal{Q}$ with Eq. (3.2.7).
   Update $\hat{H}$ with Eq. (3.2.10).
   Recompute probability distribution $\mathcal{Q}$ with Eq. (3.2.7).
   Update $\hat{B}$ with Eq. (3.2.11).
end for
4: Compute $H, B$ with the sign function.
5: Compute the projection matrix $W$ with Eq. (3.2.12).
Figure 3.2: Example images of dataset CIFAR-10.
**Parameters**  For the setting of parameters, we set the number of nearest neighbors $k$ as 1,000 for two datasets empirically. The size of the anchor set is set as 10,000 in all experiments. For the hyperparameters $\alpha$ to control the weight of L2 norm on $W$ and $\lambda$ to balance the effect of quantization loss, we set $\alpha = 1$ and $\lambda = 0.01$ respectively. In our experiments, we observed that the objective typically converges at about 50 iterations. Therefore, for the optimization, we set the maximum number of iterations as 50. The real-valued matrices $\hat{H}$ and $\hat{B}$ are initialized with a Gaussian distribution of mean 0 and standard deviation 1. The results of the compared methods are obtained with the codes provided by the authors, and the setting of parameters follows the suggestions in the original papers.

**Experimental Protocol**  Following the previous methods [45], [28], we evaluate the proposed APHash with three criterions: mean average precision (mAP), precision-recall curve, and precision curve. Specifically, the widely-used mAP criteria is defined as follows:

$$mAP = \frac{1}{|O|} \sum_{i=1}^{\left|O\right|} \frac{1}{s_i} \sum_{j=1}^{s_i} \Delta R_{i,j}$$

where $|O|$ is the size of query dataset $O$, and $s_i$ is the number of groundtruth relevant samples in the database for query $i$. The $R_{i,j}$ is defined as the ranking list of the returned instances according to the Hamming distance to query $i$. $\Delta R_{i,j}$ is defined as the precision computed for the $j$th instance if it is groundtruth relevant with the query $i$, otherwise $\Delta R_{i,j}$ is set to zero. The another important evaluation criteria is the precision-recall curve, which is also widely used in the retrieval area. Most of the time, the model is expected to obtain high precision while maintaining high recall at the same time. Since these two datasets are fully-annotated, the groundtruth relevant instances for a query are defined as those instances sharing the same label with it.
Table 3.2: Mean Average Precision of Hamming Ranking for different numbers of bits on two datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (mAP)</th>
<th>Youtube Faces (mAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>8 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td>LSH</td>
<td>0.1170</td>
<td>0.1222</td>
</tr>
<tr>
<td></td>
<td>0.1116</td>
<td>0.1586</td>
</tr>
<tr>
<td>SH</td>
<td>0.1295</td>
<td>0.1301</td>
</tr>
<tr>
<td></td>
<td>0.3900</td>
<td>0.5857</td>
</tr>
<tr>
<td>AGH</td>
<td>0.1507</td>
<td>0.1580</td>
</tr>
<tr>
<td></td>
<td>0.4527</td>
<td>0.6362</td>
</tr>
<tr>
<td>DSH</td>
<td>0.1470</td>
<td>0.1487</td>
</tr>
<tr>
<td></td>
<td>0.1440</td>
<td>0.1483</td>
</tr>
<tr>
<td>SpH</td>
<td>0.1460</td>
<td>0.1487</td>
</tr>
<tr>
<td></td>
<td>0.2646</td>
<td>0.3865</td>
</tr>
<tr>
<td>OEH</td>
<td>0.1373</td>
<td>0.1531</td>
</tr>
<tr>
<td></td>
<td>0.2182</td>
<td>0.2946</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.1545</td>
<td>0.1572</td>
</tr>
<tr>
<td></td>
<td>0.2646</td>
<td>0.3865</td>
</tr>
<tr>
<td>APHash</td>
<td>0.1630</td>
<td>0.1515</td>
</tr>
<tr>
<td></td>
<td>0.1610</td>
<td>0.1440</td>
</tr>
<tr>
<td></td>
<td>0.1779</td>
<td>0.1625</td>
</tr>
<tr>
<td></td>
<td>0.1850</td>
<td>0.1725</td>
</tr>
<tr>
<td></td>
<td>0.1980</td>
<td>0.2182</td>
</tr>
<tr>
<td></td>
<td>0.4900</td>
<td>0.5857</td>
</tr>
<tr>
<td></td>
<td>0.6975</td>
<td>0.7500</td>
</tr>
<tr>
<td></td>
<td>0.7499</td>
<td>0.7500</td>
</tr>
</tbody>
</table>
Figure 3.3: Performance evaluated with precision-recall curve (@8 bits and @64 bits respectively) and precision curve w.r.t. top returned samples (@8 bits) on CIFAR-10 Dataset.
3.3.2 Comparison with State-of-the-art Methods

We follow the recently published paper [67] and compare the proposed APHash with several state-of-the-art methods, including Locality Sensitive Hashing (LSH) [25], Spectral Hashing (SH) [106], Density Sensitive Hashing (DSH) [45], Spherical Hashing (SpH) [34], Anchor Graph Hashing (AGH) [71], Iterative Quantization (ITQ) [28], and Ordinal Embedding Hashing (OEH) [67]. We implement our proposed APHash method with Matlab on a PC with Intel Core i7-6700 3.40GHz. All the experiment results reported in the following sections are the average of performance over 10 runs.

The quantitative results of comparison with seven state-of-the-art methods listed above are shown in Table 3.2, Figure 3.3, and Figure 3.4. For the comparison with mAP criteria, the length of hash codes varies from 8 bits to 64 bits. From Table 3.2, we can observe that the proposed APHash consistently outperforms all the compared approaches and achieves the state-of-the-art retrieval performance on both CIFAR-10 and Youtube Faces datasets. As for the precision-recall curve, we show the experimental results with short hash codes (8 bits) and long hash codes (64 bits) on two datasets in Figure 3.3 (a) (b) and Figure 3.4 (a) (b). We also show the precision curves to demonstrate the effectiveness of the APHash on two datasets in Figure 3.3 (c) and Figure 3.4 (c). As we can see, APHash also outperforms state-of-the-art methods. Experimental results show that APHash can learn more discriminative hash codes and more powerful hash function. Especially, with short hash codes (e.g. 8 bits), the proposed APHash outperforms the compared methods by a large margin, which demonstrates that such an asymmetric structure leads to the accommodation of much more information as discussed in Section 3.2.1.
Figure 3.4: Performance evaluated with precision-recall curve (@8 bits and @64 bits respectively) and precision curve w.r.t. top returned samples (@8 bits) on Youtube Faces Dataset.
Figure 3.5: The effect of choosing different numbers of nearest neighbors. APHash achieves better performance on both datasets at around $k = 1000$.

Table 3.3: Mean Average Precision (mAP) on different size of anchor set $|C|$ on CIFAR-10. The best mAP is in bold while the second best one is underlined.

<table>
<thead>
<tr>
<th>CIFAR-10</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C</td>
</tr>
<tr>
<td>8 bits</td>
<td>0.1156 0.1390 0.1446 <strong>0.1635</strong> 0.1630</td>
</tr>
<tr>
<td>32 bits</td>
<td>0.1203 0.1547 0.1563 <strong>0.1784</strong> 0.1779</td>
</tr>
</tbody>
</table>

### 3.3.3 Model Analysis

In this part, we will briefly give some analysis about the proposed APHash model.

Figure 3.5 shows the effect of choosing different $k$, which means the number of nearest neighbors chosen in this model. It can be seen that the method achieves better mAP when the number of nearest neighbors is set as around 1000 on CIFAR-10 and Youtube Faces datasets.

We also conduct experiments to show the effectiveness of different size of anchor set $|C|$. By varying the value of $|C|$, we show the experimental results in Table 3.3 and 3.4. It is observed that when the size of anchor set is increasing, the
Table 3.4: Mean Average Precision (mAP) on different size of anchor set $|C|$ on Youtube Face. The best mAP is in bold while the second best one is underlined.

<table>
<thead>
<tr>
<th>Youtube Face</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C</td>
</tr>
<tr>
<td>8 bits</td>
<td>0.2459</td>
</tr>
<tr>
<td>32 bits</td>
<td>0.2883</td>
</tr>
</tbody>
</table>

Table 3.5: Mean Average Precision (mAP) obtained by using k-mean clustering. The best mAP is in bold while the second best one is underlined.

<table>
<thead>
<tr>
<th>Youtube Face</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C</td>
</tr>
<tr>
<td>8 bits</td>
<td>0.1879</td>
</tr>
<tr>
<td>32 bits</td>
<td>0.3049</td>
</tr>
</tbody>
</table>

performance of model is more or less consistently increasing. Note that to make the model implementable, we thus choose the possibly largest value $|C| = 10,000$ in our implementation.

Finally, we empirically demonstrate the rationality of constructing anchor set by random sampling. Comparing Table 3.5 and 3.4, one can see that the performance obtained by using random sampling consistently outperforms that obtained by using k-mean clustering for different size of anchor set. One reason we believe should be that the anchor set obtained by random sampling actually follows the true distribution of training data which can represent the manifold of training data to some extent. However, the distribution of anchor set obtained by k-mean clustering no longer aligns to the underlying distribution of training data, which thus degrades the performance.
3.4 Conclusions

In this chapter, an unsupervised Anchor-based Probability Hashing (APHash) method is proposed by preserving the distribution of data points. This method makes use of the correlation between anchors and data points to learn informative hash codes by treating them asymmetrically. It transforms the distances between data points into probability distributions and minimizes the KL-divergence to preserve the data structure. An alternating optimization algorithm is proposed to solve the problem. Experiments conducted on two publicly available datasets demonstrate that the proposed APHash outperforms the compared approaches and achieves state-of-the-art retrieval performance.
Chapter 4

DeepQuan: Learning Deep
Unsupervised Binary Codes for
Image Retrieval

4.1 Motivation

While supervised methods make use of provided labels to mitigate the gap between high-level semantic and low-level feature representation of items, the performance is always hindered by the availability of supervision information. This is especially challenging when deep models are considered as a large amount of labeled data is typically needed for the training. As it is much easier to collect a high volume of unlabeled data, deep unsupervised hashing learning methods become attractive. Deep unsupervised hashing learns multiple hierarchical transformations to capture the nonlinear manifold structure of data [21]. While some kernel-based methods [70] have been developed for the hashing task, they are however not scalable.

With the recent progress of deep learning [96], many deep unsupervised hashing methods have been proposed. For example, Deep Hashing (DH) [21] builds hierarchical neural networks to learn binary codes. DeepBit [62] and Unsupervised Triplet Hashing (UTH) [37] learn hashing functions with data argumentation. Binary Deep
Figure 4.1: The framework of the proposed DeepQuan model, which is built upon a deep autoencoder. All the layers are fully connected. A weighted triplet loss and a manifold preserving loss are employed to learn discriminative compact representations for better quantization.
Neural Network (UH-BDNN) [19] was developed by maintaining a discrete binary constraint during training. The recently proposed DBD-MQ [20] obtains the hashing function by using K-AutoEncoders network to minimize the reconstruction errors. Among the existing deep unsupervised hashing methods proposed for image retrieval [20, 21, 62, 37], most of them make use of a simple quadratic constraint to minimize the difference between the compact representations and the target binary codes during the training, which inevitably causes severe information loss and thus produces suboptimal binary codes. Also, as discussed in [8, 22], to quantize all input vectors effectively with vector quantization (VQ) [30], the data should exhibit a cluster structure. Hence it is also important that to learn the image representations so that such a cluster structure can be resulted.

In this chapter, I will present an unsupervised deep hashing model called DeepQuan (Deep Quantization) to address the aforementioned problems. Our contributions can be summarized as follows:

• Instead of forcing the outputs of the deep network to be binary using a quadratic constraint, we propose to generate binary codes with product quantization (PQ) [42, 22]. To the best of our knowledge, the proposed model DeepQuan is the first attempt to incorporate PQ into a deep model to learn binary code in an unsupervised way. This model is built upon a deep autoencoder, and an alternating optimization method is proposed for the model learning. By optimizing the objective function, more discriminative feature representations (i.e. discriminative cluster structure) can be obtained, so that PQ can quantize the feature vectors more effectively.

• To avoid trivial solutions and poor generalization during training, we propose a novel weighted triplet loss to learn more discriminative features and demonstrate its effectiveness empirically.

• Experimental results on two standard datasets (i.e. CIFAR-10 and MNIST) show that DeepQuan outperforms the existing unsupervised deep methods and achieves the state-of-the-art performance in the image retrieval application.
4.2 Methodology

In this section, we describe the proposed DeepQuan in detail. We use bold lowercase letters like \( x \) to indicate vectors and the uppercase letters like \( X \) to denote matrices. \( \| \cdot \|_2 \) and \( \| \cdot \|_0 \) are L2 and L0 norm respectively. Given a training set with \( N \) image feature vectors \( X = \{x_i\}_{i=1}^N \), where \( x_i \) denotes the \( D \)-dimensional feature vector of the \( i^{th} \) image. The unsupervised hashing problem is defined as learning a mapping function \( f: x \mapsto h \in \{0,1\}^B \), which encodes a feature vector into a \( B \)-bit binary vector, so that the pairwise similarity in the original feature space is preserved as much as possible.

Our proposed model DeepQuan is built upon a deep autoencoder network, and the whole architecture is shown in Figure 4.1. It accepts feature vectors (i.e. hand-crafted features or deep features) as inputs, and consists of three components: (1) multiple fully-connected layers used as an encoder to map high-dimensional inputs to low-dimensional compact representations; (2) a symmetric decoder for preserving the manifold structure within data; (3) a proposed weighted triplet loss on low-dimensional representations for training the deep model and learning the binary codes.

4.2.1 Model Formulation

The model configuration of our deep network follows the setting of [109]. It is a symmetric architecture, in which both the encoder and decoder consist of five fully connected layers but reversed to each other. Each fully connected layer learns a nonlinear mapping \( z_i^l = \alpha^l(W^l z_i^{l-1} + b^l) \), where \( z_i^l \) is the \( l^{th} \) hidden representation of input data \( x_i \), \( W^l \) and \( b^l \) are the weight and bias parameters of the \( l^{th} \) layer respectively. The activation functions for all the layers except the last fully-connected layer are rectifier units (ReLU) \( \alpha(x) = \max(0, x) \). Unless otherwise specified, we use \( z_i \), \( f(\cdot) \) and \( g(\cdot) \) to represent the middle hidden representation of \( x_i \), the encoder and the decoder respectively in the rest of the chapter.
Figure 4.2: Computing the binary codes on MNIST dataset @ 32bit. (a) Average distance between codewords which is computed using Eq. (4.2.1) and Eq. (4.2.3) keeps decreasing, resulting in trivial solutions (i.e. all the codewords become the same) and thus poor generalization. It is obvious that Eq. (4.2.4) ends up with a much larger average distance, which leads to more discriminative structure. (b) Loss value decreases until convergence as the training proceeds.
Weighted Triplet Loss  We employ product quantization (PQ) to generate binary codes from the middle hidden representation \( z_i \in \mathcal{R}^L \) by minimizing the quantization error. The product quantization is one special case of vector quantization (VQ), where any codeword is taken from the Cartesian product of \( M \) sub-codebooks [22]. Thus, we first decompose the representation \( z_i \) into \( M \) parts \( z_i = [z_{i1}, z_{i2}, \ldots, z_{iM}]^T \), where \( z_{im} \in \mathcal{R}^{L/M} \) is the subvector related to \( m^{th} \) subspace. In each subspace, \( K \) codewords are generated using \( K \)-means clustering to quantize all the subvectors \( \{z_{im}\}_{i=1}^N \). Finally, there will be \( K^M \) codewords in total. The way to measure the quantization error, as adopted in [8], is to take a quadratic form, formulated as:

\[
\sum_{m=1}^{M} \sum_{i=1}^{N} \|z_{im} - C_m h_{im}\|_2^2
\]

subject to \( h_{im} \in \{0, 1\}^K, \|h_{im}\|_0 = 1 \)

(4.2.1)

where \( C_m \) denotes the codebook associated with the \( m^{th} \) subspace. The codebook \( C_m \) is composed of \( K \) codewords \( C_m = [C_{m1}, C_{m2}, \ldots, C_{mK}] \), where \( C_{mk} \in \mathcal{R}^{L/M} \). The constraint indicates that \( h_{im} \) is a one-hot \( K \)-dimensional binary vector, which means each subvector is represented by only one codeword. Following [22, 8, 73], to represent each data point \( x_i \) with a compact binary vector, each \( h_{im} \) is compressed into one \( \log_2 K \)-bit vector, with which the final binary vector is the concatenation of \( M \log_2 K \)-bit vectors. In other words, the length of hash codes \( B = M \log_2 K \). For all the experiments we conduct, we set \( K = 256 \). Each component thus is represented with 8 bits.

One can simply take Eq. (4.2.1) as the objective function and alternatively optimize it with respect to \( z_i \) and \( C_m \). However, the result of learning the deep model this way easily is not desirable. We conducted a preliminary experiment and found that all the codewords within a codebook are more or less the same as indicated by the average distance among them as shown in Figure 4.2(a). This indicates that the hidden representations \( z_i \) do not exhibit obvious clustering structure for the product quantization to leverage on. We further conduct experiments to visualize the learned feature space in Figures 4.4 and 4.5.
To promote clustering structure to be inferred in the hidden representations, we propose a novel weighted triplet loss that takes all the codewords (i.e. clusters) into consideration instead of only the assigned codewords to learn $z_i$ during the training. This is to allow more discriminative clustering structure to be obtained. In particular, the weighted triplet loss can be formulated as:

$$J(z_i, C_i^+, C_i^-) = \max\{s - (\lambda\|z_i - C_i^-\|_2 - \|z_i - C_i^+\|_2), 0\}$$ \hspace{1cm} (4.2.2)

subject to $\lambda \in (0, 1)$

where $C_i^+ = [C_{i1}^+, C_{i2}^+, \ldots, C_{iM}^+]$ and $C_i^- = [C_{i1}^-, C_{i2}^-, \ldots, C_{iM}^-]$ denote the concatenations of all positive and negative codewords respectively. Here, positive codewords are defined as the cluster centers to which $z_{im}$ is assigned via K-means clustering. Negative codewords denote those cluster centers to which $z_{im}$ is not assigned. $s$ denotes a margin parameter. Negative codewords are generated by randomly sampling.

Compared with the simple quadratic loss in Eq. (4.2.1) which only considers the relation with positive codewords, our proposed weighted triplet loss simultaneously enlarges the distance between negative codewords $C_i^-$ and $z_i$ so that the discriminative clustering structure can be obtained, which will further benefit the quantization process. We add a weighting parameter $\lambda \in (0, 1)$ to the negative term in Eq. (4.2.2) to control the emphasis on learning discriminative representations $z_i$.

**Manifold Preserving Loss** To ensure that the hidden representation $z_i$ can well recover the data manifold in the original feature space, we employ a decoder to define the regularization term formulated as follows:

$$R(z_i) = \|x_i - g(z_i)\|^2_2$$ \hspace{1cm} (4.2.3)

where $x_i$ is the input feature vector and $g(\cdot)$ denotes the decoder network.

**Overall Objective** We formulate the overall objective function for model training as:

$$L_\theta = \sum_{i=1}^{N} J(z_i, C_i^+, C_i^-) + \eta R(z_i)$$ \hspace{1cm} (4.2.4)
where $\theta$ denotes the parameters of the deep neural network, and $\eta$ is a trade-off parameter to control the balance between feature discrimination and reconstruction quality so as to lead to better generalization.

### 4.2.2 Approximate Nearest Neighbor Search

For testing, given a new query point $x_q$, we compute a symmetric distance between $x_q$ and the data points $x_i$ in the image database based on the inferred binary codes $\{h_i\}_{i=1}^N$ and the codebooks $C$ for the retrieval. Specially, we adopt the one used in [42], defined as:

$$SD(x_q, x_i) = \sum_{m=1}^{M} \|q(x_q)_m - C_m h_{im}\|^2. \quad (4.2.5)$$

To compute that, we first precompute the distance look-up table of size $K \times K$ between the codewords. Thus, there will be $M K \times K$ look-up tables altogether. With the $M$ pre-computed look-up tables, the distance $SD(x_q, x_i)$ can be efficiently computed by summing up the codewords distance directly obtained from look-up tables. It is only slightly more costly than computing the Hamming distance [42].

Note that the distance computed with Eq. (4.2.5) is an approximation of the Euclidean distance between $x_q$ and $x_i$. As discussed in [42], we show that the error is statistically upper bounded by the quantization error.

**Theorem 1** The statistical error between the approximated distance $SD(x_q, x_i)$ and the Euclidean distance $d(x_q, x_i)$ is upper bounded as:

$$|SD(x_q, x_i) - d(x_q, x_i)| \leq |d(x_q, q(x_q)) + d(x_i, q(x_i))| \quad (4.2.6)$$

**Proof** The upper bound can be derived by simply using the triangle inequality:

$$|SD(x_q, x_i) - d(x_q, x_i)|$$

$$= |d(q(x_q), q(x_i)) - d(x_q, x_i)|$$

$$\leq |d(x_q, q(x_q)) + d(x_q, q(x_q)) - d(x_q, x_i)|$$

$$\leq |d(x_i, q(x_i)) + d(x_q, q(x_q))|$$

49
4.2.3 Alternating Optimization

We propose to optimize Eq. (4.2.4) in an alternating way. We learn the compact representation $z$, the codebooks $C$, and the binary codes $h$ iteratively until convergence.

Learning Model Parameters $\theta$ and $z$ Standard back propagation (BP) is adopted to update the model parameters $\theta$ and $z$. We simply take the gradient of the objective function $L_{\theta}$ with respect to $z$ as follows:

$$
\frac{\partial L_{\theta}}{\partial z_i} = \frac{\partial J(z_i, C_i^+, C_i^-)}{\partial z_i} \cdot \mathbb{1}\{s - (\lambda\|z_i - C_i^-\|_2 - \|z_i - C_i^+\|_2)\} \\
+ 2\eta(x_i - g(z_i)) \frac{\partial g(z_i)}{\partial z_i}
$$

(4.2.7)

where $\mathbb{1}\{s - (\lambda\|z_i - C_i^-\|_2 - \|z_i - C_i^+\|_2)\}$ is an indicator function which will return 1 if $s - (\lambda\|z_i - C_i^-\|_2 - \|z_i - C_i^+\|_2) > 0$, and 0 otherwise. To optimize the deep model, we back propagate the gradient $\frac{\partial z_i}{\partial \theta}$ through the chain rule and update the parameters $\theta$ using the gradient descent method.

Learning $C$ and $h$ As shown in Theorem 1, the quality (i.e. measured by quantization error) of codebooks $C$ has crucial impact on the final retrieval performance. By minimizing the quantization error, we can obtain the optimal codebooks $C$. For the $m^{th}$ subspace, we have

$$
\sum_{i=1}^{N} \|z_{im} - C_m h_{im}\|_2^2
$$

and the estimated values of $C$ and $h$ are obtained using $K$-means following [42]. In our implementation, we update the codebook $C$ and $h$ every epoch. As Eq. (4.2.4) tries to drive the hidden features to have a clustering structure, the updating of the cookbook for each cluster will eventually be only affected by the hidden representations assigned to it, but not the others. This makes the alternating optimization to converge. Figure 4.2(b) shows that the objective function Eq. (4.2.4) converges based on the optimization algorithm proceeds.
Table 4.1: Mean Average Precision (%) for different numbers of bits on two datasets. The best MAPs are shown in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST (MAP@All)</th>
<th>CIFAR-10-GIST (MAP@1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>DeepQuan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UH-BNN</td>
<td>45.38</td>
<td>17.83</td>
</tr>
<tr>
<td></td>
<td>43.14</td>
<td>28.64</td>
</tr>
<tr>
<td></td>
<td>44.97</td>
<td>13.64</td>
</tr>
<tr>
<td></td>
<td>46.74</td>
<td>13.61</td>
</tr>
<tr>
<td></td>
<td>16.17</td>
<td>13.54</td>
</tr>
<tr>
<td></td>
<td>16.62</td>
<td>13.54</td>
</tr>
<tr>
<td></td>
<td>16.96</td>
<td>13.54</td>
</tr>
<tr>
<td></td>
<td>16.17</td>
<td>13.54</td>
</tr>
<tr>
<td></td>
<td>16.62</td>
<td>13.54</td>
</tr>
<tr>
<td></td>
<td>16.96</td>
<td>13.54</td>
</tr>
<tr>
<td>ITQ</td>
<td>41.18</td>
<td>32.12</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
<tr>
<td>Spherical</td>
<td>26.64</td>
<td>24.85</td>
</tr>
<tr>
<td>SH</td>
<td>25.81</td>
<td>31.71</td>
</tr>
<tr>
<td></td>
<td>30.77</td>
<td>13.98</td>
</tr>
<tr>
<td></td>
<td>35.78</td>
<td>12.55</td>
</tr>
<tr>
<td></td>
<td>15.67</td>
<td>12.55</td>
</tr>
</tbody>
</table>
4.3 Experimental Results

To evaluate the performance of our proposed DeepQuan method for image retrieval, we conduct extensive experiments on two publicly available datasets. The details of the experiments and the results are described in the following subsections.

4.3.1 Datasets and Experimental Setting

To make our performance comparison consistent with some recent related work [75, 37, 21], we conduct our experiments on two widely used benchmark datasets: CIFAR-10 and MNIST.

CIFAR-10 The CIFAR-10 dataset consists of 60,000 labeled color images in 10 classes. Each class contains 6,000 images of size $32 \times 32$.

MNIST The MNIST dataset contains 70,000 gray-scale images in 10 classes of digits from “0” to “9”. Each image is of size $28 \times 28$. Example images are shown in Figure 4.3.

To demonstrate the effectiveness of the proposed DeepQuan model, we compare it with several state-of-the-art unsupervised hashing methods including: DH [75, 21], DeepBit [62], DBD-MQ [20], UH-BNN [19], UTH [37], ITQ [28], KMH [32], SH [106], LSH [18], AGH [71], Spherical [34], PCAH [103]. For the comparison on dataset MNIST, we follow the setting of DH [75, 21]. Specifically, we randomly sample 1,000 images (100 images per class) to form the test (i.e., query) set and take the rest 69,000 images as the training set and the gallery database. Each image is represented by a 784-D gray-scale feature vector with the pixel values as its elements. For the experiments on the dataset CIFAR-10, we divide the them into two categories according to the input feature vector. Following the settings of DH and UH-BNN, we compare DeepQuan with those methods whose inputs are 512-D GIST [84] feature vectors extracted from the raw images, including DH, UH-
BNN, ITQ, KMH, SH, LSH, AGH, Spherical and PCAH. We denote the CIFAR-10 dataset with these features as CIFAR-10-GIST. For the CIFAR-10-GIST dataset, again 1,000 images (100 images per class) are randomly sampled as the query data. The rest 59,000 images are used as the training set. For fair comparison with the other deep models (i.e., DeepBit, DBD-MQ, UTH) with raw pixel images as the input and using convolutional neural network (CNN), we extract 7th fully-connected layer from the deep VGG-16 [96] network as the feature vector inputs for our model. This VGG-16 model is also used in DBD-MQ [20] as part of the model. We denote this dataset as CIFAR-10-CNN, where 10,000 images are sampled as the query set and the rest 50,000 images are used for training.

Following [109], we configure our model as a deep autoencoder with the number of unit of encoder as $[D - 500 - 500 - 2000 - L]$, where $D$ and $L$ are the dimensions of the input vector and the middle hidden layer respectively. The dimension $L$ is essentially the length of binary codes, specifically $L = 16M$. The trade-off parameter $\eta = 1.0$ for all the experiments. Also, we have tested the cases empirically with $s = 1.0, 1.0, 0.001$ and $\lambda = 0.1, 0.2, 0.2$ for the datasets MNIST, CIFAR-10-CNN and CIFAR-10-GIST respectively. As adopted in most of the related work, the mean average precision (MAP) is used to measure the retrieval performance. For the performance of the existing methods, we make direct reference to the results reported in the corresponding papers. For our model, we report the average performance of 10 trials. We first pre-train the basic deep autoencoder using the training set. Then, we initialized the network accordingly and then further trained the model by optimizing Eq. (4.2.4) as explained in Section 4.2.3. We set the batch size as 512 and the learning rate as 0.01 for the stochastic gradient descent. We found that around 3,500 iterations are enough for the model to converge.
Figure 4.3: Example images of dataset MNIST.
4.3.2 Performance Comparison

In this subsection, we present the results of performance comparison between DeepQuan and the state-of-the-art methods. The MAP results on MNIST and CIFAR-10-GIST datasets are listed in Table 4.1. We can find that our DeepQuan method outperforms the state-of-the-art methods by a large margin, especially under the short-bit settings. For example, for the 16-bit experiment of MNIST, DeepQuan outperforms the best performance of compared method UH-BNN by about 15%. In Table 4.2, we show the results evaluated on CIFAR-10-CNN. Compared with deep unsupervised CNN models which take raw pixels images as inputs, DeepQuan also achieves the best performance. Noted that the MAP value is computed on the whole database for MNIST while it is computed on the top 1,000 returned samples for CIFAR-10-GIST and CIFAR-10-CNN (as what being adopted in [21, 20]), which are denoted by MAP@All and MAP@1000 respectively. We believe that these large-margin improvements are due to the capability of the product quantization design incorporated into the deep model to preserve well the underlying data manifold.

4.3.3 Empirical Analysis of DeepQuan

To better understand how different parts of the proposed DeepQuan contribute to the overall improved performance, we conduct additional experiments. We make use of the datasets MNIST and CIFAR-10-CNN for this set of experiments, and 1,000 images are sampled from each dataset to form a query set with the rest data for training.
Table 4.2: Mean Average Precision (%) (i.e. MAP@1000) for CIFAR-10-CNN dataset. The best MAPs are shown in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>16 bits</th>
<th>32 bits</th>
<th>64 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepBit</td>
<td>19.43</td>
<td>24.86</td>
<td>27.73</td>
</tr>
<tr>
<td>UTH</td>
<td>28.66</td>
<td>30.66</td>
<td>32.41</td>
</tr>
<tr>
<td>DBD-MQ</td>
<td>21.53</td>
<td>26.50</td>
<td>31.85</td>
</tr>
<tr>
<td>DeepQuan</td>
<td><strong>39.95</strong></td>
<td><strong>41.25</strong></td>
<td><strong>43.26</strong></td>
</tr>
</tbody>
</table>

Table 4.3: We compare DeepQuan with two variant methods PQ and DAE-PQ to show the encoder can learn more discriminative compact features. Results are compared with MAP@All under relatively short bit setting.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST</th>
<th>CIFAR-10-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td>PQ</td>
<td>52.98</td>
<td>48.30</td>
</tr>
<tr>
<td>DAE-PQ</td>
<td>67.52</td>
<td>54.16</td>
</tr>
<tr>
<td>DeepQuan</td>
<td><strong>74.32</strong></td>
<td><strong>60.30</strong></td>
</tr>
</tbody>
</table>

Table 4.4: We compare DeepQuan with two variants Quad and WTPs to show effectiveness of the proposed weighted triplet loss and inclusion of the reconstruction term. MAP@All and MAP@1000 are computed for MNIST and CIFAR-10-CNN respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST</th>
<th>CIFAR-10-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
<td>64 bits</td>
</tr>
<tr>
<td>Quad</td>
<td>53.18</td>
<td>28.12</td>
</tr>
<tr>
<td>WTPs</td>
<td>59.90</td>
<td>49.79</td>
</tr>
<tr>
<td>DeepQuan</td>
<td><strong>60.30</strong></td>
<td><strong>52.54</strong></td>
</tr>
</tbody>
</table>
Figure 4.4: Visualization of the first four subspaces learned with Quad-R on MNIST @32 bits, which is depicted with t-SNE. Best view in color.
Figure 4.5: Visualization of the first four subspaces learned with DeepQuan on MNIST @32 bits, which is depicted with t-SNE. Best view in color.
To evaluate the effectiveness of the proposed unified framework to learn both discriminative and compact hidden representations and product quantization in DeepQuan, we conduct experiments with relatively shorter bit-length (8 bits and 16 bits), and compute the hidden representations and the product quantization differently. In particular, we run product quantization [42] directly on the original feature vectors, and on the feature vectors extracted from the pre-trained deep autoencoder, denoted as PQ and DAE-PQ respectively. The results are reported in Table 4.3. DeepQuan achieves the best performance. In other words, it learns more discriminative compact features which in turn can ensure more effective quantization.

Next, to evaluate the effectiveness of the proposed weighted triplet loss (Eq. (4.2.2)) as compared with the standard quadratic loss (Eq. (4.2.1)), we conduct experiments by trying variants of the overall objective function. We tried the following settings: (i) We drop the reconstruction term (i.e. $\eta = 0$) and train the model by minimizing only Eq. (1) (denoted as Quad); (ii) We drop the reconstruction term and train the model by minimizing Eq. (4.2.2) (denoted as WTPs). As shown in Table 4.4, Quad gives the worst performance that is anticipated, and then followed by WTPs. DeepQuan considering all the reconstruction term during the training achieves the best results.

We further demonstrate the use of the weighted triplet loss can end up with a much more discriminative clustering structure. Figures 4.4 and 4.5 visualize the hidden features learned from MNIST under 32-bit setting. Note that for 32-bit setting, the hidden representation $z_i$ is decomposed into 4 subspaces, each of which is expected to learn $K = 256$ clusters. Figures 4.4 show the first four feature spaces learned using quadratic loss with Eq. (4.2.3) as manifold preserving term denoted as Quad-R, while Figures 4.5 visualize the first four feature spaces learned with DeepQuan. It can be seen that the features learned with Quad-R are mixed, but the features learned with DeepQuan exhibit a discriminative clustering structure, which will benefit the quantization process.
Table 4.5: Mean Average Precision (mAP) obtained by different combination of $M$ and $K$ on CIFAR-10-CNN. The best mAP is in bold while the second best one is underlined.

<table>
<thead>
<tr>
<th>CIFAR-10-CNN</th>
<th>16 bits</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>different $K$</td>
<td>different $M$</td>
</tr>
<tr>
<td>$K = 256$</td>
<td>$M = 2$</td>
<td>39.95</td>
</tr>
<tr>
<td>$K = 16$</td>
<td>$M = 4$</td>
<td>38.34</td>
</tr>
<tr>
<td>$K = 4$</td>
<td>$M = 8$</td>
<td>37.48</td>
</tr>
</tbody>
</table>

Table 4.6: Mean Average Precision (mAP) obtained by different combination of $M$ and $K$ on MNIST. The best mAP is in bold while the second best one is underlined.

<table>
<thead>
<tr>
<th>MNIST</th>
<th>16 bits</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>different $K$</td>
<td>different $M$</td>
</tr>
<tr>
<td>$K = 256$</td>
<td>$M = 2$</td>
<td>60.30</td>
</tr>
<tr>
<td>$K = 16$</td>
<td>$M = 4$</td>
<td>57.19</td>
</tr>
<tr>
<td>$K = 4$</td>
<td>$M = 8$</td>
<td>39.74</td>
</tr>
</tbody>
</table>
4.3.4 Parameters Analysis

Figure 4.6 shows the effect of changing the values of the hyper-parameters $s$ and $\lambda$. The experiments are conducted for varying one parameter at a time while fixing the other. All the results reported are under 64 bits setting. We can find that DeepQuan can achieve competitive results within the range of $0.001 \leq \lambda \leq 0.2$ and $0.001 \leq s \leq 1.0$ compared with the existing unsupervised hashing methods.

Furthermore, we conduct experiments to show the effectiveness of number of codebook $M$, i.e., the number of decomposition subvector, and the number of codeword $K$. Note that when the number of bit is fixed, the number of codeword $K$ is different for different values of $M$. One requirement for $M$ is that the length of hash codes should be divided by $M$. Take the case when the length of hash codes
is 16 as an example. If $M = 2$, then $K = 256$ since $16 = M \log_2 K = 2 \times \log_2 256$.

But if $M = 4$, then $K = 16$ since $16 = 4 \times \log_2 16$. If $M = 1$, then $K$ should equals to $2^{16}$ which is too large for the benchmark datasets (e.g. CIFAR-10 only contains 60,000 images) used here. To make the size of codebook is suitable for implementation, different combinations of $M$ and $K$ are examined, as showed in Table 4.5 and 4.6. It can be observed that when increasing the number of subvector $M$, the performance decreases consistently. By setting $K = 256$ which is used in our previous experiments, DeepQuan can give the best performance on two datasets.

\section*{4.4 Conclusions}

In this chapter, we propose a novel deep unsupervised method DeepQuan to learn binary codes. In contrast with existing unsupervised hashing methods, the intrinsic data manifold can be better preserved through the integration of product quantization and deep models. Experimental results on two widely used datasets show that our method outperforms the existing state-of-the-art models for image retrieval. An empirical analysis is also conducted to illustrate the effectiveness of the proposed weighted triplet loss in achieving discriminative hidden representations.
Chapter 5

SPDAQ: Similarity Preserving Deep Asymmetric Quantization for Multimedia Retrieval

5.1 Motivation

Most of the existing deep quantization methods use deep convolution network as the backbone for visual representations learning. However, as a large amount of parameters is needed for one convolution network, training deep quantization models given a large-scale database is highly time-consuming. For example, the complexity of triplet training in DTQ [64] is $O(N^3)$, which grows cubically as $N$ increases. To shorten the training time, existing deep quantization models only sample a subset of items from the database for training. Unfortunately, discarding a large portion of the supervision information often leads to poor retrieval performance as the inferred codebooks may be not good enough for quantizing the whole large-scale database.

In this chapter, we propose a novel quantization model named Similarity Preserving Deep Asymmetric Quantization (SPDAQ) to address the aforementioned challenge. The framework of our proposed SPDAQ is shown as in Figure 5.1. The contributions of this chapter are mainly summarized as follows:
Unlike existing deep quantization methods which merely perform metric learning on unquantized embeddings, our model takes the first attempt to adopt Asymmetric Quantizer Distance (AQD) [42] to approximate the predefined metric for optimization. Furthermore, SPDAQ can directly learn quantization codebooks and binary codes for each database item by formulating it as Maximum Inner Product Search (MIPS) [94] problem. To the best of our knowledge, it is the first time to learn directly the quantization codebooks and the explicit binary codes for each database item discretely in a deep learning framework. Also, this end-to-end training model allows that the visual representation can be jointly learned.

By sampling a subset of \( M \) items (i.e., images or videos) as input into the deep convolution network, the complexity of the pairwise training can be reduced to \( O(MN) (M \ll N) \), which allows us to utilize all the available label information efficiently. Specifically, the subset images and database items are mapped to two different but correlated distributions, where the pairwise label similarity is largely preserved by the AQD between the inferred visual representations and the composite quantized representations.

Furthermore, a well-designed alternating optimization algorithm is derived to train the model, which allows us to solve the problem efficiently.

Extensive experiments conducted on four widely-used benchmark datasets show that SPDAQ outperforms the existing deep quantization methods and achieves the state-of-the-art performance for the image retrieval task. Also, the results of experiments conducted on two video datasets further demonstrate the superiority of our model.

5.2 Methodology

In this chapter, we propose a novel model named Similarity Preserving Deep Asymmetric Quantization (SPDAQ) that tries to learn unquantized embeddings for an
subset of items (for the efficiency reason) and at the same time another set of composite quantized embeddings for the whole database so that the Asymmetric Quantizer Distance (AQD) between the unquantized and quantized representations should preserve the image similarity, which is computed with the given semantic labels. Details of the model formulation and the optimization algorithm are presented in the following, where we take image retrieval task as example for convenient discussion.

5.2.1 Notation

We use lowercase letters like \( \mathbf{b} \) to denote a vector while uppercase letters like \( \mathbf{B} \) to denote a matrix. \( \mathbf{B}_{ij} \) denotes the \((i, j)th\) element of matrix \( \mathbf{B} \). \( \mathbf{B}_{ij} \) denotes the \(jth\) column and \( \mathbf{B}_{i*} \) denotes the \(ith\) row of matrix \( \mathbf{B} \) respectively. We use \( \mathbf{B}^T \) to denote the transpose of matrix \( \mathbf{B} \). Furthermore, we will use \( \| \cdot \|_F \), \( \| \cdot \|_2 \) and \( \| \cdot \|_0 \) to denote Frobenius norm, \( L_2 \) norm and \( L_0 \) norm.
Figure 5.1: The framework of the proposed SFDAQ model which make use of a CNN network and composite quantization to learn unquantized and quantized embeddings for training set images and database images respectively.
5.2.2 Model Formulation

Suppose that we are given a large image database with \( N \) items \( \mathbf{X}^{\text{data}} = \{\mathbf{x}_i\}_{i=1}^N \) and their corresponding semantic labels \( \mathbf{Y}^{\text{data}} = \{\mathbf{y}_i\}_{i=1}^N \). As most of the deep learning methods proposed for the learning task [7, 8, 58, 59], we also first sample a small subset of \( M \) items for training \( \mathbf{X}^{\text{train}} = \{\mathbf{x}_i\}_{i=1}^M \) and \( \mathbf{Y}^{\text{train}} = \{\mathbf{y}_i\}_{i=1}^M \). Note that only the subset of images are fed into the deep model for parameter tuning to avoid linearly scanning the whole database, which is very time-consuming. The problem defined here is to learn maximum preserving the similarities between items by learning \( D \) codebooks \( \{\mathbf{C}^i\}_{i=1}^D \) for the construction of look-up tables where each entry in a look-up table is indexed by its corresponding \( D \) binary vectors \( \{\mathbf{b}^i\}_{i=1}^D \). High efficient image retrieval can thus be achieved via the look-up tables. A feature extraction function \( \mathcal{F}(\cdot) \) is also required to be learned during the optimization.

In this method, we adopt the concept of composite quantization following [7, 64] as the basis due to its superiority over other quantization methods and the matrix implementation simplicity. In the scenery of composite quantization, given a vector \( \mathbf{x} \), \( D \) codebooks \( \{\mathbf{C}^i\}_{i=1}^D \) and associated \( D \) binary vectors \( \{\mathbf{b}^i\}_{i=1}^D \) are learned to minimize the objective function

\[
\min \|\mathbf{x} - \sum_{i=1}^D \mathbf{C}^i \mathbf{b}^i\|_2
\]

Suppose that there are \( K \) codewords in each codebook, then \( K^D \) different combinations in total can be obtained to approximate the given \( \mathbf{x} \). Instead of computing \( \mathbf{C}^i \) and \( \mathbf{b}^i \) by minimizing the quantization error like general composition quantization framework, our proposed model directly optimizes codebooks and indicator vectors to obtain the composite quantized vectors when approximating the similarities.

The overall framework of our proposed SPDAQ model is depicted in Figure 5.1. It contains three parts: (i) a deep convolution network to learn the unquantized visual embeddings for the image subset, (ii) a composite quantization part to directly learn the quantization codebooks and the indexes for the database items, and (iii) an objective functions including a classification loss and a similarity preserving loss.
to guide the model training.

**Similarity Matrix Construction**  Inspired by the Maximum Inner Product Search (MIPS), we adopt pairwise training and compute the pairwise similarity matrix $S \in [0, 1]^{M \times N}$ from labels which can be single label or multi labels. For two items $x_i$ and $x_j$ associated with label vectors $y_i$ and $y_j$, we compute their similarity as

$$\text{sim}(x_i, x_j) = \frac{y_i^T y_j}{\|y_i\|_2 \|y_j\|_2} \in [0, 1]$$  (5.2.1)

If $y_i$ and $y_j$ are both single label vectors, $S_{ij}$ gives either 0 or 1. If $y_i$ and $y_j$ are multilabel vectors, $S_{ij}$ will be in the range $[0, 1]$ which gives more fine-grained similarity.

Note that the complexity of similarity matrix $S$ is $O(MN)$, which is unacceptable for large-scale datasets given a large value of $N$ even though $M \ll N$. To address this high complexity issue, we rewrite matrix $S$ as the multiplication of two matrices

$$S = L^{train}(L^{data})^T$$  (5.2.2)

where $L^{train}_{i*} = \frac{y_i^T}{\|y_i\|_2}$ is a $L_2$-norm normalized label vector. A similar normalization trick can be also applied to $Y^{data}$ to get the normalized label matrix $L^{data}$.

**Unquantized Embedding Learning**  Following [59, 58], we adopt the deep network CNN-F [12] as the backbone to learn unquantized visual embeddings for the image subset. CNN-F is a deep convolution network consisting of five convolution layers and three fully-connected layers as AlexNet [52]. The configurations of layers in CNN-F are listed in Table 5.1. We replace the last fully-connected layer with a linear projection which maps the seventh high-dimensional visual features to low-dimensional embeddings.

**Similarity Preserving Loss**  Given a query sample $q$ and a database item $x$, the AQD as in [7] is formulated as

$$AQD(q, x) = (z_q)^T \sum_{i=1}^{D} C^i b^i$$  (5.2.3)
Table 5.1: Configuration of CNN-F deep convolution network.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>kernel = 64×11×11, stride = 4×4, pool = 2×2, LRN</td>
</tr>
<tr>
<td>conv2</td>
<td>kernel = 256×5×5, stride = 1×1, pool = 2×2, LRN</td>
</tr>
<tr>
<td>conv3</td>
<td>kernel = 256×3×3, stride = 1×1</td>
</tr>
<tr>
<td>conv4</td>
<td>kernel = 256×3×3, stride = 1×1</td>
</tr>
<tr>
<td>conv5</td>
<td>kernel = 256×3×3, stride = 1×1, pool = 2×2</td>
</tr>
<tr>
<td>fc6</td>
<td>4096, dropout</td>
</tr>
<tr>
<td>fc7</td>
<td>4096, dropout</td>
</tr>
<tr>
<td>fc8</td>
<td>300</td>
</tr>
</tbody>
</table>

where \( z_q \) is an unquantized embedding of the query sample \( q \). We directly adopt Eq. (5.2.3) to approximate the similarity matrix \( S \) computed with Eq. (5.2.2). Thus, the similarity preserving loss is formulated as a constrained optimization problem, given as

\[
\min_{i=1}^{M} \sum_{j=1}^{N} \left\| \mathcal{F}(x_i)^T z_j - \gamma \cdot S_{ij} \right\|_2^2
\]

\[
\text{s.t. } z_j = \sum_{p=1}^{D} C^p b_j^p, \quad \|\mathcal{F}(x_i)\|_2 = 1
\]

\[
\text{b}_j^p \in \{0, 1\}^K, \quad \|\text{b}_j^p\|_0 = 1, \forall p \in \{1 \ldots D\}
\]

where \( \mathcal{F}(x_i) \in R^L \) is the low-dimensional unquantized embedding obtained through a deep convolution network for raw pixel image \( x_i \) from the subset.

Since the value of inner product is not only affected by the angle between vectors but also their norm scale, we add the unit norm constraint to the unquantized embedding but relax this constraint on database items for model training flexibility. \( \gamma \) is a suitably selected scaling parameter which scales the similarity. We simply set \( \gamma \) as the number of codebooks \( \gamma = D \), the main reason for which is that we expect the unit-norm unquantized embedding \( \mathcal{F}(x_i) \) to be as similar to each codeword selected from \( D \) codebooks as possible, i.e. the inner product between \( \mathcal{F}(x_i) \) and each selected codeword equals to 1 if \( S_{ij} = 1 \) and the unit-norm constraint is added to
each codeword so that their sum equals to $D$. $z_j$ is the embedding for $j^{th}$ item in the database obtained using the approach to compute quantized vector as in composite quantization $\sum_{p=1}^{D} C^p b_j^p$, where $C^p \in R^{L \times K}$ denotes the $p^{th}$ codebook and each codebook contains $K$ codewords. $b_j^p$ is a one-hot vector of $j^{th}$ item associated with $C^p$. The unit-norm constraint is easily satisfied using the internal operator in Tensorflow [1].

The Eq. (5.2.4) can be alternatively written with a penalty function as

$$\min \sum_{i=1}^{M} \sum_{j=1}^{N} \|F(x_i)^T z_j - \gamma \cdot S_{ij} \|^2_2 + \eta \sum_{j=1}^{N} \|z_j - \sum_{p=1}^{D} C^p b_j^p \|^2_2$$

(5.2.5)

The newly introduced parameter $\eta$ is important to balance the similarity approximation and the quantization error, which, however, is time-consuming to tune. Equivalently, we take one more step by replacing $z_j$ with $\sum_{p=1}^{D} C^p b_j^p$ then directly learn the quantized embedding for database items in a discrete way, formulated as

$$\min \sum_{i=1}^{M} \sum_{j=1}^{N} \|F(x_i)^T \sum_{p=1}^{D} C^p b_j^p - \gamma \cdot S_{ij} \|^2_2$$

s.t. $b_j^p \in \{0, 1\}^K$, $\|b_j^p\|_0 = 1, \forall p \in \{1 \ldots D\}$

$$\|F(x_i)\|_2 = 1.$$  

(5.2.6)

Note that the unquantized embeddings for subset samples are obtained from the last layer of the deep convolution network, while the quantized embeddings (i.e. the codebook $C$ and corresponding binary vector $b$) for database items are directly learned by preserving the correlations (i.e. the predefined similarity). By doing so, SPDAQ actually maps subset samples and database items to two different but correlated distributions, which makes the model more flexible and better preserve the semantic similarity. Also, such an asymmetric formulation allows us to efficiently utilize more supervision information, which ensures that the learned quantization codebooks can encode more semantic information, thus leading to better performance in retrieval stage. We will verify this in following experimental part.
The Eq. (5.2.6) can be further rewritten in a matrix form as

\[
\begin{align*}
\min_{Q} & \quad Q = \| \mathcal{F}(X)^T \mathbf{CB} - \gamma \cdot \mathbf{S} \|_F^2 \\
\text{s.t.} & \quad \mathbf{B} \in \{0, 1\}^{DK \times N} \\
& \quad \mathbf{b}_j \in \{0, 1\}^{DK}, \| \mathbf{b}_j \|_0 = 1, \forall p \in \{1 \ldots D\} \\
& \quad \| \mathcal{F}(x_i) \|_2 = 1, \forall i \in \{1 \ldots M\} 
\end{align*}
\]

(5.2.7)

where \( \mathcal{F}(X) \in R^{L \times M} \) is the embedding matrix of subset of \( M \) images obtained through CNN. \( \mathbf{C} \) is the concatenation of \( D \) codebooks \( \mathbf{C} = [\mathbf{C}^1 \ldots \mathbf{C}^D] \in R^{L \times KD} \), and \( \mathbf{B} \) is the concatenation of \( N \) binary vectors \( \mathbf{B} = [\mathbf{b}_1 \ldots \mathbf{b}_N] \in \{0, 1\}^{DK \times N} \) of \( N \) database items, and each \( \mathbf{b}_i \) is the concatenation of \( D \) one-hot vector with dimension \( K \).

**Classification Loss** To learn more discriminative embeddings of the sampled subset images, a classification loss is added

\[
\min \sum_{i=1}^{M} H(\mathcal{F}(x_i), y_i) 
\]

(5.2.8)

where label vector \( y_i \) could be one-hot (i.e. single-label dataset) or \( k \)-hot (i.e. multi-label dataset). \( H \) is a predefined classification loss function. For classifying single-label dataset, we use a softmax loss function as \( H \). For multi-label datasets, binary entropy (i.e. the sigmoid function) is applied to each label to do the multi-class classification.

**Overall Objective** We combine the above two losses and formulate the overall objective as multi-task learning:

\[
\begin{align*}
\min & \quad L(\theta, \mathbf{C}, \mathbf{B}) = Q + \lambda \sum_{i=1}^{M} H(\mathcal{F}(x_i), y_i) \\
& \quad = \| \mathcal{F}(X)^T \mathbf{CB} - \gamma \cdot \mathbf{S} \|_F^2 + \lambda \sum_{i=1}^{M} H(\mathcal{F}(x_i), y_i) \\
\text{s.t.} & \quad \mathbf{B} \in \{0, 1\}^{DK \times N} \\
& \quad \mathbf{b}_j \in \{0, 1\}^{DK}, \| \mathbf{b}_j \|_0 = 1, \forall p \in \{1 \ldots D\} \\
& \quad \| \mathcal{F}(x_i) \|_2 = 1, \forall i \in \{1 \ldots M\} 
\end{align*}
\]

(5.2.9)
where $\lambda$ is a hyper-parameter to balance the classification loss and the similarity preserving loss, which is adaptively updated during training. We use $\theta$ to denote the parameters of deep convolution network.

### 5.2.3 Optimization

We propose an alternating optimization algorithm to minimize the objective function Eq. (overall objective). The algorithm learns the visual embeddings $\mathcal{F}(X)$, the codebooks $C$ and the binary index $B$ alternatively until convergence or reaching the predefined maximum iteration. The algorithm is summarized in Algorithm 1.

**Initialization**  We randomly initialize each codeword in a codebook $C$ and then normalize each codeword to unit length using $L_2$ norm. The binary index $b_j^p$ is randomly initialized to be a one-hot vector. The parameters of first seven layers are pretrained with ImageNet dataset [89], while the projection layer and the classification layer are randomly initialized.

**Learning $\mathcal{F}(X)$ with $C$ and $B$ fixed**  The standard back propagation (BP) algorithm is used to update the model parameters $\theta$ and $\mathcal{F}(X)$. At each time, we sample one batch of items, with which we update the parameters based on the gradient computed. We take the gradient of the objective $L(\theta, C, B)$ with respect to $\theta$ as follows:

$$
\frac{\partial L(\theta, C, B)}{\partial \theta} = 2 \cdot ((CB)^T(CB)\mathcal{F}(x_i) - \gamma CBS_{i\alpha}^T) \frac{\partial \mathcal{F}(x_i)}{\partial \theta} + \lambda \frac{H(\mathcal{F}(x_i), y_i)}{\partial \theta}
$$

(5.2.10)

To optimize the deep model, we back-propagate the gradient $\frac{\partial L(\theta, C, B)}{\partial \theta}$ through the chain rule and update the parameters using the gradient descent method.

**Learning $C$ with $\mathcal{F}(X)$ and $B$ fixed**  To optimize $C$, we first filter out non-related terms in Eq. (5.2.9) and obtain

$$
\min \|\mathcal{F}(X)^T CB - \gamma \cdot S\|_F^2.
$$

(5.2.11)
Note that Eq. (5.2.11) is a convex function, which can be seen by rewriting it linear to \( \mathbf{C} \) as

\[
\|(\mathbf{B}^T \otimes \mathcal{F}(\mathbf{X})^T)\text{vec}(\mathbf{C}) - \gamma \text{vec}(\mathbf{S})\|_2
\]

where \( \text{vec}(\cdot) \) denotes the vectorizing operation and \( \otimes \) is the Kronecker product. The gradient descent method can be used to optimize the above objective function, but it would be costly to compute the Kronecker product for the two large matrices \( \mathcal{F}(\mathbf{X})^T \) and \( \mathbf{B}^T \) and hard to determine the learning rate. In fact, a closed-form solution which minimizing the objective can be obtained (by using the Theorem 2.3.2 in [97]) as

\[
\mathbf{C}_{opt} = (\mathcal{F}(\mathbf{X})^T)^+ \mathbf{S} \mathbf{B}^+ + \mathbf{R} - (\mathcal{F}(\mathbf{X})^T)^+ (\mathcal{F}(\mathbf{X})^T) \mathbf{R} \mathbf{B}^+
\]

where \( \mathbf{B}^+ \) denote the pseudo inverse of matrix \( \mathbf{B} \) and \( \mathbf{R} \) is an arbitrary matrix. However, for the large matrix \( \mathbf{B} \), it is time-consuming to compute its pseudo inverse.

Here, we seek to update matrix \( \mathbf{C} \) in an efficient way by introducing an auxiliary parameter \( \mathbf{Z} \) and deriving a new optimization problem. Let \( \mathbf{Z} = \mathcal{F}(\mathbf{X})^T \mathbf{C} \). We can rewrite Eq. (5.2.11) as a constrained optimization problem, given as

\[
\begin{align*}
\min_{\mathbf{Z}} & \quad \|\mathbf{Z} \mathbf{B} - \gamma \cdot \mathbf{S}\|_F^2 \\
\text{s.t.} & \quad \mathbf{Z} = \mathcal{F}(\mathbf{X})^T \mathbf{C}
\end{align*}
\]

(5.2.12)

which can be further rewritten with a penalty function as

\[
\begin{align*}
\min_{\mathbf{Z}} & \quad \|\mathbf{Z} \mathbf{B} - \gamma \cdot \mathbf{S}\|_F^2 + \mu \|\mathbf{Z} - \mathcal{F}(\mathbf{X})^T \mathbf{C}\|_F^2 \\
\end{align*}
\]

(5.2.13)

where \( \mu \) is a hyper-parameter. Then, we can alternatively update \( \mathbf{Z} \) and \( \mathbf{C} \) it reaches the predefined maximum iterations or less than the predefined threshold. For updating \( \mathbf{C} \), we take the derivate of Eq. (5.2.13) w.r.t \( \mathbf{C} \) as

\[
\mathcal{F}(\mathbf{X}) \mathcal{F}(\mathbf{X})^T \mathbf{C} - \mu \mathcal{F}(\mathbf{X}) \mathbf{Z} = 0
\]

(5.2.14)

The updating formulation for \( \mathbf{C} \) becomes

\[
\mathbf{C} = \mu (\mathcal{F}(\mathbf{X}) \mathcal{F}(\mathbf{X})^T)^{-1} \mathcal{F}(\mathbf{X}) \mathbf{Z}
\]

(5.2.15)
Similarly, the updating formulation for $Z$ will be

$$Z = (\gamma \cdot SB^T + \mathcal{F}(X)^T C)(BB^T + \mu I)^{-1}. \quad (5.2.16)$$

One can see from Eq. (5.2.16) that the second term in Eq. (5.2.13) acts like a smooth regulation when computing the inverse of $BB^T$. Computing $S$ explicitly incurs high complexity. With the factorization in Eq. (5.2.2), we can first compute $G = (L_{data}^T B)^T$ and then compute $SB^T = L_{train}^T G$ to avoid the high complexity problem.

Note that the solution $C$ obtained by optimizing Eq. (5.2.13) may not be the minima to Eq. (5.2.11) due to the error introduced by the second term $k^2_\mathcal{F}$. Denote the $C'$ and $C^{old}$ as the solution to Eq. (5.2.13) and the initial matrix of Eq. (5.2.11) respectively. In theory, one can take the updated $C$ as

$$C = \arg \min_{C \in \{C', C^{old}\}} \|\mathcal{F}(X)^T CB - \gamma \cdot S\|_F^2 \quad (5.2.17)$$

which can ensure that the objective is decreasing, i.e. $L(\theta, C, B) \leq L(\theta, C^{old}, B)$.

In practice, to avoid the time-consuming computation of value of the objective function, we directly set the updated matrix $C$ as $C'$, which we found can still give satisfactory performance. We simply set the parameter $\mu$ in Eq. (5.2.13) as $\mu = 1.0$.

**Learning $B$ with $\mathcal{F}(X)$ and $C$ fixed** We first rewrite the objective by keeping only the parts related to $B$, given as

$$\min \|\mathcal{F}(X)^T CB - S\|_F^2$$

s.t. $b_j \in \{0, 1\}^{DK}$, $B \in \{0, 1\}^{DK \times N}$

$$\|b^p_j\|_0 = 1, \forall p \in \{1 \ldots D\}. \quad (5.2.18)$$

For convenience, we absorb the scale term $\gamma$ into $S$. With the binary constraint on $b$, Eq. (5.2.18) becomes a combination optimization problem. To solve it, greedy local search is usually applied to find an approximated solution.

For the $i^{th}$ item, the binary vector is a concatenation of $D$ binary sub-vector $b_i = [b_i^1 \cdots b_i^D]$. We define its neighbor as $b'_i$ with only one binary sub-vector
different (e.g. \( b^i_j \neq b^j_i \)), and then perform greedy searching algorithm. That means we iteratively update \( \{ b^i_j \}_{j=1}^D \), i.e. update \( b^i_j \) with one-hot constraint while keeping other \( b^k, k \neq j, k = 1, \ldots, D \) fixed.

Note that each item \( b_i \) in \( B \) is independent of each other. Since the size of database \( N \) is typically large, we can optimize \( B \) batch by batch. We first sample a batch of index \( \Omega \in \{ 1 \cdots N \} \). Then we obtain \( B_\Omega \in \{ 0, 1 \}^{DK \times \Omega} \) and \( S_\Omega \in \{ 0, 1 \}^{M \times \Omega} \). Note that \( S_\Omega \) can be computed explicitly since \( |\Omega| \) can be small. In the following, we reuse notations \( B \) and \( S \) to denote the sampled sub-matrix for convenience.

Let \( Z = \mathcal{F}(X)^T C \in R^{M \times DK} \), which is decomposed as \( D \) components \( Z = [Z^1, \cdots, Z^D] \). Similarly, we have \( B = [B^1, \cdots, B^D] \). Eq. (5.2.17) can be rewritten as

\[
\min_k \| \sum_{i=1}^D Z^i B^i - S \|_F^2
\]

(5.2.19)

For the optimization of the \( i^{th} \) component in \( B \), we can rewrite Eq. (5.2.19) as

\[
\min_i \| Z^i B^i - \hat{S} \|_F^2
\]

(5.2.20)

where \( \hat{S} = S - \sum_{k=1, k \neq i}^D Z^k B^k \). Minimizing Eq. (5.2.20) is equivalent to

\[
\min_i Tr((B^i)^T (Z^i)^T Z^i B^i) - Tr(\hat{S}^T Z^i B^i)
\]

(5.2.21)

where \( Tr(\cdot) \) is the trace norm. Note that we keep the one-hot vector constraint \( \| b^o \|_0 = 1 \) during optimization and update \( B \) in Eq. (5.2.21) in a discrete way. Let \( G = diag((Z^i)^T Z^i) - \hat{S}^T Z^i \), where \( diag(\cdot) \) means selecting the diagonal vector. Specially, the “subtraction” here is a broadcasting operation.

We finally can reach the minimum solution of Eq. (5.2.21) by setting the one element in column (e.g. \( b^i_k \)) of \( B^i \) as 1, whose index associates with the index of the minimum value in the corresponding (e.g. \( k \)) row of \( G \).

By iteratively updating through the above three steps, we can draw the following conclusion:
Algorithm 1 Learning algorithm for SPDAQ

Input: Sampled subset of items and labels \( \{X^{tr}, Y^{tr}\} \), labels of database \( Y^{data} \), predefined maximum number of iteration \( \text{Max}_\text{Iter} \), maximum number of batch \( \text{Max}_\text{Batch} \) per iteration

1: Initialize \( C \), \( B \) and the parameters of CNN-F
2: for iter = 1, \ldots, \( \text{Max}_\text{Iter} \) do
3: \hspace{1em} for k = 1, \ldots, \( \text{Max}_\text{Batch} \) do
4: \hspace{2em} Sample a batch of \( \{X^{batch}, Y^{batch}\} \)
5: \hspace{2em} Update \( F(X) \) by optimizing (5.2.9) with gradient descent method
6: \hspace{1em} end for
7: \hspace{1em} Update \( C \) by iterating between (5.2.15) and (5.2.16) until reaching predefined number of iteration
8: \hspace{1em} Update \( B \) according to (5.2.21)
9: end for

Remark Assume that a proper descent step size is taken during optimizing \( F(X) \) to decrease the objective function. If we update \( C \) by Eq. (5.2.17) and \( B \) by Eq. (5.2.21), the objective function Eq. (5.2.9) is monotonically decreasing.

5.2.4 Out-of-Sample Extension

For testing, given a new query point \( x_q \), we first compute the feature representation by \( F(x_q) \) then compute the AQD as formulated in Eq. (5.2.3) between \( x_q \) and the data item \( x_i \) in the image database based on the inferred binary codes \( B \) and codebooks \( C \) for the retrieval.

To compute the AQD efficiently, we first pre-compute the inner product look-up table of size \( 1 \times K \) between the embedding \( z_q \) and codewords in one codebook. So, there will be \( D \times 1 \times K \) look-up tables altogether. With the \( D \) pre-computed look-up tables, the distance \( AQD(q, x) \) can be efficiently computed by summing up the codeword distance directly obtained from look-up tables, where the corresponding entries in a look-up table are indexed by the binary vector \( b \) of \( x \). It is only slightly
more costly than computing the Hamming distance [42].

5.3 Experimental Results on Image Retrieval

We apply the proposed SPDAQ model to a number of publicly available image datasets and compare its performance with a number of state-of-the-art methods.

5.3.1 Experimental Settings

Datasets Four widely used datasets including CIFAR-10 [51], NUS-WIDE-21, NUS-WIDE-81 [16] and MS-COCO [63] are adopted for performance evaluation.

The CIFAR-10 dataset contains 60,000 labeled color images labeled into 10 classes including “airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship” and “truck”. These classes are completely mutually exclusive. Each class contains 6,000 images of size $32 \times 32$. Since it is a single-label dataset, two images are considered to be similar if they share the same label.

The NUS-WIDE-81 dataset is a large-scale dataset containing 269,648 labeled images collected from Flickr in 81 classes. It is a multi-label dataset and each image is annotated with some of the 81 labels. Note that it is challenging because it is highly class-imbalanced, where some labels are associated with tens of thousands of images while some labels with only tens of images. Two images are considered to be similar if they share at least one label. Otherwise, they are dissimilar.

The NUS-WIDE-21 dataset is a subset of NUS-WIDE-81, containing 195,834 images. Images in NUS-WIDE-81 associated with the most-frequent 21 labels are selected. In this way, each label will be associated with at least 5,000 images. Two images are considered to be similar if they share at least one label. Otherwise, they are dissimilar.

The MS-COCO dataset contains 82,783 training images and 40,504 validation images, where each image is labeled by some of the 80 labels. After pruning the images without labels, we obtain 122,218 images in total. This dataset is also a class-
imbalanced dataset as NUS-WIDE-81. Two images are considered to be similar if they share at least one label.

Some images examples and their associated labels are shown in Table 5.2.

For CIFAR-10, we randomly select 1,000 images (100 images per class) to form the testing query set and take the rest 59,000 images as the database as in [64, 59]. Since training with the whole image database is time-consuming for the existing deep quantization methods, we follow the original settings as in their papers and sample a subset of 5,000 images (500 images per class) from the database for training. For NUS-WIDE-21, we adopt the widely-used protocol and randomly sample 2,100 images (100 images per class) as the testing query set while the remaining images as the retrieval database. A subset of 10,500 images (500 images per class) will be further sampled for training. Note that our proposed SPDAQ model also only has the sampled subset of images fed into the CNN for learning the parameters as the existing methods. For the conventional methods COSDISH and LFH, we use the identical training set as that of the deep methods. For fair comparison, we use the deep features extracted from the 7th layer of CNN-F as the inputs for both COSDISH and LFH. For the more challenging datasets MS-COCO and NUS-WIDE-81, we randomly sample 10,000 images as the subset for training and 5,000 images as the testing query set, as in [64]. The remaining images forms the retrieval database.

**Evaluation Protocol and Settings** For performance evaluation, we adopt two widely-used evaluation metrics: mean average precision (mAP) and Top-K precision. The Top-K precision is to measure the accuracy within top returned K samples given a query $x_q$. The mean average precision is specifically computed as follows: given a query $x_q$, we compute its average precision (AP) using
Table 5.2: Image examples of datasets. The third column contains the associated semantic labels.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Examples</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td><img src="image1" alt="Image Examples" /></td>
<td>“automobile”, “dog”, “ruck”</td>
</tr>
<tr>
<td>NUS-WIDE</td>
<td><img src="image2" alt="Image Example" /></td>
<td>“animal”, “grass”, “person”</td>
</tr>
</tbody>
</table>
\[ AP(x_q) = \frac{1}{R_q} \sum_{k=1}^{N} \text{Prec}(k) I(x_k) \]

where \( I(x_k) \) is an indicator function which equals to 1 if sample \( x_k \) is similar to query \( x_q \), otherwise 0. \( R_q \) is the total number of similar samples within \( N \) returned items. \( \text{Prec}(k) \) is the precision in the \( k^{th} \) position of the returned ranking list. The mAP is further computed with

\[ mAP = \frac{1}{Q} \sum_{q=1}^{Q} AP(x_q) \]

where \( Q \) is the number of query samples.

We compare the performance of the proposed SPDAQ model with seven state-of-the-art models hashing models including Column Sampling based Discrete Supervised Hashing (COSDISH) [46], Latent Factor Hashing (LFH) [112], Deep Supervised Hashing with Generative Adversarial Network (DSH-GAN) [86], Deep Asymmetric Pairwise Hashing (DAPH) [90], Deep Supervised Discrete Hashing (DSDH) [58], Deep Pairwise Supervised Hashing (DPSH) [59], Deep Hashing Network (DHN) [114] and four quantization models including Product Quantization Network (PQN) [110], Deep Quantization Network (DQN) [8], Deep Triplet Quantization (DTQ) [64], Deep Visual Semantic Quantization (DVSQ) [7]. All of them are deep models, except COSDISH and LFH.

For the dimension of the inferred embedding, we set \( L = 300 \) for CIFAR-10, NUS-WIDE-81 and MS-COCO as in [7]. For NUS-WIDE-21, we set \( L = 128 \) which works well for our implementation of SPDAQ. The learning rate is fine-tuned in the range of \( [10^{-3}, 10^{-7}] \) for each dataset. For the composite quantization, we set the number of codewords in each codebook as \( K = 256 \). Therefore, each one-hot vector \( \mathbf{b} \) can be encoded using a binary vector with \( \log_2 K \) bits. Suppose there are \( D \) codebooks, the length of the binary code will be \( D \log_2 K = 8D \). We set the number of epochs as 50 for all the datasets.
Table 5.3: Mean Average Precision (%) for different numbers of bits on CIFAR-10 and NUS-WIDE-21 datasets. The best mAPs are shown in bold. DVSQ\(^\ast\) denotes we run the codes provided by the authors to get the results.

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (mAP@All)</th>
<th>NUS-WIDE-21 (mAP@5000)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 bits</td>
<td>24 bits</td>
</tr>
<tr>
<td>DVSQ</td>
<td>80.4</td>
<td>80.8</td>
</tr>
<tr>
<td>DPFQ</td>
<td>78.5</td>
<td>79.2</td>
</tr>
<tr>
<td>DQN</td>
<td>52.7</td>
<td>56.4</td>
</tr>
<tr>
<td>DSDH</td>
<td>65.0</td>
<td>80.1</td>
</tr>
<tr>
<td>DSH-GAN</td>
<td>75.7</td>
<td>83.1</td>
</tr>
<tr>
<td>DAPH</td>
<td>72.7</td>
<td>77.1</td>
</tr>
<tr>
<td>LFH</td>
<td>22.2</td>
<td>33.8</td>
</tr>
<tr>
<td>COUSDISH</td>
<td>33.7</td>
<td>42.3</td>
</tr>
<tr>
<td>SPDAQ</td>
<td>88.4</td>
<td>88.4</td>
</tr>
<tr>
<td>PQN</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DQN</td>
<td>52.7</td>
<td>55.8</td>
</tr>
<tr>
<td>DTQ</td>
<td>79.0</td>
<td>81.8</td>
</tr>
<tr>
<td>DQN</td>
<td>56.4</td>
<td>58.0</td>
</tr>
<tr>
<td>PQN</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DSDH</td>
<td>65.0</td>
<td>80.1</td>
</tr>
<tr>
<td>DSH-GAN</td>
<td>75.7</td>
<td>83.1</td>
</tr>
<tr>
<td>DAPH</td>
<td>72.7</td>
<td>77.1</td>
</tr>
<tr>
<td>LFH</td>
<td>22.2</td>
<td>33.8</td>
</tr>
<tr>
<td>COUSDISH</td>
<td>33.7</td>
<td>42.3</td>
</tr>
<tr>
<td>SPDAQ</td>
<td>88.4</td>
<td>88.4</td>
</tr>
<tr>
<td>PQN</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DQN</td>
<td>52.7</td>
<td>55.8</td>
</tr>
<tr>
<td>DTQ</td>
<td>79.0</td>
<td>81.8</td>
</tr>
<tr>
<td>DQN</td>
<td>56.4</td>
<td>58.0</td>
</tr>
<tr>
<td>PQN</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DSDH</td>
<td>65.0</td>
<td>80.1</td>
</tr>
<tr>
<td>DSH-GAN</td>
<td>75.7</td>
<td>83.1</td>
</tr>
<tr>
<td>DAPH</td>
<td>72.7</td>
<td>77.1</td>
</tr>
<tr>
<td>LFH</td>
<td>22.2</td>
<td>33.8</td>
</tr>
<tr>
<td>COUSDISH</td>
<td>33.7</td>
<td>42.3</td>
</tr>
<tr>
<td>SPDAQ</td>
<td>88.4</td>
<td>88.4</td>
</tr>
<tr>
<td>PQN</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DQN</td>
<td>52.7</td>
<td>55.8</td>
</tr>
<tr>
<td>DTQ</td>
<td>79.0</td>
<td>81.8</td>
</tr>
<tr>
<td>DQN</td>
<td>56.4</td>
<td>58.0</td>
</tr>
<tr>
<td>PQN</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Figure 5.2: Retrieval performance evaluated with Top-K precision curve (@8 bits and @24 bits respectively) on CIFAR-10 datasets. Best view in color.
Figure 5.3: Retrieval performance evaluated with Top-K precision curve (@8 bits and @24 bits respectively) on NUS-WIDE-21 datasets. Best view in color.
5.3.2 Retrieval Accuracy

We compute mAP for CIFAR-10 with respect to the whole database. For NUS-WIDE-21, NUS-WIDE-81 and MS-COCO datasets, we compute mAP@5000 with respect to the top 5,000 returned images as in [58, 64]. All the results of the existing works reported in this chapter are obtained either from their corresponding papers or by running the codes provided by the authors. For the proposed SPDAQ model, we report the average results of five runs.

The performance comparison results on CIFAR-10 and NUS-WIDE-21 in terms of mAP are shown in Table 5.3. We see that the proposed SPDAQ model outperforms the state-of-the-art methods by a large margin, including the deep hashing models and the deep quantization models. In terms of Top-K precision (we set $K = 10,000$), the proposed SPDAQ also significantly outperforms the existing models as shown in Figure 5.2 and 5.3. Regarding the two more challenging datasets NUS-WIDE-81 and MS-COCO, we find that the proposed SPDAQ also achieves the best results (Table 5.4.).

We believe the significant improvement is due to two reasons: (1) By effectively utilizing all the supervision information of the database, SPDAQ directly learns quantization codebooks and binary codes for the database items, so that more semantic information is encoded. (2) SPDAQ tried to map the subset of the data and the database items to two different but correlated distributions via learning, which can better preserve the similarity.

5.3.3 Efficiency for Model Training

We further compare our proposed SPDAQ model with the existing deep models in terms of training efficiency under 8 bits and 16 bits settings. The large-scale NUS-WIDE-21 dataset is used for this evaluation. To show the superiority of our model, we use the whole database as input for all the models we tested so that all the supervised labels can be used, denoted as DPSH-ALL, DSDH-ALL and DVSQ-ALL. The learning curves of the different models in terms of mAP@5000 are shown in
For fair comparison, we run all the models for 50 epochs except that for DVSQ, we run about 30 epochs because it took much longer time. As shown in Figure 5.4, our model converges in around 3.5 hours and gives a much better performance in mAP. For DPSH-ALL and DSDH-ALL, both converge slowly taking more than 10 hours, but with mAP achieving only \( \approx 84\% \) even with the whole database used. For the deep quantization method DVSQ-ALL, it also converges much slower than the proposed SPDAQ model and achieves a much lower precision than ours.

Figure 5.5 further shows mAP performance and training efficiency on different size of training set on the large-scale dataset NUS-WIDE-21. We measure the training efficiency with \( 1.0 / \) (training time). It can be seen that as the size of training size increases, the training efficiency monotonically decreases and the mAP performance monotonically increases. Also interestingly, we observe that on the NUS-WIDE-21 dataset, when number of images per class is closed to 500 (i.e. 10500 training images), the mAP performance converges.

### 5.3.4 Variants of SPDAQ

We also evaluated the performance of some variants of SPDAQ using CIFAR-10 and NUS-WIDE-21. In particular, SPDAQ-nQ learns the embeddings without quantization and can be considered as the precision upper bound for SPDAQ. Also, we implement SPDAQ-LD as defined in Eq. (5.2.5). SPDAQ-LL is another version that learns quantization with only the input image subset, defined as

\[
\min \sum_{i=1}^{M} \sum_{j=1}^{N} \| \mathcal{F}(x_i)^T z_j - \gamma \cdot S_{ij} \|^2 + \beta \sum_{i=1}^{M} \| \mathcal{F}(x_i) - \sum_{p=1}^{D} C^p b_p^i \|^2.
\]

SPDAQ-nC denotes SPDAQ without the classification term. The empirical results are tabulated in Table 5.5.

We observe that: (1) SPDAQ and SPDAQ-LD achieve similar performance as these two models are equivalent in principle. However, SPDAQ-LD has a hyperparameter \( \eta \) (Eq. (5.2.5)) which critically influences the performance but is time-
consuming to tune (e.g., using grid search). This time-consuming parameter tuning process makes SPDAQ-LD undesirable. (2) We find that the performance of SPDAQ-LL severely drops compared to SPDAQ. Basically, the codebooks learned based on only the image subset fail in quantizing well the whole database, which may verify the claim that SPDAQ is able to map the subset items and database items two different distributions. (3) SPDAQ-nQ shows the upper bound of our method. We find that the performance of SPDAQ is approaching that of SPDAQ-nQ as the length of binary codes increases. (4) The comparison between SPDAQ and SPDAQ-nC shows that the classification term can help to learn more discriminative embeddings and thus further improve the retrieval accuracy.

5.4 Experimental Results on Video Retrieval

In this section, we further evaluate our proposed model SPDAQ on the video retrieval task.

5.4.1 Formulation

Video can be considered as a consecutive sequence of frames. Suppose $T$ frames of one video are sampled as input. We make a slight modification so that SPDAQ can be compatible with it. The formulation is then accordingly changed from Eq. (5.2.4) as
\[
\min_{i=1}^{M} \sum_{j=1}^{N} \| \mathbf{v}_{i}^T \mathbf{z}_{j} - \gamma \cdot \mathbf{S}_{ij} \|_2^2 \\
\text{s.t.} \quad \mathbf{z}_{j} = \sum_{p=1}^{D} \mathbf{C}^p \mathbf{b}_j^p, \| \mathbf{v}_{i} \|_2 = 1, \mathbf{v}_{i} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{F}(\mathbf{x}_{it}) \\
\mathbf{b}_j^p \in \{0,1\}^K, \| \mathbf{b}_j^p \|_0 = 1, \forall p \in \{1 \ldots D\}
\]
where $\mathcal{F}(\mathbf{x}_{it})$ is the visual representation of the $t^{th}$ frame of the $i^{th}$ video obtained through a CNN network $\mathcal{F}(\cdot)$. The representation of $i^{th}$ video is computed as the average of features of $T$ frames. A classification loss $\sum_{i=1}^{M} H(\mathcal{F}(\mathbf{x}_{i}), \mathbf{y}_{i})$ as in Eq. (5.2.8) is added to help to learn more discriminative feature.
5.4.2 Experimental Settings and Datasets

Two widely used action recognition datasets including UCF-101 [98] and HMDB-51 [53] are used for the performance evaluation.

The UCF-101 is an action recognition dataset of realistic action videos, which are collected from YouTube. They are classified into 101 action categories such as “Apply Eye Makeup”, “Brushing Teeth”, “Biking”, etc. It has totally 13320 videos, which are originally divided into testing set containing 3783 videos and training set containing 9537 videos. The duration of each video varies from 4 to 16 seconds. Two videos are considered to be similar if they share the same label.

The HMDB-51 is also an action recognition dataset, which contains 6676 videos in 51 action classes such as “brush hair”, “climb”, “climb stairs”, etc. Each class contains about 100 videos, and the duration of most of the videos are from 2 to 5 seconds. Two videos are considered to be similar if they share the same label.

Some video examples of the datasets can be found in Table 5.6.

For the UCF-101 dataset, we directly use the original testing division (3783 videos) as testing query samples and the remaining videos as database. A subset of 3030 videos (30 videos per category) are further sampled as training set for input. For the HMDB-51 dataset, a set of 1000 videos are randomly sampled as testing set and the remaining samples are as database. A subset of 2000 videos are further randomly sampled as for training. The evaluation metric mean average precision (mAP) is adopted as a criteria. We set number of frames $T = 10$ for all the experiments and the $T$ frames are evenly sampled from one video. The learning rate is fine-tuned in the range of $[10^{-4}, 10^{-6}]$ for each dataset.
Figure 5.4: Training time evaluation of SPDAQ and other state-of-the-art deep models on large-scale dataset NUS-WIDE-21. Best view in color.
Figure 5.5: The mAP performance and training efficiency on different size of training set. X-aixs is the number of images per class. Best view in color.
Table 5.4: Mean Average Precision (%) on NUS-WIDE-81 and MS-COCO datasets for different deep models. The best results are in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>NUS-WIDE-81 (mAP@5000)</th>
<th>MS-COCO (mAP@5000)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>24 bits</td>
</tr>
<tr>
<td></td>
<td>24 bits</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>80.5</th>
<th>84.2</th>
<th>85.1</th>
<th>85.1</th>
<th>80.1</th>
<th>84.4</th>
<th>84.5</th>
<th>84.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b i t s</td>
<td>DQN</td>
<td>72.1</td>
<td>72.5</td>
<td>74.7</td>
<td>75.2</td>
<td>76.7</td>
<td>76.7</td>
<td>76.5</td>
</tr>
<tr>
<td>16 bits</td>
<td>DHN</td>
<td>70.2</td>
<td>71.3</td>
<td>71.6</td>
<td>71.3</td>
<td>67.7</td>
<td>69.7</td>
<td>70.1</td>
</tr>
<tr>
<td>24 bits</td>
<td>SPDAQ</td>
<td>66.8</td>
<td>67.7</td>
<td>69.7</td>
<td>70.2</td>
<td>65.3</td>
<td>66.6</td>
<td>68.5</td>
</tr>
<tr>
<td>32 bits</td>
<td>60.7</td>
<td>65.3</td>
<td>66.6</td>
<td>70.1</td>
<td>70.4</td>
<td>71.2</td>
<td>71.7</td>
<td>72.0</td>
</tr>
</tbody>
</table>
Table 5.5: Mean Average Precision (%) for variants of the proposed SPDAQ model. The best results are in bold, while the second best ones are underlined.

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 (mAP @ 5000)</th>
<th>NUS-WIDE-21 (mAP @ 5000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b i t s</td>
<td>68.3%</td>
<td>93.1%</td>
</tr>
<tr>
<td>24 bits</td>
<td>73.1%</td>
<td>93.1%</td>
</tr>
<tr>
<td>32 bits</td>
<td>88.6%</td>
<td>93.1%</td>
</tr>
<tr>
<td>48 bits</td>
<td>88.6%</td>
<td>93.1%</td>
</tr>
<tr>
<td>SPDAQ-LD</td>
<td>88.8%</td>
<td>94.1%</td>
</tr>
<tr>
<td>SPDAQ-LL</td>
<td>93.0%</td>
<td>94.1%</td>
</tr>
<tr>
<td>SPDAQ-nQ</td>
<td>93.3%</td>
<td>94.1%</td>
</tr>
<tr>
<td>SPDAQ-nC</td>
<td>88.3%</td>
<td>94.1%</td>
</tr>
<tr>
<td>SPDAQ</td>
<td>88.4%</td>
<td>94.1%</td>
</tr>
</tbody>
</table>
Table 5.6: Video examples (consecutive frames) of datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Examples</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCF-101</td>
<td><img src="image1" alt="Examples" /></td>
<td>“cliff diving”</td>
</tr>
<tr>
<td></td>
<td><img src="image2" alt="Examples" /></td>
<td>“haircut”</td>
</tr>
<tr>
<td></td>
<td><img src="image3" alt="Examples" /></td>
<td>“pizza tossing”</td>
</tr>
<tr>
<td>HMDB-51</td>
<td><img src="image4" alt="Examples" /></td>
<td>“push”</td>
</tr>
<tr>
<td></td>
<td><img src="image5" alt="Examples" /></td>
<td>“handstand”</td>
</tr>
<tr>
<td></td>
<td><img src="image6" alt="Examples" /></td>
<td>“hug”</td>
</tr>
</tbody>
</table>

Figure 5.6: Training time comparison between SPDAQ and the state-of-the-art deep quantization model DTQ on video dataset UCF-101. Best view in color.
We compare our model on video retrieval task with several state-of-the-art models including Deep Triplet Quantization (DTQ) [64], Deep Visual Semantic Quantization (DVSQ) [7], Column Sampling based Discrete Supervised Hashing (COSDISH) [46], Latent Factor Hashing (LFH) [112] and Deep Hashing Network (DHN) [114]. All the deep model are originally designed for image retrieval task. We make the same modification so that they are compatible with the input of consecutive frames of videos. The input features for the non-deep models (i.e., COSDISH and LFH) are computed by averaging the deep CNN-F features of $T$ input frames. Note that the text associating one semantic label is generally a short phrase, such as “pizza tossing” and “brushing teeth”. To compute the final semantic text embedding $\mathbf{p}$ associated with one label for DVSQ, we apply the Generalized Max Pooling [80] to aggregate the word2vec embedding $\mathbf{e}_i$ of each word in one phrase by $\min_{\mathbf{p}} \sum_i (\mathbf{e}_i^T \mathbf{p} - 1)^2 + \eta \lVert \mathbf{p} \rVert_2$, where we set $\eta = 0.001$ for all the experiments.

### 5.4.3 Experimental Results

We compute the mAP for UCF-101 and HMDB-51 with respect to the whole dataset. All the results of the existing models reported in this chapter are obtained by running the codes provided by the authors. For the proposed SPDAQ model, we report the average results of five runs where the number of epoch is set as 100.

From Table 5.7, we can see that our proposed method outperforms the existing models by a large margin in terms of the mAP@All criteria. For example, our model outperforms the second best method DTQ by 4% on UCF-101 dataset and by 7% on HMDB-51 dataset respectively when the length of bit equals to 64.

Furthermore, we show that our model is much faster than the compared state-of-the-art deep quantization model DTQ in terms of the training efficiency even though we have the same size of training subset. The main reason is that the number of input for our SPDAQ model is linear to the size of training subset (i.e. $\mathcal{O}(M)$), while DTQ tends to generate cubic number of triplets (i.e. $\mathcal{O}(M^3)$) for input. We verify it by conducting evaluation with Nvidia Tesla K80 Dual GPU Module.
run SPDAQ model 100 epochs where we observe convergence, while the DTQ model is only run for 50 epochs because it took much longer time. From the comparison in Figure 5.6, we can see that SPDAQ took about 9 hours while DTQ took much longer time, about 20 hours to complete. Thus, our model is more efficient in training procedure.

5.5 Conclusions

We propose the Similarity Preserving Deep Asymmetric Quantization (SPDAQ) model for fast multimedia retrieval task to address the high complexity problem for training the deep model with large-scale datasets. In contrast with the existing deep quantization methods, SPDAQ learns the quantization codebooks and the binary codes directly for the database items. By effectively mapping the subset and the database items into two different but correlated distributions, similarity can be better preserved. A comprehensive empirical study is conducted for performance evaluation in both image retrieval and video retrieval tasks. Experimental results based on six benchmark datasets including image datasets and video datasets show that our method outperforms the existing state-of-the-art models for multimedia retrieval in terms of both accuracy and training efficiency.
Table 5.7: Mean Average Precision (%) for different numbers of bits on UCF-101 and HMDB-51 datasets. The best mAPs are shown in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>UCF-101 (mAP@All)</th>
<th>HMDB-51 (mAP@All)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method</td>
<td></td>
</tr>
<tr>
<td>COSDISH</td>
<td>24.7</td>
<td>36.0</td>
</tr>
<tr>
<td>LFH</td>
<td>17.0</td>
<td>17.0</td>
</tr>
<tr>
<td>DHN</td>
<td>0.0</td>
<td>7.0</td>
</tr>
<tr>
<td>LFH</td>
<td>22.8</td>
<td>20.8</td>
</tr>
<tr>
<td>COSDISH</td>
<td>43.2</td>
<td>44.3</td>
</tr>
<tr>
<td>Method</td>
<td>45.4</td>
<td>46.3</td>
</tr>
<tr>
<td></td>
<td>46.0</td>
<td>46.9</td>
</tr>
<tr>
<td>DTQ</td>
<td>32.1</td>
<td>31.3</td>
</tr>
<tr>
<td>DVSQ</td>
<td>22.8</td>
<td>22.3</td>
</tr>
<tr>
<td>SPDAQ</td>
<td>59.1</td>
<td>59.9</td>
</tr>
</tbody>
</table>

UCF-101 and HMDB-51 datasets. The best mAPs are shown in bold.
Chapter 6

Conclusions and Future Works

In this thesis, I present three different models to learn hash codes for multimedia retrieval, namely APHash, DeepQuan and SPDAQ. APHash is proposed to learn the hash codes in an unsupervised learning setting where a large amount of unlabeled data can be efficiently leveraged to achieve highly accurate retrieval performance. DeepQuan, to the best of our knowledge, is the first attempt to integrate quantization models into a deep unsupervised model as a unified framework, leading to more fine-grained information preservation for better retrieval. SPDAQ is a supervised learning method proposed to address the time-consuming training problem of a deep model, especially for the situations where a deep convolutional neural network with a large amount of parameters is to be trained given a large multimedia dataset.

All these three proposed hashing models have been evaluated with extensive experiments on a number of publicly available datasets. Our experimental results demonstrate their superiority over the corresponding state-of-the-art methods for multimedia retrieval.

This thesis research study can be further extended in at least the following two directions:

Disentangled Hash Bits  Existing learning-to-hash methods in fact learn the hash function to map items from a high-dimensional space to a low-dimensional
space, which however does not consider the meaning of each bit or groups of bits within a hash code vector. Inspired by some recent progress in learning disentangled representations [5, 72], it could be possible to learn meaningful hash bits, which means that each hash bit could relate to one specific object or feature of an image. For example, suppose the first bit of hash code “0010” represents whether a cat shows in an image, where 0 means there is no cat in an image. If there exists hash code “1010”, we can not only tell that its Hamming distance to “0010” is 1 but also know that there is a cat shown in the image associated with hash code “1010”. From this perspective, it seems to us that being able to learn disentangling hash bits can make important contributions to the field.

**Developing Optimization Method by using Combinatorial Structure of Hashing Learning Objective** If we keep the binary constraint on hash codes during optimization, the problem will become a combinatorial optimization problem, which is extremely hard and normally takes exponential time to solve. Although some existing works such Supervised Discrete Hashing (SDH) [91] and Discrete Graph Hashing (DGH) [69] claim that their methods optimize the objective function using discrete methods, they actually do the optimization in a kind of continuous way. A more fundamental question is: is it possible to design an objective function that has special combinatorial structure such as matroid [85] or special property such as submodularity [49]? Note that submodularity in discrete optimization is analogy to convexity in continuous optimization problem. With the submodular property, one can optimize the problem simply with greedy algorithm which is much faster. One closed related work may be the Submodular Dictionary Selection (SDS) [50], which learns spare representation with submodularity. However, learning to hash with submodularity could be a harder problem, and it would be interesting to further investigate along this direction.
Bibliography


102


CURRICULUM VITAE

Academic qualification of the thesis author, Mr. CHEN Junjie:

• Received the degree of Bachelor of Engineering from Jinan University, June 2016.

August 2019