2020

Deep Hashing for Image Similarity Search

Ali Al Kobaisi
University of Central Florida

DEEP HASHING FOR IMAGE SIMILARITY SEARCH

by

ALI AL KOBAISI
MSc Computer Engineering, University of Central Florida, 2019
MSc Computer Engineering, Amirkabir University of Technology, Tehran, 2000
BSc Electrical Engineering, Amirkabir University of Technology, Tehran, 1996

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2020

Major Professor: Pawel Wocjan

ABSTRACT

Hashing for similarity search is one of the most widely used methods to solve the approximate nearest neighbor search problem. In this method, one first maps data items from a real valued high-dimensional space to a suitable low dimensional binary code space and then performs the approximate nearest neighbor search in this code space instead. This is beneficial because the search in the code space can be solved more efficiently in terms of runtime complexity and storage consumption. Obviously, for this method to succeed, it is necessary that similar data items be mapped to binary code words that have small Hamming distance.

For real-world data such as images, one usually proceeds as follows. For each data item, a pre-processing algorithm removes noise and insignificant information and extracts important discriminating information to generate a feature vector that captures the important semantic content. Next, a vector hash function maps this real valued feature vector to a binary code word. It is also possible to use the raw feature vectors afterwards to further process the search result candidates produced by binary hash codes.

In this dissertation we focus on the following. First, developing a learning based counterpart for the MinHash hashing algorithm. Second, presenting a new unsupervised hashing method UmapHash to map the neighborhood relations of data items from the feature vector space to the binary hash code space. Finally, an application of the aforementioned hashing methods for rapid face image recognition.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Pawel Wocjan for his continuous support of my Ph.D study and research. I am deeply indebted to him for his guidance in all the time of research and writing of this dissertation. Also, I would like to thank my thesis committee: Dr. Kalpathy Sundaram, Dr. Mingjie Lin, Dr. Damian Dechev, and Dr. Qun Zhou for their comments and encouragement. Finally, I would like to thank my family for supporting me throughout my graduate life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Nearest Neighbor Search Problem

The nearest neighbor search problem (NNS) is the problem of finding an item that is the nearest (closest, or most similar) to a given item called the query item from a reference dataset. In the context of information retrieval, this dataset is called the *reference database* or the *gallery*. This problem appears in the literature under different names such as similarity search, search for similar data samples, proximity search, and close item search. In addition to be of interest by itself, this problem is a primitive for other problems in machine learning. For example, non-parametric k-NN classification algorithm is an application of this problem when the question is to find k nearest neighbors of query item. There are few different version of this problem which are defined formally in the following, see figure 1.1.

Definition 1.1 (Nearest neighbor (NN)). In this problem, given a query data point $\mathbf{q} \in \mathbb{R}^d$ and a dataset of N data points $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$, the question is to find the closest point $\mathbf{x}^\star = \underset{\mathbf{x}}{\operatorname{argmin}} \operatorname{distance}(\mathbf{x}, \mathbf{q})$ to the query point $\mathbf{q}$ according to some distance measure.

Definition 1.2 (Fixed-radius near (R-near) neighbor). In this problem, given a query data point $\mathbf{q} \in \mathbb{R}^d$ and a dataset of N data points $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$, the question is to find all points that are within the distance C of the query point $\mathbf{q}$, i.e, $\{\mathbf{x} \,|\, \operatorname{distance}(\mathbf{x}, \mathbf{q}) \leq \mathrm{C}\}$.

In low dimensional spaces, the nearest neighbor search problem has some computationally tractable algorithms. For instance, tree based indexing methods such as KD-tree [4] and ball tree [47] which are based on recursively structured partitioning of the data space. Nevertheless, because of the time complexity and storage requirements of these algorithms, the curse of dimensionality renders these algorithms impractical for high dimensional data space. For this reason, there is a great interest in

the relaxation of this problem called *approximate nearest neighbor* search (ANN).

Definition 1.3 ($(1+\epsilon)$ approximate nearest neighbor). In the $(1+\epsilon)$ approximate nearest neighbor search, given a query data point $\mathbf{q} \in \mathbb{R}^d$ and a dataset of N data points $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$, it suffices to find a data point $\hat{\mathbf{x}} \in \mathbf{X}$ such that $\text{distance}(\hat{\mathbf{x}}, \mathbf{q}) \leq (1+\epsilon)\,\text{distance}(\mathbf{x}^\star, \mathbf{q})$, where $x^\star$ is the true nearest neighbor with respect to the given distance function $\text{distance}(\cdot)$, and $\epsilon > 0$.

Definition 1.4 (c-approximate R-near (cR-near) neighbor). The cR-near neighbor search problem aims to find some item x, called cR-near neighbor, so that $\text{distance}(\mathbf{x}, \mathbf{q}) \leq \mathbf{cR}$.

Definition 1.5 (Randomized approximate nearest neighbor). The randomized search problem aims to report the (approximate) nearest (or near) neighbors with probability instead of deterministically. Given a query $\mathbf{q}$, the goal is to report some near (nearest) neighbor of the query with probability $1-\delta$, and $0 < \delta < 1$.

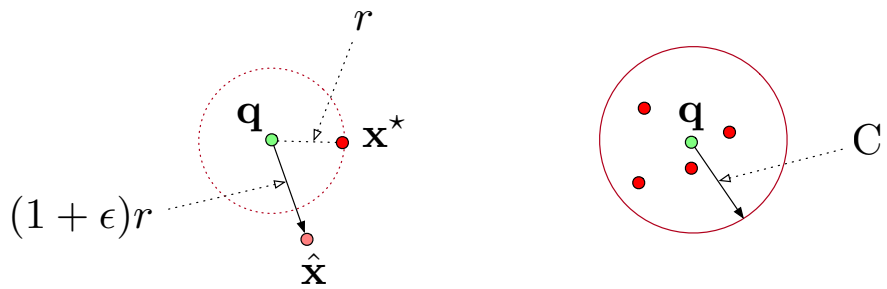

Figure 1.1: Left: $\mathbf{x}^\star$ is the nearest neighbor to the query point q, and $\hat{\mathbf{x}}$ is $(1+\epsilon)$ approximate nearest neighbor. Right: All point inside the circle are R-near neighbors of the query point q

The problem of approximate nearest neighbor search plays an important role in information retrieval systems [40, 54]. For such a practical application, it often suffices to retrieve approximate nearest neighbors of a query, that is, data items in the database that are sufficiently similar.

1.2 Hashing for Similarity Search

Definition 1.6 (Hash function). A hash function is a function that maps an arbitrary size of data to an integer value in the interval of $[0, m]$.

Hashing is a widely used approach for efficiently solving the approximate nearest neighbor search problem. In addition to the similarity search applications, hashing is used to serve in solving other problems. In digital security applications the goal for a hashing function is to generate a digital signature for an arbitrary size of data such that changing to this data gives a different digital signature. Another widely used application of hashing functions is indexing that data items. In this application, the hashing function gives a value in a range of integer numbers $[0, m]$. This value is then used as an index to specify the bucket used to store the data item in a data structure called lookup table. The goal here is to reduce the collisions and have a uniform distribution for data items in buckets.

The focus of this thesis is on the application of hashing function for similarity search.

Definition 1.7 (Hashing for similarity search). A hashing function such that the Hamming distance between the binary representation of hash values approximate the similarity between data items (a distance function of data items).

In this method, initially, we generate a short binary hash code for each item in the database such that the Hamming distance between a pair of hash codes indicates the distance of their corresponding items in the input vector space. The hash codes are generated by applying a suitable vector hash function $\mathbf{h}(\cdot) = \{h_1(\cdot), h_2(\cdot), \cdots, h_L(\cdot)\}$. The functions $h_i(\cdot)$ are usually binary-valued but in general the range of this functions may be a subset of integer numbers. None that in the literature, the term hash function is used for both the compound vector hash function $\mathbf{h}(\cdot)$ and also to refer to its scalar elements $h_i(\cdot)$ as well. We generate the hash code for the query item using the same

hash function and employ the hash codes to compute the Hamming distance as an approximation of the true distance between the query to the database items, i.e. as a measure of similarity between items. Note that the hash value can be used a key in a look-up table, as shown in figure 1.2, which results in a constant query time $O(1)$.

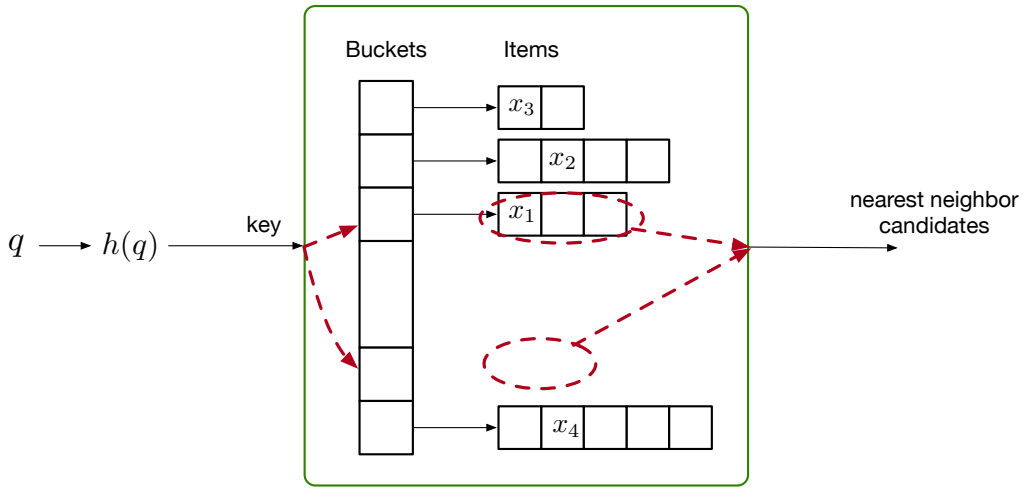

Figure 1.2: Hashing functions is used to generate a set of candidates for nearest neighbor search. For example, the union of point in the buckets which their key Hamming distance to the query point key is less or equal to one are considered as potential nearest neighbor candidates.

On the other hand, since the binary hash codes are short strings compared to the high-dimensional real-valued data items, using the Hamming distance between low dimensional hash codes as a proxy for the similarity between high dimensional items reduces the cost of distance computation and improves information retrieval algorithm both in time and space. The size of the data structures that is needed to be processed during search is reduced, and the number of operations needed to calculate the similarity between items is decreased. Figure 1.3 shows how to find a set of nearest neighbor candidates by using the Hamming distance instead of the input space distance. Note that the Hamming distance can be computed efficiently with logical xor operation between binary hash code codes, i.e. for two hash codes h_1 and h_2 the Hamming distance is $\mathrm{d_H}\left(\mathrm{h}_1, \mathrm{h}_2\right) = \mathrm{h}_1 \oplus \mathrm{h}_2$.

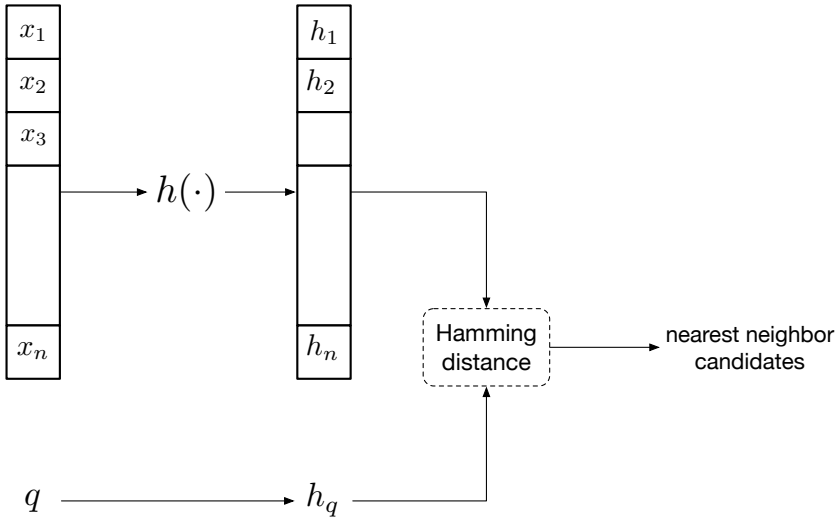

Figure 1.3: When doing an exhaustive search, employing hashing functions reduces the space and time complexity of the algorithm. Instead of computing the distance in the input space, the Hamming distance between the hash codes can be used as a proxy to this distance.

1.3 Feature Generation

Another aspect of similarity search problem is that for complex data domains such as images, any distance that is directly defined on the raw data items does not provide a good measure of their semantic similarity. Therefore, the procedure of hashing images is usually divided into two stages. In the first stage, images are encoded into a useful feature representation so that the semantic information of input is preserved and its irrelevant information is neglected. The second stage is the hash function and maps the feature vectors to compact hash codes. This mapping aims to preserve the similarity between the input feature vectors in the generated hash codes.

For image data, the feature representations are usually either hand-crafted visual descriptors such as GIST [46], and HOG [12], (SIFT) [39], or generated by a deep convolutional neural network. It is well known that deep feed-forward networks learn abstract representations of data items. For instance, deep convolutional networks learn a hierarchy of semantic representations for images.

While the lower layers specialize to detect features such as edges or simple shapes, deeper layers learn semantically higher level features that are responsible for detecting objects in images [68]. Therefore, the distance between the higher-level feature vectors can serve a measure of similarity of the images. Most recent hashing algorithms for image similarity use some combination of dimensionality reduction and quantization to generate the hash codes from the higher-level feature vectors. Note that when both the feature extraction algorithm and hash function are learning based models, they can be trained as a whole model end-to-end which usually improves the overall performance.

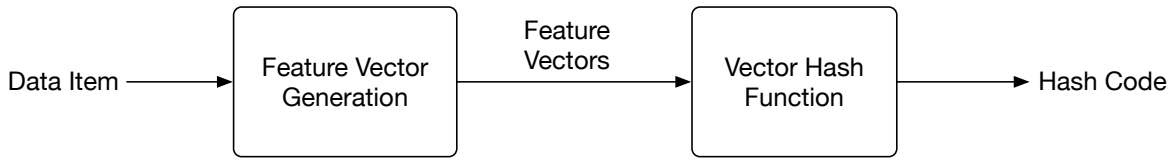

Figure 1.4: For complex data domains, the raw distance between input data points is not a good measure of similarity, and usually a feature generation system is used prior to application of distance function and/or hashing function.

1.4 Overview

In this thesis, we first present a novel learning based hash function, MaxHash [2], that is based on the MinHash [6] algorithm. We show that when integrated with a deep convolutionl neural network for feature extraction, the resultant model achieves state-of-the-art performance for image similarity search. The entire model is trained end-to-end, supervised by pairwise label similarity.

The second contribution is a new loss function that can be employed for unsupervised training of hash functions. Minimizing this loss function drives the model to preserve the topological structure of data items. This method is inspired by UMAP visualization algorithm. Since this visualization algorithm considers only a small neighborhood for each data point it has low memory and compute

footprints and can be used for large datasets. Moreover, it gives well separated clusters in the target binary space. Observe that our method trains the neural network that realizes the hash function. This is different from the visualization objective, which only seeks to obtain the points in the low-dimensional target space, and not to construct the mapping itself.

We applied this loss function to train the MaxHash model in unsupervised setting. The performance of this method surpasses or competes with the state-of-the-art unsupervised hashing methods.

The rest of this thesis is organized as follow. Chapter 2 presents an overview of the most popular data independent, and learning based hashing algorithms. The focus of this chapter is on algorithms that are closely related to our work. In chapter 3 we will present the MaxHash hashing algorithm. Chapter 4 presents the unsupervised UMAPHash hashing method.

CHAPTER 2: LITERATURE REVIEW

In this chapter we will see an overview of some common hashing algorithms for similarity search, with the emphasis on the most important algorithms that are closely related to our work. Hashing algorithms can be divided into two categories: data independent and data dependent. Data independent hashing algorithms in general utilize a randomized mapping. Each component of this mapping should map two data points which are close to each other to the same value with higher probability. Noticeable examples of this category are Locality-Sensitive Hashing (LSH) [14] and MinHash algorithm [5, 6]. Because of the importance of LSH and its central role in hashing for similarity search, we will describe this algorithm and some of its variants in more details. Also, we will see an overview on MinHash algorithm which is the base for our first contribution.

The second category of hashing algorithms employ data sets to design hash functions. The algorithms in this category are called with different names such as data dependent, learning based, and learn to hash algorithms. Learning based hashing algorithms can be divided into two subcategories; unsupervised methods and supervised methods. In unsupervised methods, such as Kernelized Locality-Sensitive Hashing [30], Iterative Quantization [15], and Semantic Hashing [49], the algorithm uses an unlabeled dataset to learn the hash function. An advantage of these methods is that, it is often easier to obtain unlabeled data, however they can only explore the distribution of data points without incorporating the information that the labels of data points may provide.

Finally, supervised hashing methods try to benefit from supervised information such as class labels or the similarity labels of the data points in the dataset. Using the supervised information these algorithms learn a better hash function for that specific data domain, and this results in a better performance compared to other categories for the same hash code length. For example, Supervised Hashing with Kernels [37] is a kernel-based supervised hashing that minimizes/maximizes

the Hamming distances between hash codes for similar/dissimilar data points. Binary Reconstruction Embedding [29] learns hash functions by minimizing the reconstruction error between the distances of points in the dataset and the Hamming distances of the corresponding binary hash codes.

2.1 Locality Sensitive Hashing

Locality sensitive hashing is a randomized hashing method for probabilistic approximate nearest neighbor search problem in high dimensional data space. In this method, a family of hash functions $\mathcal{H}$ is defined that map similar data items to the same hash code with a higher probability than dissimilar data items. The family of of hash functions $\mathcal{H}$ is called (R, cR, P_1, P_2)-sensitive, where $c > 1$ and $P_1 > P_2$, if for two data points $\mathbf{x}_1$ and $\mathbf{x}_2$, the probability that the hashing function $h(\cdot)$ give the same hash value for these two data points satisfy

- if $\mathrm{distance}(\mathbf{x}_1, \mathbf{x}_2) \leqslant R$ then $\mathrm{Prob}[h(\mathbf{x}_1) = h(\mathbf{x}_2)] \geqslant P_1$
- if $\mathrm{distance}(\mathbf{x}_1, \mathbf{x}_2) \geqslant cR$ then $\mathrm{Prob}[h(\mathbf{x}_1) = h(\mathbf{x}_2)] \leqslant P_2$

Note that here we aim to maximize the probability of collision for two data points if they are close to each other, and minimize the probability of collision for two data points if they are far from each other, so that when using hash tables for storing data items every bucket will contain similar items. Thus, given a query $\mathbf{q}$, we will expect the data points lying in the bucket $h(\mathbf{q})$ be similar data points.

To make the gap between the high probability P_1 and the low probability P_2 larger, the LSH algorithm uses multiple hashing functions $h(\cdot)$ which are chosen at random uniformly and independently from the function family $\mathcal{H}$. The resulting hash value for a data item $\mathbf{x}$ then will be

the output of the compound vector hash function $\mathbf{h}(\cdot)$ which is the concatenation of all L hash functions, i.e. $(h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_L(\mathbf{x}))$, where $h_i(\mathbf{x}) \in \mathcal{H}$.

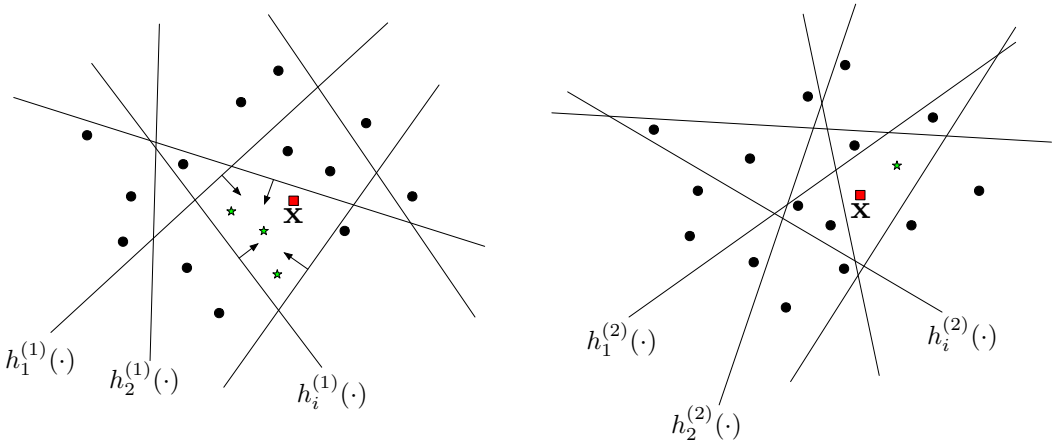

Figure 2.1: Left: If two data points are on the same side with respect to a hyperplane they are likely to be close to each other. If they are on the same side for more hyperplanes this probability increases. Right: Using another locality sensitive set of hash functions to retrieve another set of items. The union of items in the left cell (bucket) and the right cell are returned as similar items.

Figure 2.1 (Left) depicts this hashing method for the case when the hashing function family is the set of all hyper-planes in d dimensional space, where d is the dimension of input data items. Every hyper-plane partitions the data space into two parts. The hash value for every hash function $h_i(\cdot)$ is either 0 or 1 depending on the side of the data point with respect to the corresponding hyper-plane. Therefore, if the distance between two data points is small, it is more probable that they will reside on the same side for more hash functions, and therefore have the same hash value elements. Therefore two close data points share more equal bits which means the Hamming distance between their hash codes is small.

However, increasing the number of hash functions will reduce the probability that two data points have the same hash value, i.e reside in the same bucket in the lookup table. This means that increasing the hash code length decreases recall. For example if the number of examples in the

database is 1 million items and we use a hash code of length 32 then on average each bucket have $1000000/(2^{32}) \approx 0.0002$. Thus, the probability of finding a similar item that has the same hash code is very low. This effect can be seen in figure 2.1, when the number of cutting edges increased, the continues cells become smaller and smaller, and the probability that these cells contain any data point will decrease. So, given a query item, the probability of retrieving any items will decrease, and this means that the recall value for the the similar item search using the LSH hashing function is decreased.

One solution to increase the recall is to use K compound hashing functions $\mathbf{h}(\cdot)$ and K corresponding look up tables. For item retrieval, the hash value of query item is used as index for all K look up tables, and the union of the K buckets that have that index value is returned. This solution can be seen in figure 2.1 (Right). Here instead of one set of partitioning hyper-planes, we use $K = 2$ different sets. The similar items for the query item $\mathbf{x}$ is the union of sets of the data points shown by $\star$. Clearly, using more sets of hashing function will increase the probability that for a query item the more close neighbors are retrieved. However, this increases the time complexity and storage requirement of the algorithm.

Another way to improve the recall is to visit more buckets in the hash table. So instead of checking the bucket that has the exact same hash code of the query item, all the bucket whose corresponding hash codes are close (with respect to the distances in the hash code space) to query item hash code $\mathbf{h}(q)$ are investigated. Usually, only hash codes that have distance one from $\mathbf{h}(q)$ are used. Thus for a hash code of length 32, instead of retrieving only the elements is the bucket whose hash code is $\mathbf{h}(q)$, all the items in the buckets whose hash code are generated by flipping one bit in $\mathbf{h}(q)$ are also retrieved.

There are different kinds of LSH families for different distances or similarities. In genera in all locality sensitive hashing methods the probability of collision between hash codes for two data

items can be written as:

$$\text{Prob}[h(\mathbf{x}_1) \neq h(\mathbf{x}_2)] = \text{norm\_dist}(\mathbf{x}_1, \mathbf{x}_2) \tag{2.1}$$

where $\text{norm\_dist}(\cdot, \cdot) \in [0, 1]$ is a normalized distance function defined over pairs of items defined using a similarity function such as Jaccard coefficient or a distance function such as angular (cosine) distance, l_p distance, Hamming distance and so on. The locality sensitive hash function then turns the data points to hash codes such that the Hamming distance in the code space are related to the distances in the input space. We have:

$$\begin{aligned}
\text{E}[d_H(\mathbf{h}(\mathbf{x}_1), \mathbf{h}(\mathbf{x}_2))] &= \text{E}\left[\frac{1}{L}\sum_1^L \|h - h\|\right] && (2.2)\\
&= \frac{1}{L}\sum_1^L \text{E}\left[\|h - h\|\right] && (2.3)\\
&= \frac{1}{L}\sum_1^L \text{Prob}[h(\mathbf{x}_1) \neq h(\mathbf{x}_2)] && (2.4)\\
&= \text{Prob}[h(\mathbf{x}_1) \neq h(\mathbf{x}_2)] && (2.5)\\
&= \text{norm_dist}(\mathbf{x}_1, \mathbf{x}_2) && (2.6)
\end{aligned}$$

Therefore, given a hashing function that for a pair of data points gives different hash codes with probability equal to the normalized distance between data points the expected value of normalized Hamming distance equal between the corresponding hash codes equal to the normalized distance in the input space. Some examples of locality sensitive hashing families and the distance metric that they approximate are gives in the next sections.

2.1.1 Locality-Sensitive Hashing for Angular Similarity

When the metric for the similarity (distance) in the input space is the angle between data points, i.e. for two data points $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\text{distance}(\mathbf{x}_1, \mathbf{x}_2) = \theta(\mathbf{x}_1, \mathbf{x}_2) = \arccos\left(\frac{\mathbf{x}_1^T \mathbf{x}_2}{\|\mathbf{x}_1\|_2 \, \|\mathbf{x}_2\|_2}\right) \tag{2.7}$$

the hash function family $\mathcal{H}$ for estimating this distance by Hamming distance has the form $h(\mathbf{x}) = \text{sign}((w)^T \mathbf{x})$, where $\mathbf{w}$ is sampled from Gaussian distribution $\mathcal{N}(0, \mathrm{I})$ [8].

In a d-dimensional data space, let w be a random vector sampled from the normal distribution $\mathcal{N}(0, \mathrm{I})$, and $\mathbf{x}_1$ and $\mathbf{x}_2$ two data points, when the hash function is defined as above it can be proven that [8]:

$$\text{Prob}\left(h(\mathbf{x}_1) = h(\mathbf{x}_2)\right) = 1 - \frac{\theta\left(\mathbf{x}_1, \mathbf{x}_2\right)}{\pi} \tag{2.8}$$

This property reveals the relation between Hamming distance and angular similarity, and shows that Hamming distance of LSH algorithm with this hashing family is an unbiased estimate of angular similarity for data items. However this LSH hashing family suffers from the large variance of its estimation, which leads to large estimation error. The variance of estimating the angular distance using the Hamming distance for a hash code of length L, is [25]:

$$\text{Var}\left[\text{d}_{\text{Hamming}}(h(\mathbf{x}_1), h(\mathbf{x}_2))/L\right] = \frac{\theta_{\mathbf{x}_1,\mathbf{x}_2}}{L\pi}\left(1 - \frac{\theta_{\mathbf{x}_1,\mathbf{x}_2}}{\pi}\right) \tag{2.9}$$

An improvement to this algorithm is Super Bit LSH [25]. In this algorithm L random vectors $\{v_1, v_2, \cdots, v_L\}$ are independently sampled from the normal distribution $\mathcal{N}(0, \mathrm{I})$, and then these vectors are divided into K batches. Then using Gram–Schmidt process every vectors of every batch

are orthogonalized. After orthogonalization, these L random vectors can no longer be viewed as independently sampled, thus they are grouped into batches that are called Super-Bits. The Hamming distance over the super bits is an unbiased estimation for the angular distance and it has the advantage that its variance is smaller than the random projection algorithm [25].

2.1.2 Locality-Sensitive Hashing for Hamming Distance

To estimate the Hamming distance between binary data points, Indyk et al [23] proposed the family of hashing functions $\mathcal{H} = \{ h_i : h_i((b_1, b_2, \cdots, b_d)) = b_i, i \sim \mathcal{U}\{1, d\}\}$, where i is a randomly-sampled integer index in the range $[1, d]$. They show that for a random index i:

$$\text{Prob}\left(h(x) = h(y)\right) = 1 - \frac{\|x_i - y_i\|_h}{d} \tag{2.10}$$

This probability equation shows the relationship between Hamming distance in the input space and the Hamming distance the hash codes. Furthermore, [23] prove that this hash function family is $\left(R, (1+\epsilon)R, 1 - \frac{R}{d}, 1 - \frac{R(1+\epsilon)}{d}\right)$ sensitive.

2.1.3 Locality-Sensitive Hashing for Similarity Measure between Two Sets

The Jaccard similarity coefficient, also known as Intersection over Union is a measure of similarity between finite sample sets. It is the size of the intersection of two sample sets divided by the size of the union of them:

$$\text{J}(\text{A}, \text{B}) = \frac{\|\text{A} \cap \text{B}\|}{\|\text{A} \cup \text{B}\|} \tag{2.11}$$

The Jaccard coefficient is 0 when two sets are disjoint, 1 for equal sets, and a value between 0 and

1 otherwise. If two sets have more members in common, which means that they are more similar, their Jaccard coefficient is closer to 1. Min-hash [5, 6], or min-wise independent permutations locality sensitive hashing scheme, is an LSH function for estimating the Jaccard similarity $\mathrm{J(A, B)}$ without computing the intersection and union.

Let $g(\cdot)$ be a function that maps every possible member of any set to a distinct integer number. This function can be seen as the index of element in a dictionary of every possible element. Let $\pi(\cdot)$ be a random permutation that maps every integer value to an integer value with a fixed random ordering. MinHash function $h_\pi(A)$ is defined as:

$$h_\pi(A) = \min_{a \in A} \pi\left(g(a)\right) \tag{2.12}$$

It first uses the function $g(\cdot)$ to map all elements of the set A to integer values, then uses $\pi(\cdot)$ to generate a new set of integers according to that specific permutation, and finally reports the minimum value of this last set. It is shown that the Hamming distance between MinHash components is an estimation of Jaccard coefficient for the sets [5]:

$$\mathrm{Prob}\left(h_\pi(A) = h_\pi(B)\right) = \mathrm{J}(A, B) \tag{2.13}$$

When using L hash functions, the Jaccard similarity is estimated with the L hash values of two sets as $\frac{1}{L}\sum_{l=1}^{L} 1\left[h_{\pi_l}(\mathrm{A}) = h_{\pi_l}(\mathrm{B})\right]$, where each π_l corresponds to a random permutation that is independently generated.

2.1.4 Locality-Sensitive Hashing for Rank Correlation Measure

The Winner Take All [65] hash family is an extension of MinHash algorithm [6] for estimating the rank correlation measure. The rank correlation measure makes this algorithm robust with respect to noise and small variations which are important characteristics of image data.

The rank correlation measure defines similarity between two points by the degree to which their feature dimension rankings agree regardless of the absolute values of the feature dimensions. For example, a simple pairwise order similarity measures is defined as:

$$R(\mathbf{x}, \mathbf{y}) = \sum_{i} \sum_{j<i} T\left((x_i - x_j)(y_i - y_j)\right) \tag{2.14}$$

The equation 2.14 measures the number of pairs of feature dimensions in $\mathbf{x}$ and $\mathbf{y}$ that have the same ordering, and $T(x)$ is used to detect the order of a pair of values and defined as:

$$T(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{2.15}$$

In its original form, the WTA algorithm uses L different random permutations. Each permutation reorders the entries of input array and then the index of maximum entry in the first K entries of the permuted array is given as the lth entry of hash code. By repeating this process with different L permutations, a hash code of length L in the base K is generated which can be represented as a binary code of length $\lfloor \log_2(K^L - 1) \rfloor + 1$.

The WTA hash algorithm is a data-independent method, since the permutations are random and there is no feasible method to find the set of permutations that give the best performance. Our first

contribution is a novel formulation for this algorithm as a differentiable function. This makes it possible to optimize the optimal hashing for a specific data set.

2.2 Binary Reconstruction Embedding

Binary Reconstruction Embedding [29] is an extension of standard random hyperplane locality-sensitive hashing [8]. In the random hyperplane locality-sensitive hashing, we have a set of hash functions $h_i(\mathbf{x}) = \text{sign}(\mathbf{w}_i^T \mathbf{x})$ where the hyperplane normal vector $\mathbf{w}_i$ is a random vector from a multivariate Gaussian with zero-mean and identity covariance. In BRE, the hash functions are dependent on one another, and defined as parametric functions:

$$h_i(\text{x}) = \text{sign}\left(\sum_{j=1}^{s} W_{ij} \kappa\left(\mathbf{x}_{ij}, \mathbf{x}\right)\right) \tag{2.16}$$

In this function $\mathbf{x}_{i\cdot}$ are a set of data points, and $W_{i\cdot}$ are the parameters that are used to define the function $h_i(\cdot)$. Note that each hash function $h_i(\cdot)$ may use a different sets of s points. The function $\kappa\left(\cdot,\cdot\right)$ is kernel function over the data which can be the standard inner product. This kernelized form, makes the algorithm amenable to work over a variety of input data.

This algorithm does not assume anything about the distribution of the data, and instead of being random, the matrix W is learned from the data by minimizing the difference between the normalized Euclidean distance in the item space and the normalized Euclidean distance in the hash code space, i.e.

$$\mathcal{L} = \sum_{i,j} \left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2} - \frac{\|h(\mathbf{x}_i) - h(\mathbf{x}_j)\|_2^2}{L}\right)^2 \tag{2.17}$$

In this equation, we assuming that the vectors $\mathbf{x}_i$ are normalized to have L_2 norm of one, and L is the number of dimensions of the binary space. Binary Reconstruction Embedding utilizes

coordinate descent algorithm to optimize the objective function $\mathcal{L}$. They fix all but one weight W_{ij}, and optimize the objective $\mathcal{L}$ with respect to this weight.

2.3 Spectral Hashing

Spectral hashing [62] is another learning based hashing algorithm. Let $\{\mathbf{x}_i\}_{i=1}^n$ be the data points in the input space and $\{\mathbf{y}_i\}_{i=1}^n$ be the codewords in $\{-1,1\}^L$. Assuming that the Euclidean distance in the input space correlates with the similarity between data points, the matrix $W_{n\times n}$ is defined:

$$W_{i,j} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\epsilon^2}\right) \tag{2.18}$$

Here, the parameter ϵ defines the distance which corresponds to similar items. Note that for data points with small distance in input space $W_{i,j}$ will be a close to one and for far apart data point this value goes toward zero. The average Hamming distance between similar neighbors is $\sum_{i,j} W_{i,j}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2$, and the goal is to minimized the average Hamming distance under the constraint that the codewords have the following desirable properties:

$$\begin{aligned} &\text{minimize: } \sum_{ij} W_{ij}\,\|y_i - y_j\|_2^2 \\ &\text{subject to: } y_i \in \{-1,1\}^k \\ &\qquad\qquad \sum_i y_i = 0 \\ &\qquad\qquad \frac{1}{n}\sum_i y_i y_i^T = I \end{aligned} \tag{2.19}$$

such that the code words are uncorrelated and balanced, i.e. In this optimization problem the constraint $\sum_i y_i = 0$ is in order to make the number of zeroes and ones in the code words balanced,

and the constraint $\frac{1}{n}\sum_i y_i y_i^T = I$ demands the bits of code words to be uncorrelated.

The above optimization problem is NP hard, and to solve it [62] define the matrix $Y_{(n\times k)}$ whose jth row is y_j^T and a diagonal matrix $D_{(n\times n)}$ where $D(i,i) = \sum_j W(i,j)$ and removing the constraint that $Y(i,j) \in \{-1, 1\}$ to rewrite the problem as the following relaxation:

$$\begin{aligned} \text{minimize: } & \operatorname{trace}\left(Y^T(D-W)Y\right) \\ \text{subject to: } & Y^T 1 = 0 \\ & Y^T Y = I \end{aligned} \tag{2.20}$$

Which is a simple problem whose solutions are k eigenvectors of $D-W$ with minimal eigenvalue. Clearly, the projection directions are the principal directions of the data.

To extend this algorithm for out of sample data points, they assume that the data points a a probability distribution $p(x)$. This makes the above optimizing problem to be written as follows:

$$\begin{aligned} \text{minimize: } & \int \|y(x_1) - y(x_2)\|^2 W(x_1, x_2)\, p(x_1)\, p(x_2)\, dx_1 x_2 \\ \text{subject to : } & \int y(x) p(x) dx = 0 \\ & \int y(x) y(x)^T p(x) dx = I \end{aligned} \tag{2.21}$$

The solution to this problem are functions that satisfy $L_p f = \lambda f$ with minimal eigenvalue. As discussed in [16, 15, 13], with proper normalization, the eigenvectors of the discrete Laplacian defined by n points sampled from $p(x)$ converges to eigenfunctions of Lp as $n \to \infty$.

2.4 Deep Networks for Hashing

Recently, deep learning methods have shown significant advances in many machine learning tasks such as feature representation learning, classification, clustering, and sample data point generation. A deep neural network model defines a parametric function $f(\mathbf{x}, \theta)$ and learns the parameters θ that approximate a specific function optimally. Deep neural networks have the flexibility to encode any nonlinear function, and this resulted in state-of-the-art results for many machine learning tasks on many benchmarks. One of the successful applications of this method is to learn nonlinear hash functions [63, 72]. In a feedforward neural network model the data flows through the layers of the model starting from the input layer x sequentially to the output layer y. This is equivalent to a composition of multiple different functions.

The basic method to train a neural network model is optimizing the model by the gradient decent. In order to use the gradient decent method to find the optimal parameters θ, every single function in the network should be differentiable. In this method, a cost function is defined that shows the ability of our model to

2.4.1 Convolutional Neural Network Hashing (CNNH)

In this method, the goal is to learn L hash functions using a dataset of images $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ and the pairwise similarity labels which is defined as [63]:

$$S_{ij} = \begin{cases} +1, & I_i, I_j \text{ are semantically similar} \\ -1, & I_i, I_j \text{ are semantically dissimilar.} \end{cases}$$

The goal is to generate hash codes that preserve semantic similarities over pairs of images.

This method, similar to [69] and [33], adopts a two stage method to learn the hash functions. First, approximate hash codes for images are learned, and the next stage is to learn image feature representation and hash functions simultaneously using the approximate hash codes found in the first stage.

2.4.1.1 Stage 1: Learn Approximate Hash Codes

In this step, we define an n by L binary matrix H such that the row i represent the hash code for image i. The goal of this step is to find H such that the Hamming distance between two rows H_i and H_j be correlated with S_{ij} the semantic similarity of images i and j. It is known that when the bit values are -1 and 1, i.e. $H_i \in \{-1, 1\}^L$ the inner product $H \cdot H^T$ has one-to-one correspondence to the Hamming distance between hash codes [37]. Note that $H_i \cdot H_j^T$ is in the range $[-L, L]$. Finding the approximate hash codes for the image dataset in then is reduced to minimizing the reconstruction errors:

$$\min_H \sum_{i=1}^{n} \sum_{j=1}^{n} \left(S_{ij} - \frac{1}{L} H_i \cdot H_j^T \right)^2 = \min_H \left\| S - \frac{1}{L} H \cdot H^T \right\|_F^2 \tag{2.22}$$

Where $\| \cdot \|_F$ is Frobenius norm, and the Hamming distance has been normalized to be comparable to the pairwise semantic similarities.

Because of integer constraints $H_i \in \{-1, 1\}^L$, optimizing 2.22 is difficult. [63] solves this optimization problem by relaxing the constraints to $H_i \in [-1, 1]^L$, i.e letting the binary bit to be in the real value range $[-1, 1]$, and by employment of coordinate descent algorithm using Newton directions. The optimization process is to repeatedly choose a random entry in H and optimize it while keeping all other entries fixed.

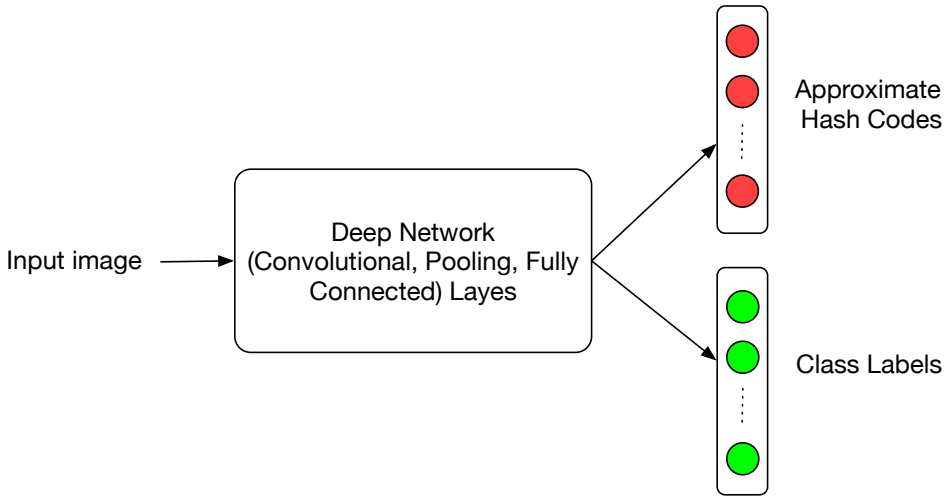

Figure 2.2: The convolutional neural network model to learn the feature representation and hash functions simultaneously. The network consists of convolutional, pooling and fully connected layers. The model hash two output layers, one to generate the hash codes and one to predict the class label for input images.

2.4.1.2 Stage 2: Learn Feature Representation And Hash Codes

In the second stage, a neural network model with two heads is used. As depicted in figure 2.2, one output is a fully connected layer for hash codes and the other output is a fully connected layer for class labels. This neural network model is then trained with the approximate hash codes found in the first step and class labels for images. Note that if class labels are not available, it is still possible to train the neural network model using similarity information between items. This leads to learning both image feature representation and hash functions at the same time.

The class label output layer enforces the network to learn an image representation that is beneficial for both the approximate hash codes and the image class labels. This architecture can be considered as a transfer learning case in which class labels help to learn image representations that are used to generate the hash codes.

2.4.2 Deep Neural Network Hashing (DNNH)

This is hashing method based on a deep neural network model to learn the image feature representation and the hash codes in a single stage [31]. Given a dataset of images $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, the goal is to find a mapping $\mathcal{F}$ that directly maps images to binary hash codes preserving similarity between images. This method follows some recent work, such as [32, 45], that uses relative similarity of a triple of images. The relative similarity is of the form "image I is similar to image I^+ more than image I^-". The mapping $\mathcal{F}$ should generate hash codes such that the more similar the images the smaller their Hamming distance. This can be achieved by using the following loss function [31]:

$$\ell_{\text{triplet}}\left(\mathcal{F}(I), \mathcal{F}\left(I^+\right), \mathcal{F}\left(I^-\right)\right) = \max\left(0, 1 - \left(\left\|\mathcal{F}(I) - \mathcal{F}\left(I^-\right)\right\|_H - \left\|\mathcal{F}(I) - \mathcal{F}\left(I^+\right)\right\|_H\right)\right)$$

To use this loss function for optimization the Hamming distance $\|\cdot\|_H$ is replaced by L_2 norm, and the mapping $\mathcal{F}$ generates a value in the range of $[-1, 1]$ instead of binary codes. Later the output of the mapping is converted to integer binary values using a threshold.

As shown in figure 2.3, the mapping $\mathcal{F}$ is a deep neural network model. The model consist of a shared sub-network with a stack of convolution layers, followed by a fully connected layer or a divide and encode module.

The shared sub-network part uses Network in Network architecture [34]. Instead of a fully connected layer usually used in traditional architectures, in this architecture an average pooling layer is used as output layer. The parameters of the sub-network are shared for each one of the three input images. This shared sub-network maps images to intermediate image feature vectors.

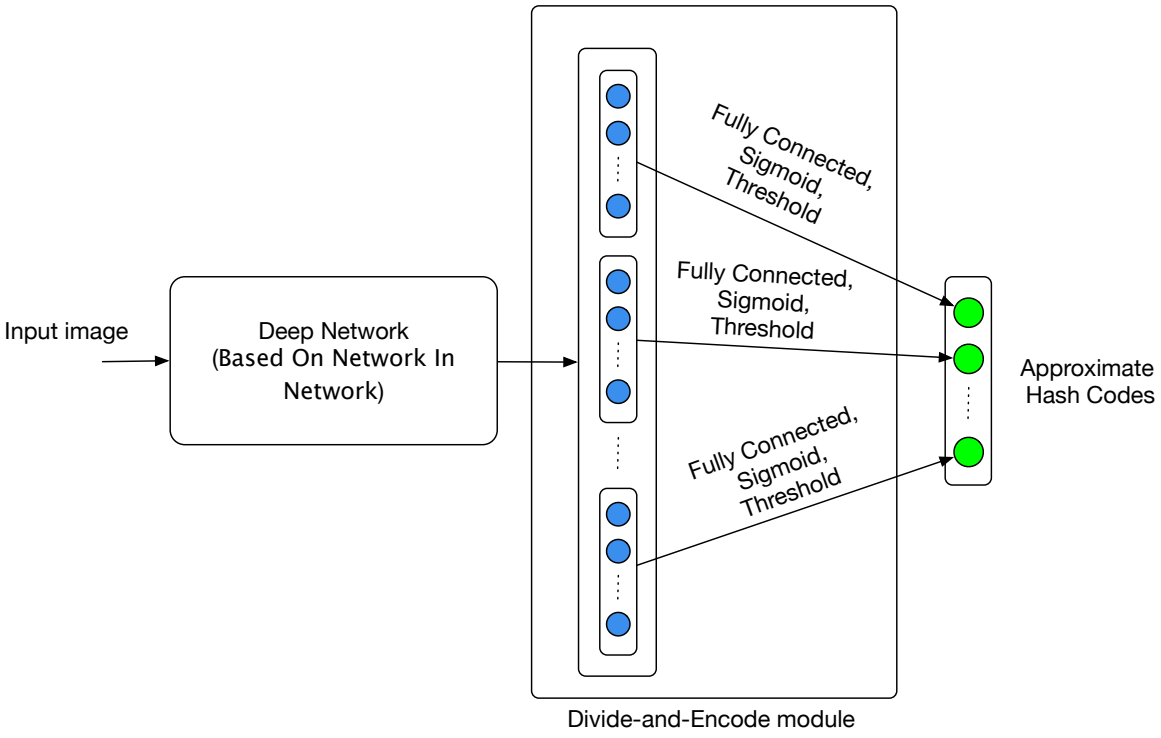

Figure 2.3: The framework for DNNH hashing method. It consists of a deep network based on network-in-network architecture, a divide-and-encode module which is a set of fully-connected layers followed by a sigmoid function and a piece-wise threshold function.

Divide-and-Encode module maps the image features to approximate hash codes. For a target hash code of length L, the output of the shared sub-network is designed to be o f length $50L$. Then the divide-and-encode module, divides this feature vector to L slices. Then, every slice is mapped to a single number in the range $[0, 1]$ using a fully connected layer of length one followed by a sigmoid activation function and a piece-wise threshold function to push the outputs to be binary hash bits. Then, these L hash bits are concatenated to form an approximate L-bit hash code. The idea behind using the divide-and-encode module is to reduce the redundancy between the hash bits. Since each hash bit is generated using a separate slice, they have less dependency on each

other. The piece-wise threshold function is defined as:

$$g(s) = \begin{cases} 0, & s < 0.5 - \epsilon \\ s, & 0.5 - \epsilon \leq s \leq 0.5 + \epsilon \\ 1, & s > 0.5 + \epsilon \end{cases} \tag{2.23}$$

where ϵ is a parameter and s is the output of the sigmoid function. This function is used in the training phase, and in the test phase it is replaced by a simple quantization function (a threshold at 0.5).

This method also uses a simple fully-connected layer of length L as an alternative to the divide-and-encode module, then a sigmoid functions transforms the output vector of the fully connected layer to a vectors in the range $[0, 1]$. The hash codes generated using divide-and-encode module outperform the ones generated with a simple fully-connected layer.

2.4.3 Deep Semantic Ranking Based Hashing (DSRH)

This hashing method is based on deep learning and semantic ranking and learns hash functions that preserve similarity between multi-label images [71]. Similar to other deep learning methods, this method does not use a separate feature extraction module and incorporates all components in a unified deep convolutional neural network model.

As depicted in Figure 2.5, the hash function is a deep neural network model that has the first five convolutional layers of AlexNet [28]. The input image is of sized 224×224. After the convolutional layers. there is two fully connected layers, and then a special fully connected layer that is connected to the concatenated vectors of the previous fully connected layes with the $\mathrm{Sign}(\cdot)$ activation function. The concatenation of two fully connected layers is to provide the hash layer

with more information about the detailed content and appearance of the input image, because the feature vector of the second fully connected layer have high level semantic information useful for classifying the input image as a whole. The hash layer at the end is defined as:

$$h(\mathbf{x}; \mathbf{w}) = \text{sign}\left(\mathbf{w}^{\mathrm{T}}\left[f_a(\mathbf{x}); f_b(\mathbf{x})\right]\right) \tag{2.24}$$

This hashing method works for multilabel datasets. For the simple case of single label items, two items are either similar if they have the same labels or dissimilar if their labels are different, and the goal is to learn hash functions that make the Hamming distances between binary codes small/large for similar/dissimilar pairs of items. However, in the case of multilabel data items the similarity of two items depends on how many labels they have in common, and the objective is to preserve this multilevel similarity. To do that the ranking order of neighbors computed using the hash codes should be in accordance with the ranking orders computed using the the class label similarity.

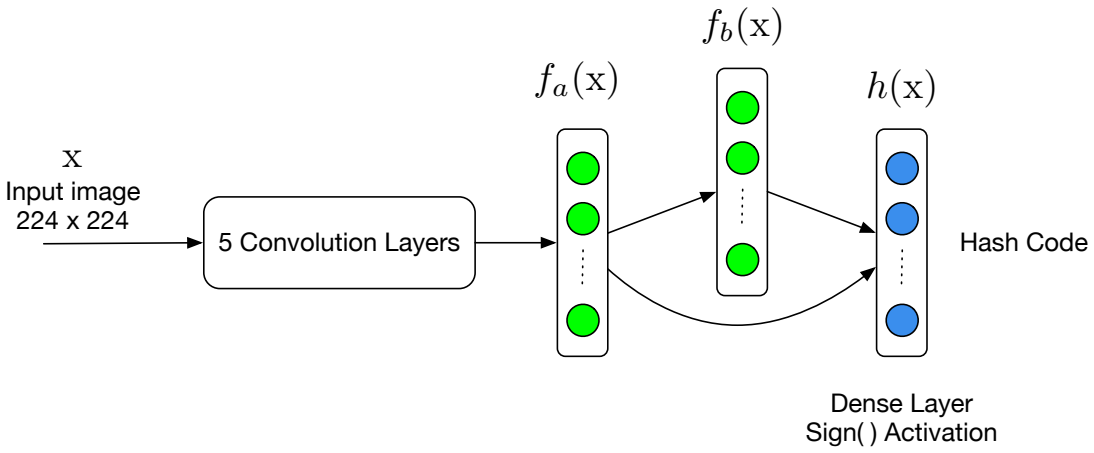

Figure 2.4: The deep semantic ranking based hashing model. Images go through first five convolution layers of AlexNet framework. Then two fully connected layers generates a deep feature representation for the input image. At the end, a hash layer is used to output the hash code. The hash layer is a fully connected layer that is connected to both previous layers, and hash a threshold as the activation function

For a query data item $\mathbf{q}$ and given data item x $\in \mathcal{D}$ the semantic *similarity-level* is defined as the number of common labels between $\mathbf{q}$ and x. So, the ground-truth ranking list for $\mathbf{q}$ can be calculated by sorting the dataset $\mathcal{D}$ in decreasing order of their similarity. Based on ground truth ranking an evaluation criteria, such as Normalized Discounted Cumulative Gain (NDCG) score [24], is used to measure the consistency of the rankings when calculated using hash code.

$$NDCG@p = \frac{1}{Z} \sum_{i=1}^{p} \frac{2^{r_i} - 1}{\log(1+i)} \tag{2.25}$$

In equation 2.25, p is the truncated position in a ranking list, Z is a normalizeration constant, and and r_i is the *similarity-level* of the i-th database point in the ranking list.

Optimizing this ranking criteria needs minimizing a nonsmooth and multivariate loss function. Thus, a surrogate loss function is used in learning procedure. For a query data item $\mathbf{q}$ and a ranking list $\{\mathbf{x}_i\}_{i=1}^{M}$, the ranking loss is defined on a set of triplets as follows:

$$L\left(\mathbf{h}(\mathbf{q}), \{\mathbf{h}\left(\mathbf{x}_i\right)\}_{i=1}^{M}\right) = \sum_{i=1}^{M} \sum_{j:r_j<r_i} \max\left(0, d_H\left(\mathbf{h}(\mathbf{q}), \mathbf{h}(\mathbf{x}_i)\right) - d_H\left(\mathbf{h}(\mathbf{q}), \mathbf{h}(\mathbf{x}_j)\right) + \rho\right) \tag{2.26}$$

where ρ is a margin parameter to control the minimum distances between two pairs and $d_H(\cdot,\cdot)$ is the Hamming distance. Further more, the Hamming distance in replaced by the form of inner product $d_H(\mathbf{x}, \mathbf{y}) = \frac{n - \mathbf{x}^{\mathrm{T}}\mathbf{y}}{2}$, where n is the length of vectors $\mathbf{x}$ and $\mathbf{y}$.

This loss function is a convex and it can be shown that it is the upper bound of the number of incorrectly ranked triplets. Finally, the model is trained using stochastic gradient descent to minimize the objective function.

2.4.4 *Deep Hashing (DH) and Supervised Deep Hashing (SDH)*

Deep Hashing (DH) [35] is another hashing method that uses a deep convolutional network to learn hash functions. The main contribution of this method is the loss function that is used to train the model. The model is deep network of M layers that has a threshold function at the top layer to obtain the binary hash codes:

$$\mathbf{h}_i = \text{sign}\left(\mathbf{y}_i^M\right) \tag{2.27}$$

In equation 2.27, $\mathbf{h}_i$ is the hash code for image $\mathbf{x}_i$, and $\mathbf{y}_i^M$ is output of the last layer (M-th layer) of the deep convolutional neural network.

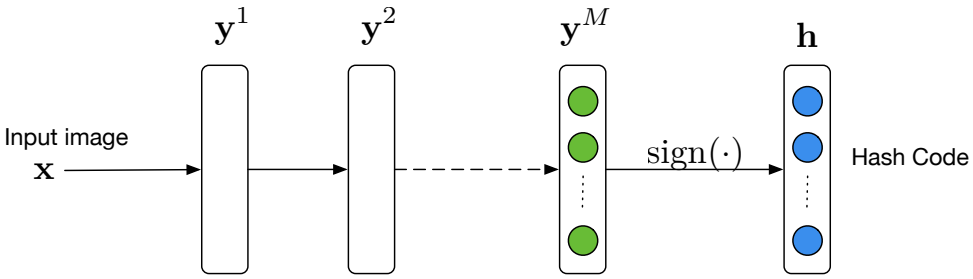

Figure 2.5: The deep hashing model. Images are fed to a stack of convolutional and fully connected layer and at the end a $\text{sign}(\cdot)$ function quantize the output of the last layer. In the supervised setting, the loss function aims to minimize the difference between the output of all layers close for two similar items and maximize the difference for two dissimilar items.

Given a dataset of images $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, define the matrix $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_N] \in \{-1, 1\}^{L \times n}$ where L is the length of the hash code, and n the size of the dataset. The column i in this matrix is the hash code for the item i. Also, define the matrix $\mathbf{Y}^m = [\mathbf{y}_1^m, \ldots, \mathbf{y}_N^m]$ for the layer m of the network. Every column of this matrix is the output of layer m for the input image. To learn the

hash function this method propose to minimize the following loss function:

$$
\begin{aligned}
J &= J_1 - \lambda_1 J_2 + \lambda_2 J_3 + \lambda_3 J_4 \\
&= \frac{1}{2} \left\| \mathbf{H} - \mathbf{Y}^M \right\|_F^2 \\
&- \frac{\lambda_1}{2n} \operatorname{tr} \left(\left(\mathbf{Y}^M \mathbf{Y}^M \right)^T \right) \\
&+ \frac{\lambda_2}{2} \sum_{m=1}^{M} \left\| \mathbf{W}^m \left(\mathbf{W}^m \right)^T - \mathbf{I} \right\|_F^2 \\
&+ \frac{\lambda_3}{2} \left(\| \mathbf{W}^m \|_F^2 \right)
\end{aligned}
\tag{2.28}
$$

In this equation $\mathbf{W}$ represent the parameters of the deep network model, and $\operatorname{tr}(\cdot)$ is the trace of the matrix. The term J_1 is to minimize the quantization error between the real valued outputs of the last layer and the binary hash codes. The second term J_2 is to maximize the variance of the binary bits of the hash codes, in order to have balanced bit which are used uniformly. The term J_3 aims to maximize the independence of each layer transformation by putting a relaxed orthogonality constraint on the projections. Finally, the last term J_4 is to regularize the model parameters. To optimize the model with this loss function the stochastic gradient decent method is used.

The above mentioned method (DH) is an unsupervised method and does not use the similarity information of the items. The performance of this method can be improved by using label information in the training phase. The improved version of this method is called Supervised Deep Hashing (SDH) [35]. In SDH for each pair of data items $(\mathbf{x}_i, \mathbf{x}_j)$ we have the similarity label s_{ij} which is one when the two items are similar and zero when they are dissimilar. To incorporate this similarity information in the loss function we replace J_1 with the following loss:

$$
J_1 = \operatorname{tr} \left(\frac{1}{N} \mathbf{Y}^M \left(\mathbf{Y}^M \right)^T \right) + \alpha \operatorname{tr} \left(\Sigma_B - \Sigma_W \right) \tag{2.29}
$$

where:

$$\Sigma_W = \frac{1}{Z_s} \sum_{s_{ij}=1} \left(\mathbf{y}_i^M - \mathbf{y}_j^M\right) \left(\mathbf{y}_i^M - \mathbf{y}_j^M\right)^T \tag{2.30}$$

$$\Sigma_B = \frac{1}{Z_d} \sum_{s_{ij}=0} \left(\mathbf{y}_i^M - \mathbf{y}_j^M\right) \left(\mathbf{y}_i^M - \mathbf{y}_j^M\right)^T \tag{2.31}$$

In equations 2.30 and 2.31, Z_s is a normalizing constant which is the number of pairs of items that are similar, and Z_s is the number of pairs of items that are dissimilar. Clearly, minimizing Σ_W pushes the output of every layer in the deep network to be close for two similar items and maximizing Σ_B leads to increase the output of every layer in the deep network to be different for two similar items. Similar to DH the total loss function is minimized using stochastic gradient descent method.

2.5 Evaluation Measures

In information retrieval, there are some metrics that are commonly used to evaluate the performance of the different systems and algorithms. Given a query item the system is supposed to rank the items in the database according to their relevance to the query items. The items are called "positive" (P) when they are relevant to the query items, and the one that are irrelevant are called "negative" (N). Thus, when the system return a relevant item it is called a "true positive" (TP), and when the retrieved item is irrelevant it is a "false positive" (FP) item. Note that the term "true positive" and "false positive" are also used to indicate the "number" of these items. Similarly, the item considered by the system as irrelevant are called "true negative" (TN) if they are indeed irrelevant and "false negative" (FN) if they relevant items. Given these naming conventions, the following metrics are defined.

2.5.1 Precision

Precision is the fraction of relevant items (true positive items) among the retrieved items (all returned items i.e both true positive and false negative items), or the probability that a retrieved item is relevant. Precision is also called positive predicted value. When does not specified, precision takes all retrieved items into account. i.e.

$$\text{precision} = \frac{|\{ \text{relevant retrieved items} \}|}{|\{ \text{retrieved items} \}|} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.32}$$

However, this metric can be evaluated considering only the n top ranked items returned by the system. In this case, it is called precision at n (precision@n).

2.5.2 Recall

Recall is the fraction of the number of relevant items retrieved among all relevant items in the dataset, or in other words the probability that the relevant item/items are retrieved. This metric is also known as *sensitivity*, and *true positive rate*.

$$\text{recall} = \frac{|\{ \text{relevant retrieved items} \}|}{|\{ \text{relevant items} \}|} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.33}$$

When this metric is evaluated with threshold n for the top ranked items returned by the system, it is called recall at n (recaall@n).

2.5.3 Precision Recall Curve

The precision-recall curve shows the relation between precision and recall for different thresholds. The x-axis is for recall values and the y-axis for precision values both from 0 to 1. The curve start from left top corner and ends in a point with recall 1 and precision of a random classifier.

While recall expresses the ability of the system to find all relevant items in a dataset, precision expresses the proportion of the data points the system indicates as relevant where they are actually relevant. The high the area under the curve the closer the system is to the perfect information retrieval system.

2.5.4 Average Precisiona and Mean Average Precision

Mean average precision (mAP) is a single number which quantifies the quality of data retrieval system. Given a query item, a retrieval system return a ranked set of items from the reference dataset or gallery. Figure 2.6 illustrate this metric by an example. First, at every threshold if the item is true positive the precision is calculated. The average precision is the average of precision values given that the precision at thresholds with false positive is replaced by zero. Intuitively when the more relevant items at the top of the ranking of retrieved items this metric has higher values.

Mean average precision (mAP) is the the mean value of average precisions for a set of query items. When these metrics are evaluated for only top n retrieved items, it is called average precision at n (AP@n), and mean average precision at n (mAP@n) respectively.

This metric summarizes the quality of a specific item retrieval algorithm. However, note that when an algorithm employs hash codes for document retrieval the relevance (similarity) of items

is quantified with discrete values. This puts many retrieved items at the same rank with respect to a specific query item. When the items with the same rank are both relevant and irrelevant, the order of the items will affect the mean average precision value significantly. Despite this problem, researchers report the mean average precision without further indication on the order of items with the same rank.

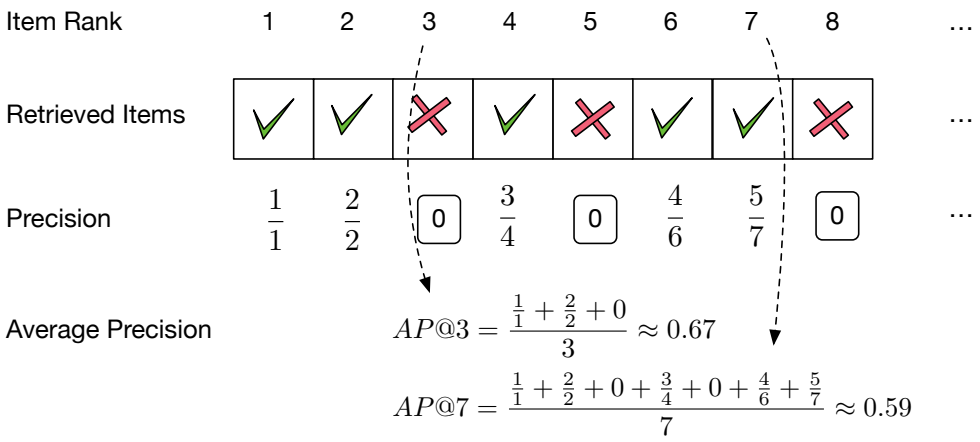

Figure 2.6: The process of calculating average precision for a query item. The relevant and irrelevant retrieved items are show respectively with tics and crosses.

2.5.5 Accuracy

For a classifier, accuracy is the fraction of the number of true positive and false negative items among all items in dataset.

$$\text{accurcy} = \frac{\text{TP} + \text{FN}}{\text{P} + \text{N}} \tag{2.34}$$

When the dataset is not balanced, i.e, the number of positive items is noticeably different than the number of negative items, this metric does not give a good indication of system performance. A solution for this problem is balanced accuracy in which the number of positive and negative samples are first normalized before applying the above equation.

2.6 Summery

In this chapter, we reviewed some important approaches to solve the hashing for similarity search problem. We started with the local sensitivity hashing and some of its variants for different distance metrics. Then, studied few other fundamental data independent and data dependent hashing algorithms. At the end, we reviewed some of the main evaluation metric of hashing systems.

CHAPTER 3: SUPERVISED MAXHASHING

3.1 Introduction

The Winner Take All (WTA) [65] hash family which is an extension of Min-Hash algorithm [6]. WTA algorithm uses the rank correlation as a measure of similarity instead of quantizing a real valued vector using a threshold function. The rank correlation measure makes this algorithm robust with respect to noise and small variations which are important characteristics of image data. In its original form, the WTA algorithm uses L different random permutations. Each permutation reorders the entries of input array and then the index of maximum entry in the first K entries of the permuted array is given as the l-th entry of hash code. By repeating this process with different L permutations, a hash code of length L in the base K is generated which can be represented as a binary code of length $\lfloor \log_2(K^L - 1) \rfloor + 1$ [2].[1]

Clearly, WTA algorithm is a data-independent hashing method, since the permutations are random and there is no feasible method to find the set of permutations that give the best performance.

In this chapter, we present a learning based extension of WTA hash algorithm that resolves the problems of linear subspace ranking hash and can be connected seamlessly to other neural network layers. The result is a consistent model in which both feature learning and hash coding parts are trained together optimally.

[1]

3.2 Method

Let's suppose we have a training set $X = \{\mathbf{x}^{(i)}\}_{i \in 1 \ldots N}$ of N examples along with their semantic labels or relevance information. Using the semantic labels we generate a semantic similarity set $S = \{s_{ij}\}_{i,j \in 1 \ldots N}$ such that $s_{ij} = 1$ if two examples $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are semantically similar and $s_{ij} = 0$ if they are not. In the training phase, pairs of training examples along with their similarity labels are fed to the model. The goal of supervised learning based hash algorithms is to train the model so that it produces two hash codes $\mathbf{h}^{(i)}$ and $\mathbf{h}^{(j)}$ that have small Hamming distance when the examples $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are semantically similar, and vice versa. Once the model is trained, it acts as a nonlinear mapping function that generates a hash code $\mathbf{h}^{(i)}$ corresponding to the item $\mathbf{x}^{(i)}$.

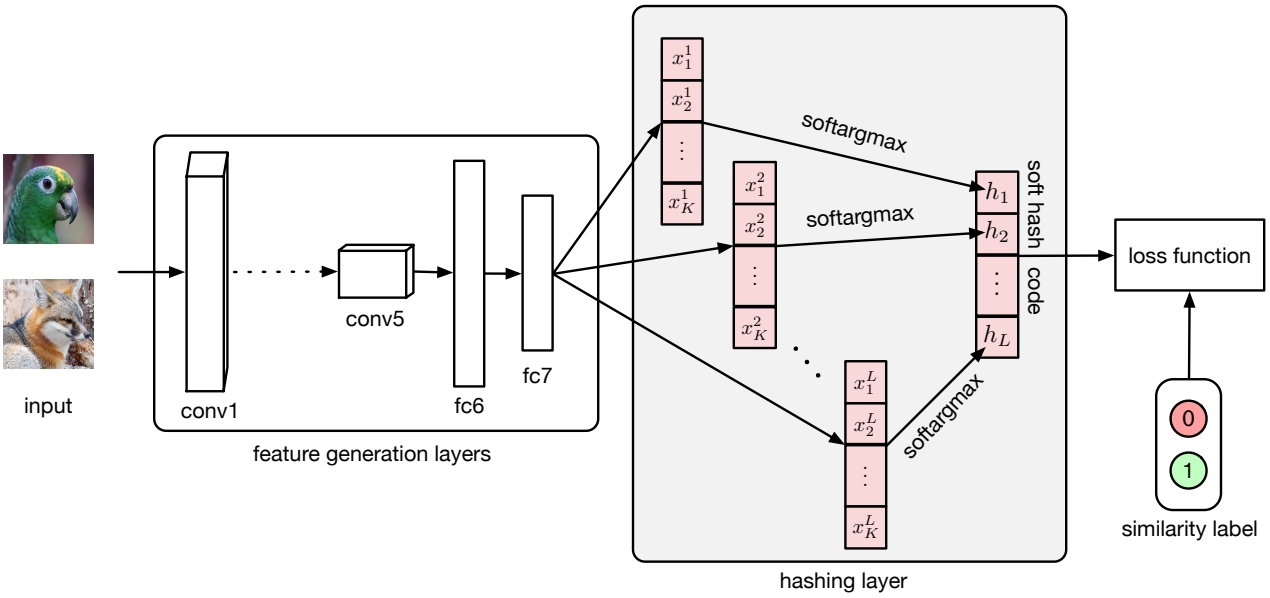

Figure 3.1: The proposed supervised max hashing network which is comprised of: (1) a standard deep convolutional neural network for learning image representation (in our case AlexNet [28]), (2) a hashing layer for transforming the image representation into K-ary L-dimensional hash code, and (3) a pairwise cross-entropy loss function for learning a similarity-preserving mapping.

In this work, we present a supervised learning to hash method with a ranking based hash function

that is differentiable and therefore can be connected to other layers of neural networks. Figure 3.1 shows the pipeline of our framework. It consists of the following three components:

- A deep convolutional network to produce a rich image representation as a features vector. To be comparable with other works, we used the first seven layers of AlexNet [28].
- A ranking hash layer. It is comprised of L parallel linear mappings of the feature vector followed by a differentiable approximation of argmax function. We call this differentiable approximation of argmax function soft-argmax.
- A novel pairwise loss function which: (1) minimizes the quantization error for each hash code, (2) and maximizes hash code similarity for similar items and vice versa.

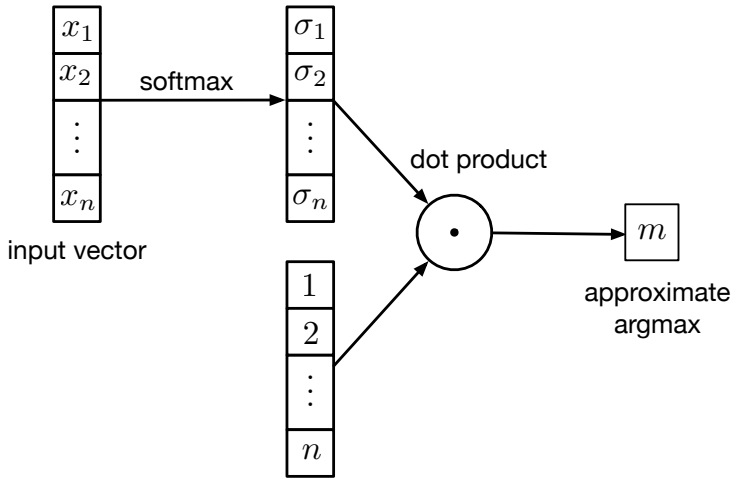

Figure 3.2: Differentiable approximation of argmax function to find the index of maximum entry in an array. This function is the composition of a softmax function followed by the dot product by a vector whose entries are equal to their indices.

3.2.1 Convolutional Neural Network

The first step of every hash code generation system is the conversion of input images into compact feature vectors. These feature vectors should preserve semantic similarity between input images and avoid those differences that do not reflect real semantic disparity. While most of image hashing methods use hand-crafted visual descriptors or pre-trained deep convolutional neural networks, training a deep convolutional neural network in conjunction with hash coding system results in a better performance. This is because the feature vector generator is optimally trained for the specific hash function.

Since the performance of hashing algorithms depends on the quality of feature vectors, in order to present a fair comparison of other approaches with our novel contributions (hashing layer and differentiable approximation of argmax function), we have used the AlexNet [28] deep neural network for generating feature vectors. The other approaches such as DHN [72] and DNNH [31] have also used the same network for feature generation.

3.2.2 Permutation

It is well known that a permutation can be represented by a square binary matrix M of size $n \times n$ obtained by permuting the columns of the identity matrix $I_{n\times n}$ [7]. For example the permutation $\pi : \{1,2,3,4\} \rightarrow \{4,3,1,2\}$ or $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$ or simply $\pi = (4,3,1,2)$ can be represented by the matrix: $M = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$. In this example, given a vector $\mathbf{v} = (a,b,c,d)^T$ the permuted

vector is $M\mathbf{v} = (d, c, a, b)^T$. Therefore, a permutation is indeed a special case of a projection, and we can replace it with a matrix multiplication. This gives us a differentiable equivalent of permutation operation.

3.2.3 Hashing Layer and Soft-ArgMax

Our hashing layer produces an L dimensional hash code $\mathbf{h} = \{h_1, h_2, h_3, \ldots h_L\}$. Each entry in the L dimensional hash code is computed as follows: first, we project the feature vector to a K dimensional subspace, then we find the index of the feature that has the maximum value. This index is one of L entries of the hash code. In order to get an L dimensional hash code, this process is repeated with L different projections. Therefore, for the l-th entry of the hash code, we can write:

$$h_l = \underset{1 \leq k \leq K}{\arg\max} \, (\mathbf{z}^T \mathbf{W}_l)_k \tag{3.1}$$

here $\mathbf{z}$ is the feature vector, $\mathbf{W}_l$ is projection matrix for the l-th entry of the hash code, and argmax returns the index of the maximum element in the vector $\mathbf{z}^T \mathbf{W}_l$.

The argmax function in equation 3.1 is not differentiable, thus we cannot use it in this form for training using gradient descent method. In order to train a model comprising argmax function, we have proposed the following approximation for it, which is differentiable:

$$\underset{1 \leq k \leq K}{\arg\max} \, \mathbf{x}_k \approx [1, 2, 3, ..., K] \cdot \sigma(\mathbf{x}) \tag{3.2}$$

here $\sigma(\mathbf{x})$ is the softmax function. It converts a vector $\mathbf{x}$ of arbitrary real values into a normalized

vector of real values in the range [0, 1]. The j-th entry of $\sigma(\mathbf{x})$ is given by the following equation:

$$\sigma(\mathbf{x})_j = \frac{e^{\mathbf{x}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}_k}}, \text{ for } j = 1, \ldots, K. \tag{3.3}$$

The softmax function in equation 3.2, converts the vector $\mathbf{x}$ into another vector of the same dimension. In this new vector, the element with the maximum value in $\mathbf{x}$ is replaced by a value close to 1, while other elements are close to zero. Dot product of this vector with the vector $[1, 2, 3, ...K]$ returns approximate position of the maximum element in the vector $\mathbf{x}$. Figure 3.2 depicts this approximate argmax function which we will call soft-argmax from now on.

After using equation 3.2, the complete ranking hash function for a single entry of the hash code is:

$$\begin{aligned} h_l &\approx [1, 2, 3, ..., K] \cdot \sigma(\mathbf{z}^T \mathbf{W}_l) \\ &= \sum_{k=1}^{K} k \cdot \sigma(\mathbf{z}^T \mathbf{W}_l)_k \end{aligned} \tag{3.4}$$

This function is continuous and differentiable. Let's suppose $\mathbf{x}_l = \mathbf{W}_l \mathbf{z}$ is the input of the soft-argmax, then the derivative of the hash element h_l with respect to the input $\mathbf{x}_l$ is given by:

$$\frac{dh_l}{d\mathbf{x}_{l,i}} = \sum_{k} k \cdot \sigma(\mathbf{z}^T \mathbf{W}_l)_k \left(\delta_{ik} - \sigma(\mathbf{z}^T \mathbf{W}_l)_k\right) \tag{3.5}$$

here δ_{ik} is the Kronecker delta. Please note that the soft-argmax operation will be used in training phase only, and in the test phase, when there is no need for error back-propagation, this operation can be replaced by the faster argmax operation.

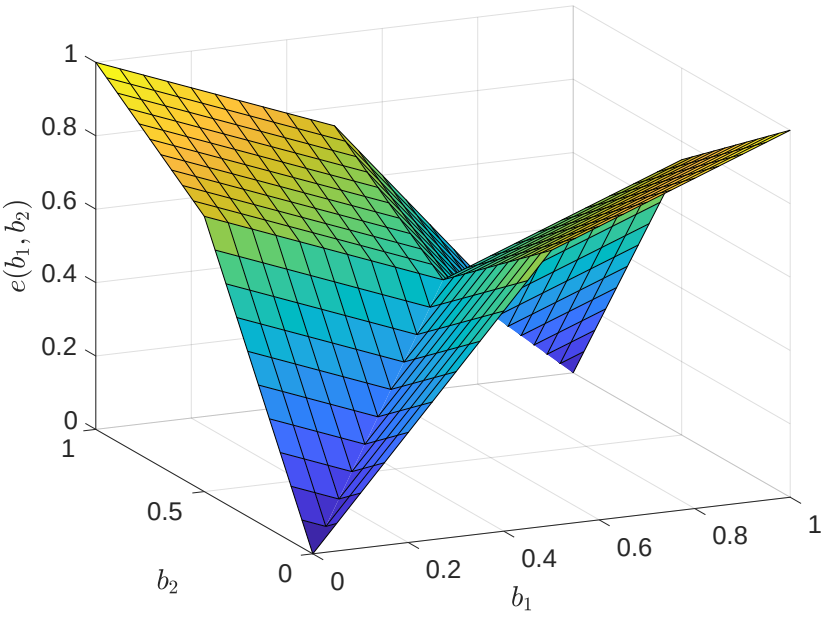

Figure 3.3: Total distance between two bits for $\alpha = 0.5$. Minimizing this distance simultanuosly pushs two bit toward integer values and minimize their Hamming distance. Tuning the parameter α moves the "saddle point" vertically and changes the slopes of of planes.

3.2.4 Loss function

For two hash codes $\mathbf{h}^{(i)}$ and $\mathbf{h}^{(j)}$, we define the total error $e(\mathbf{h}^{(i)}, \mathbf{h}^{(j)})$ as the weighted sum of the disparity error $e_d(\mathbf{h}^{(i)}, \mathbf{h}^{(j)})$ of two hash codes and quantization error $e_q(\mathbf{h}^{(i)})$ and $e_q(\mathbf{h}^{(j)})$ of each hash code:

$$e(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}) = e_d(\mathbf{h}^{(i)}, \mathbf{h}^{(j)}) + \alpha \left(e_q(\mathbf{h}^{(i)}) + e_q(\mathbf{h}^{(j)})\right) \tag{3.6}$$

In this equation, the disparity error $e_d(\mathbf{h}^{(i)}, \mathbf{h}^{(j)})$ is defined as the normalized Manhattan distance

between two hash codes $\mathbf{h}^{(i)}$ and $\mathbf{h}^{(j)}$:

$$e_d(\mathbf{h}^i, \mathbf{h}^j) = \frac{1}{L} \sum_{l=1}^{L} |\mathbf{h}_l^{(i)} - \mathbf{h}_l^{(j)}| \tag{3.7}$$

The quantization error is the normalized distances of entries of a soft hash code from its rounded integer vector, and defined as:

$$e_q(\mathbf{h}^{(i)}) = \frac{1}{L} \sum_{l=1}^{L} |\mathbf{h}_l^{(i)} - \lfloor \mathbf{h}_l^{(i)} \rceil| \tag{3.8}$$

where $\lfloor x \rceil$ is the nearest integer to x.

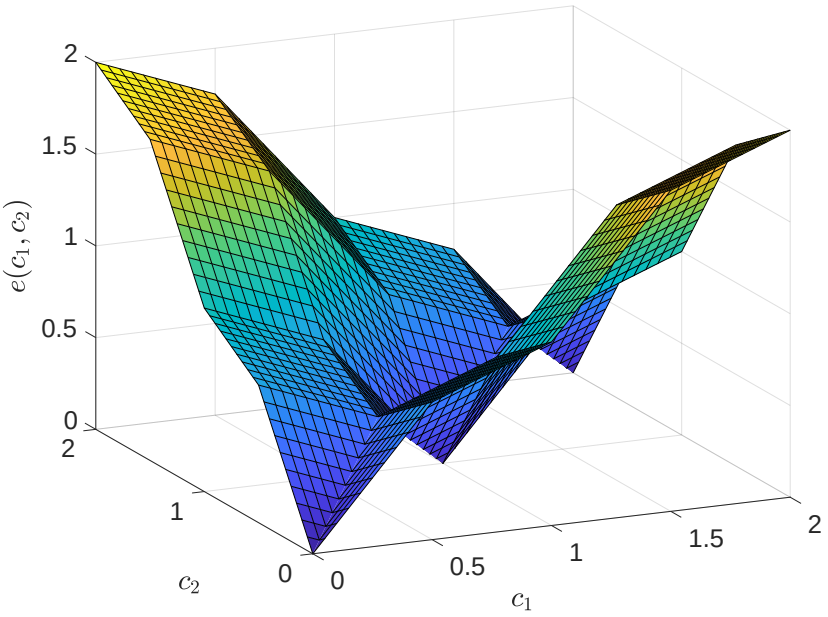

Figure 3.4: For hash codes with digits in base 3, the total error $e(c_1, c_2)$ has its minimums at locations where two digits are equal integers.

Note that, when two entries of two hash codes are different but close to integer values, decreasing

the quantization error will increase the disparity error with the same rate. These relations are visually depicted in figure 3.3. Figure 3.4 show the effect of increasing the base of hash code digits on the total error between soft hash codes. So, to give the priority to disparity error the parameter α should be less than one. In this case, the total error is a real number in the range $[0, K-1]$.

Now, using the total error function we define the likelihood of getting hash codes $\mathbf{h}^{(i)}$ and $\mathbf{h}^{(j)}$ for two images with similarity label s_{ij} as:

$$p(s_{ij}|\mathbf{h}^{(i)}, \mathbf{h}^{(j)}) = \begin{cases} S\left(\gamma\left(\beta - e(\mathbf{h}^{(i)}, \mathbf{h}^{(j)})\right)\right) & s_{ij} = 1 \\ 1 - S\left(\gamma\left(\beta - e(\mathbf{h}^{(i)}, \mathbf{h}^{(j)})\right)\right) & s_{ij} = 0 \end{cases} \tag{3.9}$$

where $S(x) = 1/(1+e^{-x})$ is the sigmoid function and γ and β are two hyper-parameters that control the slope and the inflection point of the sigmoid function. This Bernoulli probability distribution setting leads to likelihood function:

$$\mathcal{L}(W) = (p_{ij})^{s_{ij}}(1-p_{ij})^{(1-s_{ij})} \tag{3.10}$$

and finally to minimize the error we use gradient descent and minimize:

$$l = -\log(\mathcal{L}) = -\log\left((p_{ij})^{s_{ij}}(1-p_{ij})^{(1-s_{ij})}\right) \tag{3.11}$$

Finally, it is worth to mention that proposed hashing layer can be put in the middle of a deep neural network and in this case it can use the back-propagated error to train. However, when the purpose is to use this layer as hash code function, the best results will be generated when it is put as the last layer of the neural network model and is fed by highest level image representation which is

the feature vectors of previous fully connected layer. In this case, we have to use the pairwise loss function to train the entire network.

3.2.5 Hyper-Parameters α, β and γ

The proposed loss function has three hyper-parameters α, β and γ. The first hyper-parameter, α is the weight of quantization error. Since the quantization error acts at opposite of disparity error when two digits of two hash code are different but close to integer numbers, α should be less than 1. Based on experiments, a good initial suggestion for this hyper parameter is 0.5. The hyper-parameter β controls the center point of sigmoid function, and a value of 0.5 makes the argument of sigmoind function symmetric around zero in the range -0.5 to 0.5. Finally, the hyper-parameter γ controls the slope of sigmoid function, and a higher value for this hyper-parameter gives a steeper sigmoid function. A small value for this parameter make the sigmoid function simply close to a line for its argument range, and a value around 10 is a good initial guess for γ.

3.3 Experimental Results

3.3.1 Setup

To evaluate the efficiency of the proposed method against several state of the art and classical methods, we performed extensive experiments with the same three standard datasets NUS-WIDE[2] [10], CIFAR-10[3] [27], and MIRFlickr[4] [22] that are used by DHN [72] and CNNH [63]:

[2]http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

[3]https://www.cs.toronto.edu/~kriz/cifar.html

[4]http://press.liacs.nl/mirflickr

- The NUS-WIDE multi-label dataset which contains of 269,648 image links downloadable from Flickr.com. Since this is public Web image dataset, it loses some of its images every day and currently the actual usable dataset size is only about 223,000 images. The images are annotated with one or more semantic tags associated to 81 different concepts. Following [72] we use only 21 concepts that have more than 5,000 sample images. This results in a subset of 158,529 images.

- The standard CIFAR-10 dataset containing 60,000 color images of size 32×32 pixels in 10 classes.The classes contain equal number of samples. We resize all images to 227×227 and the images are fed to the network without any further preprocessing.

- The MIRFLICKR-25000 dataset consists of 25,000 images collected from the social photography site MIRFlickr. Each image manually annotated with one or more of 38 semantic concepts.

Table 3.1: Mean average precision of Hamming ranking for NUS-WIDE, CIFAR-10, and MIRFlickr image datasets. The highest MAPs for each category are shown in boldface.

Method	NUS-WIDE				CIFAR-10				MIRFlickr			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
LSH	0.403	0.421	0.426	0.441	0.121	0.126	0.120	0.120	0.499	0.513	0.521	0.548
ITQ	0.452	0.468	0.472	0.477	0.162	0.169	0.172	0.175	0.544	0.555	0.560	0.570
SH	0.433	0.426	0.426	0.423	0.131	0.135	0.133	0.130	0.531	0.533	0.531	0.529
BRE	0.485	0.525	0.530	0.544	0.159	0.181	0.193	0.196	0.571	0.592	0.599	0.604
MLH	0.500	0.514	0.520	0.522	0.182	0.195	0.207	0.211	0.610	0.618	0.629	0.634
ITQ-CCA	0.435	0.435	0.435	0.435	0.264	0.282	0.288	0.295	0.513	0.531	0.540	0.555
KSH	0.556	0.572	0.581	0.588	0.303	0.337	0.346	0.356	0.690	0.702	0.702	0.706
CNNH	0.611	0.618	0.625	0.608	0.429	0.511	0.509	0.522	0.732	0.734	0.741	0.740
CNNH*	0.617	0.663	0.657	0.688	0.484	0.476	0.472	0.489	0.749	0.761	0.768	0.776
DNNH	0.674	0.697	0.713	0.715	0.552	0.566	0.558	0.581	0.783	0.789	0.791	0.802
DHN	0.708	0.735	0.748	0.758	0.555	0.594	0.603	0.621	0.810	0.828	0.829	0.841
MaxHash (Ours)	**0.759**	**0.784**	**0.793**	**0.797**	**0.753**	**0.767**	**0.763**	**0.765**	**0.837**	**0.854**	**0.862**	**0.868**

We use the same protocol as in [72] and [63] to have a fair comparison with other algorithms. Hence, for CIFAR-10 dataset we used a set of 1,000 images as test query set and a set of 5,000 images as training set and the remaining images are used as the database. Both training and query sets have equal numbers of images from each class. For NUS-WIDE dataset the test query set has 2,100 images and the training set has 10,500 images with equal number of images from each class, the rest of the dataset is used as database. Finally, for MIRFlickr dataset the test query set consist of 1,000 and the training set has 4,000 randomly selected images as in [72] and the remaining 20,000 images make the database. All images are resized to 227×227 pixels, then the mean value of three channels is subtracted and the resulting images are used for training and evaluation without any further preprocessing.

To evaluate the quality of proposed hashing algorithm, we adopt four evaluation metrics that are widely used in previous work like [72] and [63]: the mean average precision at 5,000 top returned samples, the precision recall curves, precision curves within Hamming distance 2, and precision curves with respect to different number of top returned samples. We compare these retrieval performance metrics of proposed method to eleven state-of-the-art and classical hashing methods. Three of these methods, LSH [14], SH [62], and ITQ [15] are unsupervised and eight other method DHN [72], DNNH [31], CNNH [63], CNNH* [31], KSH [37], MLH [44], BRE [29], and ITQ-CCA [15] are supervised.

The similarity labels for training the model and for all different evaluation methods mentioned above are constructed from image labels. Two images $x^{(i)}$ and $x^{(j)}$ are considered similar and $s_{ij} = 1$ if they share at least one label, otherwise the images are considered dissimilar and $s_{ij} = 0$.

We implement our method based on the Theano [59] Python library[5] and used AlexNet [28] architecture to have the quality of feature vectors and network strength equal to [72] for a fair com-

[5]http://deeplearning.net/software/theano/

parison. We trained the proposed hashing method and fine tuned the model simultaneously using stochastic gradient descent (SGD) with 0.9 momentum, 0.001 weight decay, and step decay learning rate annealing.

3.3.2 Results

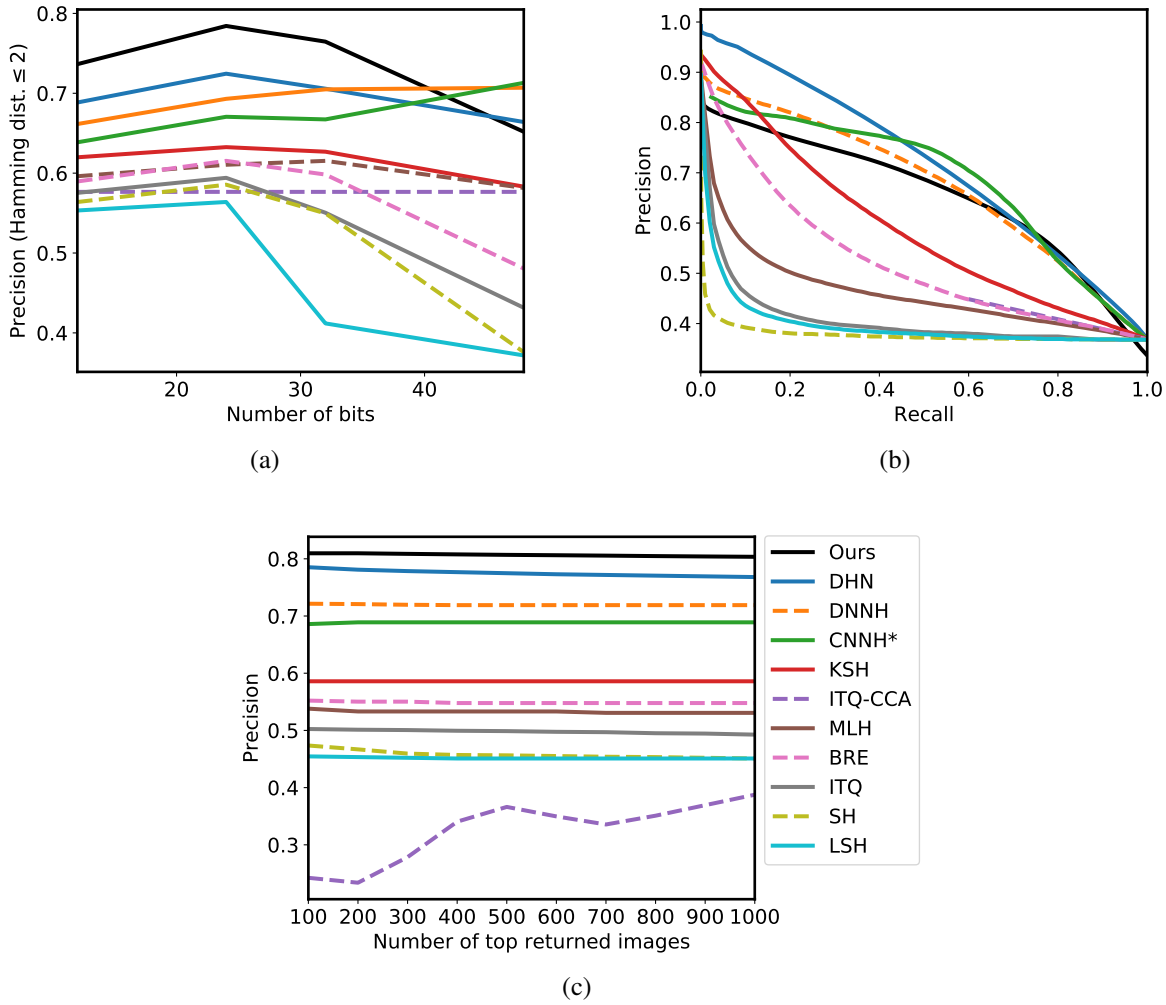

Figure 3.5: Comparisons with state-of-the-art approaches on NUS-WIDE dataset. (a) Precision within Hamming radius 2. (b) Precision-Recall curves with 48-bits. (c) Precision curves with 48-bits with respect to different number of top returned samples.

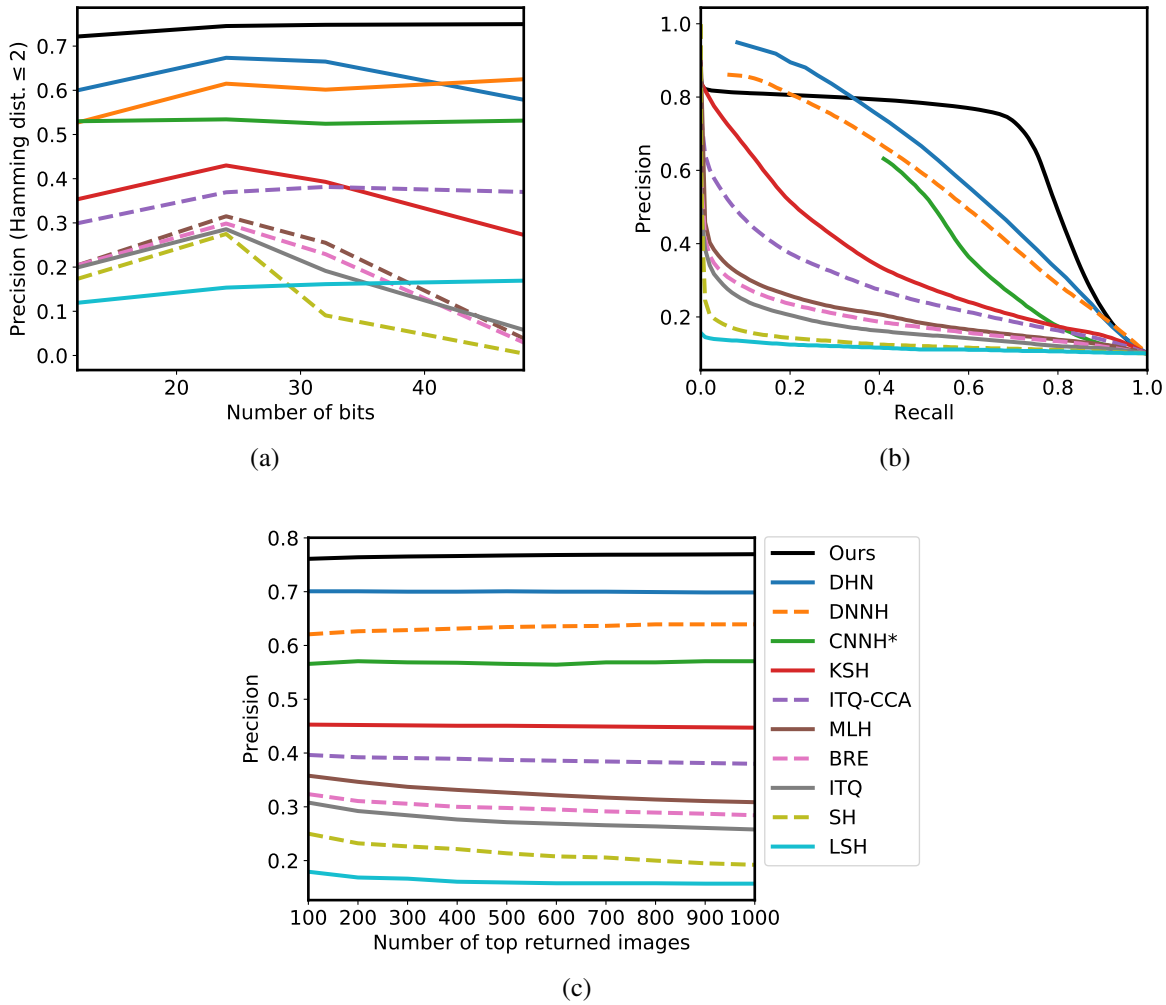

Figure 3.6: Comparisons with state-of-the-art approaches on CIFAR-10 dataset. (a) Precision within Hamming radius 2. (b) Precision-Recall curves with 48-bits. (c) Precision curves with 48-bits with respect to different number of top returned samples.

Table 3.1 shows the mean average precision for the three above mentioned datasets for all methods. Our proposed method substantially outperforms all the comparison methods in all cases. For NUS-WIDE dataset, we achieve mean average precision improvements of 5.1% for hash code length 12 bits, 4.9% for hash code length 24 bits, 4.5% for hash code length 32 bits and 3.9% for hash code length 48 bits compared to the best baseline. For CIFAR10 the improvements are 19.8%, 17.3%, 16% and 14.4% respectively, and for MIRFlickr the boosts in MAP are 2.7%, 2.6%, 3.3% and

2.7% for different hash code lengths. The results for LSH [14], SH, ITQ, DHN, DNNH, CNNH, CNNH*, KSH [37], MLH, BRE, and ITQ-CCA are directly reported from [72]. In all cases α, β and γ are set to 0.5, 0.4, and 40 respectively.

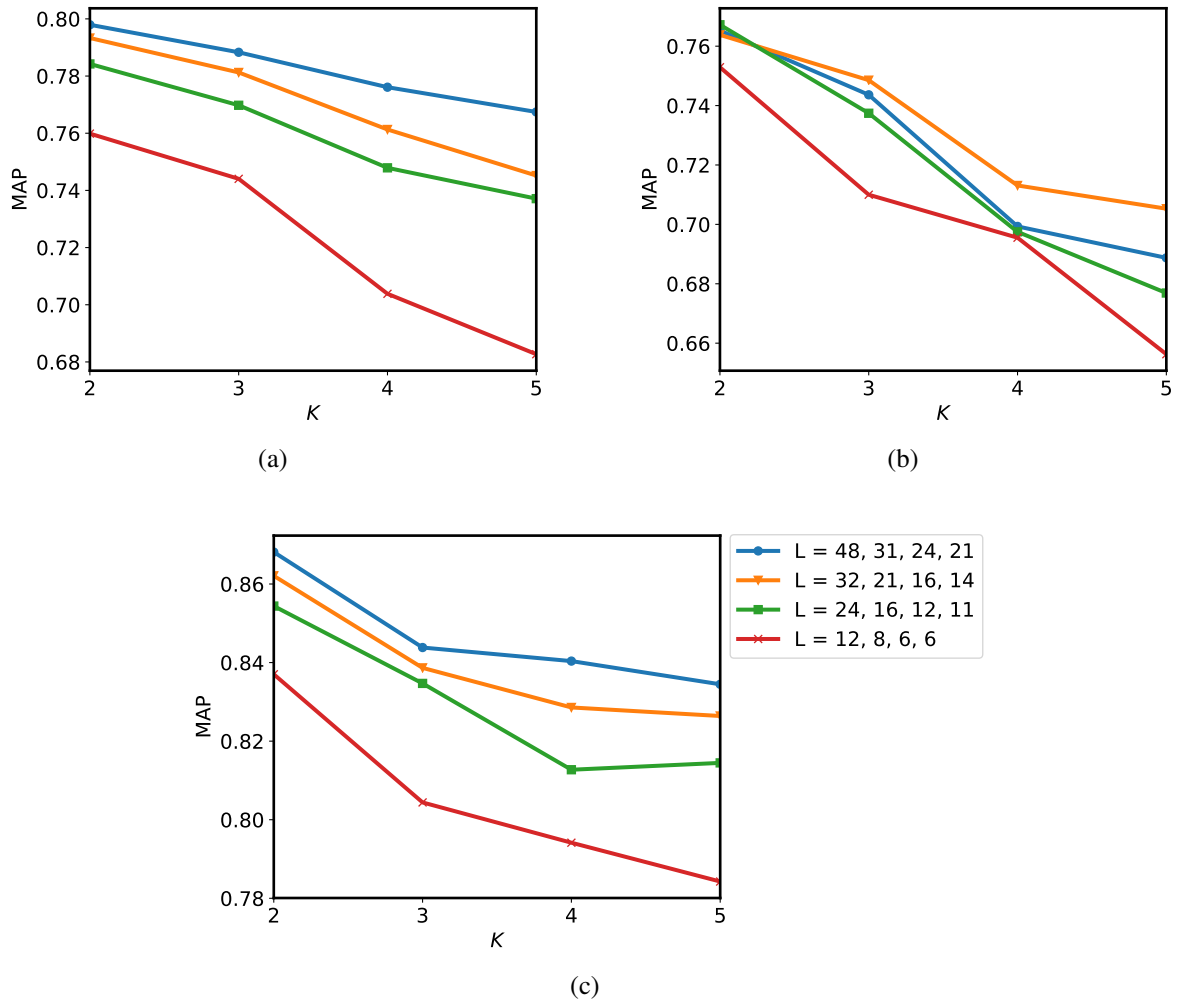

Figure 3.7: The effect of hash code base K on the mean average precision for the three datasets: (a) NUS-WIDE, (b) CIFAR-10, and (c) MIRFlickr. In all cases when we increase the base and decrease the hash code length the precision of hash code decreases.

As shown in Figures 3(a) and 4(a), the proposed method achieves the highest precision within Hamming radius 2 on NUS-WIDE dataset for hash code lengths 12, 24, and 32 bits, and on CIFAR-10 dataset for all hash code lengths. The importance of this performance metric comes from the

efficient retrieval with binary hash codes since the time complexity of this Hamming ranking for each query is $O(1)$.

The performance in terms of precision-recall curves are shown in Figures 3(b) and 4(b). These curves show that our method resists increase in recall and tends to keeps high precision for larger values of recall. For NUS-WIDE data set the precision of our method for recall equals one is lower than other methods, and the reason is that many image links in the NUS-WIDE dataset are broken, and the usable dataset size has been reduced to about 158,000 images currently compared to the work done by [72].

Figures 3(c) and 4(c) depict precision curves with respect to different numbers of top retrieved samples which shows that our method have higher precision both for small and large number of retrieved samples. This means that our method behaves more like a perfect document retrieval system.

Finally, we evaluated the effect of the parameter K on the mean average precision. For a binary code of length L we evaluated the method with K-ary code of length $\lfloor \log_2(K^L - 1) \rfloor + 1$. So, as equivalent to binary hash code of length 48, we evaluated the mean average precision for the cases with hash code length and hash code base as follows: $L = 31, K = 3$ and $L = 24, K = 4$, and $L = 21, K = 5$. The other equivalent cases are $L = 32, 21, 16, 14$ and $L = 24, 16, 12, 11$ and $L = 12, 8, 6, 6$ for $K = 2, 3, 4, 5$ respectively. Figure 3.6a, 3.6b, and 3.6c depicts the result of this experiment on the three datasets NUS-WIDE, CIFAR10 and MIRFlickr. The graphs show that in general increasing K reduces the mean average precision, and the bast results are when the hash code base $K = 2$.

3.4 Conclusion

In this chapter, we proposed a novel deep learning based hashing method that integrates image representation and feature learning step with the ranking based hash function into a unified deep convolutional network. This method consists of two parts: a neural network layer that performs ranking based hashing, and a pairwise loss function designed for training the model. Extensive experiments show that this model outperform state of the art and classical hashing methods both learning based and data-independent.

CHAPTER 4: UNSUPERVISED UMAPHASH

4.1 Introduction

The purpose of hashing algorithms for information retrieval is to find a hash function that maps similar items to binary hash codes with small Hamming distances and dissimilar items to binary hash codes with large Hamming distances. This is similar to other techniques in machine learning, notably dimensionality reduction and data visualization. In the dimensionality reduction techniques the aim is to obtain a smaller set of principal variables among all random variables that data is presented in, but retaining the most important variance in the input data. For example, in principal component analyses (PCA) method the original data space is reduced to the space spanned by a few eigenvectors corresponding to the highest variation in the data distribution.

On the other hand, data visualization is the pictorial representation of the data, such that the produced images show the relationships in the represented data. Remarkable methods in this techniques are Isomap [58], SNE [19], t-SNE [48], and the recent UMAP [41]. In this chapter we will present a new method to train hash functions inspired by UMAP. In this method, minimizing the loss function leads to mappings that retain the topological structure of the training datasets.

4.2 Method

We start by describing how our method for training hash functions is used for image similarity search. In brief, the images are processed by a deep neural network to obtain the feature vectors, which are then further processed using a certain number of fully connected layers and a final MaxHash layer [2]. This two-stage architecture is depicted in Figure 4.1. We train the two stages

of the model sequentially. First, we train a generative model in order to use its feature vector generation component. Next, we train the hash function to map the feature vectors to hash codes preserving item similarity in this mapping. To train the hash function, we used a method inspired by the UMAP manifold learning and projection algorithm [41].

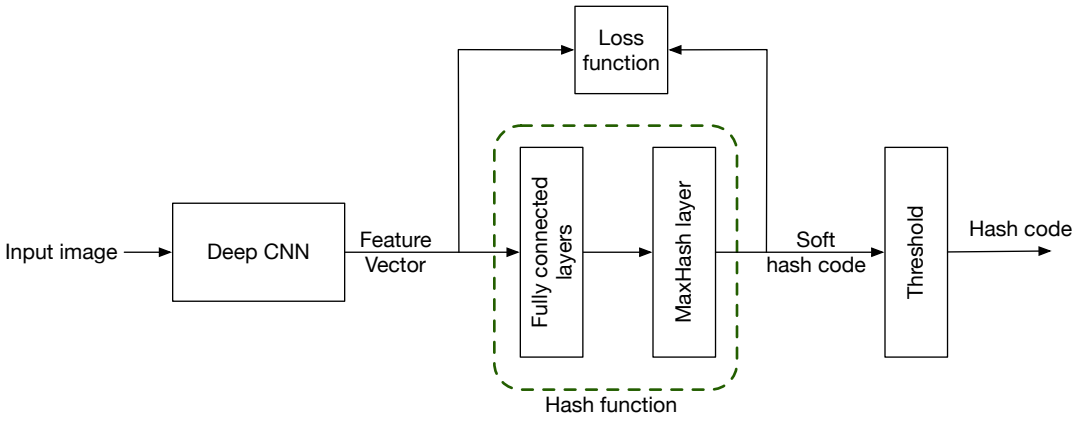

Figure 4.1: The proposed unsupervised hashing network which is comprised of: (1) a deep convolution network, (2) a trainable hash function, and (3) a loss function that penalize the divergence between the probability distribution in the feature vectors space and probability distribution in the hash codes space.

The feature generation subsystem can be a discriminator of a generative adversarial network (GAN) [16] or an encoder of a variational autoencoder (VAE) [26]. In the discriminator case, the output of next to last layer can be used as the feature vector, and in the decoder case the mean parameter of the latent variable. We assume that the deep model non-linearly maps the space of images to a new feature space $\mathcal{F} \subset \mathcal{R}^D$, such that in this new feature space the Euclidean distance between pairs of feature vectors reflects the semantic similarity between the images. In other words, we assume that the model non-linearly maps the complicated manifold images onto a simpler structure in the low-dimensional space $\mathcal{F}$, in which the semantic features of the images become disentangled. Therefore, under this assumption if we train a deep hash function that generates a distribution in the hash code space $\mathcal{H} \subset \{0,1\}^L$ that resembles the distribution of those feature vectors in $\mathcal{F}$,

then this hash function will express the similarity between images in a much simpler binary vector space, and at the same time enjoy the advantages of deep models in producing rich disentangled feature vectors.

We now describe the hashing subsystem. The configuration of the hashing subsystem is depicted in the middle of Figure 4.1. It consists fully connected layers (if any) followed by the trainable hashing layer MaxHash [2]. MaxHash is a trainable variant of MinHash algorithm [6] capable of generating hash codes with digits in arbitrary base K. The i-th element in the output vector $\mathbf{h}$ of MinHash layer for the input vector $\mathbf{x} \in \mathcal{R}^d$ is given by:

$$\mathbf{h}_i = [0, 1, 2, \ldots, K-1]^T \, \sigma\left(\mathbf{W}_i \mathbf{x}\right) \tag{4.1}$$

where $\mathbf{W}_i$ is a $K \times d$ projection matrix, and $\sigma(\cdot)$ is the softmax function, and $\mathbf{h}_i$ is a real value between zero and $K-1$, with the tendency to be closer to integer numbers. We call the output of MaxHash layer "soft hash code" to distinguish it from the final hash code after thresholding. In this work, we used MaxHash with $K = 2$ in order to directly obtain a binary hash code without any further processing.

We now describe our novel loss function for training the hashing subsystem. We are given a set of feature vectors $\{v_1, v_2, \ldots, v_N\} \subset \mathcal{F}$ and seek to obtain a corresponding collection of hash codes $\{c_1, c_2, \ldots, c_N\} \subset \mathcal{H}$ such that the topological structure of the feature vectors in $\mathcal{F}$ and that of the hash codes in $\mathcal{H}$ are as similar as possible. To achieve this goal, we need construct a suitable loss function that quantifies how dissimilar the topological structures are in the spaces $\mathcal{F}$ and $\mathcal{H}$.

For each feature vector v_i, we determine the approximate k nearest neighbors $\{v_{i,1}, v_{i,2}, \ldots, v_{i,k}\}$ with respect to the Euclidean metric, where k is a hyper-parameter chosen between 5 and 100.

We compute the distance between the feature vector v_i and its closest neighbor, ρ_i:

$$\rho_i = \min_{1 \leq j \leq k} \|v_i - v_{i,j}\|_2 \tag{4.2}$$

Using binary search, we determine a normalization factor σ_i such that the following equality holds:

$$\sum_{j=1}^{k} \exp\left(-\frac{\|v_i - v_{i,j}\|_2 - \rho_i}{\sigma_i}\right) = \log_2(k) \tag{4.3}$$

We define the non-symmetric distance metric $d_{i|j}$ between any pair of different feature vectors v_i and v_j as follows:

$$d_{i|j} = \frac{\|v_i - v_j\|_2 - \rho_i}{\sigma_i} \tag{4.4}$$

This normalization gives us a local distance metric $d_{i|\cdot}$ for each feature vector v_i that captures the distribution of its k closest neighbors.

Now, we define a weighted directed graph with feature vectors as vertices and directed edges with weights $p_{i|j}$ defined by

$$p_{i|j} = \exp\left(-d_{i|j}\right) \tag{4.5}$$

Finally, we define a weighted undirected graph with feature vectors as vertices and edges with

weight p_{ij} defined by

$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}\, p_{j|i} \tag{4.6}$$

The arguments in the UMAP paper [41] show that this weighted graph gives an approximate topological representation of the collection of the feature vectors v_i.

Given a collection of hash codes $c_1, \ldots, c_n$, we construct an undirected weighted graph with hash codes as vertices and weights q_{ij} given by

$$q_{ij} = \left(1 + a\, \|c_i - c_j\|_1^{2b}\right)^{-1} \tag{4.7}$$

The parameters a and b in equation 4.7 depend on the minimum distance m and spread s, which are two hyperparameters in the UMAP method. The parameters a and b are chosen by fitting the function $\left(1 + a\, x^{2b}\right)^{-1}$ to the function $\exp(-(x - m)/s)$ for some values x in the interval $[0, 3s]$.

Now, the task is to make the topological representation of the hash codes resembles that of the features vectors. This can be accomplished by minimizing the fuzzy set cross entropy of the edges weights p_{ij} and q_{ij}

$$\mathcal{L}_{\text{map}} = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) + (1 - p_{ij}) \log\left(\frac{1 - p_{ij}}{1 - q_{ij}}\right) \tag{4.8}$$

The other requirement for the quality of soft hash codes is to minimize the quantization error for

each bit of the soft hash code c_i which is given by:

$$\mathcal{L}_{\text{quant}} = \sum_{i=1}^{n} \sum_{\ell=1}^{L} |c_i(\ell) - \lfloor c_i(\ell) \rceil| \tag{4.9}$$

In equation 4.9, $\lfloor . \rceil$ denotes the nearest integer function, and $c_i(\ell)$ is the ℓ-th bit of the hash code of item i.

Finally, combining objectives (4.8) and (4.9), we define the total loss function $\mathcal{L}$ as:

$$\mathcal{L} = \mathcal{L}_{\text{map}} + \mathcal{L}_{\text{quant}} \tag{4.10}$$

By minimizing the loss function (4.10) the soft hash code embedding is pushed to resemble the topological structure of manifold of feature vectors while having small quantization error.

4.3 Experimental Results

4.3.1 Setup

To verify the efficiency of our algorithm, we carry out different experiments. For all experiments, we first trained a slightly modified Improved-GAN [50] network in unsupervised setting. Then, we pass the training images through the discriminator of this network and extract the output of the next to the last layer as their corresponding feature vectors (deep CNN part in Figure 4.1). Finally, we train the hash function which is a composition of two fully connected layers followed by a MaxHash layer using these feature vectors (hash function part in Figure 4.1). We used the following datasets for our experiments:

- MNIST dataset[1] includes 70,000 gray scale images of size 28×28 of hand written digits across 10 classes. For this dataset, we replace the pooling layer of Improved-GAN with a fully connected layer with 150 units, employ weight normalization [51], and utilize sigmoid function as the activation function for the last layer of generator. The images are normalized to the range of $[0, 1]$ and resized to 32×32. We follow the standard protocol for evaluating algorithms with this dataset and randomly sample 1000 images with equal items from each class as the query set and the remaining data samples as the database.

- CIFAR-10 dataset[2] dataset contains 60,000 colored images of size 32×32 pixels in 10 classes. The classes contain equal number of samples. As for MNIST data set, we use the discriminator of a modified version of Improved-GAN as feature extraction model. We remove the pooling layer, utilize tanh function as the activation function for the last layer of generator, and to stabilize the training use weight normalization. The images of this dataset are normalized to the range of $[-1, 1]$.

- STL-10[3] that has 13,000 labeled colored images of size 96×96 pixels in 10 classes with equal items per class and 100,000 unlabeled images of the same size. We resize images to 32×32 and normalize them to the range of $[-1, 1]$. We use the same Improved-GAN structure as with CIFAR-10 dataset.

- USPS Handwritten Digits[4] is a dataset that includes 11,000 grayscale images of size 16×16 pixels. The images are of handwritten digits with 1,100 examples of each class. Images are resized to 32×32 and normalized, and we use the same Improved-GAN structure as with MNIST dataset.

---

[1] http://yann.lecun.com/exdb/mnist/

[2] https://www.cs.toronto.edu/~kriz/cifar.html

[3] https://cs.stanford.edu/~acoates/stl10/

[4] https://cs.nyu.edu/~roweis/data.html

Table 4.1: Image retrieval results (mAP and mAP@1000) on CIFAR-10 and MNIST datasets, for hash code lengths 16, 32 and 64. The results of alternative models are reported from [13].

Method	Dataset	CIFAR-10						MNIST					
	Metric	mAP (%)			mAP@1000 (%)			mAP (%)			mAP@1000 (%)		
	Code Length	16	32	64	16	32	64	16	32	64	16	32	64
KMH		13.59	13.93	14.46	24.08	23.56	25.19	32.12	33.29	35.78	59.12	70.32	67.62
SphH		13.98	14.58	15.38	24.52	24.16	26.09	25.81	30.77	34.75	52.97	65.45	65.45
SpeH		12.55	12.42	12.56	22.10	21.79	21.97	26.64	25.72	24.10	59.72	64.37	67.60
PCAH		12.91	12.60	12.10	21.52	21.62	20.54	27.33	24.85	21.47	60.98	64.47	63.31
LSH		12.55	13.76	15.07	12.63	16.31	18.00	20.88	25.83	31.71	42.10	50.45	66.23
ITO		15.67	16.20	16.64	26.71	27.41	28.93	41.18	43.82	45.37	70.06	76.86	80.23
DH		16.17	16.62	16.96	-	-	-	43.14	44.97	46.74	-	-	-
DAR		16.82	17.01	17.21	-	-	-	-	-	-	-	-	-
HashGAN		29.94	31.47	32.53	44.65	46.34	48.12	91.13	92.70	**93.93**	94.31	95.48	96.37
UMAPHash (Ours)		**38.53**	**40.65**	**43.55**	**52.41**	**54.38**	**55.39**	**93.66**	**93.40**	92.66	**96.77**	**97.15**	**97.20**

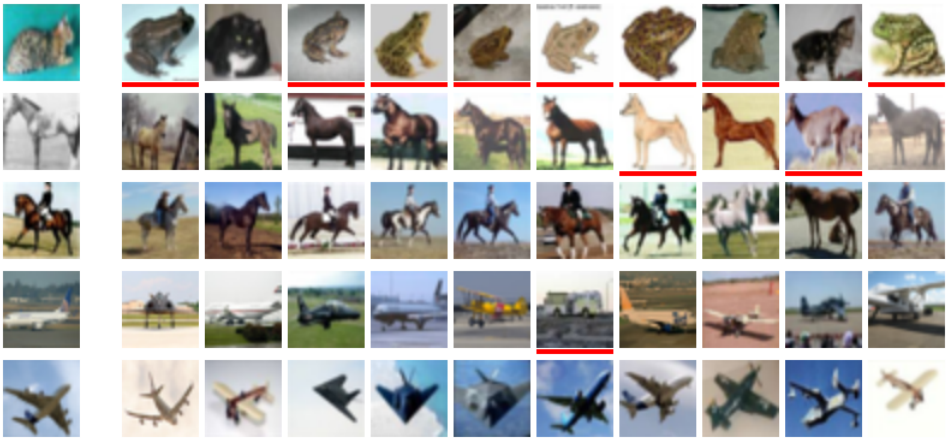

Figure 4.2: Top 10 retrieved images for query data on CIFAR-10 dataset with 64 bits hash codes. In each row, the image on the left side is the query image, and retrieved images with red underline have different labels than the query image.

We implement the algorithm in Tensorflow and use a Tesla V100 GPU for training and tests. The source code to reproduce the results of experiments is available on GitHub.

4.3.2 Image Retrieval

Firstly, we present the results of application of UmapHash for image retrieval. We compare our method with several unsupervised methods using precision at top 1000, mean average precision (mAP), and mean average precision at top 1000, and precision-recall curves. We also show the top ten retrieved images for some samples to demonstrate the quality of data retrieval.

Table 4.1 shows the mean average precision and mean average precision at 1000 for hash code of length 16, 32, and 64 bits. The unsupervised hash functions including K-means hashing (KMH) [17], spherical hashing (SphH) [18], spectral hashing (SpeH) [62], PCA-based hashing (PCAH) [61], locality sensitive hashing (LSH) [14], iterative quantization (ITQ) [15], deep hashing (DH) [35], and discriminative attributes representations (DAR) [20].

Figure 4.3: Top 10 retrieved images for different images for digit 8 in MNIST dataset with 64 bits hash codes. In each row, the image on the left side is the query image, and retrieved images with red underline have different labels than the query image.

The precision@1000 which is the fraction of true positive retrieved images from the top 1000 retrieved images in the database is shown in Table 4.2. As the table shows our method outperforms all other algorithms.

Figures 4.2 and Figure 4.3 show the top 10 retrieved images for few query samples from CIFAR-10 and MNIST datasets respectively. Note how the system return digits similar to the query image shape regardless of its label. In the last row the query is an image of 8, but since it is very similar to a 9, the images with closest hash code are all images of digit 9.

4.3.3 Raw Feature Vectors

When feature extraction/hash function is used for image retrieval, the expectation is that passing the feature vectors through additional hashing layers will reduce the accuracy in the retrieved images. However, experiments show the contrary. If we use the Euclidean distance of raw feature vectors for image retrieval, as we can see in Table 4.4, for MNIST dataset mAP is 80.04% and mAP@1000

is 96.71%, and for CIFAR-10 dataset mAP is 37.37% and mAP@1000 is 55.24%, which is lower than when using the hash codes.

Table 4.2: Image retrieval results (precision@1000) of unsupervised hash functions on CIFAR-10 and MNIST datasets, when the number of hash bits are 16, 32 and 64.

	CIFAR-10			MNIST		
	precision@1000 (%)			precision@1000 (%)		
Method	16	32	64	16	32	64
KMH	18.83	19.72	20.16	51.08	53.82	54.13
SphH	18.90	20.91	23.25	46.31	54.74	62.50
SpeH	18.83	19.72	20.16	51.08	53.75	54.13
PCAH	18.89	19.35	18.73	51.79	51.90	48.36
LSH	16.21	19.10	22.25	31.95	45.05	55.31
ITQ	22.46	25.30	27.09	61.94	68.80	71.00
DH	16.17	16.62	16.96	-	-	-
DAR	24.54	26.62	28.06	-	-	-
HashGAN	44.65	46.34	48.12	94.31	95.48	96.37
UMAPHash (Ours)	**50.64**	**52.13**	**53.32**	**96.47**	**96.98**	**97.01**

Our explanation is that, the topological similarity of feature vectors defines the similarity of images better than the simple Euclidean distance between their feature vectors, especially for distant data points. As Figure 4.5 shows, the Euclidean distance between two points A and B is less than the distance from A to C, which is misleading. Since UMAP Hash maps the topological structure of the dataset to the binary code space, it is filtering out some of the Euclidean distance information that does not reflect the local neighborhood structure for every item, therefore improving the retrieval accuracy. The substantial difference of mAP when sorting the entire database, 80.04% vs 92.66% and 37.37% vs 43.55%, confirms this explanation.

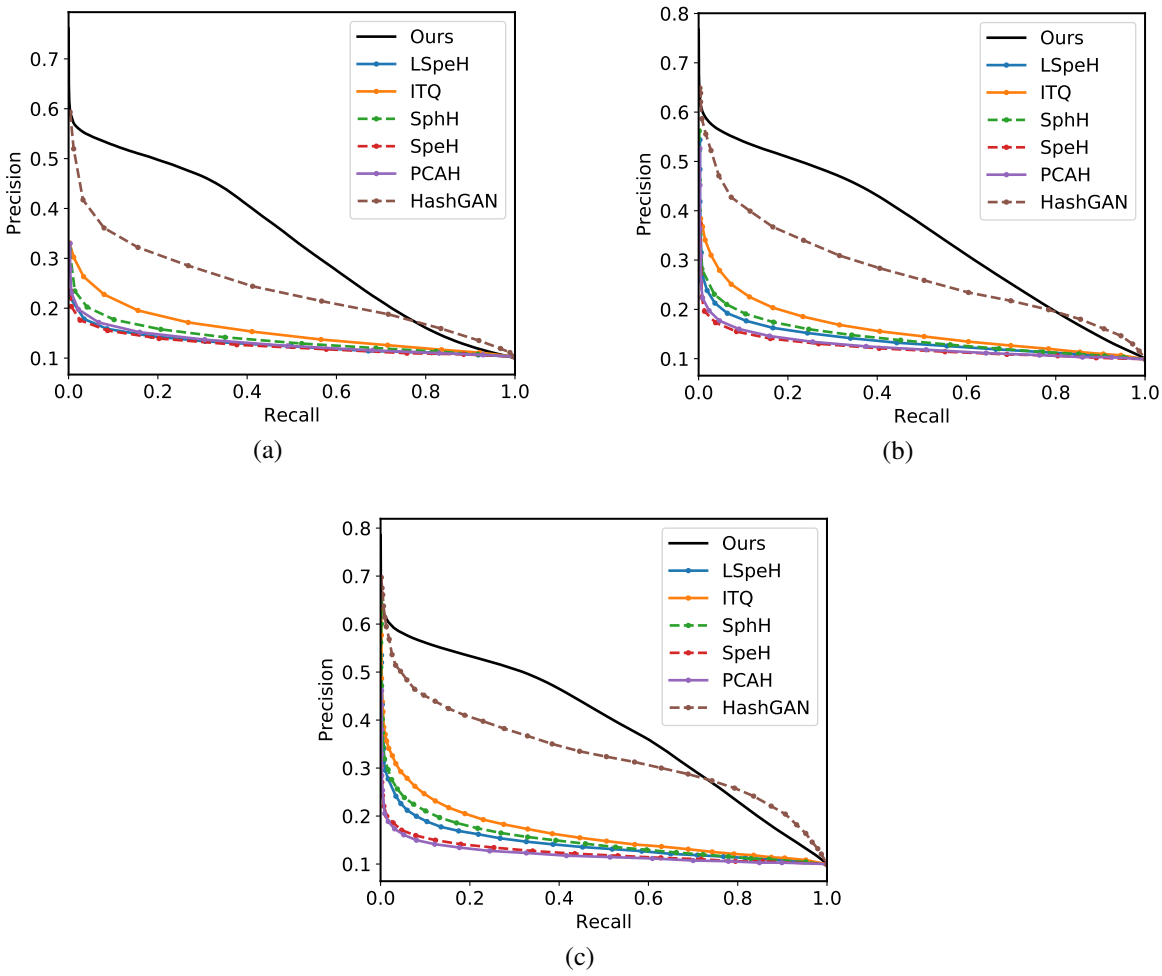

Figure 4.4: Precision-Recall curves compared with state-of-the-art approaches on CIFAR-10 dataset with hash codes of length (a) 16, (b) 32, and (c) 64 bits.

4.3.4 Clustering

To evaluate the quality of our hash method, we also used the hash codes with K-mean algorithm to cluster the images. We compare the performance with K-means, normalized cuts (N-Cuts) [53], deep embedded clustering (DEC) [64], joint unsupervised learning (JULE-RC) [66], spectral embedded clustering (SEC) [43], large-scale spectral clustering (SC-LS) [9], and HashGAN [13] hashing method. Table 4.4 shows the results in normalized mutual information (NMI), and accu-

racy.

NMI gives the normalized mutual dependence between the predicted and true labels, and accuracy is the accuracy of predicted labels after matching the predicted labels and the true labels using the Hungarian method for the assignment problem. Table 4.4 shows that the generated hash codes are performing well for clustering purpose, and give superior or competitive results.

Table 4.3: Image retrieval results (mAP and mAP@1000) by the real valued feature vectors (v columns) and binary hash code of length 64 (c columns) on CIFAR-10 and MNIST datasets.

Dataset	mAP (%)		mAP@1000 (%)	
	v	c	v	c
MNIST	80.04	92.66	96.71	97.20
CIFAR-10	37.37	43.55	55.24	55.39

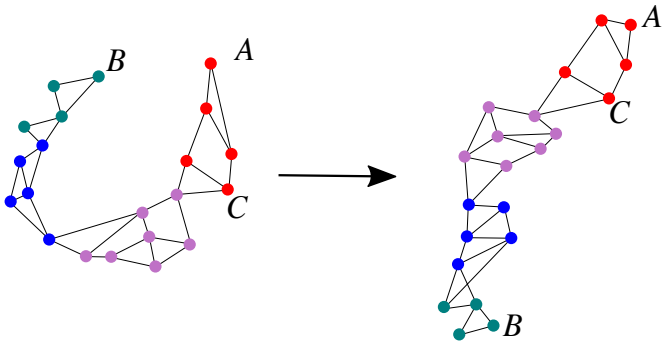

Figure 4.5: The mapping retains the local neighborhood structure of input space in the target space.

4.4 Conclusion

In this chapter, we presented UMAP Hash, a novel unsupervised learning based hashing method that maps the feature vectors of images to binary hash codes. This method employs a MaxHash layer preceded by one or two fully connected layers to map the feature vectors to the target space.

In the training phase, the loss function tunes the hash network such that the topological similarity of feature vector neighborhood is retained in the target binary space. UMAP Hash superior or competitive results compared to baseline and state of the art learning based and data-independent hashing methods according to different standard experiments.

Table 4.4: Accuracy (Acc) and Normalized Mutual Information (NMI) of clustering using the binary hash code and other methods on MNIST, USPS, and STL-10 datasets.

	MNIST		USPS		STL-10	
Method	NMI	Acc	NMI	Acc	NMI	Acc
K-means	0.500	0.534	0.450	0.460	0.209	0.284
N-Cuts	0.411	0.327	0.675	0.314	-	-
SC-LS	0.706	0.714	0.681	0.659	-	-
AC-PIC	0.017	0.115	0.840	0.855	-	-
SEC	0.779	0.804	0.511	0.544	0.245	0.307
LDMGI	0.802	0.842	0.563	0.580	0.260	0.331
NMF-D	0.152	0.175	0.287	0.382	-	-
DEC	0.816	0.844	0.586	0.619	0.284	0.359
JULE-RC	0.913	0.964	0.913	0.950	-	-
DEPICT	0.917	0.965	**0.927**	**0.964**	0.303	0.371
HashGAN	0.913	0.965	0.920	0.958	0.316	**0.394**
UMAPHash (Ours)	**0.941**	**0.976**	0.876	0.934	**0.342**	0.374

CHAPTER 5: HASHING FOR RAPID FACE RECOGNITION

5.1 Introduction

Face recognition is an important method for identification and authentication in many security, financial and personal applications. The popularity of this machine vision research field started with the introduction of Eigenface approach [60] which projects the face images onto a feature space that spans the substantial variations of face image dataset. [1]

The traditional face recognition methods attempted to solve the problem by employing handcrafted features such as Gabor [36], Scale Invariant Feature Transform (SIFT) [39], and Local Binary Patterns [1]. These feature vector are used to classify items using classifiers such as Support Vector Machine (SVM) [11]. A drawback of these methods is that they aimed to present solutions for a few aspect of facial changes such as illuminations, expressions, or poses, and fail to deal with the uncontrolled facial changes that are not in accordance with the prior assumptions. Moreover, face images have large similarity with each other and the differences between face images are usually subtle. This means that face images form a very dense cluster in the data space and this makes it hard for traditional pattern recognition approaches to give accurate recognition systems [42].

The advances in deep convolutional neural networks [28] made it possible to learn a cascade of multiple layers for feature extraction and transformation. These models learn multiple levels of representations that express different levels of abstraction. In these deep model, the bottom layers simply learn features such as Gabor and SIFT which are designed over years of research, and the later layers learn higher level abstractions. High level representations are then used for detecting and classifying the underlying patterns. However, since optimizing parameters of a large model

[1]

to learn a multilevel representations of a dataset from scratch requires millions of training items, the transfer learning method is efficiently utilized to apply previously learned domain of a relevant visual recognition problem to the new data domains. This usually done either by fine tuning a model with the new dataset, or by using an entire or part of a trained model as a subsystem of a new model and to train the rest of the model accordingly. In this chapter, we will use a trained model as a feature extractor for our system.

Face recognition application can be put in two categories: face verification and face identification. Face verification applications aim to determine if two face images belong to the same subject or not. On the other hand, in face identification there is a set of known subjects called the gallery, and given a new query image the goal is to identify a subject in the gallery that is similar to the query image.

Figure 5.1: The proposed face identification method which is comprised of: (1) Multitask cascaded convolutional networks for face detection and face alignment. (2) FaceNet: a standard deep convolutional neural network for learning face features, (3) MaxHash layers for transforming the face representation to binary hash code.

In this chapter, we propose a framework for transforming face images to binary hash codes. Using the Hamming distance between low dimensional hash codes as a proxy for the similarity between high dimensional items reduces the cost of distance computation. In the next step, we use exhaustive search using the L2 distance between the deep feature vectors. This search is performed inside this small set of candidates to find feature vectors that match the query item. Note that, binary hash codes can be used as keys in lookup tables to retrieve a set of candidate images for any query face image. Using the hash codes as keys in lookup tables, it is possible to identify a small set of

candidates for the query image with time complexity $O(1)$.

5.2 Related Work

With the emergence of deep learning algorithms, many deep learning models have been developed for different face recognition tasks. DeepFace [57] that has eleven layers trained on four million face images from about 4000 subjects. This model achieved about 91.4% accuracy on Labeled Faces in the Wild (LFW) dataset [21].

Another successful face recognition and verification model is deep hidden identity features (DeepID) [55] which consist of thirteen neural network layers. This network was trained on Celebrity Faces dataset and achieved accuracy 97.45% on the LFW dataset.

FaceNet [52] is another deep convolutional model which is based on Inception-Resnet-v1 [56] framework. This model hash been trained on different datasets using Triplet Loss and also as a classifier. The accuracy of this model on LFW dataset is 99.63%.

In this chapter, we will use FaceNet as deep face feature extractor, and for fast face image retrieval, we transform the deep face features to the Hamming space.

5.3 Method

Generally, a deep learning based face recognition system consists of three components: face detection, deep feature extraction and face matching. The goal of face detection component is to reduce the effect of illuminations, expressions, occlusions and more importantly poses which is a major difficulty in face recognition problem. This step localize faces in the image and specifies a bounding box around their extent.

The next component, deep feature extraction, utilizes a convolutional neural network trained with massive data to extract a feature vector that spans the data space linearly. The face feature vectors are then used to find the similarity between two feature vectors by a simple distance metrics such as L2 distance or cosine distance. Using the distance between this deep feature vectors, the identity verification is done by a threshold comparison, and identification can be done by using a nearest neighbor search. However, subject identification in a large dataset needs exhaustive search for similar items. Therefore, since the near neighbor search in high dimensional data space using the raw deep feature vectors is impractical, in this chapter, we employ approximate nearest neighbor search using hashing to improve the performance of identification.

In this work, we propose a face identification system which is a composition of (1) multitask cascaded convolutional networks (MTCNN) [70] deep learning model for face detection, (2) the FaceNet [52] model to extract the deep face feature, and finally, (3) MaxHash algorithm [2] to transform the feature vectors to binary hash codes. Figure 5.1 depicts components of this model. [3]

5.3.1 Face Detection

Face detection and alignment are fundamental parts of any face recognition application. Face detection is the process of specifying a bounding box for each face in the image, and face alignment is the process of detecting the location of facial points. Multitask cascaded convolutional networks [70] is a framework that integrates face detection and face alignment in a unified cascaded convolutional network.

MTCCN consist of three stages. The first stage is a shallow CNN which produces candidate windows quickly. A more complex CNN, then, refines the windows to reject most of the non-faces windows. Lastly, a more powerful CNN refine the result and outputs facial landmarks locations. For training this model CelebA [38] and WIDER FACE [67] datasets are used and the training

is done in consecutive phases. We used this model to crop the face images before face feature extraction.

5.3.2 Face Feature Extraction

FaceNet [52] is a deep convolutional network trained to learn a mapping from face images to a compact Euclidean space. It is a unified system for face verification, recognition, and clustering. The model is trained such that the squared L2 distances in the embedding space corresponds to similarity between face images. In other words, the feature vector in the Euclidean space of faces of the same person have small distance while faces of different people have large distances.

FaceNet uses Inception-Resnet-v1 deep model [56] of nineteen layers, and train the output using triplet based loss function. Each triplet is consisting of two matching face images and a non-matching face image. The triplet loss function goal is to separate the matching pairs from non-matching pairs by a large margin. Thus, for an anchor image $\mathbf{x}^a$ of a specific person another positive image of the same person $\mathbf{x}^p$, and a negative image $\mathbf{x}^n$ of a different person are selected. The loss function aims to increase the margin α in this inequality:

$$\|f\left(\mathbf{x}^a\right) - f\left(\mathbf{x}^p\right)\|_2^2 + \alpha < \|f\left(\mathbf{x}^a\right) - f\left(\mathbf{x}^n\right)\|_2^2 \tag{5.1}$$

In equation 5.1, $f(\mathbf{x})$ is the feature vector for input image $\mathbf{x}$. This is equivalent to minimizing the loss function:

$$L = \sum \left[\|f\left(\mathbf{x}^a\right) - f\left(\mathbf{x}^p\right)\|_2^2 - \|f\left(\mathbf{x}^a\right) - f\left(\mathbf{x}^n\right)\|_2^2 + \alpha\right] \tag{5.2}$$

Since many of triplets satisfy equation (5.1) even before training, to increase the speed of the training only those triplets are used that do not satisfy this equation.

We used a FaceNet model pre-trained on MS-Celeb-1M dataset.[2] As a pre-processing, the input images are resized to 160×160 and the pixel values are whitened.

5.4 Supervised MaxHash

Despite the advances in the face recognition methods, using face identification at scale is a challenging problem. Even with highly accurate models such as FaceNet, the problem of search for a person in a large dataset requires comparing the feature vector of the query image with all images in the dataset. To increase the speed of such a face recognition problem, we transform the real values feature vectors to binary hash codes. Using hash codes with lookup tables provide a set of possible items we call candidate set. This set consists of all images in the bucket that have the same key as the query image or have as small Hamming distance with the hash code of the query image. This process speeds up the retrieval system by reducing the need to comparing the raw feature vectors to a small subset of the dataset. In this work, we used MaxHash algorithm [2], which is a learning based variant of WTA [65] hashing algorithm.

MaxHash hashing function transforms the feature vector space to the binary code space such that the distance in the Hamming binary space resemble the similarity between items. It uses the rank correlation measure [65] as the distance between feature vectors. The rank correlation measure for two feature vectors is smaller when the order of their elements are similar. This makes this algorithm robust with respect to noise and small variations which are important characteristics of image data.

[2]MS-Celeb-1M is a dataset of 10 million face images collected from the Internet for the purpose of developing face recognition technologies. This dataset and its website msceleb.org has been terminated recently.

We train the hash function, with face features as input and the similarity label is $s_{ij} = 1$ if two images belong to the same person otherwise the similarity label is $s_{ij} = 0$.

5.5 Supervised MaxHash Experimental Results

We used Labeled Faces in the Wild (LFW) dataset [21] to test our model. This dataset consists of 13,233 images of 5749 people only 1680 of them has two or more pictures. We used 100 pictures of people that have 12 to 32 pictures as query set, and the rest of the dataset for training.

An open source python/tensorflow[3] implementation of Multi-task CNN used for face landmark detection. For deep face feature extraction we used a FaceNet[4] model pre-trained on MS-Celeb-1M dataset.

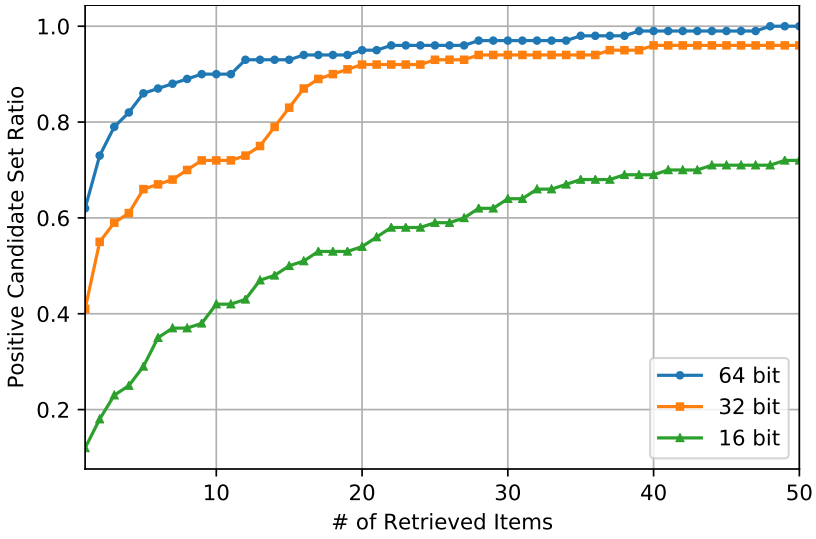

Figure 5.2: The ratio of candidate sets with query person in the candidate set plotted with respect to the length of candidate set retrieved using hash codes for supervised MaxHash model.

[3]https://github.com/davidsandberg/facenet/tree/master/src/align

[4]https://github.com/davidsandberg/facenet

Figure 5.3: The first column on the left side shows the query pictures. The ten picture on the right are the first ten pictures out of fifty retrieved according to the Hamming distance to the query item. Retrieved pictures from wrong persons are underlined with a red line.

We used two fully connect layers of 256 and 512 nodes and a MaxHash [2] layer to transform deep face feature vectors to binary hash codes. The hashing layers are trained with the training portion of LFW dataset. We trained the hash function for binary codes of length 64, 32, and 16 bits. We implement the algorithm in Tensorflow and use a Tesla V100 GPU for training and tests.

First, we use the hash codes to retrieve a set of candidate items from the gallery. This is done by using the Hamming distance to find the near neighbors. Finding the Hamming distance can be done with one XOR operation and a Hamming weight which is a single POPCNT instruction on many processors, while finding the distance of two float vectors of length 128 (FaceNet feature vectors) needs 128 subtraction operations and 128 multiplications and 128 additions. Therefore, using the Hamming distance as a proxy for L2 distance give the algorithm about 192x speed up.

After finding the candidate set, we use the L2 distance to find the best matching face inside this candidate set only. As we see in figure 5.2, for a hash code of length 64, a candidate set of length 48 will have at least one picture for the query person with probability 1. For 32 bit and

16 bit binary hash codes, as figure 5.2 shows, the candidate sets for the same length has lower probability to contain an images of the corresponding person. However with larger candidate sets this probability can be increased, and for 32 bit and 16 bit binary hash codes a candidate set of length 621 and 1461 are also contain a picture of query person with probability 1.

Figures 5.3 and 5.4 show the quality of retrieved images. Each row in the figure 5.3 shows a query on LFW dataset using only the hash codes. The first picture on the left is the query item, and the rest ten pictures are the first ten pictures out of fifty which are retrieved using only the hash codes. When we sort the candidate set of length fifty using the L2 distance of deep face feature vector, we find the best matching candidates as we see in figure 5.4.

Figure 5.4: The first column on the left side shows the query pictures. The ten picture on the right are the first ten pictures out of fifty retrieved according to the Hamming distance to the query item and then sorted with respect to L2 distance. Retrieved pictures with red underline have wrong persons.

5.6 Unsupervised UmapHash

The deep face features form an embedding in the real value 128 dimensional space. This embedding has the desirable property that the L2 distances are related to the similarity between faces. Therefore, the other possibility is to map this deep face feature space to binary hash code space without using the supervision labels such that the relations between data point are preserved in the binary code space.

In this section, we propose to use UmapHash unsupervised hashing method. The framework used is similar to the supervised MaxHash framework. It is composed of a multitask cascaded convolutional network for face detection and face alignment (MTCCN) [70]. Then, the FaceNet feature extraction deep convolutional network [52]. At the end, two fully connected layers of size 256 and 512 and MaxHash layer map the feature vectors to binary hash codes and UmapHash loss function which takes its input from the feature vectors and binary hash codes, see figure 5.5. In this framework, the hashing component is trained by minimizing the UmapHash loss function instead of MaxHash loss function and pairwise similarity labels. The hashing layers are trained with training subset of LFW dataset, and the model is trained for hash code lengths 64, 32, and 16.

5.7 Unsupervised UmapHash Experimental Results

We used the same Labeled Faces in the Wild (LFW) dataset [21] to test our model. As in the supervised setting, we used 100 pictures of people that have 12 to 32 pictures as query set, and the rest of the dataset for training. We used open source python/tensorflow[5] implementation of Multi-task CNN for face landmark detection, and FaceNet[6] model pre-trained on MS-Celeb-1M

[5]https://github.com/davidsandberg/facenet/tree/master/src/align

[6]https://github.com/davidsandberg/facenet

dataset for deep face feature extraction.

Compared to figure 5.2, for a hash code length of length 64 a candidate set of length 10 always have at least one picture for the person in the query image, see figure 5.6. Accordingly, for hash codes of length 32 the candidate set of length 44 and for hash codes of length 16 the candidate set of length 476 has this property. Figure 5.6 also shows that the probability of candidate set to have an image of the query person are higher compared to the MaxHash method in all similar cases.

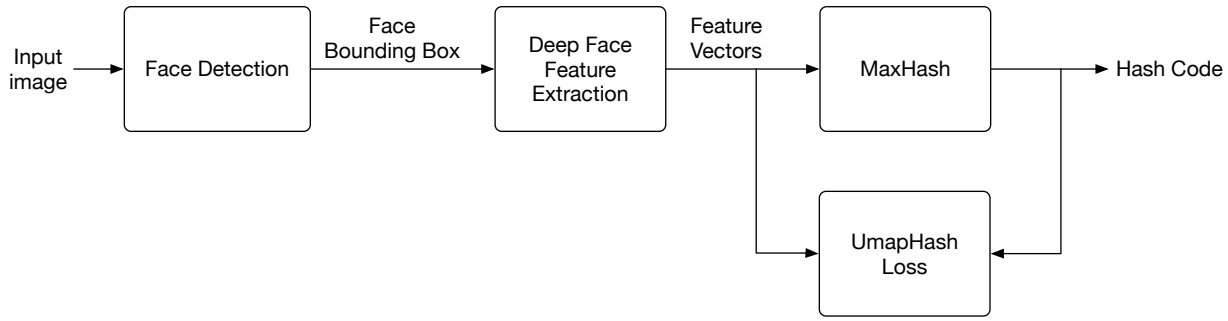

Figure 5.5: Face identification model with unsupervised UmapHash method.

Figures 5.7 and 5.8 show samples of this image retrieval system. As before the the first image in each row is a query images and the next ten images are the closest images according to Hamming distance of the hash code for figure 5.7 and according to L2 distance of the deep face feature vector for figure 5.8.

5.8 Conclusion

In this chapter, we presented a deep learning based hashing approach for face identification. The main purpose of the proposed system is fast face recognition in large datasets of face images. Firstly, given an image a face detection algorithm locate a bounding box for the face images. Then, the truncated, resized, and whitened face image is fed to deep face feature extraction model.

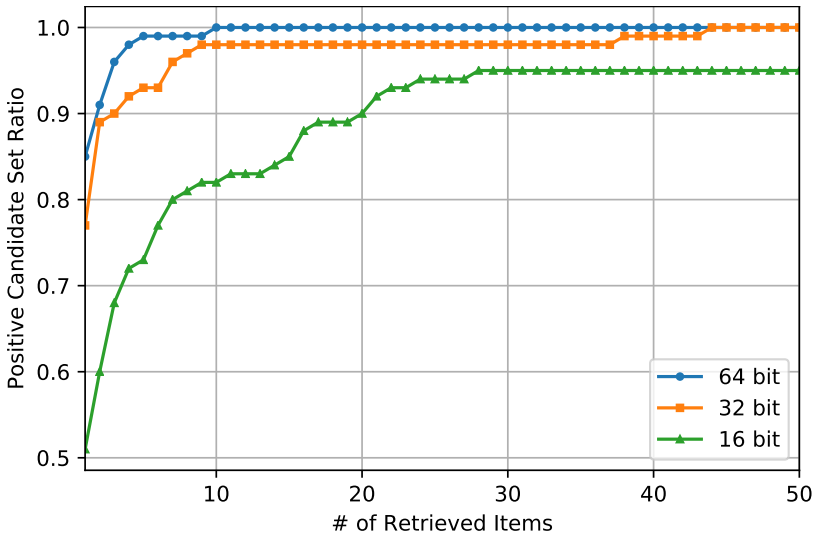

Figure 5.6: The ratio of candidate sets with query person in the candidate set plotted with respect to the length of candidate set retrieved using hash codes for unsupervised UmapHash model.

The output of this model a real valued vector such that the squared L2 distances in the Euclidean space corresponds to similarity between face images. The last component of the system transforms the face feature vector to binary hash code that can be used as a key to retrieve approximate near neighbor items from a lookup table.

Experimental results on LFW dataset show that with a hash code of length 64 a set of only 48 items retrieved using the hash codes always contains the corresponding person for a given query face image.

Figure 5.7: The first column on the left side shows the query pictures. The ten picture on the right are the first ten pictures out of fifty retrieved according to the Hamming distance to the query item. Retrieved pictures from wrong persons are underlined with a red line.

Figure 5.8: The first column on the left side shows the query pictures. The ten picture on the right are the first ten pictures out of fifty retrieved according to the Hamming distance to the query item and then sorted with respect to L2 distance. Retrieved pictures with red underline have wrong persons.

CHAPTER 6: CONCLUSION AND PROPOSED FUTURE WORK

The focus of this dissertation is the design of hashing methods and their application in similar image retrieval and face recognition. Hashing is one of the research fields that facilitates information retrieval through approximate nearest neighbor (ANN) search. Hashing benefits approximate nearest neighbor search in two ways; first, using hash codes as indices to store the items in hash table, which exploits the higher probability of storing similar items in the same hash bucket compared to probability of dissimilar items. Second, using the Hamming distance between low dimensional hash codes as a proxy for the similarity between high dimensional items which reduces the cost of distance computation. Furthermore, since the hash codes are short binary strings, storing the hash codes as keys to the data items benefits from their storage efficiency.

This dissertation focuses on improving the speed and the accuracy of approximate nearest neighbor search by utilizing the Hamming space as an approximation to the space that represent the similarity between data items. In this dissertation, we propose a learning based hash algorithm that utilize ordinal information of feature vectors. We have proposed a novel mathematically differentiable approximation of argmax function for this hash algorithm. It has enabled seamless integration of hash function with deep neural network architecture which can exploit the rich feature vectors generated by convolutional neural networks. We have also proposed a loss function for the case that the hash code is not binary and its entries are digits of arbitrary k-ary base. This loss function has its minima where its pair of arguments are equal and both integer values. Using these new components, we proposed a novel deep learning based hashing function that unifies image representation and feature learning step with the ranking based hash function into a unified deep convolutional network. This method consists of two parts: a neural network layer that performs ranking based hashing, and a pairwise loss function designed for training the model. These parts are put at the top of deep convolutional neural network that generates a real valued feature vector. The training can

be done for the entire model or a pretrained feature generator model can be fine tuned and used as the input to the hashing layers. Extensive experiments show that this model outperform state of the art and classical hashing methods both learning based and data-independent measured according to data retrieval metrics such as the mean average precision (mAP) and precision recall curves.

The second contribution presented in this dissertation is a new loss function to train a differentaible hash function in unsupervised setting. Minimizing this loss function pushes the hash function to map the topological structure of the data points in the input real valued feature space to the output binary space of hash codes. This has been accomplished by using ideas from the Uniform Manifold Approximation and Projection (UMAP) method for dimensionality reduction and visualization. With the help of this loss function, we propose unsupervised UMAP Hash method. It is a neural network that consists of one or two fully connected layers to map the followed by a MaxHash layer. In the training phase, the loss function tunes the hash network such that the topological similarity of feature vector neighborhood is retained in the target binary space. Experimental results show that UMAP Hash is superior or competitive compared to baseline and state of the art learning based and data-independent hashing methods according to different standard experiments.

In the last chapter of this dissertation, we presented a face recognition method based on hash functions. Face recognition refers to identification and authentication of person using face images and it is used in many security, financial and personal applications. We proposed to use the hash codes as a fast method for recognition and retrieval of face images in large face datasets. Unlike the distance of real valued feature vectors, the distance in the Hamming space is computed with a simple XOR operation. In this approach, we generate binary hash codes for deep face features extracted using FaceNet model. We evaluated two hashing methods; supervised MaxHash method and and unsupervised UmapHash method. For a given query item, we generate a candidate set of nearest neighbors according to the Hamming distance metric of the query items to the items in the dataset. Then we sort this candidate set according to the L2 distance to find the closest items to

the query item in this small candidate set. Experiments on LFW dataset show that an image of the corresponding person is always in a small set of candidates items. For supervised MaxHash method a candidate set of 48 items always has an images of the corresponding person. In the case of unsupervised UmapHash method the result are better and a set of only 10 items has this property; moreover, in the case of unsupervised UmapHash method, since the Hamming distance of query item hash code to the hash codes of items in the database are more likely equal to zero, the hash code can be used as keys in hash tables.

APPENDIX A: PERMISSION TO REUSE PUBLISHED MATERIAL

Supervised Max Hashing for Similarity Image Retrieval

Conference Proceedings:
2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)

Author: Ali Al Kobaisi

Publisher: IEEE

Date: Dec 2018

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE

From: **IEEE Support Center** SupportCenter@help.ieee.org
Subject: Copyright permission [Reference #: 200331-001266]
Date: April 1, 2020 at 9:36 AM
To: ali.alkobaisi@knights.ucf.edu

Subject: Copyright permission

Reference #: 200331-001266

Response By Email (Kathy) (04/01/2020 09:36 AM)

Dear Ali:

The IEEE policy reaffirms the principle that authors are free to post their own version of their IEEE periodical or conference articles on their personal Web sites, those of their employers, or their funding agencies for the purpose of meeting public availability requirements prescribed by their funding agencies. Authors may post their version of an article as accepted for publication in an IEEE periodical or conference proceedings. Posting of the final PDF, as published by IEEE *Xplore*®, continues to be prohibited, except for open-access journal articles supported by payment of an article processing charge (APC), whose authors may freely post the final version.

For more info, please see, Author Posting Policy at
https://www.ieee.org/publications/rights/index.html

Warmest Regards,
IEEE Meetings, Conferences, and Events (MCE) | 445 Hoes Lane, Piscataway, NJ 08854 USA
t: +1 732 562-3878 | www.ieeemce.org| ieee-mce@ieee.org
Why Conferences Matter - The Global Technical Community

Auto-Response By (Administrator) (03/31/2020 05:56 PM)

Thank you for contacting IEEE Meetings, Conferences & Events! We have received your email and we have begun our work on your request. Our response may be taking a little longer than usual so if your request is urgent please call IEEE Meetings, Conferences & Events at +1 855 340 4333 (Toll-Free US & Canada) or +1 732 562 3878 (worldwide). For any future communications related to this inquiry please use the reference number which is listed in the email subject line.

We thank you for taking the time to contact us, and we look forward to helping you with your inquiry.

Kind regards,
IEEE Meetings, Conferences, and Events (MCE)

Customer By CSS Email (Patrick Kellenberger) (03/31/2020 05:56 PM)

Dear Ali,

We have added Conference Operations to this email. They will be able to provide you with reuse permission instructions. Thanks!

Regards,
Patrick

On Tue, Mar 31, 2020 at 2:44 PM Ali Al Kobaisi <ali.alkobaisi@knights.ucf.edu> wrote:

> Dear Conference Publishing Services ,

This is Ali Al Kobaisi. I am writing to kindly ask you to give me a permission to reuse my published paper in my dissertation. The permission I want is for the following paper:

Ali Al Kobaisi and Pawel Wocjan, "MaxHash for Fast Face Recognition and Retrieval". In 6th Annual Conference on Computational Science and Computational Intelligence, Las Vegas, Nevada, 5-7 December 2019, pages 652-656, 2019.

Kind Regards,

Ali Al Kobaisi

--
Patrick Kellenberger
Manager
Conference Publishing Services (CPS)
IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1264
Email: pkellenberger@computer.org
Phone: + 1 714 816 2105
Fax: +1 714 761 1784

LIST OF REFERENCES

[1] T. Ahonen, A. Hadid, and M. Pietikäinen. Face description with local binary patterns: Application to face recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(12):2037–2041, 2006.

[2] A. Al-Kobaisi and P. Wocjan. Supervised max hashing for similarity image retrieval. In *17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, December 17-20, 2018*, pages 359–365, 2018.

[3] A. Al-Kobaisi and P. Wocjan. Maxhash for fast face recognition and retrieval. In *6th Annual Conference on Computational Science and Computational Intelligence, 2019*, pages 652–656, 2019.

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

[5] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, pages 21–29. IEEE Computer Society, 1997.

[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[7] R. A. Brualdi. *Index*, page 2. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2006.

[8] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 380–388, 2002.

[9] X. Chen and D. Cai. Large scale spectral clustering with landmark-based representation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011.

[10] T. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. NUS-WIDE: a real-world web image database from national university of singapore. In *Proceedings of the 8th ACM International Conference on Image and Video Retrieval, CIVR 2009, Santorini Island, Greece, July 8-10, 2009*, 2009.

[11] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[12] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 886–893, 2005.

[13] K. G. Dizaji, F. Zheng, N. Sadoughi, Y. Yang, C. Deng, and H. Huang. Unsupervised deep generative adversarial hashing network. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3664–3673, 2018.

[14] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 518–529, 1999.

[15] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, pages 817–824, 2011.

[16] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[17] K. He, F. Wen, and J. Sun. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pages 2938–2945, 2013.

[18] J. Heo, Y. Lee, J. He, S. Chang, and S. Yoon. Spherical hashing. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2957–2964, 2012.

[19] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 833–840, 2002.

[20] C. Huang, C. C. Loy, and X. Tang. Unsupervised learning of discriminative attributes and visual representations. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 5175–5184, 2016.

[21] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[22] M. J. Huiskes and M. S. Lew. The mir flickr retrieval evaluation. In *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*, New York, NY, USA, 2008. ACM.

[23] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613, 1998.

[24] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 24-28, 2000, Athens, Greece*, pages 41–48, 2000.

[25] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 108–116, 2012.

[26] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[27] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[29] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1042–1050, 2009.

[30] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 2130–2137, 2009.

[31] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3270–3278, 2015.

[32] X. Li, G. Lin, C. Shen, A. van den Hengel, and A. R. Dick. Learning hash functions using column generation. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 142–150, 2013.

[33] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 2552–2559, 2013.

[34] M. Lin, Q. Chen, and S. Yan. Network in network. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[35] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 2475–2483, 2015.

[36] C. Liu and H. Wechsler. Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition. *IEEE Trans. Image Processing*, 11(4):467–476, 2002.

[37] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang. Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2074–2081, 2012.

[38] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[39] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.

[40] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.

[41] L. McInnes and J. Healy. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR*, abs/1802.03426, 2018.

[42] C. Nastar and M. Mitschke. Real-time face recognition using feature combination. In *3rd International Conference on Face & Gesture Recognition (FG '98), April 14-16, 1998, Nara, Japan*, pages 312–317, 1998.

[43] F. Nie, Z. Zeng, I. W. Tsang, D. Xu, and C. Zhang. Spectral embedded clustering: A framework for in-sample and out-of-sample spectral clustering. *IEEE Trans. Neural Networks*, 22(11):1796–1808, 2011.

[44] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 353–360, 2011.

[45] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1070–1078, 2012.

[46] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[47] S. M. Omohundro. Efficient algorithms with neural network behavior. *Complex Systems*, 1(2), 1987.

[48] P. E. Rauber, A. X. Falcão, and A. C. Telea. Visualizing time-dependent data using dynamic t-sne. In *Eurographics Conference on Visualization, EuroVis 2016, Short Papers, Groningen, The Netherlands, 6-10 June 2016.*, pages 73–77, 2016.

[49] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009.

[50] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2226–2234, 2016.

[51] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016.

[52] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 815–823, 2015.

[53] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[54] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[55] Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10, 000 classes. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1891–1898, 2014.

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015.

[57] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1701–1708, 2014.

[58] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319, 2000.

[59] Theano Development Team. Theano: A python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, 2016.

[60] M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, Jan. 1991.

[61] J. Wang, O. Kumar, and S. Chang. Semi-supervised hashing for scalable image retrieval. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pages 3424–3431, 2010.

[62] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1753–1760, 2008.

[63] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2156–2162, 2014.

[64] J. Xie, R. B. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 478–487, 2016.

[65] J. Yagnik, D. Strelow, D. A. Ross, and R. Lin. The power of comparative reasoning. In *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 2431–2438, 2011.

[66] J. Yang, D. Parikh, and D. Batra. Joint unsupervised learning of deep representations and image clusters. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 5147–5156, 2016.

[67] S. Yang, P. Luo, C. C. Loy, and X. Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[68] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833, 2014.

[69] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 18–25, 2010.

[70] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Process. Lett.*, 23(10):1499–1503, 2016.

[71] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1556–1564, 2015.

[72] H. Zhu, M. Long, J. Wang, and Y. Cao. Deep hashing network for efficient similarity retrieval. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2415–2421, 2016.