Deep Learning for Approximate Nearest Neighbour Search: A Survey and Future Directions

Mingjie Li, Yuan-Gen Wang, *Senior Member, IEEE*, Peng Zhang, Hanpin Wang, Lisheng Fan, Enxia Li, and Wei Wang

Abstract—Approximate nearest neighbour search (ANNS) in high-dimensional space is an essential and fundamental operation in many applications from many domains such as multimedia database, information retrieval and computer vision. With the rapidly growing volume of data and the dramatically increasing demands of users, traditional heuristic-based ANNS solutions have been facing great challenges in terms of both efficiency and accuracy. Inspired by the recent successes of deep learning in many fields, substantial efforts have been devoted to applying deep learning techniques to ANNS for learning to index and learning to search, resulting in numerous algorithms that achieve state-of-the-art performance compared with conventional methods. In this survey paper, we comprehensively review the different types of deep learning-based ANNS methods according to two learning paradigms: *learning to index* and *learning to search*. We provide a comprehensive overview and analysis of these methods in a systematic manner. Based on the overview, we point out that *end-to-end learning* will be a new and promising research direction for deep learning-based ANNS, i.e., applying deep learning techniques to jointly learn the indexing and searching together, such that the underlying knowledge learned from data can directly contribute to the final searching performance. Finally, we conduct experiments and provide general performance analyses for the representative deep learning-based ANNS algorithms.

Index Terms—Approximate nearest neighbour search, similarity search, high-dimensional space, learning to index, learning to search

1 Introduction

Nearest neighbour search aims at finding the closest items to a query item under a specified distance measure from a reference database, which is a fundamental research problem in many fields such as computer vision, multimedia database, information retrieval and recommendation system. Due to the expensive computation cost of exact nearest neighbour search, many efforts focus on the approximate nearest neighbour search (ANNS) to achieve a good trade-off between the search efficiency and the search accuracy. Over the past decades, a large number of algorithms were proposed to support ANNS in the literature, which can mainly be classified into three categories according to the difference of their indexing paradigms: *Hashing-based*, *Partition-based*, and *Graph-based* approaches. With the growing volume of data and the increasing demands for the querying efficiency and accuracy, traditional ANNS solutions face great challenges. For the hashing-based methods, data-oblivious locality sensitive hash-based (LSH) methods [1], [2], [3], [4], [5], [6], [7] have rigorous theoretical guarantees but with poor practical performance in terms of the search accuracy. For partition-based methods, optimized KD-tree [8], hierarchical k-means tree [9] and random projection tree [10] propose to use heuristic-based strategies to find the data partition, which cannot take advantages of the data distribution to produce good partitions for better performance. In terms of graph-based approaches, such as KGraph [11], [12], [13], Small World Graph [14], [15] and Navigating Spreading-out Graph [16], they mainly focus on the proximity graph construction with special merits in the degree distribution and the graph connectivity. However, their searching stages still adopt simple greedy heuristic, which may easily make the algorithms track into the local optimum.

Deep learning has been enjoying an increasing popularity in the past years due to its significant successes in many domains, including machine learning [17], computer vision [18], natural language processing [19], etc. Thus, to tackle the above challenges in ANNS, tremendous efforts have been devoted to applying deep learning techniques to

- *Mingjie Li, Yuan-Gen Wang, Hanpin Wang, and Lisheng Fan are with Guangzhou University, Guangzhou, Guangdong Province 510006, China. E-mail: {limingjie, wangyg, wanghp, lsfan}@gzhu.edu.cn.*
- *Peng Zhang is with the Shandong University of Science and Technology, Qingdao, Shandong 266590, China. E-mail: pengzhang_skd@sdust.edu.cn.*
- *Enxia Li is with the University of Technology Sydney, Ultimo, NSW 2007, Australia. E-mail: Enxia.Li@student.uts.edu.au.*
- *Wei Wang is with the Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong Province 511466, China. E-mail: weiwcs@ust.hk.*

Manuscript received 21 August 2021; revised 30 August 2022; accepted 30 October 2022. Date of publication 8 November 2022; date of current version 8 August 2023.
The work of Yuan-Gen Wang was supported in part by NSFC under Grant 61872099. The work of Peng Zhang was supported in part by Shandong NSF Project under Grant ZR2021QF017. The work of Lisheng Fan was supported in part by NSFC under Grant 61871139. The work of Wei Wang was supported in part by HKUST(GZ) under Grants G0101000028 and GZU22EG04.
(Corresponding author: Enxia Li.)
Recommended for acceptance by L. Chen.
Digital Object Identifier no. 10.1109/TKDE.2022.3220683

support ANNS for leaning to index (e.g., [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30]) and learning to search (e.g., [31], [32], [33]), to achieve better performance and hence meet the increasing real-world demands. These efforts result in many state-of-the-art deep learning-based ANNS methods. However, to the best of our knowledge, little effort has been made to systematically survey and summarize the differences and connections between these diverse methods.

In this paper, we try to fill this knowledge gap by comprehensively reviewing the deep learning-based ANNS methods. We mainly divide the existing methods into two categories according to two learning paradigms: *learning to index* and *learning to search*. For the learning to index, algorithms in this category mainly use the deep learning techniques to learn data-sensitive index structures to better serve the querying search. For the learning to search, methods in this category employ deep learning techniques to learn a good searching policy on the constructed index structures. Based on the systematic overview and analysis for the current deep learning-based ANNS methods, we point out that *end-to-end learning*, i.e., simultaneously learning the indexing and searching together, is a new and promising research direction for deep learning-based ANNS community. The key motivation is that the underlying knowledge learned from data can directly contribute to the search performance through end-to-end learning. We will elaborate the technique details of our end-to-end learning framework and discuss its underlying issues when applying it to ANNS.

Related Work. There are some surveys in literature that are closely related to ANNS on high dimensional data. The works in [34] and [35] reviewed the learning to hash methods for ANNS according to different similarity preserving schemes. However, deep hashing methods that utilized deep learning techniques for hashing do not receive much attention in these two surveys. Li et al. [13] conducted an experimental survey mainly for heuristic-based ANNS methods as well as learning to hash methods. Wang et al. [36] conducted an experimental evaluation for proximity graph-based ANNS methods. Our work differs from these surveys as we focus on reviewing the deep learning-based ANNS methods, including the methods based on *learning to index* paradigm as well as the methods in *learning to search* paradigm. Note that the deep learning to hash is a only branch of learning to index-based ANNS methods.

The rest of this paper is organized as follows. In Section 2, we introduce the notations and the problem definition. Then, we briefly introduce some deep learning techniques in Section 3. After that, we review and analyze the deep learning-based ANNS methods in terms of learning to index and learning to search paradigms in Sections 4 and 5, respectively. Our end-to-end learning framework for ANNS is presented in Section 6. We provide some discussions about deep learning in Section 7. The performance evaluation study is conducted in Section 8. Finally, we conclude this paper in Section 9.

2 Problem Definition

For the notations, we use bold lower case letters to represent (column) vectors and bold upper case letters to denote matrices. We denote a dataset of N data items as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, where each item $\mathbf{x}_i$ is a d-dimensional vector, i.e., $\mathbf{x}_i \in \mathcal{R}^d$. Given a query item $\mathbf{q} \in \mathcal{R}^d$, the *nearest neighbour search* (NNS) aims at finding an item in $\mathbf{X}$ that is closest to $\mathbf{q}$. The distance between two items $\mathbf{x}$ and $\mathbf{q}$ is the euclidean distance by default, defined as $\|\mathbf{x} - \mathbf{q}\|_2 = \sqrt{\sum_{i=1}^{d}(x_i - q_i)^2}$, which is widely studied in the literature. Other distance measures like cosine similarity and l_1 norm may also be used, which usually depends on the studied tasks. The k-NNS can be accordingly defined, where we need to find k nearest neighbours.

Due to the intrinsic difficulty of the exact nearest neighbor search, most of efforts are devoted to its approximate version. For a query item $\mathbf{q}$, *c-approximate nearest neighbour search* (ANNS) focuses on finding an item $\mathbf{x} \in \mathbf{X}$ such that $\|\mathbf{x} - \mathbf{q}\|_2 \leq c\|\mathbf{x}^* - \mathbf{q}\|_2$, where c is an approximation ratio ($c > 1$) and $\mathbf{x}^*$ is the nearest neighbour for $\mathbf{q}$ in $\mathbf{X}$. Similarly, c–kANNS is to return k items $\mathbf{x}_i \in \mathbf{X}$ ($1 \leq i \leq k$) such that $\|\mathbf{x}_i - \mathbf{q}\|_2 \leq c\|\mathbf{x}_i^* - \mathbf{q}\|_2$, where $\mathbf{x}_i^*$ is the i-th nearest item of $\mathbf{q}$ in $\mathbf{X}$. Randomized hashing methods like LSH-based methods [3], [4], [5], [7] support c-ANNS with theoretical accuracy guarantee. Another version of ANNS, time-constrained ANNS [9], [35], [37], aims to approximate the nearest neighbours by limiting the search time, which has been used in many real-life applications, though it lacks the accuracy guarantee. The goal is to make the search as accurate as possible while using as less querying time as possible, which is also the purpose of deep learning-based ANNS methods. Thus, time-constrained ANNS is the focus of this paper.

The ANNS methods usually first construct an index structure to organize data items and then perform a querying search algorithm based on this index to retrieve the nearest neighbour results for the given queries.

Indexing. In the index construction, a variety of data structures can be adopted to organize the data items (or the representations after learning), such as the linear-based structure like inverted file [38], tree-based structures like binary tree [8], R-tree [4] and B$^+$-tree [39], graph-based structures like relative neighborhood graph [14] and k-nearest neighbor graph [12].

Searching. In the search procedure, the ANNS method will perform a searching algorithm over the constructed index to retrieve the target results. In terms of the linear-based index, the linear scan integrated with a pruning scheme is usually adopted. For the tree-based and graph-based indices, the heuristic-based or learning-based best-first-search (BFS) strategy can be employed.

3 Deep Learning

Deep learning [17] aims to learn high-level feature representations for data by constructing deep neural networks that consist of hierarchical processing layers. It has been successfully applied into many applications from many domains, including the image recognition [64], [65], [66], speech recognition [67], [68], object detection [69], [70], [71], natural language understanding [72], [73], [74] and many other domains like drug discovery [75] and genomics [76].

Since Krizhevsky et al. [18] demonstrated the significant success of deep convolutional neural networks (CNNs) on image classification, deep learning has rapidly become the most popular branch of artificial intelligence. After that, many efforts were devoted to deeper neural networks to seek superior performance, resulting in several very deep CNN-based models such as "GoogleNet" [77] and the "VGGNet" [64]. However, as the depth of neural network increases, the accuracy degradation problem occurs due to the difficulties of optimizing such an extremely deep network. To address the degradation issue, He et al. [78] introduced a deep residual learning framework, called "RestNet", which is substantially deeper but has shown to be much easier to optimize than the previous CNN-based networks, which is mainly due to the use of residual learning strategy. With deeper networks (with a depth of up to 152 layers), ResNet can gain accuracy improvement compared with previous deep networks.

When dealing with the tasks involving sequential inputs like speech and time series, it is recommended to utilize the recurrent neural networks (RNNs). RNNs process an input sequence one element at a time, and the final representations implicitly contain the information of all the previous elements of the sequence. Long short-term memory (LSTM) [79] is a kind of RNNs that was proposed to address the memory issue and is able to remember the input information for a long time. LSTM has become a crucial ingredient in RNN-based network models.

Graph Neural Networks (GNNs) [80], [81], [82] were proposed to learn the representations for graph data. The key point is to aggregate feature information from local graph neighborhoods by using neural networks. The node information can be propagated over graph after some iterations. Thus, GNNs can integrate both the content information and graph topology information, and have been shown to be effective for graph representation learning [83], [84].

Reinforcement learning (RL) [85] is a goal-directed learning paradigm that is widely used for decision-making problems like playing games [86] and Go [87]. RL aims to improve the decision-making ability of models by trial-and-error learning and reward-maximized planning. Currently, RL is usually integrated into other deep learning techniques like CNNs, RNNs and GNNs, to address the routing problem [88], recommendation problem [89] and combinatorial optimization problem [90].

In the following sections, we will demonstrate how the current existing methods use these deep learning techniques to deal with ANNS problem by learning to index and learning to search, respectively. A summary of the representative deep learning-based ANNS methods is reported in Table 1.

4 Learning to Index

The ANNS methods in this category aims at employing deep learning techniques to learn the data-sensitive index by fully exploring the data distribution. And then, the searching algorithm can make use of this learned index to provide a better performance in terms of the storage efficiency, I/O efficiency, CPU time efficiency, and search accuracy. According to the difference of the learned indices, we divide the methods into three sub-categories: *deep learning to hash*, *learning to partition*, and *learning to graph construction* class. Next, we review and analyze the methods in each sub-category in details.

4.1 Deep Learning to Hash

The goal of deep learning to hash methods is to map the data items in high-dimensional space into low-dimensional hash codes, so that the ANNS can be efficiently performed within this low-dimensional hash coding space. The hash functions adopted by these methods are based on deep learning networks in order to maximally preserve the similarities of data items between original space and hash coding space.

After the hash codes for data items are learned, there are two main types of strategies to index these hash codes and perform the querying search. The first one is hash table lookup. The key idea is to construct a hash table consisting of several buckets with each bucket being indexed by a hash code. Then the items sharing with the same hash code are mapped into the same bucket. Given a query, the items lying in the same bucket as that of query will be retrieved as candidates, which is followed by a re-ranking in the original space to output the final results. The performance can be further improved by retrieving more relevant buckets or constructing more independent hash tables, then more candidates can be retrieved and re-ranked. The second strategy is the hash code ranking, which is suitable for the binary hash codes. The key point is to perform an exhaustive search by sorting the distances of the reference items to query item based on their binary hash codes and finally output the top k items. The computation of Hamming distance can be efficiently performed by using the *XOR* operation, which is much faster than the computation of euclidean distance, thus achieving a better trade-off performance in terms of the efficiency and accuracy. To avoid the exhaustive search, some indexing techniques like multi-index hashing [91] and inverted multi-index [38] are adopted to organize the binary codes to further accelerate the search in Hamming space.

A deep learning to hash method needs to address four issues: what neural network architecture (hash function) is employed, what similarity information is extracted from the hash coding space and original space, what loss function is selected, and what optimization technique is utilized to train the neural networks. An illustration for deep hashing-based ANNS methods can be seen in Fig. 1.

Hash Function. The hash function is based on deep neural network, which is formulated as:

$$\mathbf{y} = sign(\mathbf{h} = \mathcal{F}(\mathbf{x}, \boldsymbol{\Theta})), \tag{1}$$

where $\mathcal{F}$ is the neural network with parameters $\boldsymbol{\Theta}$. $\mathbf{h}$ is the real-valued output vector for input item $\mathbf{x}$ after neural network transformation. $sign(z)$ is an element-wise function, where $sign(z) = 1$ if $z \geq 0$ and -1 (equivalently 0) otherwise. Thus, the output $\mathbf{y} \in \{-1, 1\}^t$ is a binary hash code with length t for item $\mathbf{x}$.

The selection of neural network architecture is crucial for deep hashing-based ANNS methods, as it affects the training cost, time cost for hash code generation and the final searching accuracy. The widely studied network models for hashing methods includes deep neural networks (DNNs) [21], deep convolutional neural networks (CNNs) (e.g., AlexNet [18], VGGNet [64], GoogleNet [77] and

TABLE 1
A Summary of Representative Deep Learning-Based ANNS Methods With Respect to Indexing and Searching Schemes

Paradigm	Category	Approach	Indexing	Searching
Learning to Index	Deep Learning to Hash	Deep Supervised Hashing [40]	PSP(PM) + CNNs	Hash Table Lookup/ Hash Code Ranking
		Deep Self-taught Hashing [41], [42]	PSP(PM) + CNNs	
		Supervised Deep Hashing [21]	PSP(PM) + DNNs	
		Semantic Guided Deep Hashing [43]	PSP(PM) + CNNs	
		Convolutional Neural Network Hashing [20]	PSP(DM) + CNNs	
		Binary Deep Neural Network [44]	PSP(DM) + DNNs	
		Asymmetric Deep Supervised Hashing [45]	PSP(DM) + CNNs	
		Deep Incremental Hashing [46]	PSP(DM) + CNNs	
		Dual Hinge Loss-based Hashing [47]	PSP(DM) + CNNs	
		Distance Preserving Hashing [48]	PSP(DM) + DNNs	
		Deep Pairwise-supervised Hashing [49]	PSP(LM) + CNNs	
		DistillHash [50]	PSP(LM) + CNNs	
		HashNet [51]	PSP(LM) + CNNs	
		Deep Hashing Network [24]	PSP(LM) + CNNs	
		Deep Priority Hashing [52]	PSP(LM) + CNNs	
		Deep Cauchy Hashing [53]	PSP(LM) + CNNs	
		Deep Neural Networks Hashing [22]	MSP + CNNs	
		Deep Semantic Ranking Hashing [54]	MSP + CNNs	
		Bit-scalable Deep Hashing [55]	MSP + CNNs	
		Neural Order Preserving Hashing [30]	MSP + DNNs	
		Deep Progressive Hashing [56], [57]	MSP + RNNs	
		Learning to Rank-based Hashing [58]	MSP + CNNs	
		Deep Discriminative Hashing [59]	ISP + CNNs	
		Similarity-adaptive Deep Hashing [26]	ISP + CNNs	
		Deepbit [23]	ISP + CNNs	
		Deep Quantization Network [60]	PQ + CNNs	
		Deep triplet quantization [25]	TQ + CNNs	
		Deep Product Quantization [27]	PQ + CNNs	
		Deep Semantic Multimodal Hashing [61]	MM + CNNs	
		Deep Semantic-preserving Ordinal Hashing [62]	MM + CNNs	
	Learning to Partition	Locality-Sensitive Partitioning [28]	Graph Partitioning + DNNs	Best-First-Search
	Learning to Graph Construction	Graph Construction by Reinforcement Learning [63]	Graph Refinement + DRL	Heuristic-based Greedy Routing
Learning to Search	Learning to Rank	Learning to Rank over Partitions [32]	Partition-based Indexing	DNN-based Ranking + Best-First-Search
	Learning to Route	Learning to Route over Graphs [31]	Graph-based Indexing	Learning-based Greedy Routing (GNNs+LL)
	Learning to Termination	Learning to Early Termination for ANNS [33]	Partition-based / Graph-based Indexing	Learning-based Early Termination

PSP = Pairwise Similarity Preservation, MSP = Multiwise Similarity Preservation, ISP = Implicit Similarity Preservation. PM = Product-based Minimization, DM = Difference-based Minimization, LM = Likelihood-based Minimization. PQ = Product Quantization, TQ = Triplet Quantization, MM = Multimodality. DNNs = Deep Neural Networks, CNNs = Convolutional Neural Networks, RNNs = Recurrent Neural Networks, DRL = Deep Reinforcement Learning, GNNs = Graph Neural Networks, LL = Limitation Learning.

ResNet [78]) and recurrent neural networks (RNNs) (e.g., LSTM [79]). Here, the DNNs are the deep networks that do not contain the convolution layers. Usually, the last layer of deep network is replaced with a linear fully-connected layer as hash functions for producing the hash codes as illustrated in Fig. 1.

Similarity. In original space, the distance d^o_{ij} between items $\mathbf{x}_i$ and $\mathbf{x}_j$ can be euclidean distance, $\|\mathbf{x}_i - \mathbf{x}_j\|_2$, or other forms. The similarity s^o_{ij} can be defined as Gaussian function: $s^o_{ij} = \exp(-\frac{(d^o_{ij})^2}{2\sigma^2})$, or cosine similarity: $s^o_{ij} = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}$.

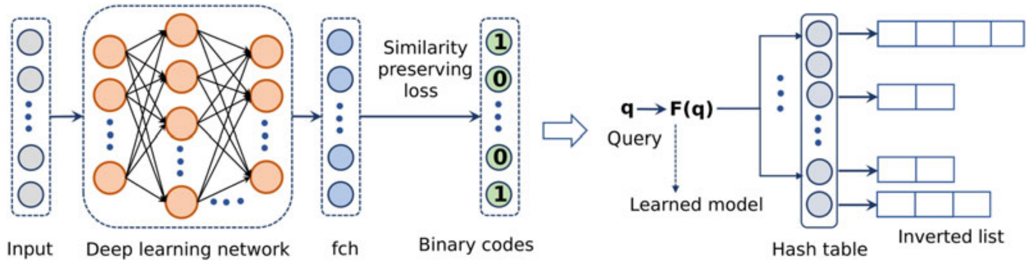


Fig. 1. An illustration for deep hashing-based ANNS methods where the fch is a fully-connected layer which is regarded as hash functions. Here, the hash table lookup is used for querying search.

For the semantic search, the similarity $s^o_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same semantic class, and 0 (or -1) otherwise.

In hash coding space, the distance d^h_{ij} between $\mathbf{y}_i$ and $\mathbf{y}_j$ is usually the Hamming distance, which is defined as $d^h_{ij} = \sum_{m=1}^{t} \mathbf{1}[y_{im} \neq y_{jm}]$, where $\mathbf{1}[p]$ is an indicator function which returns 1 if the predicate p is true and 0 otherwise. This formula is equivalent to $d^h_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|_1$ if the hash code is valued by 1 and 0, and the similarity s^h_{ij} is defined as $s^h_{ij} = t - d^h_{ij}$. If the hash code is valued by 1 and -1, then $d^h_{ij} = (t - \mathbf{y}_i^T \mathbf{y}_j)/2$ and $s^h_{ij} = \mathbf{y}_i^T \mathbf{y}_j$.

Loss Function. The key point for designing loss function is to preserve the similarity order, i.e., the ANNS results obtained from hash codes should be maximally consistent with the true search results computed from original space. The pairwise similarity preserving scheme is widely studied, which expects the similarity between a pair of two items in original space to be preserved in hash coding space. Another solution is the multiwise similarity preserving scheme, which aims to make the similarity orders among multiple items obtained from the two spaces are as consistent as possible. Quantization-based scheme aims at finding the optimal approximation of an item by minimizing the reconstruction error. Besides the similarity preserving terms, the loss function needs to consider other important factors like the parameters scale, binarization, bit balance and independence as extra constraints, for obtaining high-quality compact hash codes.

Optimization. The challenge for optimizing neural network parameters is to handle the vanishing gradient problem caused by the *sign* function which is widely adopted to generate binary codes.

There are several ways to deal with the *sign* function. The first one is the widely-used continuous relaxation, including tanh relaxation, Sigmoid relaxation, and directly dropping sign function $sign(z) = z$. Then, the Hamming distance is usually replaced with euclidean distance or inner product form. The relaxed problem can be solved by using stochastic gradient descent algorithm (SGD) (e.g., Adam optimizer [92]) via the back-propagation [93]. The second way is the two-step strategy [20], [94]: optimize the binary codes without using the hash functions, and then learn the neural networks based on the optimized hash codes. The third solution is the discretization: drop the *sign* function, i.e., $sign(z) = z$ and treat the hash code y as a discrete approximation of z, which is formulated as a loss $(y - z)^2$. In some complex optimizations involving multiple parameters, the alternative optimization [26] is usually adopted: fix all but one parameter, and then optimize the loss function in terms of a single parameter in each iteration.

Categorization. Similar to [35], we categorize the existing deep hashing-based ANNS methods into four classes: pairwise similarity preserving class, multiwise similarity preserving class, implicit similarity preserving class, quantization class. For each class, we will analyze and summarize the corresponding approaches in details. Finally, we review some multimodal deep hashing methods which has been attracting the interest in deep hashing field recently.

4.1.1 Pairwise Similarity Preservation

The algorithms in this class mainly focus on preserving the pairwise similarities or distances computed from the input space and the embedded space, which can be roughly divided into three groups according to the types of their loss functions: *product-based minimization*, *difference-based minimization*, and *likelihood-based minimization*.

Product-Based Minimization. The loss function in this group is in the form of product of distances or similarities calculated from hash coding space and original space. For instance, the similarity-distance product minimization, i.e., min $\sum_{(i,j)\in\varepsilon} s^o_{ij} d^h_{ij}$, which expects that the distance in hash coding space is smaller if the similarity in original space is larger. The ε is a set of pair items. Note that there are three other forms of product-based loss function that can be similarly defined.

Deep supervised hashing (DSH) [40] utilizes a CNN network as its hash function, and aims at minimizing the following objective function:

$$J = \sum_{(i,j)\in\varepsilon} \frac{1}{2} s^o_{ij} d^h_{ij} + \frac{1}{2}(1 - s^o_{ij}) \max(m - d^h_{ij}, 0) \tag{2}$$

where the $s^o_{ij} = 1$ for similar pairs and 0 for dissimilar pairs, and $m > 0$ is a margin threshold. This loss function encourages that the similar items have the same hash codes and the dissimilar items have different hash codes. To optimize problem (2), DSH drops the $sign$ function: $sign(z) = z$ and replaces the Hamming distance with euclidean distance. In addition, it imposes an regularizer to encourage the output of network to be binarized. Then the loss function in (2) is rewritten as:

$$\begin{aligned} J^* = \sum_{(i,j)\in\varepsilon} \Bigg(& \frac{1}{2} s^o_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 \\ & + \frac{1}{2}(1 - s^o_{ij}) \max(m - \|\mathbf{y}_i - \mathbf{y}_j\|_2^2, 0) \\ & + \alpha(\||\mathbf{y}_i| - \mathbf{1}\|_1 + \||\mathbf{y}_j| - \mathbf{1}\|_1) \Bigg) \end{aligned} \tag{3}$$

where $\mathbf{1}$ is a vector of all ones, and α is a weighting parameter. Then the network model is optimized by using the standard SGD algorithm with back-propagation.

Deep Self-taught hashing (DSTH) [41], [42] is a two-step based approach, which consists of the hash code learning and hash function learning. In the first step, it uses a pre-trained CNN-based model to generate features for each item, based on which it applies a k-nearest neighbour (NN) search approach to achieve the k-NN items for each item. Then the pairwise similarity in this feature space is defined as cosine similarity: $s^o_{ij} = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are k-nearest neighbours of each other, and $s_{ij} = 0$ otherwise. The loss function for learning the hash codes is defined as:

$$J = \sum_{(i,j)\in\varepsilon} \frac{1}{4} s^o_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 \tag{4}$$

The optimization of (4) is similar to that of *Spectral Hashing* [95]. In the second step, a CNN-based network is adopted to learn the hash functions with labels being the hash codes achieved from the first step, where the euclidean loss is employed.

Supervised deep hashing (SDH) [21] adopts a DNN-based model as hash function and aims to minimize a loss function based on a variant of $\sum_{(i,j)\in\varepsilon} s^o_{ij} d^h_{ij}$. Other than the similarity preserving loss, SDH also considers a quantization loss between the final binary vectors $\mathbf{y}$ and the real-valued vectors $\mathbf{h}$: $\|\mathbf{y} - \mathbf{h}\|_2^2$. Some regularizers are used to guarantee the bit balance and independence. Similar to SDH, *semantic guided deep hashing* (SGDH) [43] optimizes an objective function including a loss based on $\sum_{(i,j)\in\varepsilon} s^o_{ij} d^h_{ij}$ and a quantization loss. To make use of the tag information, SGDH introduces a binary matrix factorization-based loss as well as a loss for preserving data structure. All the components are incorporated into an unified framework to guide the hashing learning.

Difference-Based Minimization. The loss function in this group is to minimize the difference between the distances or the similarities calculated from the two spaces, i.e., $\min \sum_{(i,j)\in\varepsilon} (d^o_{ij} - d^h_{ij})$ or $\min \sum_{(i,j)\in\varepsilon} (s^o_{ij} - s^h_{ij})$.

Convolutional neural network hashing (CNNH) [20] aims to minimize $\sum_{(i,j)\in\varepsilon} (s^o_{ij} - s^h_{ij})$, where the $s^o_{ij} = 1$ for similar pairs and $s^o_{ij} = -1$ otherwise. In the first step, it obtains the hash codes for input items without using hash functions by minimizing the following reconstruction error:

$$J = \sum_{(i,j)\in\varepsilon} \left(\left(s^o_{ij} - \frac{1}{t} \mathbf{y}_i^T \mathbf{y}_j \right) \right)^2 = \|\mathbf{S} - \frac{1}{t} \mathbf{Y}^T \mathbf{Y}\|_F^2 \tag{5}$$

where $\mathbf{Y}$ actually encodes the hash codes that preserve the similarities in $\mathbf{S}$. It optimizes (5) by a coordinate descent algorithm, i.e., choose one entry in $\mathbf{Y}$ to optimize while keeping other entries fixed. In the second step, a CNN-based network is adopted to learn hash functions with labels being the hash codes obtained from the first step. *Semantic structure-based deep hashing* [96] modifies problem (5) to support unsupervised learning where the similarity matrix $\mathbf{S}$ is constructed from the features extracted from a pre-trained CNN network. *Deep ordinal hashing* [97] also utilizes a loss function similar to (5), where the ranking-based hash codes are learned from a spatial attention-based model by making use of the local spatial and global semantic information of input data.

Binary deep neural network (BDNN) [44] is an extension of CNNH [20]. Unlike CNNH that separates the learning of hash codes and hash functions, BDNN simultaneously learns them together. It drops the sign function and introduces an auxiliary variable $\mathbf{B} \in \{-1, 1\}^{t\times N}$, to encourage the network output to approach the binary form. The final objective function is formulated as follows:

$$\begin{aligned} J =& \frac{1}{2N} \|\mathbf{S} - \frac{1}{t} \mathbf{Y}^T \mathbf{Y}\|_F^2 + \frac{\lambda_1}{2N} \|\mathbf{Y} - \mathbf{B}\|_F^2 \\ &+ \frac{\lambda_2}{2} \| \frac{1}{N} \mathbf{Y}^T \mathbf{Y} - \mathbf{I}\|_F^2 + \frac{\lambda_3}{2N} \|\mathbf{Y}\mathbf{1}\|_F^2 + \frac{\lambda_4}{2} \sum_{l=1}^{L} \|\mathbf{\Theta}^l\|_F^2 \end{aligned} \tag{6}$$

where L is the number of network layers. The first term is the same as CNNH, and the other terms are the regularizers to control the binarization, parameter scale, bit independence and balance. Problem (6) is then solved by using an alternating optimization in terms of $\mathbf{\Theta}$ and $\mathbf{B}$.

Asymmetric deep supervised hashing (ADSH) [45] treats the query items and reference items in an asymmetric manner. Specifically, a CNN-based model is adopted to learn hash codes for queries while the binary codes for reference items are directly learned. Its objective function is formulated as:

$$J = \|\mathbf{Y}^T \mathbf{B} - c\mathbf{S}\|_F^2 \tag{7}$$

where $\mathbf{Y}$ is the network output for query items and $\mathbf{B}$ is the binary code matrix for reference items that needs to be learned. The optimization of problem (7) is similar to (6). *Deep discrete supervised hashing* [98] extends this technique by dividing the whole reference items into two subsets whose hash codes are learned in an asymmetric manner. *Deep incremental hashing* [46] proposes to handle the case for new coming items. Its loss function contains two losses where one is used to learn hash codes for new items while another one is for keeping the hash codes of old items unchanged. Both losses are designed in a manner similar to (7).

Dual hinge loss-based hashing (DHLH) [47] aims at preserving the pairwise similarity by minimizing the Kullback-Leibler (KL) divergence between the similarity matrix $\mathbf{S}$ and a probability matrix with respect to the hash codes $\mathbf{Y}$ instead of the difference between them $\|\mathbf{S} - \frac{1}{t} \mathbf{Y}^T \mathbf{Y}\|_F^2$. It proposes a symmetric KL divergence-based loss:

$$\begin{aligned} SKL(\mathbf{S}, \mathbf{Y}) = & \sum_{(i,j)\in\varepsilon} p_{ij} \log \frac{\alpha p_{ij}}{(\alpha - 1) p_{ij} + s_{ij}} \\ & + \sum_{(i,j)\in\varepsilon} s_{ij} \log \frac{\alpha s_{ij}}{(\alpha - 1) s_{ij} + p_{ij}} \end{aligned} \tag{8}$$

where $\alpha \geq 1$ is used to control the scale of $SKL(\mathbf{S}, \mathbf{Y})$, and p_{ij} is formulated as a probability function based on Gamma distribution:

$$p_{ij} = c \cdot (d^h_{ij})^{\gamma - 1} e^{-\lambda d^h_{ij}} \tag{9}$$

where γ and λ are hyper-parameters, and c is a constant. In Equation (9), larger Hamming distance means the probability p_{ij} is smaller.

Distance preserving hashing (DPH) [48] aims to minimize a loss function based on $\sum_{(i,j)\in\varepsilon} (d^o_{ij} - d^h_{ij})$. Specifically, its final loss function is formulated as:

$$J = \frac{\lambda}{N}\|\mathbf{D}^h - a\mathbf{D}^o - b\|_F^2 + \frac{\alpha}{N}\|\mathbf{Y} - \mathbf{C}\|_F^2 + \beta\|\boldsymbol{\theta}^T\boldsymbol{\theta} - \mathbf{I}\|_F^2 \tag{10}$$

where $\mathbf{D}^h$ denotes the Hamming distance matrix for N pair items in hash coding space, and $\mathbf{D}^o$ is the euclidean distance matrix for N pair items in original space. $\mathbf{C}$ is the hash codes generated by an existing hashing method, which is used for supervision. $\boldsymbol{\theta}$ is the weight of the output layer of network.

Likelihood-Based Minimization The loss function in this group is based on the maximum likelihood estimation. Given the pairwise similarity matrix $\mathbf{S}$ and hash code matrix $\mathbf{Y}$, the logarithm maximum likelihood estimation of $\mathbf{Y}$ is formulated as:

$$\log p(\mathbf{S}|\mathbf{Y}) = \sum_{(i,j)\in\varepsilon} \log p(s_{ij}^o|\mathbf{y}_i, \mathbf{y}_j) \tag{11}$$

where the $p(\mathbf{S}|\mathbf{Y})$ is the likelihood function. For each pair, $p(s_{ij}^o|\mathbf{y}_i, \mathbf{y}_j)$ is the conditional probability of similarity s_{ij}^o given the hash codes $\mathbf{y}_i$ and $\mathbf{y}_j$, which is formulated as a pairwise logistic function:

$$\begin{aligned} p(s_{ij}^o|\mathbf{y}_i, \mathbf{y}_j) &= \begin{cases} \sigma(\mathbf{y}_i^T\mathbf{y}_j), & s_{ij}^o = 1 \\ 1 - \sigma(\mathbf{y}_i^T\mathbf{y}_j), & s_{ij}^o = 0 \end{cases} \\ &= \sigma(\mathbf{y}_i^T\mathbf{y}_j)^{s_{ij}^o}(1 - \sigma(\mathbf{y}_i^T\mathbf{y}_j))^{(1-s_{ij}^o)} \end{aligned} \tag{12}$$

where the $\sigma = \frac{1}{1+e^{-x}}$ is the Sigmoid function. By taking (12) into (11), the final loss function is rewritten as:

$$J = \sum_{(i,j)\in\varepsilon} ((\log(1 + \exp(\mathbf{y}_i^T\mathbf{y}_j)) - s_{ij}^o\mathbf{y}_i^T\mathbf{y}_j) \tag{13}$$

This loss function expects the similarity of a pair of items in hash coding space to be large if their similarity in original space is large.

Deep pairwise-supervised hashing (DPSH) [49], *deep supervised discrete hashing* (DSDH) [99], *deep asymmetric pairwise hashing* (DAPH) [100] and *DistillHash* [50] all use the same loss function as (13) with some regularizers to learn hash codes for data items. Different from DPSH and DSDH whose networks for processing pairwise items share the same wights, DAPH proposes to use two networks with different parameters to process the pairwise items. Other than the likelihood-based loss, DSDH imposes a linear reconstruction loss for the label information and the learned binary codes. *DistillHash* proposes to learn hash functions from a distilled data set that has confidence similarity signals.

HashNet [51] is proposed to handle the issue of imbalanced data where the number of dissimilar pairs is much more than that of similar pairs. Thus, it adopts a weighted maximum likelihood-based loss to preserve the pairwise similarity, which is formulated as:

$$\log p(\mathbf{S}|\mathbf{Y}) = \sum_{(i,j)\in\varepsilon} w_{ij}\log p(s_{ij}^o|\mathbf{y}_i, \mathbf{y}_j) \tag{14}$$

where w_{ij} is used to weight the pairs according to the significance of misclassifying that pair [101], which is designed as

$$_{ij} = c_{ij} \cdot \begin{cases} |\mathbf{S}|/|\varepsilon^-|, & s_{ij}^o = 0 \\ |\mathbf{S}|/|\varepsilon^+|, & s_{ij}^o = 1 \end{cases} \tag{15}$$

where the ε^- is the set of dissimilar pairs and ε^+ is the set of similar pairs. $c_{ij} = \frac{\mathbf{f}_i \cap \mathbf{f}_j}{\mathbf{f}_i \cup \mathbf{f}_j}$ for multi-labeled data and $c_{ij} = 1$ for single labeled data. The definition of $p(s_{ij}^o|\mathbf{y}_i, \mathbf{y}_j)$ is similar to Equation (12). HashNet proposes to deal with the *sign* function by a continuation strategy which is formulated as: $\lim_{\beta\to\infty} tanh(\beta z) = sign(z)$. *HashGAN* [102] is a variant of *HashNet*, where the network architecture is based on the generative adversarial networks [103].

Deep hashing network (DHN) [24] employs the logarithm Maximum a Posteriori (MAP) estimation for hash codes $\mathbf{Y}$, which is defined as follows:

$$\begin{aligned} \log p(\mathbf{Y}/\mathbf{S}) &\propto \log p(\mathbf{S}|\mathbf{Y})p(\mathbf{Y}) \\ &= \sum_{(i,j)\in\varepsilon} \log p(s_{ij}^o|\mathbf{y}_i, \mathbf{y}_j)p(\mathbf{y}_i)p(\mathbf{y}_j) \end{aligned} \tag{16}$$

where the $p(\mathbf{S}|\mathbf{Y})$ is the likelihood function, and $p(\mathbf{Y})$ is the prior distribution. The definition of $p(s_{ij}^o|\mathbf{y}_i, \mathbf{y}_j)$ is the same as Equation (12). For the $p(\mathbf{y}_i)$, DHN proposes an unnormalized bimodal Laplacian prior that can be regarded as a quantization loss, which is defined as:

$$p(\mathbf{y}_i) = \frac{1}{2\epsilon}\exp\left(-\frac{\||\mathbf{y}_i| - \mathbf{1}\|_1}{\epsilon}\right) \tag{17}$$

where the ϵ is the diversity parameter. By taking (12) and (17) into (16), the loss function is formulated as:

$$\begin{aligned} J = &\sum_{(i,j)\in\varepsilon} ((\log(1 + \exp(\mathbf{y}_i^T\mathbf{y}_j)) - s_{ij}^o\mathbf{y}_i^T\mathbf{y}_j) \\ &+ \frac{1}{\epsilon}(\||\mathbf{y}_i| - \mathbf{1}\|_1 + \||\mathbf{y}_j| - \mathbf{1}\|_1)). \end{aligned} \tag{18}$$

To address non-smooth problem caused by the absolute function in (18), DHN adopts a smooth surrogate which is $|x| \approx \log coshx$ [104]. The binary codes are learned by a CNN network with a binarization operation on the last output layer of the network.

Deep priority hashing (DPH) [52] and *deep cauchy hashing* (DCH) [53] are two variants of DHN [24]. DPH improves DHN by adding a weight for each different pair items such that the objective function can concentrate more on the difficult pairs than the easy pairs. DCH replaces the Sigmoid function in Equation (12) with a probability function based on Cauchy distribution in order to penalize more on similar pairs with large Hamming distances.

4.1.2 Multiwise Similarity Preservation

The methods in this class define the objective function by minimizing the difference of the similarity orders over multiple items that obtained from the two spaces.

Deep neural networks hashing (DNNH) [22] uses a triplet loss similar to that in [105] to preserve the relative similarities of items. Given a triplet of items $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ where the $\mathbf{x}_i$ is more similar to $\mathbf{x}_j$ than to $\mathbf{x}_k$, then the triplet loss function is defined as:

$$J(\mathbf{y}_i, \mathbf{y}_j, \mathbf{y}_k) = \max(0, 1 - (d_{ik}^h - d_{ij}^h)) \tag{19}$$

where $\mathbf{y} \in \{0,1\}^t$. This loss function expects the hash code $\mathbf{y}_i$ to be closer to $\mathbf{y}_j$ than $\mathbf{y}_k$. The objective function over a set of triplets can be accordingly defined. To optimize the above problem, the binary constraint is replaced by continuous relaxation (i.e., $\mathbf{y} \in [0,1]^t$) and the Hamming distance is replaced by l_2 norm. For network architecture, a CNN-based model is used and the intermediate features are transformed into hash codes using a divide-and-encode module with the help of a Sigmoid function.

Deep semantic ranking hashing (DSRH) [54] is based on a triplet-wise similarity ranking preservation. For an item $\mathbf{x}_i$, a similarity level r of an item $\mathbf{x}_j$ with respect to $\mathbf{x}_i$ is computed by the number of their common labels: $r = |\mathbf{f}_i \cap \mathbf{f}_j|$. Then a ranking list for $\mathbf{x}_i$ is obtained by sorting the reference items in descending order of their similarity levels. Given a query $\mathbf{x}_i$ and a ranking list $\{\mathbf{x}_j\}_{j=1}^N$ for $\mathbf{x}_i$, the weighted ranking loss on a set of triplets is defined as follows:

$$J(\mathbf{y}_i, \{\mathbf{y}_j\}_{j=1}^N) = \sum_{j=1}^{N} \sum_{k: r_k < r_j} w_{jk} \max(0, d_{ij}^h - d_{ik}^h + \rho) \tag{20}$$

where the weight $w_{jk} = \frac{2^{r_j} - 2^{r_k}}{Z}$, and Z is a normalization constant. The higher the relevance of $\mathbf{x}_i$ and $\mathbf{x}_j$ is than that of $\mathbf{x}_i$ and $\mathbf{x}_k$, the larger weight is assigned to this triplet. By combining the regularizers for bit balance and large weight penalty, the final loss function over a training set can be accordingly defined.

Bit-scalable deep hashing (BSDH) [55] proposes to use a pairwise loss and a triplet loss together to preserve the similarity. A weight vector $\mathbf{w} = [w_1, w_2, ..., w_c]$ is used to weight each bit of hash codes, which denotes the importance of each bit in evaluating the similarity. Given a set of triplet items ϵ and a set of pair items ε, the final objective function is formulated as:

$$J = \sum_{(i,j,k)\in\epsilon} \max(\hat{d}_{ij}^h - \hat{d}_{ik}^h, \rho) + \lambda \sum_{(i,j)\in\varepsilon} \hat{d}_{ij}^h s_{ij}^o \tag{21}$$

where $\hat{d}_{ij}^h = \mathbf{y}_i^T \mathbf{W} \mathbf{y}_j$, and $\mathbf{W}$ is a diagonal matrix with $\mathbf{W}(i,i) = w_i$. The definition of $\hat{d}_{ij}^h$ is similar to the weighted Hamming affinity [106].

Neural order preserving hashing (NeOPH) [30] employs a DNN-based network to learn hash functions by aligning the similarity orders calculated from original space and the ones in hash embedding space. Unlike previous deep hashing methods that learn binary codes, NeOPH learns real-valued hash codes for input items, which are then organized by an I/O efficient index for large-scale ANNS. Given a query item $\mathbf{x}_i$, the ranking position of a reference item $\mathbf{x}_j$ with respect to $\mathbf{x}_i$ in hash coding space is defined as:

$$r(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{N} \mathbf{1}[\|\mathbf{y}_i - \mathbf{y}_j\|_2 > \|\mathbf{y}_i - \mathbf{y}_k\|_2] + 1 \tag{22}$$

where $\mathbf{1}[p]$ is an indicator function. Intuitively, this formulation measures how many other points (i.e., $\mathbf{x}_k$) has a smaller distance to $\mathbf{x}_i$ than that of $\mathbf{x}_j$. To make the above formulation smooth and continuous, NeOPH uses the Sigmoid function to approximate the indicator function. Then the ranking function is rewritten as:

$$\tilde{r}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{N} \sigma(\|\mathbf{y}_i - \mathbf{y}_j\|_2^2 - \|\mathbf{y}_i - \mathbf{y}_k\|_2^2) + 1 \tag{23}$$

Then, the final loss function for a training set $\mathbf{X}$ can be formulated as:

$$J = \sum_{\mathbf{x}_i \in \mathbf{X}} \sum_{j=1}^{N} \beta_j \log\left((\tilde{r}(\mathbf{x}_i, \mathbf{x}_j) - g(\mathbf{x}_i, \mathbf{x}_j))^2 + 1\right) \tag{24}$$

where the $g(\mathbf{x}_i, \mathbf{x}_j)$ is the ground-truth ranking position of $\mathbf{x}_j$ in terms of query $\mathbf{x}_i$ in original space $\mathcal{R}^d$, and β_j is the weight for $\mathbf{x}_j$, defined as $\beta_j = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2}{\max_{1 \le k \le N} \|\mathbf{x}_i - \mathbf{x}_k\|_2})$. This loss penalizes how far is the ranking position of $\mathbf{x}_j$ away from its ground-truth in space $\mathcal{R}^d$, and $\log(1+z)$ is used to encourage the network to concentrate more on nearby items than faraway items.

The learned t-dimensional hash codes are then organized by t sorted lists where each list corresponds to one dimension of hash codes. The querying search is to retrieve a candidate set by locating the position of the query on the corresponding sorted lists, and then scanning the lists based on the ranking position of the reference items.

Others. Deep progressive hashing [56], [57] adopts a triplet loss similar to DNNH [22] for similarity preservation. Different from DNNH, it employs a CNN-based model followed by a RNN network [79] to learn the hash codes for data items. *Learning to rank-based hashing* [58] aims to directly optimize the ranking-based evaluation metrics such as the Normalized Discounted Cumulative Gain (NDCG) [107] and Average Precision (AP). *Mutual information-based hashing* [108] proposes to optimize a kind of mutual information as a ranking surrogate for hash code learning.

4.1.3 Implicit Similarity Preservation

The loss functions of the methods in this class are not explicitly based on pairwise or multiwise similarity preservation, but they still utilize the similarity information from data to guide the hash code and hash function learning. Thus, we classify this class of methods as implicit similarity preserving class.

Deep discriminative hashing (DDH) [59] first employs a pre-trained CNN-based model to extract features for data items, which are then clustered by a k-means algorithm. The generated cluster labels of items are used as pseudo labels for the subsequent hashing step. Then a softmax classification loss is adopted to preserve the similarity from the pseudo labels:

$$J = -\sum_{i=1}^{N} \sum_{j=1}^{M} \mathbf{1}[f_i = j] \log\left(\frac{\exp(\boldsymbol{\theta}_j^T \mathbf{y}_i + v_j)}{\sum_{l=1}^{M} \exp(\boldsymbol{\theta}_l^T \mathbf{y}_i + v_l)}\right) \tag{25}$$

where M is the number of clusters, f_i is the cluster label of item i, and $\boldsymbol{\theta}_j$ and v_j are weights and bias of the softmax layer, respectively. To solve problem (25), DDH drops the *sign* function and imposes a quantization loss to J. Then the problem can be solved by standard optimization methods.

Similar to DDH [59], many hashing methods in this class utilize the deep features of data items extracted from a pre-trained CNN-based model to construct similarity information for hash code and hash function learning. *Hybrid-similarity hadamard hashing* [29] utilizes the deep features to build a hybrid-similarity matrix for hash code learning. *Similarity-adaptive deep hashing* [26] employs the deep features of items to build a similarity graph, then the hash code learning is converted to a graph hashing problem [109].

Deepbit [23] employs a CNN-based model as its network architecture and trains the network parameters by minimizing a bit balance loss and a quantization loss. In addition, a rotation invariant loss is added by augmenting training data with different rotations.

Very deep supervised hashing [110] trains a very deep neural network by minimizing a linear reconstruction loss between the learned hash codes (i.e., the output of network) and the label information.

4.1.4 Quantization

The methods in this class aim to apply the quantization approaches into deep hashing-based ANNS methods to better control the hashing quality and quantization error.

Deep quantization network (DQN) [60] proposes to apply deep learning techniques into product quantization [111] to construct binary codes for data items. The similarity preservation is conducted by a pairwise cosine loss:

$$J_1 = \sum_{(i,j)\in\varepsilon} \left(s_{ij}^o - \frac{\mathbf{h}_i^T \mathbf{h}_j}{\|\mathbf{h}_i\|_2 \|\mathbf{h}_j\|_2} \right) \tag{26}$$

where $\mathbf{h}_i$ is the output of network for item $\mathbf{x}_i$, and $tanh$ function is employed as activation. On the other hand, a product quantization-based loss is imposed to control the hashing quality and quantization error. The key point is to decompose the original vector space into the Cartesian product of M low dimensional subspaces and quantize each subspace into k codewords (clusters) through k-means clustering. Specifically, $\mathbf{h}_i \in \mathcal{R}^t$ is divided into M subspaces, i.e., $\mathbf{h}_i = [\mathbf{h}_{i1}; ...; \mathbf{h}_{iM}]$, where $\mathbf{h}_{im} \in \mathcal{R}^{t/M}$ is the sub-vector of $\mathbf{h}_i$ associated with the m-th subspace. Then all sub-vectors $\{\mathbf{h}_{im}\}_{i=1}^N$ of each subspace m are quantized into k clusters (codewords) independently via k-means:

$$J_2 = \sum_{m=1}^{M} \sum_{i=1}^{N} \|\mathbf{h}_{im} - \mathbf{C}_m \mathbf{y}_{im}\|_2^2$$
$$\mathbf{y}_{im} \in \{0,1\}^k, \|\mathbf{y}_{im}\|_1 = 1 \tag{27}$$

where $\mathbf{C}_m = [\mathbf{c}_{m1}; ...; \mathbf{c}_{mk}] \in \mathcal{R}^{\frac{t}{M}\times k}$ represents the codebook of k codewords (cluster centers) in the m-th subspace, and $\mathbf{y}_{im}$ is the code indicating which codeword in $\mathbf{C}_m$ is selected to approximate $\mathbf{h}_{im}$. Then $\mathbf{y}_i = [\mathbf{y}_{i1}, ..., \mathbf{y}_{iM}]$ is the complete binary code for item $\mathbf{x}_i$. By combining the J_1 and J_2 together, the final optimization problem is accordingly defined, which can be solved by algorithms based on the alternative optimization scheme. Finally, the *Asymmetric Quantizer Distance* is employed as distance metric to perform the querying search:

$$AQD(\mathbf{q}, \mathbf{x}_i) = \sum_{m=1}^{M} \|\mathbf{h}_{qm} - \mathbf{C}_m \mathbf{y}_{im}\|_2^2 \tag{28}$$

where the $\mathbf{h}_{qm}$ is the m-th sub-vector of the representation of query $\mathbf{q}$.

Deep triplet quantization (DTQ) [25] proposes to use a triplet quantization loss and a triplet loss to preserve similarity. Different from DQN employing the product quantization loss, the triplet quantization loss is established on a set of codebooks that need to be learned via the entire objective function and are shared across different items in all triplets, which is formulated as:

$$J = \sum_{i=1}^{N} \sum_{*\in\{1,2,3\}} \|\mathbf{h}_i^* - \sum_{m=1}^{M} \mathbf{C}_m \mathbf{y}_{im}^*\|_2^2$$
$$+ \gamma \sum_{m=1}^{M} \sum_{m'=1}^{M} \|\mathbf{C}_m^T \mathbf{C}_{m'} - I\|_F^2 \tag{29}$$

where $\{1,2,3\}$ is the indices of items in a triplet and $\mathbf{C}_m \in \mathcal{R}^{t\times k}$ is the m-th codebook. The second term is a weak orthogonality penalty that reduces the dependency of multiple codebooks and improves the compactness of binary codes. The final loss function is built based on the above quantization loss and a triplet loss similar to DNNH [22]. Finally, the *Asymmetric Quantizer Distance* based on inner-product similarity is adopted to perform the search:

$$AQD(\mathbf{q}, \mathbf{x}_i) = \mathbf{h}_q^T \left(\sum_{m=1}^{M} \mathbf{C}_m \mathbf{y}_{im} \right) \tag{30}$$

Deep product quantization (DPQ) [27] combines the merits from DQN and DTQ. It employs the product quantization to construct the quantization loss as in DQN, but the codebooks are not built from the k-means approach executed on the representation of network output. Instead, the codebooks are learned via the final loss function as in DTQ.

Deep asymmetric quantization (DAQ) [112] employs a composite quantization [113] loss as well as a classification loss to learn the hash functions.

4.1.5 Multimodal Deep Hashing

With rapid development of social networks and the increasing demands of users, the search involving multimodal data has been enjoying the popularity in recent years, such as the image-text search [114] and video-text search [61]. Methods in this class focus on leveraging the deep learning networks to jointly learn multiple types of hash functions (embedded spaces) for multiple types of data. Then the similarities between the hash codes of involved data can be easily calculated for efficient ANNS. The techniques used in multimodal deep hashing is similar to that of general deep hashing methods as reviewed above, except that the similarity preservation is based on the inter-modality similarity or the combination of inter-modality and intra-modality similarities. The representative algorithms in this class include the *deep semantic multimodal hashing* [61], *deep semantic-preserving ordinal hashing* [62], and *deep collaborative embedding* [115].

4.1.6 Comparison Between Deep Hashing and Traditional Learning to Hash

The key difference between deep hashing methods and traditional learning to hash methods lies in their hash functions. The former employs deep learning networks to learn hash functions while the latter usually leverages the linear transformations or kernel functions. If the deep networks are replaced with the traditional hash functions, then the deep hashing methods becomes the traditional hashing methods.

The main advantage of applying deep learning networks into hashing learning is that deep learning models can well simulate different kinds of complicated non-linear transformations to explore the real-world data distributions, thus can better capture and preserve the similarity relationships from the input space by learning the hash coding space. Further more, deep hashing methods enable simultaneously learning the representation of raw image data and the hash functions under a unified loss function. Thus, deep hashing methods are shown to outperform traditional hashing methods on many tasks, which has been reported and discussed in the existing work [40], [45], [58], [97].

4.1.7 Analyses and Summary

Based on the above review, the common point of the five types of deep hashing methods is that their network architectures can actually be designed as any deep learning networks for hash function learning, which depends on the computing resources and the studied tasks. The main difference between them lies in their similarity preservation schemes, which is the key component of objective function for hash code learning.

Pairwise similarity preserving-based methods focus on preserving the pairwise similarities while the multiwise similarity preserving-based approaches aim at aligning the relative similarities from the two spaces. Compared with pairwise similarities, relative similarities have several advantages: 1) They are more easily obtained than pairwise similarities (e.g., the click-through data from image retrieval systems, or the ranking information from original euclidean distance). 2) Given the information of pairwise similarities, one can easily produce a set of triplet samples to achieve the relative similarities. 3) Relative similarities contains more multilevel information which is useful for hash code learning. Some empirical results of the comparison between these two types of methods have been discussed in [58], [108]. Implicit similarity preserving-based approaches usually make use of the feature information extracted from the output of networks to construct the similarities of items for iterative hashing learning, which is a good choice for data without labeling information. However, this type of methods lack the label knowledge to guide the hash code learning, limiting their final performance to some extent.

The quantization-based methods take both label-based similarities (i.e., pairwise or relative similarities) and feature-based similarities (i.e., PQ-based loss on the output of deep networks) into account for similarity preservation, thus having the advantages on some tasks compared with other methods. Some empirical results can be seen in [27]. For the multi-modality deep hashing, almost all techniques used in single-modality methods can be applied into this type of methods, which depends on the studied problems.

Due to the non-linear transformation power of deep learning, and the storage and computation efficiency of low dimensional hash codes, deep learning to hash is suitable for dealing with large-scale ANNS problem, attracting the interests of ANNS community.

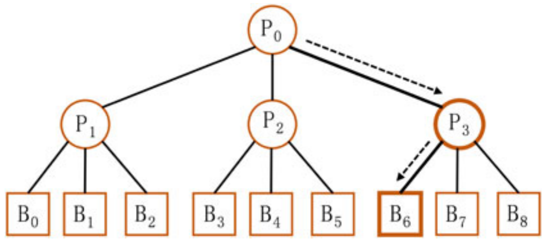


Fig. 2. An illustration for partition-based ANNS methods. P_i's are the partitions and B_j's are the bins of data items.

4.2 Learning to Partition

The main idea of partition-based methods is to divide the high dimensional space $\mathcal{R}^d$ into multiple disjoint regions, where the partitioning procedure can be executed recursively. Then, the data items can be organized by a tree or a forest structure. Given a query item, the searching algorithm aims to find a subset of data items that are closest to query by traversing the index in a best-first-search manner according to the adopted partitioning rule, and then a re-ranking operation is performed to retrieve the final target results. An illustration for partition-based methods can be seen in Fig. 2.

Conventional methods perform the data partition based on the heuristic strategies, such as the random direction-based hyperplane [10], axis-aligned hyperplane [8], PCA-based hyperplane [116] and compact partitioning [9]. By contrast, learning to partition-based methods try to employ deep learning techniques to find a good partitioning rule for reference data for efficient ANNS. There are some learning-based algorithms that have been proposed in literature for learning the hyperplane-based partition [117], [118], [119] and the neural network-based partition [28]. As our study focuses on the deep learning-based ANNS methods, we only review and analyze the work in [28] in this subsection.

Neural LSH [28] develops a framework for finding high-quality space partitions for answering ANNS. This method considers addressing three issues when designing the framework: 1) Balanced: the number of data items in each partition should be roughly the same. 2) Locality sensitive: for each item $\mathbf{q} \in \mathcal{R}^d$, most of its nearest neighbors should belong to the same partition. 3) Simple: the partition procedure should be computationally efficient. Formally, a good partition $\mathcal{P}$ aims to minimize the loss:

$$J = \mathbf{E}_{\mathbf{q}}\left[\sum_{\mathbf{x}\in N_k(\mathbf{q})} \mathbf{1}_{\mathcal{P}(\mathbf{x})\neq\mathcal{P}(\mathbf{q})}\right]$$
$$s.t.\ \forall_{\mathbf{x}\in\mathbf{X}}|\mathcal{P}(\mathbf{x})| \leq (1+\eta)(N/M) \tag{31}$$

where $\mathbf{q}$ is sampled from the query distribution, $N_k(\mathbf{q}) \subset \mathbf{X}$ is the set of its k-nearest neighbors, $\eta > 0$ is a balance parameter, $\mathcal{P}(\mathbf{x})$ represents the partition containing $\mathbf{x}$, and M represents the number of partitions.

Given a dataset $\mathbf{X}$, let G be the k-nearest neighbour (k-NN) graph of $\mathbf{X}$, whose vertices are the data items, and each vertex is connected to its k nearest neighbors. The motivation is that under mild conditions on the dataset, the

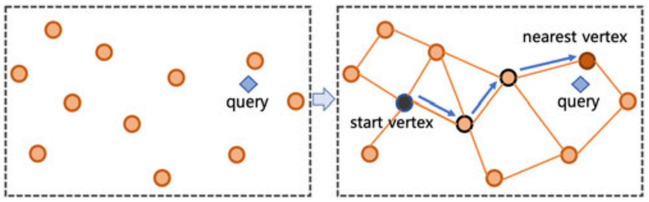


Fig. 3. An illustration for graph-based ANNS methods.

k-NN graph of $\mathbf{X}$ can be partitioned with a hyperplane into two parts of comparable size such that only few edges get split by the hyperplane [28], [120]. Thus, the problem of partitioning $\mathbf{X}$ can be converted into balanced graph partitioning problem which is widely studied. Motivated by that, this method first builds a k-NN graph G for $\mathbf{X}$, and then divides the graph G into M parts by using the well-known graph partitioner KaHIP [121]. Finally, it builds a DNN-based classifier for data items with labels being the parts of the partitions generated by the graph partitioner.

Specifically, for an item $\mathbf{x}$, the partitioning model $\mathcal{F}$ can be formulated as:

$$\mathcal{F}(\mathbf{x}) = (p_1, p_2, ..., p_M) \tag{32}$$

where the $(p_1, ..., p_M)$ are the predicted probability distribution on the M partitions for which one item $\mathbf{x}$ should be allocated. The target distribution $(p_1^*, ..., p_M^*)$ is a probability distribution over the partitions (generated by KaHIP) containing items from the k-nearest neighbors of $\mathbf{x}$. Then the final loss function aims at minimizing the KL divergence between these two distributions: $\sum_{i=1}^{M} p_i^* \log \frac{p_i^*}{p_i}$.

For an unseen item $\mathbf{x}$, the learned partitioning model $\mathcal{F}$ is utilized to predict which of the M bins $\mathbf{x}$ belongs to. To improve the efficiency of the resulting partition, it pays off to perform the data partition in a hierarchical manner. The model is accordingly designed in a hierarchical way.

Comments. Partition-based ANNS has several advantages compared with other methods. First, it is suitable for distributed settings as different partitions can be stored in different machines [122], [123]. Second, partition-based indexing is also suitable for GPUs for fast ANNS due to the simple and predictable memory overhead [124]. Thus, applying deep learning to learn good data partitions for efficient ANNS still needs the efforts from ANNS community.

4.3 Learning to Graph Construction

Graph-based ANNS methods aim to build a proximity graph where a vertex corresponds to a data item and the edges define a kind of neighbor-relationship between the connected vertices. The key motivation is that *"a neighbor's neighbor is also likely to be a neighbor"*. Then starting from a graph vertex (a data item), the querying search is efficiently executed by iteratively expanding neighbors' neighbors in a best-first-search manner following the edges, until stop condition is met. Among the search, a candidate list of size k can be used to maintain the top-k nearest neighbours found so far. An illustration for graph-based ANN methods can be seen in Fig. 3.

The search efficiency over a graph can be approximately formulated as $\mathcal{O}(el)$, where the e represents the average out-degree of the graph and l denotes the length of search path. Thus, the conventional graph-based ANNS methods aim to construct proximity graphs with special properties that have smaller size of out-degree and shorter length of search path, while maintaining the connectivity of graph, such that the overall search efficiency can be improved without accuracy degradation. Representative proximity graphs include k-nearest neighbour graph [12], [13], relative neighborhood graph [16], [125], and small world graph [14], [15]. However, the construction of all these graphs is based on the heuristic strategies.

By contrast, learning to graph construction-based methods try to employ deep learning techniques to learn a good proximity graph from data to better serve the querying search. So far, there is only one work in [63] that belongs to this category, which is reviewed and analyzed in details.

The method in [63] proposes to learn the graph construction from data to serve ANNS by deep reinforcement learning (RL) [85]. The key idea is to refine the connectivity of a given proximity graph by deleting the unnecessary edges for achieving a better querying performance. Specifically, it models the existence of each edge of the graph as a probability, and then uses reinforcement learning to learn these probabilities from data. Finally, the graph will be refined by deleting the edges with probabilities smaller than 0.5, i.e., these edges cannot help improve the search efficiency and accuracy.

The graph refinement is formulated as a Markov Decision Process (MDP). Specifically, a given graph G and a search approach are regarded as an environment ξ. A MDP agent interacts with ξ using two actions: "keep" or "remove" on an edge. The environment state $\mathcal{S} = (q, v_i, v_{adj}, V, H)$ includes the query item q, current vertex v_i, the adjacent vertices v_{adj} of v_i, visited vertices V and a candidate heap H. The probability learning procedure is based on a beam search algorithm [126], [127], which is presented in Algorithm 1, where $d(v, q)$ represents the euclidean distance between vertex v and query q.

Algorithm 1. Beam Search Algorithm(G, q, v_0, k)

```
Input : $G$ : Initial Graph,
        $q$ : query item,
        $v_0$ : start vertex,
        $k$ : the number of output nearest neighbours
Input: $C$: The set of top-$k$ nearest neighbours
1: $V \leftarrow v_0$;                          /* a set of visited vertices */;
2: $H \leftarrow \{v_0 : d(v_0, q)\}$;               /*a candidate heap */;
3: $C \leftarrow \{v_0 : d(v_0, q)\}$;    /* a heap for top-$k$ results */;
4: while stop condition is not met do
5:     /* one search session */;
6:     $v_i \leftarrow$ select the nearest vertex from $H$ w.r.t $q$;
7:     $v_{adj} \leftarrow$ obtain the adjacent vertices of $v_i$;
8:     $\mathcal{S} \leftarrow (q, v_i, v_{adj}, V, H)$;          /* collect environment
       state */;
9:     $\hat{v}_{adj} \leftarrow$ Agent($\mathcal{S}$) /* predict what edges to keep */;
10:    for $\hat{v} \in \hat{v}_{adj} \backslash V$ do
11:        $V \leftarrow$ Add($V, \hat{v}$);
12:        $H \leftarrow$ Insert($H, \hat{v}, d(\hat{v}, q)$);
13:        $C \leftarrow$ Update($C, k, \hat{v}, d(\hat{v}, q)$);
14: return $C$
```

One round of the *while* loop in Algorithm 1 is regarded as a search session τ. In each search session, the algorithm accesses a vertex and updates $\mathcal{S}$. The agent takes $\mathcal{S}$ as input and determines what edges are to be kept. Then, the algorithm processes the kept edges and selects the next vertex. Once the session terminates, the *Agent* gets a reward $\mathcal{R}$, which is defined as:

$$\mathcal{R}(\tau) = I[\tau] \cdot \max(DC_{max} - DC, 1) \tag{33}$$

where $I[\tau] = 1$ if the true nearest neighbor (w.r.t q) is found in session τ, and 0 otherwise. DC is the number of distance calculations during session τ, and DC_{max} is a distance computation budget. This reward is used to encourage the agent to find the actual nearest neighbor while reducing the complexity without the accuracy degradation. The search trajectories in one complete session are then collected for the subsequent model optimization.

The architecture of the probability model (agent) is a DNN network which takes the environment state as input and predicts the edge probabilities. Then the policy-based RL method *TRPO* [128] is adopted to optimize the network in order to get the maximum reward $\mathcal{R}$.

Comments. As mentioned in [63], one can consider refining the proximity graph by the combination of deleting old edges and adding new edges through learning from data, such that final searching performance can be further improved. On the other hand, learning a proximity graph directly from data instead of the refinement from a given graph is still an open research problem. Thus, learning to graph construction is still an interesting research direction for ANNS problem.

5 Learning to Search

After the index structure for data items is constructed, ANNS methods will perform the querying search algorithm to retrieve the final target results. Thus, the searching policy is crucial for ANNS methods as it determines the final search performance. Different ANNS methods have different searching policies, which depends on their index types. For hashing-based methods, the searching policy can be roughly regarded as a kind of linear scan on the hash codes (hash tables). For partition-based methods, the index is usually based on a tree structure, then the searching policy needs to determine which node (cluster) is to be accessed next, which is usually based on a specified access order. In terms of graph-based methods, the greedy-based routing over the constructed graph is widely adopted.

In traditional ANNS methods, the searching policies are usually heuristic and independent from data. By contrast, learning to search-based ANNS methods aim at utilizing deep learning techniques to learn a good searching policy from the given indices and data in order to achieve better querying efficiency and accuracy. In the past, there have been a number of deep learning-based ANNS methods in literature that belong to this paradigm, which are carefully reviewed and analyzed in details below.

5.1 Learning to Rank

The work in [32] proposes a ranking model by learning the underlying neighborhood relationships in the partition-based index structure. In conventional k-means clustering-based methods [38], [39], [111], the reference dataset $\mathbf{X} \in \mathcal{R}^{d\times N}$ is partitioned into M clusters $\{C_m | m = 1, ..., M\}$ via k-means clustering, where each cluster C_m is with a centroid $\mathbf{u}_m \in \mathcal{R}^d$. Given a query item $\mathbf{q} \in \mathcal{R}^d$, a typical search policy is to retrieve the top-T ranked clusters whose centroids are closest to $\mathbf{q}$ according to their euclidean distances. In contrast, this method models the ranking order of clusters for a query by a network function $\mathcal{F}$:

$$\mathcal{F}(\bar{\mathbf{q}}) = (p_1, p_2, ..., p_M) \tag{34}$$

where $\bar{\mathbf{q}}$ is a query-dependent feature vector, and p_m represents the neighbourhood probability of the m-th cluster. Then the learned search policy is used to rank the clusters based on their neighbourhood probabilities rather than their euclidean distances, thereby improving the retrieval quality.

Specifically, given a query $\mathbf{q}$, its k nearest neighbours $\{g_1, ..., g_k\}$, and the associated weight for each nearest neighbour $\{w_1, ..., w_k\}$. The $\bar{\mathbf{q}}$ consists of three terms: the query item feature (i.e., $\mathbf{q}$), query-centroid similarities, and their concatenation. Let $\mathbf{d} = [d_1, ..., d_M]$, where $d_m = \|\mathbf{q} - \mathbf{u}_m\|_2$, then the query-centroid similarity vector is defined as $\mathbf{a} = [a_1, ..., a_M]$, where $a_m = \frac{max(\mathbf{d}) - d_m}{max(\mathbf{d})}$. Then the input vector $\bar{\mathbf{q}} = \{\mathbf{q}, \mathbf{a}, concat(\mathbf{q}, \mathbf{a})\}$.

The target label that the network needs to learn is an approximation of the neighbourhood probabilities of M clusters, denoted as $\mathbf{Y} = \{y_1, ..., y_M\}$, where y_m is defined by:

$$y_m = \frac{\sum_i \{w_i | g_i \in C_m\}}{\sum_i \{w_i | g_i \in C_m\} + |C_m|} \tag{35}$$

then the y_m is normalized by sum to one. Finally, this method employs a DNN network as the function $\mathcal{F}$ to learn the ranking policy for each cluster, which is then used for querying search. The ranking model can be learned in a hierarchical manner for hierarchical querying search to further improve the performance.

Comments. For the partition-based ANNS methods, their searching algorithms need a good access order of the partitions (clusters) for more efficient retrieval, as a good access order means a less amount of search work to find the nearest neighbours. The idea of *learning to rank* serves this purpose, and can be performed in a hierarchical manner, thus it can be applied into the searching stage of partition-based ANNS methods. The challenging point is to determine the input and label (supervision) information for the network model according to the index structure.

5.2 Learning to Route

The method in [31] proposes a routing model over graph based on limitation learning. Specifically, this method develops a model that learns a routing policy to find nearest neighbors over a given proximity graph. It reformulates the graph routing algorithm as a probabilistic model and trains it by maximizing the probability of optimal routing for a large set of training queries.

The typical routing algorithm over a proximity graph is based on a greedy-based beam search, which is similar to Algorithm 1. The key idea of conventional heuristic-based greedy routing is to iteratively expand the nearest vertex from a heap of candidates based on their euclidean distances to the query. However, heuristic-based greedy routing would easily make the algorithm track into a local optimal solution.

Instead of selecting a vertex that has the smallest distance to a query, this method selects the next vertex v_i based on a Softmax probability distribution over vertices in the candidate heap H:

$$\mathcal{P}(v_i|q, H, \mathbf{\Theta}) = \frac{\exp(\langle \mathcal{F}_{\boldsymbol{\theta}}(v_i), \mathcal{G}_{\hat{\boldsymbol{\theta}}}(q)\rangle)}{\sum_{v_j \in H} \exp(\langle \mathcal{F}_{\boldsymbol{\theta}}(v_j), \mathcal{G}_{\hat{\boldsymbol{\theta}}}(q)\rangle)} \tag{36}$$

where $\mathcal{F}_{\boldsymbol{\theta}}(\cdot) \in \mathcal{R}^t$ is a neural network for reference items with parameters $\boldsymbol{\theta}$, $\mathcal{G}_{\hat{\boldsymbol{\theta}}}(\cdot) \in \mathcal{R}^t$ is another network for query items with parameters $\hat{\boldsymbol{\theta}}$, $\mathbf{\Theta} = \{\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}\}$, and $\langle \cdot \rangle$ is inner product operation. The key idea of this method is to learn two kinds of embeddings for the reference items and query items, respectively, such that the inner product of their embeddings can help the routing algorithm reach the actual nearest neighbor v^* with smaller number of hops (over graph) compared with their euclidean distance. Once the search terminates, the top-k vertices with the highest probabilities are retrieved as the target results for query.

The above probabilistic model (36) is learned by the paradigm of imitation learning [127], [129]. Specifically, an oracle function $Ora(H)$ is defined as a function which chooses vertices from H that are closest to true nearest neighbor v^* in terms of hops over graph. The values of $Ora(H)$ for each training query can be achieved by a breadth-first search approach. Then, the goal is to maximize the log-likelihood of probability routing:

$$\begin{aligned} J = \mathop{\mathbf{E}}_{q,v^*} \Big[& \sum_{v_i, H_i \in Rout_{\mathbf{\Theta}}(q)} \log \mathcal{P}(v_i \in Ora(H_i)|q, H_i, \mathbf{\Theta}) \\ & + \log \mathcal{P}(v^* \in TopK|q, V, \mathbf{\Theta}) \Big] \end{aligned} \tag{37}$$

where $v_i, H_i \in Rout_{\mathbf{\Theta}}(q)$ denotes the selected vertices and the corresponding heap states based on the routing algorithm with parameters $\mathbf{\Theta}$. The probabilistic learning procedure is similar to Algorithm 1. In each search session, routing algorithm is performed using the current parameters of $\mathcal{F}$ and $\mathcal{G}$, i.e., $\mathbf{\Theta}$, during which the search trajectories are collected. Once the session terminates, $\mathbf{\Theta}$ will be updated via gradient descent algorithm, which is guided by the oracle function $Ora(H)$ (i.e., supervision) to encourage the algorithm to find the optimal routing with as less as number of hops. Then another session starts again, until the algorithm converges to a good performance.

In terms of the network architecture, this method utilizes the graph convolutional network [80] as function $\mathcal{F}$ to learn the embeddings for reference items. For the query branch $\mathcal{G}$, the identity transformation or linear transformation can be employed, as it should be computationally efficient for online query pre-processing. The network $\mathcal{F}$ and $\mathcal{G}$ are jointly trained in order to maximize the objective function (37).

Comments. The key idea of *learning to route* is to replace the heuristic routing policy with the policy learned by deep learning techniques, thus can be applied into current existing graph-based ANNS methods. The challenging point is to extract the routing metric from the studied task, and then design the network architecture and loss function to learn a new routing metric from index structure and data. Once the model is learned, the routing algorithm with the learned routing metric can help make the algorithm jump out of local optimum, finally reaching the global optimum.

5.3 Learning to Early Termination

The method in [33] proposes to learn a termination condition for each individual query to speed up the querying search of the existing ANNS methods. Specifically, this method builds a learning model to learn and predict the stop condition for a certain query, so as to obtain similar accuracy with less amount of search overhead compared with those fixed configurations.

Conventional methods adopt fixed configurations (i.e., termination conditions) for all queries. However, due to the item vector distributions and the index structures, the number of reference items that need to be accessed to retrieve the nearest neighbor varies greatly for different queries. In addition, the intermediate searching result after a certain amount of search is a crucial information which implies how much more search needs to be executed. Motivated by this, this method builds a model that takes the current query-based state information as input, and outputs a numerical value showing how much more work needs to be done, which can be formulated as:

$$\mathcal{F}(query\_f, index\_f, intermediate\_f) = r \tag{38}$$

where the $query\_f$ is the query item feature, $index\_f$ is an index-based feature, $intermediate\_f$ is a kind of intermediate feature, and r is a numerical value which is directly related to the search work. The target label $\hat{r}$ that the model needs to learn is produced by executing an actual ANN search until the true nearest neighbor is found.

Here is an instance of applying the above model into a graph-based ANNS method HNSW [15]. In HNSW, the search amount for a query is quantified by the number of distance evaluations η_{dist} in the base layer. Thus, the task of this model is to predict the minimum η_{dist} for each query during search. For the input features of model $\mathcal{F}$, the $query\_f$ is straightforward, i.e., $\mathbf{q}$. The $index\_f$ is defined as the distance between query and the start node in base layer, $f_1 = \|\mathbf{q}, \mathbf{v}_{star}\|_2$. The $intermediate\_f$ consists of four features $\{f_2, f_3, f_4, f_5\}$. Features f_2 and f_3 are represented by the distances between query and the 1st, and the 10-th neighbors after γ distance evaluations, respectively. γ is a hyperparameter showing how much search should be performed before using the results as intermediate features, which can be tuned by grid search. Then the ratios between these two features and f_1 are adopted as the last two features, i.e., $f_4 = \frac{f_2}{f_1}$ and $f_5 = \frac{f_3}{f_1}$. The target (label) value, i.e., minimum η_{dist} in base layer, is generated by performing an actual ANN search until the ground-truth nearest neighbor is found. The existing networks can be designed for the model architecture, and l_2 norm loss can be employed for training.

Once the model is learned, for each query, the algorithm performs a fixed amount of search until the intermediate results are ready. Then the input information is collected and fed to the learned model to predict the stop condition. If the predicted stop condition has passed, the algorithm terminates immediately. Otherwise, the searching keeps going until the stop condition is met.

Comments. Since almost all ANNS methods need to first construct the index and then perform the querying search, the idea of *learning to early termination* can be integrated into current existing ANNS methods for speeding up the search with little accuracy degradation.

5.4 Summary

From the above review, *learning to early termination* can be applied into existing ANNS methods, which leads to a straightforward question: why not apply this idea into *learning to rank* and *learning to route*, for achieving a similar accuracy while getting a smaller latency. The challenging point is that the network model should be carefully designed to not only learn the searching policy but also learn the early stop conditions for different queries.

Overall, the key idea of *learning to search* is to replace the heuristic searching policy with the learned searching policy, thus can be applied into the searching stages of the current existing ANNS methods for achieving a better performance. With the advances of deep learning, *learning to search* for ANNS methods is still worth investigating and exploring.

6 End-To-End Learning

Based on the above review and analysis of existing deep learning-based ANNS methods, we can observe that current deep learning methods aim at either learning to index or learning to search. Why not combine the index learning and search learning together, resulting a new learning paradigm for ANNS, *end-to-end learning*, which might lead to an extraordinary searching performance.

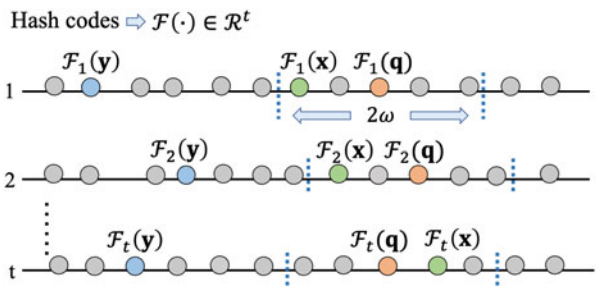
6.1 Motivation

Learning to index-based methods focus on learning a data-sensitive index, and then the querying search is performed on this learned index with traditional heuristic-based searching policy. Learning to search-based approaches aim at learning an index-sensitive searching policy according to a constructed heuristic-based index. Then the querying search is executed on this heuristic index with the learned searching strategy. However, both two learning paradigms have a common issue that the final searching performance cannot directly benefit from learning from data. In contrast, *end-to-end* learning eliminates this issue by jointly learning these two stages without using the heuristic strategies, such that the knowledge learned from data can directly contribute to the final searching performance.

End-to-end learning is a widely adopted learning paradigm in machine learning. However, there is currently no existing work that is about end-to-end learning for ANNS problem. Thus, in this section, we try our best to explore and discuss this important learning paradigm for ANNS. Specifically, next, we will propose and elaborate our ideas in details on how to apply end-to-end learning into the three categories of ANNS methods, respectively.

6.2 End-To-End Learning for Hashing-Based Methods

For hashing-based methods, the challenge of applying end-to-end learning is to parameterize the query processing algorithm. Here, we take hash code ranking for example to elaborate our idea.



Hash codes ⇨ $\mathcal{F}(\cdot) \in \mathcal{R}^t$
$\mathcal{F}_1(\mathbf{y})$
$\mathcal{F}_1(\mathbf{x})$
$\mathcal{F}_1(\mathbf{q})$
1
2ω
$\mathcal{F}_2(\mathbf{y})$
$\mathcal{F}_2(\mathbf{x})$
$\mathcal{F}_2(\mathbf{q})$
2
$\mathcal{F}_t(\mathbf{y})$
$\mathcal{F}_t(\mathbf{q})$
$\mathcal{F}_t(\mathbf{x})$
t

Fig. 4. An illustration of end-to-end learning for hashing-based methods (for non-binary case). The orange circles denote the hash code of query **q**. Suppose that $\alpha = 1$, then the item $\mathbf{x}$ represented by green circles is a candidate item for **q** while the item $\mathbf{y}$ denoted by blue circles is a non-candidate item.

Given a dataset **X**, and a query **q**, the query processing algorithm is actually to find a candidate set for **q** by sorting the (Hamming) distances of data items to query based on their (binary) hash codes in ascending order. We introduce a scan range variable w to parameterize this procedure, then by combining the hashing and querying together, our end-to-end learning model is formulated as:

$$C(\mathbf{q}, w) = \left\{ \mathbf{x} \in \mathbf{X} \middle| \sum_{i=1}^{t} \mathbf{1}[|\mathcal{F}_i(\mathbf{x}) - \mathcal{F}_i(\mathbf{q})| \le w] \ge \lceil \alpha \cdot t \rceil \right\} \tag{39}$$

where $\mathbf{1}[\cdot]$ is the indicator function, $\mathcal{F}_i(\mathbf{x})$ represents the i-th hash bit of $\mathbf{x}$, t is the length of a complete hash code, and α is a hyperparameter. Scan range w measures the closeness of the hash codes of $\mathbf{x}$ and **q**. If $\mathcal{F}(\mathbf{x})$ is in the binary form, then the w is set to be 0, which means that if $\mathcal{F}_i(\mathbf{x})$ and $\mathcal{F}_i(\mathbf{q})$ collides on the i-th hash bit, then they should be the same. If $\mathcal{F}(\mathbf{x})$ is not in the binary form (e.g., the NeOPH [30]), then the w is a real value, which measures how close $\mathcal{F}_i(\mathbf{x})$ is to $\mathcal{F}_i(\mathbf{q})$ can they be regarded as a "collision" case. This formulation means that a reference item can be retrieved as a candidate if the "collision" count with respect to query **q** based on scan range w is at least $\lceil \alpha \cdot t \rceil$.

The idea of our end-to-end learning is that we directly maximize the recall attained by the items in $C(\mathbf{q}, w)$ by learning t hash functions (e.g., a neural network). Assume that the k nearest neighbours (NN) of **q** is denoted as k-NN (**q**), then we introduce a quality evaluation function $\mu(C(\mathbf{q}, w))$ ($\mu(\cdot) \in [0, 1]$) such that the higher value means that the k-NN items of **q** have a higher collision count compared with other items and more k-NN items of **q** can be included in $C(\mathbf{q}, w)$. The $\mu(\cdot)$ can be the Mean Average Precision (MAP) or the NDCG score [107]. Then the final loss function over a set of training queries is defined as:

$$J = -\frac{1}{N} \sum_{i=1}^{N} \mu(C(\mathbf{q}_i, w)) \tag{40}$$

We can see that formulation (40) is non-smooth and non-continuous due to the two logical operators, which makes the standard gradient descent algorithm infeasible on this problem. To address this issue, we can employ the relaxation strategy used in NeOPH [30]. An illustration of our end-to-end learning for hashing-based methods (for non-binary case) can be seen in Fig. 4.

Another important issue that we need to consider for the non-binary case is to generalize the proposed model to varying w. One simple way is to optimize an expected μ when w varies according to a probability distribution $p(w)$, which is formulated as:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\mathbf{E}_{w\sim p(w)}[\mu(C(\mathbf{q}_i, w))] \tag{41}$$

Another way is to use some heuristic methods, such as gradually changing the w when $|\mathcal{F}_i(\mathbf{x}) - \mathcal{F}_i(\mathbf{q})|$ does not meet the requirements during the training process.

The final issue needs to be considered is that increasing w might increase the μ value, however it incurs additional cost as more items need to be accessed. Therefore, we may design a loss function that combines the μ value with the query processing cost modelled by $\mathcal{G}(w)$.

6.3 End-To-End Learning for Partition-Based Methods

Learning to partition [28] and *learning to rank* [32] belong to partition-based class, where the former is based on *learning to index* paradigm and the latter is based on *learning to search* paradigm. However, they cannot be trivially combined together to result in the *end-to-end learning* paradigm.

Here, we present our idea on how to conduct end-to-end learning for data partitioning for ANNS. The challenge here is to take the query processing algorithm into consideration when designing the learning model. We only consider the one-level partitioning case, and the hierarchical partitioning scheme can be similarly designed. Given a dataset $\mathbf{X}$, a query $\mathbf{q}$ and its k-nearest neighbours $\{\mathbf{x}_1^*, ..., \mathbf{x}_k^*\}$ with respect to $\mathbf{X}$. We randomly sample a subset of items from $\mathbf{X}$ without $\{\mathbf{x}_1^*, ..., \mathbf{x}_k^*\}$ to form a negative sample set $\{\bar{\mathbf{x}}_1, ..., \bar{\mathbf{x}}_n\}$. Suppose that we partition $\mathbf{X}$ into M bins, and the partitioning function can be defined similar to [28]:

$$\mathcal{F}(\mathbf{x}) = (p_1, ..., p_M) \tag{42}$$

where the $(p_1, ..., p_M)$ are the predicted probability distribution on the M bins for which one item $\mathbf{x}$ should be allocated. For end-to-end learning, our loss function is designed to realize two goals: 1) The query processing action should be included into the learning procedure. 2) The learned partitions should be balanced and locality sensitive.

To realize the first goal, we design a loss function that seeks to minimize the KL divergence between $\mathcal{F}(\mathbf{q})$ and $\mathcal{F}(\mathbf{x}_i^*)$, and to maximize the KL divergence between $\mathcal{F}(\mathbf{q})$ and $\mathcal{F}(\bar{\mathbf{x}}_j)$ at the same time, which is formulated as:

$$\bar{J}(\mathbf{q}) = \frac{1}{k}\sum_{i=1}^{k} KL(\mathcal{F}(\mathbf{q}), \mathcal{F}(\mathbf{x}_i^*)) - \frac{1}{n}\sum_{j=1}^{n} KL(\mathcal{F}(\mathbf{q}), \mathcal{F}(\bar{\mathbf{x}}_j)) \tag{43}$$

where the $KL(\cdot,\cdot)$ is to measure the KL divergence between two distributions. This function encourages that query $\mathbf{q}$ and its neighbours $\mathbf{x}_i^*$ should be allocated into the same bins while $\mathbf{q}$ and the faraway items $\mathbf{x}_j$ should be allocated into different bins.

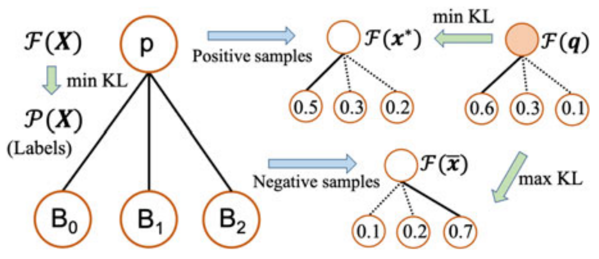

$\mathcal{F}(\boldsymbol{X})$
p
min KL
$\mathcal{P}(\boldsymbol{X})$
(Labels)
B$_0$
B$_1$
B$_2$
Positive samples
$\mathcal{F}(\boldsymbol{x}^*)$
min KL
$\mathcal{F}(\boldsymbol{q})$
0.5
0.3
0.2
0.6
0.3
0.1
Negative samples
$\mathcal{F}(\bar{\boldsymbol{x}})$
max KL
0.1
0.2
0.7

Fig. 5. An illustration of end-to-end learning for partition- based methods (for one-level partitioning case). The number of partitions is set to be 3, $\mathbf{X}$ is the entire dataset, and $\mathbf{x}^*$ and $\bar{\mathbf{x}}$ are the positive and negative sample sets for query $\mathbf{q}$, respectively.

To realize the second goal, we can adopt the idea in [28] that first builds a k-NN graph for $\mathbf{X}$ and then partitions this graph into M balanced bins using a graph partitioner. Then these M partitioned bins can be used as label (supervision) information to learn the partitioning function. Finally, our loss function in terms of query $\mathbf{q}$ can be formulated as:

$$J(\mathbf{q}) = \frac{1}{N}\sum_{\mathbf{x}\in\mathbf{X}} KL(\mathcal{F}(\mathbf{x}), \mathcal{P}(\mathbf{x})) + \bar{J}(\mathbf{q}) \tag{44}$$

where the $\mathcal{P}(\mathbf{x})$ is a ground-truth distribution for $\mathbf{x}$ directly obtained from the M partitioned bins. The loss function over a set of training queries can be accordingly defined. An illustration of our end-to-end learning for partition-based methods can be seen in Fig. 5.

Once the partitioning function $\mathcal{F}$ is learned, it can be used not only to allocate unseen reference items to the most suitable bins, but also to predict a good ranking distribution for the partitioned bins for query items. By retrieving through more bins in the order predicted by $\mathcal{F}$, we can achieve better search accuracy, allowing for a better trade-off between computational resources and accuracy.

6.4 End-To-End Learning for Graph-Based Methods

Graph-based methods contain two steps: 1) Proximity graph construction. 2) A routing algorithm over the proximity graph. The work in [63] proposes to learn a proximity graph to serve the heuristic-based routing algorithm while the method in [31] proposes to learn a routing strategy over a given proximity graph. Thus, it is naturally leading to a question: can we combine these two learning stages together, such that the learned graph can better serve the learned routing algorithm, so as to further improve the final performance? The main challenge here is how to jointly learn these two stages together, resulting in a kind of end-to-end learning. Based on the review of these two work [31], [63], we can observe that there is a common point between these two learning procedures: collect training samples using the current model parameters in one search session and then update the model parameters via optimization by using the training samples generated in previous steps when the session stops. This procedure is performed iteratively, until the algorithm converges or the stop condition is met. In other words, the graph learning and routing algorithm learning can be trained together, which provides the opportunity for us to design an end-to-end learning model for graph-based methods.

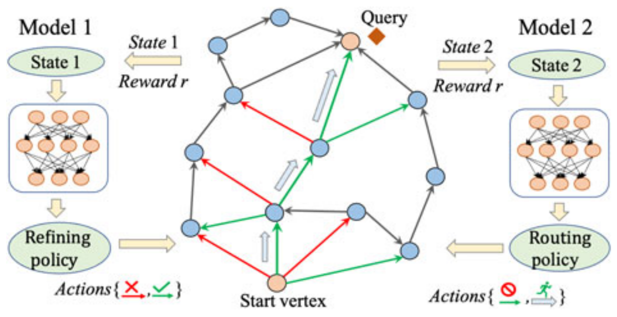


Fig. 6. An illustration of end-to-end learning for graph-based methods. Model 1 is used to learn the refining policy while model 2 is for learning the routing policy.

To realize the above goal, we can design an asymmetric architecture with two network models where one model is used to learn the graph construction (refinement) and another model is responsible for learning the routing policy. These two models are jointly trained in a way that the final routing algorithm can directly benefit from learning from data. Specifically, recall the Algorithm 1, in line 9, the *Agent* (model 1) is used to predict what connections need to be kept according to the current search state. Here, we add another *Agent* (model 2) in line 6 to predict which vertex needs to be extracted from the candidate heap H (w.r.t query) rather than using the euclidean distance for selection. Then, the Algorithm 1 with two agents (models) will involve in the subsequent training procedures. Specifically, after the initialization for the two models, Algorithm 1 is performed to collect routing trajectories. Once a search session is finished, we can get a reward that is related to the routing cost and the routing accuracy. Smaller routing cost and higher routing accuracy can get a higher reward, and vice versa. Then, the collected trajectories are utilized to train the two agents by reinforcement learning, where the updating of model parameters is guided by the reward of the current session. And then the next search session starts again with the updated model parameters, followed by the next round of model training. Note that the training trajectories are generated with the interaction of both two models. Thus, the training for these two models is actually an end-to-end learning procedure. An illustration of our end-to-end learning for graph-based methods can be seen in Fig. 6.

For the design of model architecture and objective function, we can borrow the ideas from the two work [31], [63], or we can propose a new model architecture and objective function that can well fit our end-to-end learning setting. For instance, in [63], the agent can just predict what edges to be kept and delete others, so we can consider enabling the agent to predict what vertices need to be connected over graph to make more possibilities. In [31], the graph neural network (GNN) is adopted to learn the vertex embeddings over graph, however, in the end-to-end learning setting, the graph topology is changing, which makes GNN unfeasible in this task. Thus, more suitable models need to be investigated and designed.

6.5 Summary

End-to-end learning can be generalized to other kinds of indices and the associated query processing methods. The key point is to parameterize the query processing algorithm (e.g., the scan range variable w for hash code ranking) when designing the model function. It aims to fully make use of the knowledge learned form data to directly serve the final querying search, which is a new learning paradigm for ANNS. With the development of deep learning techniques, end-to-end learning for ANNS can well address the challenges from big data and well meet the increasing demands of users. Thus, we believe that end-to-end learning will become a new and promising research direction for ANNS community in the near future.

7 Discussions About Deep Learning

Although deep learning achieves great success in many domains, it still has some inherent shortcomings. Deep learning can be regarded as a non-linear function simulator to some extent. With the increase in the number of network layers and the massive use of different kinds of activation functions and operations in each layer, the simulation capability of deep learning is becoming more and more powerful. However, its interpretability is getting worse and worse at the same time, as the distributions of data streams inside the networks do not follow the regular distributions and we cannot easily analyze them using known mathematical methodology. Thus, deep learning currently cannot support the c-ANNS with theoretical guarantee.

Although many efforts are now devoted to investigate methods to interpret the inner working mechanism of deep learning, there is not much theoretical progress that has been made so far. The interpretability of deep learning is currently an open problem, attracting the interests of researchers.

8 Evaluation

8.1 Evaluation Measures

There are four main concerns for deep learning-based ANNS methods: preprocessing time, index size, search efficiency and search quality. Preprocessing time usually includes the training time of deep learning model and the index construction time. Index size is measured as the actual space cost of index.

The search efficiency is measured as the time taken to retrieve the target results for a query, which is usually calculated as the average time over a set of queries. For the search quality, there are several metrics that are widely used: recall, accuracy, precision-recall curve, and mean average precision (MAP). These metrics are usually computed over a certain number of retrieved target results. For each query, we retrieve its T nearest items, and then rerank these candidates using the original feature representations and return the top k items as the final output. Recall is computed as the ratio of the number of true nearest neighbours in the k returned items to k. Accuracy is equal to $\sum_{i=1}^{k} \frac{dist(q,kANN(q)[i])}{dist(q,kNN(q)[i])}$, where the $kANN(q)[i]$ is the i-th nearest neighbour of query q returned by an approximate algorithm, and the $kNN(q)[i]$ is the i-th true nearest neighbour computed by linear scan over the original features. The $dist$ is the euclidean distance between two items in original space. Accuracy is usually adopted by the algorithms that take the point data as input where a point is a high dimensional feature vector. Clearly, accuracy closing to 1.0 is better and vice versa.

For methods like most of deep hashing methods directly taking the raw image data as input, the precision-recall

TABLE 2
A Summary of Evaluation Datasets

Dataset	Dim	Reference set	Learning set	Query set	Type
CIFAR-10	512	50K	-	10K	Image
NUS-WIDE	500	$\approx$ 270K	-	-	Image
YoutubeFace	1770	$\approx$ 350K	-	-	Video
Msong	420	$\approx$ 1M			Audio
SIFT1M	128	1M	100K	10K	Image
GIST1M	960	1M	500K	1K	Image
GloVe1.2M	100	$\approx$ 1.2M	-	-	Text
SPACEV1B	100	1B	100M	$\approx$ 30K	Text

curve and MAP are widely adopted. The precision is defined as the ratio of the number of retrieved true items to T, i.e., the retrieved position. The recall is also computed according to these T retrieved items. Then the precision-recall curve can be achieved by varying the retrieved position T. A larger precision-recall value indicates better performance. The MAP is defined as the mean of average precision over a set of queries. The average precision for a query is computed as $\sum_{t=1}^{T} P(t)\phi(t)$, where the $P(t)$ denotes the precision of top t retrieved results in the ranked list and $\phi(t)$ is the change in recall from items $t-1$ to t.

8.2 Datasets

Various and diverse datasets have been used to evaluate deep learning-based ANNS methods, including image, video, audio and text data. The features from these data are extracted as high dimensional vectors for ANNS performance evaluation. For most of deep learning to hash methods, the original raw data is usually used directly. This subsection briefly introduces some representative datasets from small to large scale, which are summarized in Table 2.

CIFAR-10 [130][1] is a labeled subset of TinyImage dataset, which consists of 60K 32×32 color images in 10 classes, with each image being represented by a 512-dimensional GIST feature vector.

NUS-WIDE [131] is a public web image dataset which contains 269,648 images downloaded from Flickr.com. Each of the images is annotated with multiple labels referring to 81 categories, which is extracted as a 500-dimensional bag-of-words vector.

YoutubeFace [132][2] is a face video dataset, containing 3,425 videos of 1,595 different people, with a total of about 600K frames. All the videos were downloaded from YouTube.

Msong [3] is a collection of audio features and metadata for a million contemporary popular music tracks with each record being a 420-dimensional vector.

SIFT1M [111][4] is an image dataset, which consists of 1M 128-dimensional SIFT feature vectors as reference set, 100K vectors as learning set and 10K as the query set.

GIST1M [111][5] is composed of 1M 960-dimensional GIST feature vectors as reference set, 500K vectors as learning set and 1K as the query set.

GloVe1.2M [133][6] is a text dataset, containing 1.2M 100-dimensional feature vectors extracted from Twitter.

SPACEV1B [7] is a new web search related dataset released by Microsoft Bing. It consists of document and query vectors encoded by Microsoft SpaceV Superior model to capture generic intent representation.

8.3 Training Sets

There are three main ways to prepare the training sets for learning the network models of ANNS methods. The first way is to utilize a separated learning set as a training set which is not included in the reference set. Some datasets like SIFT1M and GIST1M already contain the separated learning sets that can be directly used. The second option is to sample a subset from the reference set. What percentage of reference set needs to be sampled out can be regarded as a hyper-parameter, which can be tuned according to the actual performance on the associated dataset. The third choice is directly using the whole reference set as the training set. However, this way might lead to over-fitting, which depends on the studied tasks. If the model learning involves queries, one also needs to sample a small subset from reference set as the training query set.

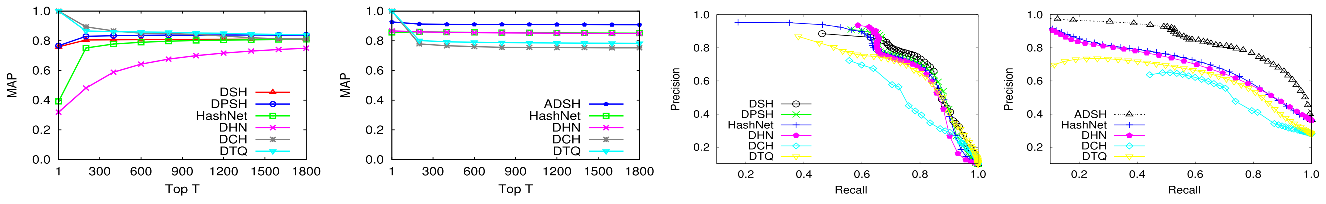
For learning to index-based methods, the model learning is mainly conducted on the training set. Sometimes, it also involves the training query set (e.g., neural order preserving hashing [30]). For learning to search-based methods, the model learning is mainly based on the constructed index and a training query set. Note that the constructed index usually contains the information of the reference set.

8.4 Experiments and Analyses

In this subsection, we conduct two groups of experiments for some representative deep learning-based ANNS methods on four datasets and provide the detailed performance analyses for them based on the empirical results. Specifically, the first group of experiment is for the algorithms that directly take the original raw data (e.g., raw image data) as input, and the second group is for the methods taking the point data (e.g., high dimensional feature vectors) as input.

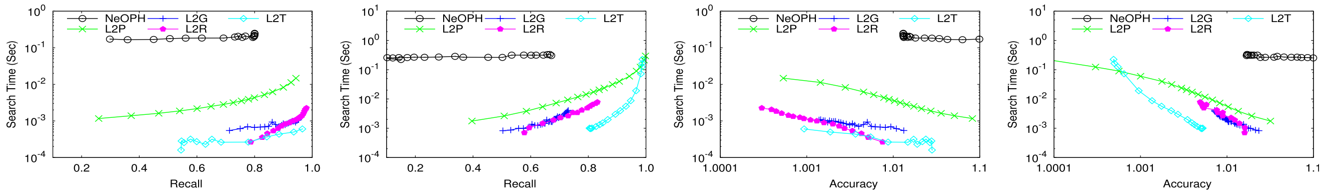
For the first group of experiment, the comparison methods include seven representative deep hashing methods: deep supervised hashing (DSH) [40], asymmetric deep supervised hashing (ADSH) [45], deep pairwise-supervised hashing (DPSH) [49], HashNet [51], deep hashing network (DHN) [24], deep cauchy hashing (DCH) [53], and deep triplet quantization (DTQ) [25]. The datasets CIFAR-10 and NUS-WIDE are used to evaluate the performance of these methods, where the raw image pixel data of these two datasets is directly used. The MAP and precision-recall curve are employed as the evaluation metrics. The source codes of all these methods are public available online and are trained according to the default settings as suggested by original authors. The number of bits of the output binary codes is set

1. http://www.cs.toronto.edu/~kriz/cifar.html
2. http://www.cs.tau.ac.il/~wolf/ytfaces/index.html
3. http://www.ifs.tuwien.ac.at/mir/msd/download.html
4. http://corpus-texmex.irisa.fr/
5. http://corpus-texmex.irisa.fr/
6. https://nlp.stanford.edu/projects/glove/
7. https://big-ann-benchmarks.com/



(a) CIFAR-10 (b) NUS-WIDE (c) CIFAR-10 (d) NUS-WIDE

Fig. 7. Search performance of the first group of deep hashing methods on CIFAR-10 and NUS-WIDE.



(a) SIFT1M (b) GloVe1.2M (c) SIFT1M (d) GloVe1.2M

Fig. 8. Search performance of the second group of ANNS methods on SIFT1M and GloVe1.2M.

to be 64 for all algorithms. As all comparison methods utilize the Hamming distance-based hash code ranking to perform the search, their search times can be viewed as the same with the same settings in the number of bits and the number of retrieved candidates. Thus, we do not report the search time in this group of experiment.

The experimental results for the first group of algorithms are reported in Fig. 7, where we vary the number of retrieved candidates T to get the MAP curve. On CIFAR-10 dataset, we can observe that DTQ, DCH and DPSH achieve relatively better performance compared with DSH, HashNet and DHN in terms of the MAP results. For the precision-recall results, the performance gap among DSH, DPSH, HashNet, DHN and DTQ for large recall is not very prominent. Relatively, DSH achieves the best performance while DCH ranks last. On NUS-WIDE dataset, according to the results from MAP and precision-recall curves, ADSH achieves the best performance, HashNet and DHN rank second, followed by DTQ and DCH. Based on the experimental results on CIFAR-10 and NUS-WIDE, we conclude that it is hard for a deep hashing method to achieve good performance on all datasets, and different methods have different advantages on different datasets.

For the second group of experiment, the comparison methods involve neural order preserving hashing (NOPH) [30], learning to partition (L2P) [28], learning to graph construction (L2G) [63], learning to route (L2R) [31], and learning to termination (L2T) [33]. The datasets SIFT1M and GloVe1.2M are utilized for performance evaluation, where their associated high dimensional feature vectors are used as input for all algorithms. The source codes of these algorithms are public available online or achieved from original authors, and their learning models are trained as suggested by authors. The recall and accuracy are adopted as the evaluation metrics, and the search efficiency is also reported.

The results of the second group of experiment are plotted in Fig. 8. We vary the number of retrieved candidates to get the curves for the recall and the accuracy with respect to search time, respectively. On SIFT1M dataset, L2T outperforms the other methods in large recall and accuracy with smaller search time. This is probably due to the early termination scheme used in L2T. L2R performs similar to L2G, followed by L2P and NeOPH. Note the NeOPH mainly focuses on the I/O efficiency, thus is not optimized for CPU search time efficiency. On GloVe1.2M dataset, the performance comparison results are consistent with that on SIFT1M dataset. Based on the empirical results, we have several observations. (1) Learning-based early termination scheme is a good choice to improve the search efficiency. (2) Learning to search-based methods performs no worse than learning to index-based methods, which is because learning to search contributes more to the final search performance.

8.5 Applications

Due to the success of deep learning and the significance of approximate nearest neighbour search (ANNS), deep learning-based ANNS methods have been applied into a wide range of real-world application scenarios, including image search and retrieval [27], [41], [54], [57], [112], video search [134], [135], object detection [136], visual recognition [137], person re-identification [55], classification [138], information retrieval [139], recommendation system [140], natural language processing [141], and chemical structure analysis [142]. In these applications, each object is usually first embedded into a high-dimensional vector by deep learning-based techniques, and then the reference vector set can be indexed and searched by using the ANNS methods to efficiently support the applications. Note that the deep learning techniques can also be applied into the indexing and searching stages, resulting in the learning to index and learning to search paradigms as reviewed in this paper. These applications show that deep learning enables ANNS methods to be applied into diverse disciplines and achieve successes, which implies that deep learning-based ANNS methods is a promising research direction and is able to support more applications in the future.

9 Conclusion

In this paper, we conduct a comprehensive survey of the deep learning-based methods for approximate nearest neighbour search (ANNS). Specifically, we divide the existing methods into two categories according to two learning paradigms: learning to index and learning to search. Then we carefully review and analyze the differences and connections among the algorithms in each category. Finally, based on the overview and analysis of the existing deep learning-based ANNS algorithms, we propose a new learning paradigm for ANNS that jointly learns the indexing and searching together, namely end-to-end learning, which will be a new and promising research direction for the ANNS community.

REFERENCES

[1] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on P-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.

[2] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Efficient and accurate nearest neighbor and closest pair search in high-dimensional space," *ACM Trans. Database Syst.*, vol. 35, no. 3, pp. 1–46, 2010.

[3] J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 541–552.

[4] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin, "SRS: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index," *Proc. VLDB Endowment*, vol. 8, no. 1, pp. 1–12, 2014.

[5] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search," *Proc. VLDB Endowment*, vol. 9, no. 1, pp. 1–12, 2015.

[6] Y. Zheng, Q. Guo, A. K. Tung, and S. Wu, "Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 2023–2037.

[7] W. Liu, H. Wang, Y. Zhang, W. Wang, and L. Qin, "I-lsh: I/O efficient c-approximate nearest neighbor search in high-dimensional space," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1670–1673.

[8] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2008, pp. 1–8.

[9] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014.

[10] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *Proc. 40th Annu. ACM Symp. Theory Comput.*, 2008, pp. 537–546.

[11] J. Chen, H. Ren Fang, and Y. Saad, "Fast approximate KNN graph construction for high dimensional data via recursive Lanczos bisection," *J. Mach. Learn. Res.*, vol. 10, no. 69, pp. 1989–2012, 2009.

[12] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proc. Int. World Wide Web Conf.*, 2011, pp. 577–586.

[13] W. Li et al., "Approximate nearest neighbor search on high dimensional data–experiments, analyses, and improvement," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 8, pp. 1475–1488, Aug. 2020.

[14] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Inf. Syst.*, vol. 45, pp. 61–68, 2014.

[15] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, Apr. 2020.

[16] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *Proc. VLDB Endowment*, vol. 12, pp. 461–474, 2019.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[19] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Learn. Representations*, 2015.

[20] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, 2014, pp. 2156–2162.

[21] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 2475–2483.

[22] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3270–3278.

[23] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 1183–1192.

[24] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, 2016, pp. 2415–2421.

[25] B. Liu, Y. Cao, M. Long, J. Wang, and J. Wang, "Deep triplet quantization," in *ACM Int. Conf. Multimedia*, 2018, pp. 755–763.

[26] F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, and H. T. Shen, "Unsupervised deep hashing with similarity-adaptive and discrete optimization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 3034–3044, Dec. 2018.

[27] B. Klein and L. Wolf, "End-to-end supervised product quantization for image search and retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 5036–5045.

[28] Y. Dong, P. Indyk, I. Razenshteyn, and T. Wagner, "Learning space partitions for nearest neighbor search," in *Proc. Int. Conf. Learn. Representations*, 2019.

[29] W. Zhang, D. Wu, Y. Zhou, B. Li, W. Wang, and D. Meng, "Deep unsupervised hybrid-similarity hadamard hashing," in *Proc. ACM Int. Conf. Multimedia*, 2020, pp. 3274–3282.

[30] M. Li, Y. Zhang, Y. Sun, W. Wang, I. W. Tsang, and X. Lin, "I/O efficient approximate nearest neighbour search based on learned functions," in *Proc. Int. Conf. Data Eng.*, 2020, pp. 289–300.

[31] D. Baranchuk, D. Persiyanov, A. Sinitsin, and A. Babenko, "Learning to route in similarity graphs," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 475–484.

[32] C.-Y. Chiu, A. Prayoonwong, and Y.-C. Liao, "Learning to index for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 1942–1956, Aug. 2020.

[33] C. Li, M. Zhang, D. G. Andersen, and Y. He, "Improving approximate nearest neighbor search through learned adaptive early termination," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2539–2554.

[34] J. Wang, W. Liu, S. Kumar, and S. Chang, "Learning to hash for indexing big data - A survey," *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.

[35] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, Apr. 2018.

[36] M. Wang, X. Xu, Q. Yue, and Y. Wang, "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search," *Proc. VLDB Endowment*, vol. 14, no. 11, pp. 1964–1978, Jul. 2021.

[37] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proc. Int. Conf. Comput. Vis. Theory Appl.*, 2009, pp. 331–340.

[38] A. Babenko and V. Lempitsky, "The inverted multi-index," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 6, pp. 1247–1260, Jun. 2015.

[39] Y. Liu, H. Cheng, and J. Cui, "PQBF: I/O-efficient approximate nearest neighbor search by product quantization," in *Proc. Conf. Inf. Knowl. Manage.*, 2017, pp. 667–676.

[40] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2064–2072.

[41] K. Zhou, Y. Liu, J. Song, L. Yan, F. Zou, and F. Shen, "Deep self-taught hashing for image retrieval," in *Proc. ACM Int. Conf. Multimedia*, 2015, pp. 1215–1218.

[42] Y. Liu et al., "Deep self-taught hashing for image retrieval," *IEEE Trans. on Cybern.*, vol. 49, no. 6, pp. 2229–2241, Jun. 2019.

[43] Z. Li, J. Tang, L. Zhang, and J. Yang, "Weakly-supervised semantic guided hashing for social image retrieval," *Int. J. Comput. Vis.*, vol. 128, no. 8, pp. 2265–2278, 2020.

[44] T.-T. Do, A.-D. Doan, and N.-M. Cheung, "Learning to hash with binary deep neural network," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 219–234.

[45] Q.-Y. Jiang and W.-J. Li, "Asymmetric deep supervised hashing," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, 2018, pp. 3342–3349.

[46] D. Wu, Q. Dai, J. Liu, B. Li, and W. Wang, "Deep incremental hashing network for efficient image retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9069–9077.

[47] C. Yan, G. Pang, X. Bai, C. Shen, J. Zhou, and E. Hancock, "Deep hashing by discriminating hard examples," in *Proc. ACM Int. Conf. Multimedia*, 2019, pp. 1535–1542.

[48] M. Wang, W. Zhou, Q. Tian, Z. Zha, and H. Li, "Linear distance preserving pseudo-supervised and unsupervised hashing," in *Proc. ACM Int. Conf. Multimedia*, 2016, pp. 1257–1266.

[49] W.-J. Li, S. Wang, and W.-C. Kang, "Feature learning based deep supervised hashing with pairwise labels," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 1711–1717.

[50] E. Yang, T. Liu, C. Deng, W. Liu, and D. Tao, "Distillhash: Unsupervised deep hashing by distilling data pairs," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2946–2955.

[51] Z. Cao, M. Long, J. Wang, and P. S. Yu, "HashNet: Deep learning to hash by continuation," in *Proc. Int. Conf. Comput. Vis.*, 2017, pp. 5608–5617.

[52] Z. Cao, Z. Sun, M. Long, J. Wang, and P. S. Yu, "Deep priority hashing," in *Proc. ACM Int. Conf. Multimedia*, 2018, pp. 1653–1661.

[53] Y. Cao, M. Long, B. Liu, and J. Wang, "Deep cauchy hashing for hamming space retrieval," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1229–1237.

[54] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1556–1564.

[55] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang, "Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification," *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 4766–4779, Dec. 2015.

[56] J. Bai et al., "Deep progressive hashing for image retrieval," in *Proc. ACM Int. Conf. Multimedia*, 2017, pp. 208–216.

[57] J. Bai et al., "Deep progressive hashing for image retrieval," *IEEE Trans. Multimedia*, vol. 21, no. 12, pp. 3178–3193, Dec. 2019.

[58] K. He, F. Cakir, S. A. Bargal, and S. Sclaroff, "Hashing as tie-aware learning to rank," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4023–4032.

[59] Q. Hu, J. Wu, J. Cheng, L. Wu, and H. Lu, "Pseudo label based unsupervised deep discriminative hashing for image retrieval," in *Proc. ACM Int. Conf. Multimedia*, 2017, pp. 1584–1590.

[60] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen, "Deep quantization network for efficient image retrieval," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, 2016, pp. 3457–3463.

[61] L. Jin, Z. Li, and J. Tang, "Deep semantic multimodal hashing network for scalable image-text and video-text retrievals," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 20, 2020, doi: 10.1109/TNNLS.2020.2997020.

[62] L. Jin, K. Li, Z. Li, F. Xiao, G.-J. Qi, and J. Tang, "Deep semantic-preserving ordinal hashing for cross-modal similarity search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 5, pp. 1429–1440, May 2019.

[63] D. Baranchuk and A. Babenko, "Towards similarity graphs constructed by deep reinforcement learning," 2019, *arXiv:1911.12122*.

[64] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015.

[65] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 1799–1807.

[66] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Aug. 2013.

[67] G. Hinton et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[68] T. N. Sainath et al., "Improvements to deep convolutional neural networks for lvcsr," in *Proc. IEEE Workshop Autom. Speech Recognit. Understanding*, 2013, pp. 315–320.

[69] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2014.

[70] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.

[71] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 1701–1708.

[72] A. Bordes, S. Chopra, and J. Weston, "Question answering with subgraph embeddings," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 615–620.

[73] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2014, pp. 1–10.

[74] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[75] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, "Deep neural nets as a method for quantitative structure–activity relationships," *J. Chem. Inf. Model.*, vol. 55, no. 2, pp. 263–274, 2015.

[76] H. Y. Xiong et al., "The human splicing code reveals new insights into the genetic determinants of disease," *Science*, vol. 347, no. 6218, 2015, Art. no. 1254806.

[77] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[78] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[79] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[80] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[81] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[82] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.

[83] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 2017, *arXiv:1709.05584*.

[84] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional networks," in *Proc. Int. Conf. Des. Mater.*, 2018, pp. 929–934.

[85] A. G. Barto and R. S. Sutton, "Reinforcement learning," *Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1995, pp. 804–809.

[86] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[87] D. Silver et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[88] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takáč, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 9861–9871.

[89] J. Wang, N. Wu, W. X. Zhao, F. Peng, and X. Lin, "Empowering a* search algorithms with neural networks for personalized route recommendation," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 539–547.

[90] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6348–6358.

[91] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3108–3115.

[92] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.

[93] Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[94] L. Gao, J. Song, F. Zou, D. Zhang, and J. Shao, "Scalable multimedia retrieval by deep learning hashing with relative similarity learning," in *Proc. ACM Int. Conf. Multimedia*, 2015, pp. 903–906.

[95] Y. Weiss et al., "Spectral hashing," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2008, Art. no. 4.

[96] E. Yang, C. Deng, T. Liu, W. Liu, and D. Tao, "Semantic structure-based unsupervised deep hashing," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 1064–1070.

[97] L. Jin, X. Shu, K. Li, Z. Li, G.-J. Qi, and J. Tang, "Deep ordinal hashing with spatial attention," *IEEE Trans. Image Process.*, vol. 28, no. 5, pp. 2173–2186, May 2019.

[98] Q.-Y. Jiang, X. Cui, and W.-J. Li, "Deep discrete supervised hashing," *IEEE Trans. Image Process.*, vol. 27, no. 12, pp. 5996–6009, Dec. 2018.

[99] Q. Li, Z. Sun, R. He, and T. Tan, "Deep supervised discrete hashing," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2479–2488.

[100] F. Shen, X. Gao, L. Liu, Y. Yang, and H. T. Shen, "Deep asymmetric pairwise hashing," in *Proc. ACM Int. Conf. Multimedia*, 2017, pp. 1522–1530.

[101] J. P. Dmochowski, P. Sajda, and L. C. Parra, "Maximum likelihood in cost-sensitive learning: Model specification, approximations, and upper bounds," *J. Mach. Learn. Res.*, vol. 11, no. 108, pp. 3313–3332, 2010.

[102] Y. Cao, B. Liu, M. Long, and J. Wang, "HashGAN: Deep learning to hash with pair conditional wasserstein GAN," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1287–1296.

[103] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017.

[104] A. Hyvärinen, J. Hurri, and P. O. Hoyer, *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision*, vol. 39. Berlin, Germany: Springer, 2009.

[105] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1061–1069.

[106] Y. Weiss, R. Fergus, and A. Torralba, "Multidimensional spectral hashing," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 340–353.

[107] K. Järvelin and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents," in *Proc. Annu. Int. ACM SIGIR Forum Conf.*, 2017, pp. 243–250.

[108] F. Cakir, K. He, S. A. Bargal, and S. Sclaroff, "Hashing with mutual information," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2424–2437, Oct. 2019.

[109] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. Int. Conf. Mach. Learn.*, 2011, pp. 1–8.

[110] Z. Zhang, Y. Chen, and V. Saligrama, "Efficient training of very deep neural networks for supervised hashing," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 1487–1495.

[111] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[112] J. Chen and W. K. Cheung, "Similarity preserving deep asymmetric quantization for image retrieval," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, 2019, pp. 8183–8190.

[113] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 838–846.

[114] Y.-W. Zhan, X. Luo, Y. Wang, and X.-S. Xu, "Supervised hierarchical deep hashing for cross-modal retrieval," in *Proc. ACM Int. Conf. Multimedia*, 2020, pp. 3386–3394.

[115] Z. Li, J. Tang, and T. Mei, "Deep collaborative embedding for social image understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 9, pp. 2070–2083, Sep. 2019.

[116] A. Abdullah, A. Andoni, R. Kannan, and R. Krauthgamer, "Spectral approaches to nearest neighbor search," in *Proc. IEEE 55th Annu. Symp. Found. Comput. Sci.*, 2014, pp. 581–590.

[117] L. Cayton and S. Dasgupta, "A learning framework for nearest neighbor search," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2007, pp. 233–240.

[118] Z. Li, H. Ning, L. Cao, T. Zhang, Y. Gong, and T. S. Huang, "Learning to search efficiently in high dimensions," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 1710–1718.

[119] P. Ram and A. G. Gray, "Which space partitioning tree to use for search?," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 656–664.

[120] A. Andoni, A. Naor, A. Nikolov, I. Razenshteyn, and E. Waingarten, "Data-dependent hashing via nonlinear spectral gaps," in *Proc. 50th Annu. ACM SIGACT Symp. Theory Comput.*, 2018, pp. 787–800.

[121] P. Sanders and C. Schulz, "Think locally, act globally: Highly balanced graph partitioning," in *Proc. Int. Symp. Exp. Algorithms*, 2013, pp. 164–175.

[122] J. Li et al., "Losha: A general framework for scalable locality sensitive hashing," in *Proc. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2017, pp. 635–644.

[123] A. Bhaskara and M. Wijewardena, "Distributed clustering via LSH based data partitioning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 570–579.

[124] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, Jul. 2021.

[125] B. Harwood and T. Drummond, "FANNG: Fast approximate nearest neighbour graphs," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5713–5722.

[126] Y. Xu and A. Fern, "On learning linear ranking functions for beam search," in *Proc. Int. Conf. Mach. Learn.*, 2007, pp. 1047–1054.

[127] R. Negrinho, M. R. Gormley, and G. J. Gordon, "Learning beam search policies via imitation learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 10 675–10 684.

[128] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[129] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4565–4573.

[130] G. H. Alex Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep. 3, 2009.

[131] T.-S. 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 *Proc. ACM Int. Conf. Image Video Retrieval*, 2009, pp. 1–9.

[132] L. Wolf, T. Hassner, and I. Maoz, "Face recognition in unconstrained videos with matched background similarity," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 529–534.

[133] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 1532–1543.

[134] L. Yuan et al., "Central similarity quantization for efficient image and video retrieval," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 3083–3092.

[135] S. Li, Z. Chen, J. Lu, X. Li, and J. Zhou, "Neighborhood preserving hashing for scalable video retrieval," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 8212–8221.

[136] T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik, "Fast, accurate detection of 100,000 object classes on a single machine," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 1814–1821.

[137] X. Li, C. Shen, A. Dick, and A. Van Den Hengel, "Learning compact binary codes for visual tracking," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 2419–2426.

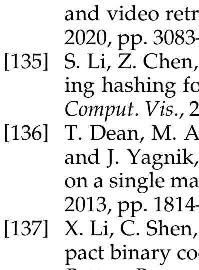
[138] S. Zhang, X. Li, M. Zong, X. Zhu, and D. Cheng, "Learning K for KNN classification," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 3, pp. 1–19, 2017.

[139] J. Zhang et al., "Fast and flexible top-k similarity search on large networks," *ACM Trans. Inf. Syst.*, vol. 36, no. 2, pp. 1–30, 2017.

[140] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for e-commerce recommendation in alibaba," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 839–848.

[141] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *Proc. Int. Conf. Learn. Representations*, 2020.

[142] J. Wang et al., "Milvus: A purpose-built vector data management system," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2021, pp. 2614–2627.



Mingjie Li received the PhD degree in computer science from the University of Technology Sydney, Australia, in 2021. He is currently an associate professor with the School of Computer Science and Cyber Engineering, Guangzhou University, China. His research interests include learning to hash, deep learning, reinforcement learning, and their applications in similarity search on high dimensional data.

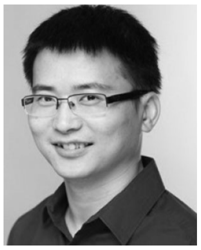
Yuan-Gen Wang (Senior Member, IEEE) received the PhD degree in communication and information system from Sun Yat-sen University, Guangzhou, China, in 2013. He is currently a full professor and the deputy dean with the School of Computer Science and Cyber Engineering, Guangzhou University, China. His research interests include digital watermarking, image processing, and computer vision.

Peng Zhang received the PhD degree from the University of Technology Sydney in 2020. He is currently working as a lecturer with the Shandong University of Science and Technology. He has published more than 20 research papers in major journals and conferences, such as *IEEE Transactions on Image Processing*, *IEEE Transactions on Circuits and Systems for Video Technology*, PR, WACV, etc. His research interests include metric learning, deep learning and their applications on person re-identification, gait recognition, etc.

Hanpin Wang received the PhD degree from Beijing Normal University, China, in 1993. He is currently a professor of Computer Science with the School of Computer Science and Cyber Engineering, Guangzhou University, China. He has served as the dean with the school since 2019. His research interests include mathematical aspects of computers, algorithms and computational complexity.

Lisheng Fan received the PhD degree from the Department of Communications and Integrated Systems of Tokyo Institute of Technology, Japan, in 2008. He is now a professor with the School of Computer Science and Cyber Engineering, Guangzhou University. His research interests span in the areas of wireless cooperative communications, physical-layer secure communications, intelligent communications, and system performance evaluation.

Enxia Li is currently working toward the PhD degree with the University of Technology Sydney (UTS). Her research focuses on efficient analytic of large scale data, with focus on finance data.



Wei Wang received the PhD degree in computer science from The Hong Kong University of Science and Technology in 2004. He is currently a professor with the Information Hub, The Hong Kong University of Science and Technology (Guangzhou), China. His current research interests include similarity query processing, artificial intelligence, knowledge graphs, and security for AI models.