Learning to Index for Nearest Neighbor Search

Chih-Yi Chiu, Amorntip Prayoonwong, and Yin-Chih Liao

Abstract—In this study, we present a novel ranking model based on learning neighborhood relationships embedded in the index space. Given a query point, conventional approximate nearest neighbor search calculates the distances to the cluster centroids, before ranking the clusters from near to far based on the distances. The data indexed in the top-ranked clusters are retrieved and treated as the nearest neighbor candidates for the query. However, the loss of quantization between the data and cluster centroids will inevitably harm the search accuracy. To address this problem, the proposed model ranks clusters based on their nearest neighbor probabilities rather than the query-centroid distances. The nearest neighbor probabilities are estimated by employing neural networks to characterize the neighborhood relationships, i.e., the density function of nearest neighbors with respect to the query. The proposed probability-based ranking can replace the conventional distance-based ranking for finding candidate clusters, and the predicted probability can be used to determine the data quantity to be retrieved from the candidate cluster. Our experimental results demonstrated that the proposed ranking model could boost the search performance effectively in billion-scale datasets.

Index Terms—Approximate nearest neighbor, asymmetric distance computation, cluster ranking and pruning, hash-based indexing, product quantization, residual vector quantization

1 Introduction

Nearest neighbor (NN) search in a high-dimensional and large-scale dataset is highly challenging and it has recently attracted much active interest from researchers. This approach is a fundamental technique in many applications, such as classification [1], feature matching [2], recommendation [3], and information retrieval [4]. An exhaustive approach is prohibitive so numerous approximate NN search algorithms have been proposed to address issues in terms of the search accuracy, computation efficiency, and memory consumption.

One solution is to generate compact codes to replace the original data during NN search. Research into compact code generation has focused on two main areas: binary embedding and codebook learning. In binary embedding, each data point is transformed into a binary pattern with a set of hashing functions. In codebook learning, a set of centroids is generated by a clustering technique and each data point is assigned to the cluster with the nearest centroid. By compiling these codes as a codebook, we can build an index structure to accelerate NN search. Common indexing approaches include cluster ranking and pruning [5], [6], where we expect a set of similar data is grouped in the same cluster. At the query time, we rank the clusters based on a similarity or distance metric with respect to the query. Only data points indexed in high-ranking clusters are retrieved for verification, and those in low-ranking clusters that are considered irrelevant to the query can be filtered out.

Inevitably, the information loss between the original data points and the corresponding quantized codes will impair the indexing performance, although we can increase the number of codewords or use a better quantization method to alleviate it to some extent. Fig. 1 shows an example of a two-dimensional euclidean space with three clusters: A, B, and C. Let a, b, and c be the centroids of the respective clusters, and q is the query point. The euclidean distances between the query and centroids are denoted as $\|\overline{qa}\|$, $\|\overline{qb}\|$, and $\|\overline{qc}\|$. Assuming that $\|\overline{qa}\| < \|\overline{qb}\| < \|\overline{qc}\|$, then the order of traversing the clusters is from A to B to $\mathbf{C}$. It should be noted that the traversal order does not reflect the NN distribution among the clusters well. In fact, cluster $\mathbf{C}$ contains more relevant data that are closer to the query (in a certain radius) than the other clusters, and thus it should be visited earlier so more true positives can be retrieved. However, the existing NN search methods (e.g., cluster ranking and pruning) rely mainly on the euclidean distances to rank clusters. Cluster $\mathbf{C}$ might not be visited if we rely on the distance-based ranking to traverse only the top one or two clusters, and thus the relevant data indexed in cluster $\mathbf{C}$ cannot be found.

In this study, we present a novel ranking model that learns neighborhood relationships embedded in the index space, which can be employed to estimate the NN distribution probabilities among clusters for a given query. The underlying concept for the proposed model is that the cluster ranking may be determined based on the NN probabilities of clusters rather than their centroid distances, with respect to the query. As shown in Fig. 1, cluster $\mathbf{C}$ contains most of the NNs to query q. Thus, it can be regarded as the most relevant cluster, so its priority should be at the topmost rank. In order to characterize the neighborhood relationships, we propose learning from a training set of

• *The authors are with the Department of Computer Science and Information Engineering, National Chiayi University, Chiayi 600, Taiwan, R.O.C. E-mail: cychiu@mail.ncyu.edu.tw, {aprayoonwong, zzzzbenny6209}@gmail.com.*

Manuscript received 12 June 2018; revised 27 Feb. 2019; accepted 6 Mar. 2019. Date of publication 25 Mar. 2019; date of current version 1 July 2020. (Corresponding author: Chih-Yi Chiu.)
Recommended for acceptance by R. Manmatha.
Digital Object Identifier no. 10.1109/TPAMI.2019.2907086

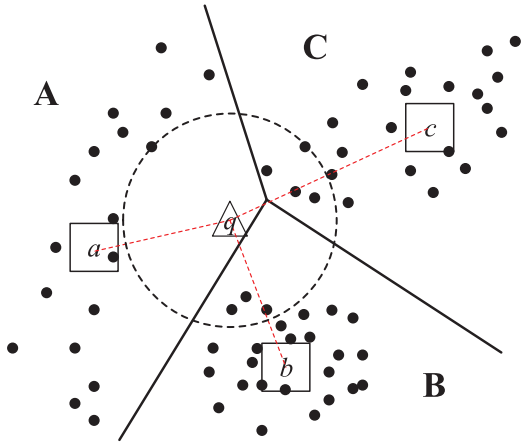


Fig. 1. A 2D euclidean space, where *A*, *B*, and *C* are clusters, *a*, *b*, and *c* are the respective centroids, and *q* is the query. The clusters can be ranked based on their query-centroid distances, which is widely used in existing NN search methods. However, the distance-based ranking does not reflect the NN distribution among the clusters well.

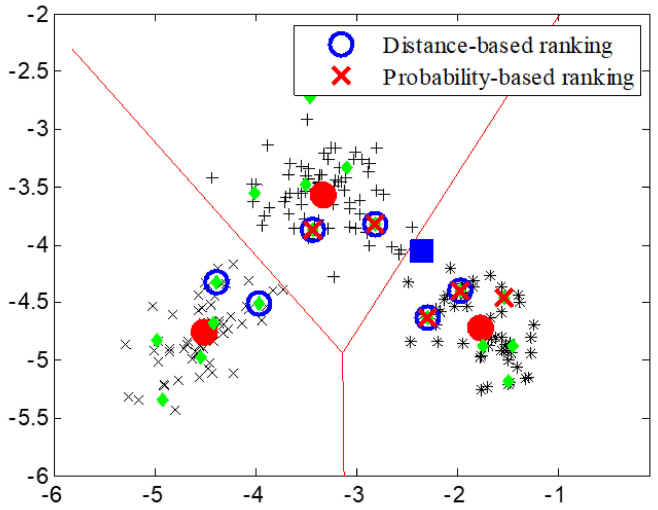


Fig. 2. Probability-based ranking versus distance-based ranking. Data points are from the wine dataset (UCI machine learning repository), a thirteen-feature set labeled by three different classes. We project the data points to a 2D space constituted by the top two eigenvectors with the highest eigenvalues for visualization purposes.

query-dependent features. That is, we formulate a nonlinear function of the query-dependent features for predicting the NN probabilities of clusters. The clusters are ranked according to the NN probabilities instead of the euclidean distances, so retrieving the high-ranking clusters is expected to obtain more true positives for the query. Moreover, the predicted probability can be used to calculate the data quantity to be retrieved from the cluster. Intuitively, we should retrieve more data from the high-ranking clusters to identify true positives, and retrieve less data from the low-ranking clusters to reduce false positives. Rather than retrieving all of the data indexed in a cluster, the data quantity that needs to be extracted from the cluster is proportional to its NN probability for a better retrieval quality. The proposed ranking model can be applied hierarchically to learn the neighborhood relationships embedded in a multi-level index space. It is time- and space-efficient while effectively boosting the search accuracy.

Fig. 2 shows an example of the retrieval results obtained by the proposed probability-based ranking and typical distance-based ranking, where the blue rectangle is a query point, large red circles are cluster centroids, and small green diamonds are subcluster centroids. The results are produced by selecting the subclusters with the highest probabilities/smallest distances to the query. The probability-based ranking presents more relevant retrieval compared to the distance-based ranking.

The proposed probability-based ranking has the following advantages compared with the distance-based ranking:

- The probability-based ranking can be used to replace the distance-based ranking during NN search. We alleviate the information loss problem caused by embedding and quantization by predicting the NN probabilities through nonlinear mappings of query-dependent features to rank clusters and estimate the quantity to be retrieved from each cluster. The retrieval quality is improved significantly through computation- and memory-efficient indexing.
- The probability-based ranking can be integrated easily with existing quantization/search methods. It returns highly relevant data as candidates, which are then reranked through asymmetric distance computation (ADC). We demonstrated the feasibility and superior performance of the proposed method based on a multi-level index structure comprising two popular quantization methods residual vector quantization (RVQ) and optimized product quantization (OPQ).

We conducted evaluation experiments using the billion-scale SIFT and DEEP datasets. We implemented the cluster ranking function of three query-dependent features and three quantity estimation functions in the probability-based ranking model to compare them with the distance-based ranking model. The results indicated that the neighborhood relationships can be learned and utilized to improve the ranking quality during NN search.

The remainder of this paper is organized as follows. In Section 2, we discuss previous research related to NN search techniques. In Section 3, we explain the proposed probability-based ranking model. In Section 4, we present the experimental results obtained using large-scale datasets. Finally, we give our conclusions in Section 5.

2 Related Work

NN search has been a highly active research area in recent decades. In the following, we review some of the fundamental techniques related to our study, including data quantization and data indexing.

2.1 Data Quantization

Data quantization techniques generate compact codes transformed from the original data and they can be divided into two categories: binary embedding and codebook learning. Binary embedding provides the most compact data representation. In general, the original data are transformed into an intermediate space by a set of hash functions. The transformed data are then quantized to binary embedding codes. The Hamming distance between two binary codes can be

calculated efficiently with hardware-supported machine instructions. Numerous learning to hash methods have been proposed recently for learning the data-dependent and task-specific hash functions to generate a short but effective binary code, including unsupervised [7], [8], [9], [10] and supervised [11], [12], [13], [14], to name a few. A comprehensive study for learning to hash can be found in some survey papers [15], [16], [17]. Applying binary embedding can save time and memory but its search accuracy is severely degraded. Due to the huge quantization loss, the measurement between two binary embedding codes is imprecise with only a few distinct Hamming distances.

The codebook learning approach produces a set of representatives for vector quantization. One of the most popular algorithms for this purpose is k-means clustering, which iteratively partitions the vector space into k Voronoi cells according to the data distribution. Product quantization (PQ) is a generalization of k-means [18]. PQ partitions the feature vector of a data point into several segments of disjoint subvectors. Each segment is encoded by a corresponding codebook, and each data point is represented by a concatenation of segment codewords. PQ produces huge quantization levels using small-size codebooks, so codebook learning and data quantization can be implemented efficiently. This becomes a common technique for processing data in the high-dimensional vector space. For example, PQk-means [19] employs PQ codes to enable fast and memory-efficient k-means clustering. Blalock and Guttag [20] presented a variation of PQ with fast encoding speed based on small codebooks, and efficient distance computation by learning a quantization function that approximates the query-centroid distances in lookup tables. Xu et al. [21] proposed an online PQ method that can update the quantization codebook dynamically based on the incoming streaming data and by using insertion/deletion operations to reflect the real-time data behavior.

To reduce the quantization distortions in PQ, some studies searched for the optimal space decomposition to learn codebooks. In OPQ [22] and Cartesian k-means [23], generalized PQ is applied by finding the optimal transformation matrix and sub-codebooks in the data space to minimize the quantization distortion from a global viewpoint. Locally OPQ (LOPQ) [24] assumes the data distribution in each cluster is unimodal, so an individual OPQ is performed per cluster with a lower distortion. Bilinear optimized PQ [25] learns a bilinear projection for PQ, which exploits the natural data structure to reduce the time and space complexity.

The PQ-based methods try to decompose the data space into orthogonal subspaces, but an alternative approach called non-orthogonal quantization approximated the data based on the sum of the codewords instead of concatenation. For example, the widely used RVQ method [26], [27], [28] quantizes the displacement between the original vector and the nearest centroid, i.e., the residual vector. Encoding the residual usually yields a lower quantization error than encoding the original vector. Additive quantization [29] eliminates the orthogonal subspace assumption and represents a vector with a sum of codewords to achieve considerably lower approximation error. Similarly, composite quantization [30] approximates a vector using the summation of codewords under the constraint of a constant inter-dictionary-element-product. Tree quantization [31] uses a tree graph of codebooks, which is constructed by minimizing the compression error through integer programming-based optimization. AnnArbor [32] employs arborescence graphs to encode vectors based on their displacements between nearby vectors.

2.2 Data Indexing

The tree-based index structure is one of the most widely used techniques for vector space indexing [2], [33], [34]. However, it usually requires a considerable amount of memory to store a huge index structure and the search time grows rapidly with the dimensionality. To address this problem, Muja and Lowe [35] presented the priority search k-means tree algorithm for effective matching in high-dimensional spaces. Houle and Nett [36] proposed the rank cover tree, which uses the ordinal ranks of the query distances to prune data points. Liu et al. [37] introduced an aggregating tree, which is a radix tree built based on RVQ encodings, and the beam search algorithm is applied to search for NNs.

Inverted file system with ADC (IVFADC) is another popular framework, which was designed to handle billion-scale datasets efficiently [18]. This system employs k-means clustering for coarser quantization and then utilizes PQ for residual quantization. ADC [38], [39], [40] is used for fast euclidean distance computation via inverted table lookup. In ADC, the query is kept in its original space and the reference data are kept in the quantized space. There is no quantization loss at the query side, so ADC can yield a more accurate evaluation compared with the computation between the quantized query and reference data points. LOPQ [24] can be regarded as a modified form of IVFADC that uses separate local PQ codebooks for data compression. Local codebooks can model the local data distribution more precisely, so its accuracy is better than that of IVFADC. Baranchunk et al. [41] proposed grouping and pruning procedures to improve IVFADC, where each cluster is split into subclusters, which form a set of convex combinations of the local neighboring centroids.

The use of multiple quantizers, known as multiple hash tables, can achieve good recall and speed in practice. For example, k-means locality sensitive hashing [42] uses several codebooks, which are generated by running k-means clustering several times for multiple hashing. The joint inverted indexing method [43] creates codewords jointly by single k-means clustering, before assigning the codewords to different codebooks so the total distortion of the quantizers is minimal. Some PQ-based indexing methods exploit the power of PQ in multiple hashing. The inverted multi-index (IMI) method [44] constructs a multi-dimensional index table, where the cell centroids are made from the Cartesian product of the codebooks. A large number of cells can provide very dense partitioning of the space, so IMI can achieve a high recall by traversing only a small fraction of the dataset. PQTable [45] was proposed to accelerate exhaustive PQ search by using the concatenation of PQ codes from multiple hash tables to index a data point. Multi-index voting [46] employs a voting mechanism to generate high-quality NN candidates by traversing across multiple hash tables derived from the intermediate spaces.

3 Probability-Based Ranking

Given a reference dataset of I l-dimensional real-valued vectors $V = \{v_i \in \{\mathbb{R}\}^l | i = 1, 2, \ldots, I\}$, we construct an index structure with M clusters $\{C_m | m = 1, 2, \ldots, M\}$ and a codebook of centroids $\{u_m | m = 1, 2, \ldots, M\}$. The mth cluster C_m is associated with the centroid u_m and an inverted list that contains the reference data indexed in the mth cluster expressed as:

$$C_m = \{v_i | Z(v_i) = m\}, \tag{1}$$

where function $Z(v_i)$ returns the nearest centroid id of v_i:

$$Z(v_i) = \underset{m \in \{1,2,\ldots,M\}}{\arg\min} (\|v_i - u_m\|, 1), \tag{2}$$

where $\|\cdot\|$ denotes the 2-norm (euclidean distance) and $\arg\min(B, k)$ returns the k arguments indexed by b for the k minimums in set B. Given a query q, a typical approach for utilizing the index structure involves retrieving the top-R ranked clusters whose centroids are nearest to q. Denote the top-R clusters as $\{C_{m_1}, C_{m_2}, \ldots, C_{m_R}\}$ and the index subscripts are expressed by:

$$\{m_1, m_2, \ldots, m_R\} = \underset{m \in \{1,2,\ldots,M\}}{\arg\min} (\|q - u_m\|, R), \tag{3}$$

where C_{m_r} is the rth-nearest cluster to q. These clusters are traversed sequentially to retrieve NN candidates from their inverted lists. This indexing process is known as cluster ranking and pruning [5] and it is usually applied as a coarse filter, which is employed widely in existing NN search methods. However, as mentioned earlier, the quantization loss in the index space will degrade the retrieval quality. In order to address this issue, we learn how to model the neighborhood relationships embedded in the index space.

3.1 Learning to Index

We model the neighborhood relationships by a function f: $f(X) = \{p_1, p_2, \ldots, p_M\}$, which maps a query-dependent feature vector X to the NN probabilities $\{p_1, p_2, \ldots, p_M\}$, where p_m represents the NN probability of the mth cluster. We then rank the clusters based on their NN probabilities instead of the euclidean distances in order to output the top-R clusters. Thus, Eq. (3) is rewritten as:

$$\{m_1, m_2, \ldots, m_R\} = \underset{m \in \{1,2,\ldots,M\}}{\arg\min} (p_m, R). \tag{4}$$

The cluster sequence ranked by the NN probabilities can reflect the neighborhood relationships between the query and clusters better than that ranked based on the euclidean distances, thereby improving the retrieval quality.

The mapping function $f(X)$ is constructed in the following training process. Let $Q = \{q^{(1)}, q^{(2)}, \ldots, q^{(T)}\}$ be the training dataset of T queries. The tth query $q^{(t)}$ is associated with the weighted ground truth of K NNs, denoted as $G^{(t)} = \{g_1^{(t)}, g_2^{(t)}, \ldots, g_K^{(t)}\}$ and $W^{(t)} = \{w_1^{(t)}, w_2^{(t)}, \ldots, w_K^{(t)}\}$, where $g_k^{(t)} \in V$ is the kth NN of $q^{(t)}$ and the corresponding weight is $w_k^{(t)}$. For input X, we propose three query-dependent features, comprising the raw query data, query-centroid similarities, and their concatenation. The raw query data feature is $q^{(t)}$ itself. The query-centroid similarity feature is derived from a distance feature $D^{(t)} = \{d_1^{(t)}, d_2^{(t)}, \ldots, d_M^{(t)}\}, d_m^{(t)} = \|q^{(t)} - u_m\|$. Then, the distance feature is transformed into a similarity feature $A^{(t)} = \{a_1^{(t)}, a_2^{(t)}, \ldots, a_M^{(t)}\}$, where $a_m^{(t)}$ is defined by:

$$a_m^{(t)} = \frac{\max(D^{(t)}) - d_m^{(t)}}{\max(D^{(t)})}.$$

越大，代表query距离质心越近

The mapping function output, also known as the target, is a vector of NN probabilities of M clusters, denoted as $Y^{(t)} = \{y_1^{(t)}, y_2^{(t)}, \ldots, y_M^{(t)}\}$. $y_m^{(t)}$ is defined by:

$$y_m^{(t)} = \frac{\sum_k \left\{ w_k^{(t)} | g_k^{(t)} \in C_m \right\}}{\sum_k \left\{ w_k^{(t)} | g_k^{(t)} \in C_m \right\} + |C_m|}, \tag{6}$$

where $|\cdot|$ denotes the set cardinality. Then $y_m^{(t)}$ is normalized by sum to one. Consequently, the tth training data $\{q^{(t)}, G^{(t)}, W^{(t)}\}$ is converted to the input-output pair $\{X^{(t)}, Y^{(t)}\}$ of f.

We employ a fully-connected neural network to learn the neighborhood relationships based on the training dataset $\{X^{(t)}, Y^{(t)} | t = 1, 2, \ldots, T\}$. The input layer receives $X^{(t)}$ and the output layer predicts NN probabilities for M clusters, denoted as $P^{(t)} = (p_1^{(t)}, p_2^{(t)}, \ldots, p_M^{(t)})$. Based on the loss between the predictions $P^{(t)}$ and the target $Y^{(t)}$, we compute the error derivative with respect to the output of each unit, which is backward propagated to each layer in order to tune the weights of the neural network.

3.2 Quantity Estimation

By using the learned mapping function, we can predict the NN probability distribution among clusters for the given query. The NN probabilities can be utilized to rank clusters but also to estimate the data quantities that needs to be retrieved from the clusters. If we control the amount of data that needs to be retrieved from the clusters, it is reasonable to retrieve more data from the high-probability clusters and less data from the low-probability clusters for the sake of better precision. To illustrate this idea, we formulate the quantity estimation problem in a two-level index structure. The first level comprises M clusters $\{C_m | m = 1, 2, \ldots, M\}$. Each C_m is further partitioned by N subclusters, i.e., the second-level clusters, denoted as $\{C_m^n | n = 1, 2, \ldots, N\}$. Thus, the data space is divided into $M \times N$ partitions. Suppose that we obtain the NN probabilities of the first-level clusters $\{p_m | m = 1, 2, \ldots, M\}$. Let $\gamma(p)$ be a monotonically increasing function and T is the total amount of the second-level clusters that need to be retrieved. The number of the second-level clusters retrieved from C_m is calculated by:

$$S = \max\left(\left\{ N, \text{round}\left(\frac{\gamma(p_m)}{\sum_1^m \gamma(p_m)} \cdot T \right) \right\} \right), \tag{7}$$

where function $\text{round}(\cdot)$ returns the closest integer. Thus, from C_m, we select S second-level clusters (at maximum of N) that are closest to query q. The selection of the top-S second-level clusters is dependent on the underlying index structure and we explain it in next subsection.

$\gamma(p)$ can be considered a normalization function that rescales p based on some assumed distribution for the NN probabilities among the top-R first-level clusters $\{C_{m_1}, C_{m_2}, \ldots, C_{m_R}\}$. Let p_{m_r} be the NN probability of the

rth ranked cluster C_{m_r}. The possible normalization functions considered in this study are as follows.

- *Sum*. Normalized by sum to one:

$$\gamma_{sum}(p_{m_r}) = \frac{p_{m_r}}{\sum_{j=1}^{R} p_{m_j}}. \tag{8}$$

- *Standard*. Rescaled under a normal distribution with a mean of 0 and a standard deviation of 1:

$$\gamma_{std}(p_{m_r}) = \frac{p_{m_r} - \mu}{\sigma}, \tag{9}$$

  where μ and σ defined as the mean and standard deviation of the NN probabilities of the top-R first-level clusters, respectively.
- *Exponent*. Defined as a probability density function of the exponential distribution:

$$\gamma_{exp}(p_{m_r}) = e^{-\frac{1-p_{m_r}}{\rho}}, \tag{10}$$

  where ρ is the inverse to the mean of the probabilities of the top-R first-level clusters:

$$\rho = 1 - \frac{1}{R}\sum_{j=1}^{R} p_{m_j}. \tag{11}$$

Our evaluations for these normalization functions are presented in the experimental section.

3.3 Indexing and Search Algorithm

In this subsection, we describe the general indexing and search algorithm integrated with the proposed ranking model, and we illustrate the capability based on some hash-based index structures for NN search. The steps are summarized in Algorithm 1. In step 1, for the given query q, we generate the query-dependent feature X. The trained mapping function $f(X)$ is used to predict the NN probabilities of the first-level clusters $\{p_1, p_2, \ldots, p_M\}$. The top-$R$ first-level clusters $\{C_{m_1}, C_{m_2}, \ldots, C_{m_R}\}$ with the highest NN probabilities are selected. In steps 4 and 5, we select the top-S second-level clusters $\{C_{m_r}^{m_1}, C_{m_r}^{m_2}, \ldots, C_{m_r}^{m_S}\}$ from each C_{m_r}. The reference data indexed in the selected second-level clusters are retrieved as candidates. We then calculate the distance between the query and each of the candidates for reranking, where ADC is employed and sort the candidates in ascending order and return k NNs in the last step.

The time complexity of Algorithm 1 is mainly from the three parts: NN probability prediction (step 2), candidate selection (steps 3-5), and reranking (steps 6-7). The NN probabilities of the first-level clusters are predicted by the fully-connected neural network. Suppose the network has τ hidden layers, each of which contains λ units. The prediction thus spends $O(\tau\lambda^2 + \lambda M)$ time. Candidate selection requires to sort M (N) clusters. We apply partial sorting (e.g., std::nth_element from C++ STL) to find top clusters with linear complexity, taking $O(M)$ $(O(N))$ time in average. The main concern of reranking is the number of candidates retrieved. For each candidate, we perform ADC to the query by table lookup. If we suppose that there are α tables and each table contains β codewords representing an $\frac{l}{\alpha}$-dimensional vector, then ADC requires $O(l\beta + \alpha|\mathbf{A}|)$ time.

The memory required by Algorithm 1 is mainly for the reference dataset and index structure, which must be loaded into memory for real-time responses in search applications. The reference data are stored in the form of PQ codes. Each data point takes $\alpha \lg \beta$ bits in memory and $\frac{\alpha \lg \beta}{8} \cdot I$ bytes for a total of I reference data points, with an additional PQ codebook of $8\beta l$ bytes. The index structure size is determined by the number of the indexed data points, index table, prediction network, and index codebook. Assume the number of reference data points I is no more than 2^{32}, we can use 4 bytes to encode the data identity, and $4I$ bytes are required for I reference data points. A two-level index table associates data identities through 64-bit pointers for inverted indexing, which take $8MN$ bytes. The prediction network occupies $4(\tau\lambda^2 + M\lambda)$ bytes, where the network coefficients are represented by 4-byte floating-point numbers. The index codebook size is dependent on the underlying quantization method, as discussed in Section 3.3.3.

Algorithm 1. Indexing and Search

Input: first-level clusters $\{C_m\}$; second-level clusters $\{C_m^n\}$; mapping function f; query q;
Output: k NNs of q;
1: Generate the query-dependent feature vector X for q;
2: Predict the NN probabilities of the M first-level clusters $f(X) = \{p_1, p_2, \ldots, p_M\}$;
3: Select the top-R first-level clusters $\{C_{m_1}, C_{m_2}, \ldots, C_{m_R}\}$ by Eq. (4);
4: For each $C_{m_r}, r \in [1, R]$, calculate the number of the second-level clusters S to be retrieved using Eq. (7);
5: For each $C_{m_r}, r \in [1, R]$, select the top-S second-level clusters $\{C_{m_r}^{m_1}, C_{m_r}^{m_2}, \ldots, C_{m_r}^{m_S}\}$ and generate a candidate set $\mathbf{A}$;
6: Perform ADC between q and each candidate in $\mathbf{A}$;
7: Sort the asymmetric distances in ascending order and return the k NNs;

The index structure and quantization method determine the cluster selection and candidate generation process specified in step 5 of Algorithm 1. In next subsections, we take two quantization methods, comprising RVQ and OPQ, to show how to integrate with the proposed ranking model.

3.3.1 RVQ

We use the multi-level RVQ method [26], [27], [28], where multiple quantizers are concatenated sequentially to approximate the quantization errors in the preceding levels. Each quantizer has its corresponding codebook. We recall that the first-level cluster set $\{C_m | m = 1, 2, \ldots, M\}$ is associated with the quantizer $Z(v_i)$ and codebook $\{u_m | m = 1, 2, \ldots, M\}$. The residual of the reference data point v_i known as the quantization error is expressed as:

$$e_i = v_i - Z(v_i). \tag{12}$$

Given the residual set $\{e_i | i = 1, 2, \ldots, I\}$, we apply k-means clustering to divide the residual space into N partitions to generate the residual codebook $\{u^n | n = 1, 2, \ldots, N\}$. The residual codebook is shared by all of the vectors in the residual space. In total, MN distinct quantization codes are provided by the two codebooks. The second-level cluster is denoted as C_m^n, $m \in [1, M]$ and $n \in [1, N]$:

$$C_m^n = \{v_i | Z(v_i) = m, E(e_i) = n\}, \tag{13}$$

where function $E(e_i)$ returns the nearest centroid id of the residual codebook to e_i. v_i is indexed in C_m^n and it can be approximated by the second-level centroid u_m^n:

$$v_i \approx u_m^n = u_m + u^n, \forall v_i \in C_m^n. \tag{14}$$

If we want to create a multi-level index structure [27], the codebooks can be created in the same manner.

Suppose that we obtain the top-R first-level clusters $\{C_{m_1}, C_{m_2}, \ldots, C_{m_R}\}$ for query q. We want to find the sth closest-distance second-level cluster from the rth highest-probability first-level cluster, denoted as $C_{m_r}^{n_s}, r \in [1, R], s \in [1, S]$ (step 5 of Algorithm 1). From each C_{m_r}, we select the top-S second-level clusters $\{C_{m_r}^{n_1}, C_{m_r}^{n_2}, \ldots, C_{m_r}^{n_S}\}$ (recall that S is determined by Eq. (7)), where the index superscripts $\{n_1, n_2, \ldots, n_S\}$ are expressed by:

$$\{n_1, n_2, \ldots, n_S\} = \underset{n \in \{1,2,\ldots,N\}}{\arg\min} \Big(\big\| q - u_{m_r}^n \big\|, S \Big). \tag{15}$$

The data indexed in $\left\{ C_{m_r}^{n_s} | r = 1, 2, \ldots, R, s = 1, 2, \ldots, S \right\}$ are considered to be the candidates.

3.3.2 OPQ

The objective of OPQ is to find the optimal space decomposition for minimum quantization distortions [22]. The optimal space decomposition is jointly determined by an orthonormal matrix and sub-codebooks, which can be solved by either non-parametric or parametric (based on a Gaussian distribution) optimization. Following the same process employed for RVQ, we generate the first-level cluster set $\{C_m\}$ and create the residual space with the residual data $\{e_i\}$. The residual space is transformed by an orthonormal matrix Φ and divided into α subspaces. Each subspace has a sub-codebook containing β sub-codewords. This process is equivalent to generating a residual codebook $\{u^n | n = 1, 2, \ldots, N\}$ in the residual space, where $N = \beta^\alpha$, and u^n is represented as the concatenation of sub-codewords from the α subspaces. The second-level cluster C_m^n is expressed as:

$$C_m^n = \{v_i | Z(v_i) = m, E(\Phi e_i) = n\}, \tag{16}$$

and its second-level centroid u_m^n as:

$$u_m^n = u_m + \Phi^{-1} u^n, \tag{17}$$

where Φ^{-1} is the inverse of the orthonormal matrix. Similarly, we apply Eq. (15) to generate the candidate set from the top-S second-level clusters $\left\{ C_{m_r}^{n_s} \right\}$ from each C_{m_r}.

3.3.3 Discussion

We have demonstrated the proposed probability-based ranking model can be integrated easily with the quantization methods described above. Both RVQ and OPQ exploit the residual space where less quantization loss occurs. Under the same quantization levels, OPQ has a more compact codebook size $O\big((M + \beta)l\big)$ compared with RVQ, which takes $O\big((M + N)l\big)$, for β is usually smaller than N. The tradeoff is that the quantization loss is higher with OPQ than that using RVQ.

In our implementation of the two-level index structure, one first-level codebook and one second-level codebook are constructed, and the second-level codebook is shared by all the first-level clusters. Alternatively, a second-level codebook can be constructed for each first-level cluster using locally-optimized quantization methods, such as hierarchical k-means (HKM) [34] and LOPQ [24]. These methods can lead to lower quantization losses (compared with RVQ and OPQ), but they must keep a large amount of second-level codebooks, e.g., $O(MNl)$ for HKM and $O(M\beta l)$ for LOPQ, which is not negligible in memory consumption.

3.4 Hierarchical Learning

The proposed ranking model can be applied hierarchically to learn the neighborhood relationships embedded in a hierarchical index space. Again, we consider the two-level index structure of RVQ for illustration. In Eq. (15), we rank the second-level clusters according to their euclidean distances from the query. It is natural to consider replacing the distance-based ranking with the probability-based ranking for the second-level clusters. However, it is impractical to generate the M functions (of the M first-level clusters) for the second-level probability prediction. Instead, we propose only generating one function that can be shared for all of the first-level clusters.

We train a mapping function h that maps a cluster-based feature X_m to NN probabilities: $\{p_m^1, p_m^2, \ldots, p_m^N\}$, where p_m^n represents the NN probability of the second-level cluster C_m^n:

$$h(X_m) = \{p_m^1, p_m^2, \ldots, p_m^N\}. \tag{18}$$

Let $X_m^{(t)}$ be the C_m-based feature of the tth query $q^{(t)}$ in the training set. $X_m^{(t)}$ comprises two different parts, which characterize the first-level and second-level clusters with respect to $q^{(t)}$. The first part is the mth first-level centroid $u_m^{(t)}, m \in \{1, 2, \ldots, M\}$, and the second part is the residual $e_m^{(t)} = q^{(t)} - u_m^{(t)}$. $X_m^{(t)}$ is defined as the concatenation of the two parts $X_m^{(t)} = \big(u_m^{(t)}, e_m^{(t)}\big)$. Therefore, $X_m^{(t)}$ generally characterizes the query representation in the second-level index space of the mth first-level cluster C_m. The target used in the training process is denoted as $Y_m^{(t)} = \big\{y_m^{1(t)}, y_m^{2(t)}, \ldots, y_m^{N(t)}\big\}$, where $y_m^{n(t)}$ is the ground truth NN probability of the second-level cluster C_m^n:

$$y_m^{n(t)} = \frac{\sum_k \left\{ w_k^{(t)} | g_k^{(t)} \in C_m^n \right\}}{\sum_k \left\{ w_k^{(t)} | g_k^{(t)} \in C_m^n \right\} + |C_m^n|}. \tag{19}$$

Then, the sum-to-one normalization is applied to $y_m^{n(t)}$.

It should be noted that only one mapping function h is learned to predict the NN probabilities in the second-level index space. Function h is modeled by another fully-connected neural network and trained with the same procedure as function f. Suppose h is characterized by the same numbers of hidden layers and units as f. The space complexity for functions f and h still requires $O(\tau\lambda^2 + \lambda M)$. The time complexity through the two-level prediction takes $O\big(R(\tau\lambda^2 + \lambda M)\big)$, where the top-$R$ first-level clusters utilize h for the second-level prediction with R times in total.

4 Experimental Results

In our experiments, we evaluated the proposed probability-based ranking and compared it to the distance-based ranking with various configurations and state-of-the-art methods.

4.1 Datasets

We examined based on two benchmark datasets comprising SIFT1B and DEEP1B. SIFT1B is a visual feature set from the BIGANN Dataset [47], which has been used widely in many large-scale NN search studies. SIFT1B contains one billion 128-dimensional SIFT descriptors and extra 10000 descriptors as the query set. For each query, SIFT1B provides 1000 NNs with the smallest euclidean distances from the query as the ground truth. DEEP1B is another visual feature set [28], which was produced by using the GoogLeNet architecture based on the ImageNet dataset. The deep descriptors were extracted from the outputs of the last fully-connected layer, compressed using principal component analysis (PCA) to 96 dimensions, and normalized to unit length. DEEP1B also contains one billion base descriptors and another 10000 as the query set. Each query only provides one NN as the ground truth, so we extended it to 1000 NNs by running exhaustive search.[1]

4.2 Implementation

The proposed method uses the PCA-compressed data for indexing but the original data for reranking. The basic idea is the coarse-to-fine approach. Therefore, in the indexing stage we perform rough but efficient filtering with the compressed data, while in the reranking stage we execute ADC on the original data for detailed verification. PCA, which is a standard tool for dimension reduction [7], [48], [49], was applied here to generate the compressed data for efficient indexing. The two benchmark datasets were transformed from the original space to a 32-dimensional PCA-compressed space as the reference datasets. We found that the indexing performance is even improved when using the compressed data instead of the original data, as discussed later. On the other hand, reranking by ADC should be performed in the original space to avoid the transformation error. We implemented ADC as follows. The original SIFT/DEEP descriptor was partitioned into 16 segments. Each segment was quantized by a codebook of 256 centroids, which were produced using *k*-means clustering. Since each segment could be encoded by 8 bits, the SIFT/DEEP descriptor was compressed to a 16-byte PQ code.

For the proposed ranking model, mapping functions f and h were characterized based on a three-layer fully-connected neural network, expressed as: I1 - H2 - H3 - O4. Function f learned the neighborhood relationships in the first-level index space. In its network, I1 was the input layer that received the query-dependent feature, and the number of input units was equal to the dimensions of the query-dependent feature. H2 and H3 were hidden layers, where each had 512 units. RELU was used as the activation function. O4 was the output layer for predicting the NN probabilities of the first-level clusters and the number of output units was equal to the number of the first-level clusters. The activation function was softmax and the loss function was cross entropy. In addition, function h characterized the relationships in the second-level index space and its network was slightly different from f in layers I1 and O4. I1 received the first-level centroid and residual of the query, and O4 predicted the NN probabilities of the second-level clusters. Layers H2 and H3 had the same setup.

To train the neural network, for each benchmark dataset, we generated a training set containing 20000 data points, which were randomly selected with stratified sampling according to the data distribution over clusters. Each training data point was associated with its 1000 NNs of ground truth found in the reference dataset. We set the batch size to 1000 and run 300 epochs for training. The 10000 queries provided by the benchmark dataset were used for test.

The NN search methods were implemented in C++ and BLAS routines in the single-thread mode. The neural networks were trained using Python and the Keras library. The programs were run on a system that operated Windows 10 with an Intel Core i7 3.4 GHz CPU and 64 GB RAM.

TABLE 1
Configurations of the Index Structures in the Reference Datasets

SIFT1B	DEEP1B
256×256	256×256
1024×1024	1024×256
4096×4096	4096×256

4.3 Results and Discussion

First, we assess the quality of the candidate set retrieved from the first-level and second-level clusters. The quantization methods RVQ and OPQ described above were investigated, where they were deployed in the two-level hierarchical index structures with various configurations, as listed in Table 1. We denote the configuration as $M \times N$, where M and N are the numbers of the first-level clusters and second-level clusters, respectively. For example, in the DEEP1B column, "4096×256" indicates that the index structure used in DEEP1B had 4096 first-level clusters, each with 256 second-level clusters.

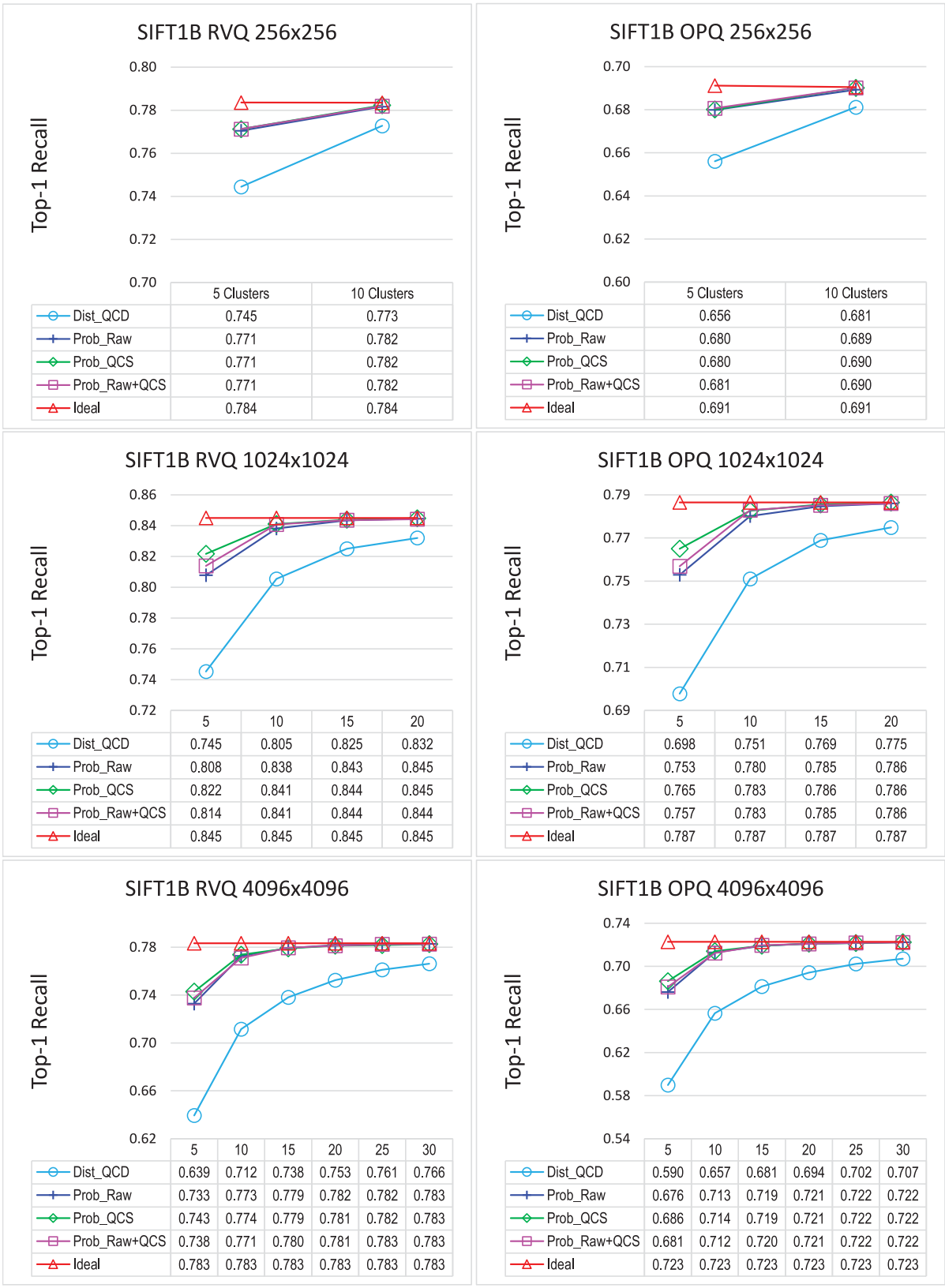
The accuracy was evaluated based on top-k recall, which measures the quality of the retrieved set $\mathbf{A}$. Let $G_k = \{g_1, \ldots, g_k\}$ be the first k NNs of ground truth for a query. Top-k recall is defined by:

$$top - k - recall@\mathbf{A} = \frac{|G_k \bigcap \mathbf{A}|}{|G_k|}. \tag{20}$$

In practice, we are often interested in the top-k NNs ($k > 1$) rather than only the first NN ($k = 1$). However, as qualitative conclusions for $k = 1$ remain valid for $k > 1$, we apply top-1 recall in most of the experiments.

Note that the weights $\{w_1, \ldots, w_K\}$ associated with G_K are hyperparameters to be set according to the evaluation metric. To obtain good top-1 recall, we will give a large weight to w_1. If we are interested in top-k recall, we can set

1. The 1000 NNs of the ground truth in DEEP1B and our source code project are available at https://github.com/AmorntipPrayoonwong/Learning-to-Index-for-Nearest-Neighbor-Search.



	5 Clusters	10 Clusters
Dist_QCD	0.745	0.773
Prob_Raw	0.771	0.782
Prob_QCS	0.771	0.782
Prob_Raw+QCS	0.771	0.782
Ideal	0.784	0.784

	5 Clusters	10 Clusters
Dist_QCD	0.656	0.681
Prob_Raw	0.680	0.689
Prob_QCS	0.680	0.690
Prob_Raw+QCS	0.681	0.690
Ideal	0.691	0.691

	5	10	15	20
Dist_QCD	0.745	0.805	0.825	0.832
Prob_Raw	0.808	0.838	0.843	0.845
Prob_QCS	0.822	0.841	0.844	0.845
Prob_Raw+QCS	0.814	0.841	0.844	0.844
Ideal	0.845	0.845	0.845	0.845

	5	10	15	20
Dist_QCD	0.698	0.751	0.769	0.775
Prob_Raw	0.753	0.780	0.785	0.786
Prob_QCS	0.765	0.783	0.786	0.786
Prob_Raw+QCS	0.757	0.783	0.785	0.786
Ideal	0.787	0.787	0.787	0.787

	5	10	15	20	25	30
Dist_QCD	0.639	0.712	0.738	0.753	0.761	0.766
Prob_Raw	0.733	0.773	0.779	0.782	0.782	0.783
Prob_QCS	0.743	0.774	0.779	0.781	0.782	0.783
Prob_Raw+QCS	0.738	0.771	0.780	0.781	0.783	0.783
Ideal	0.783	0.783	0.783	0.783	0.783	0.783

	5	10	15	20	25	30
Dist_QCD	0.590	0.657	0.681	0.694	0.702	0.707
Prob_Raw	0.676	0.713	0.719	0.721	0.722	0.722
Prob_QCS	0.686	0.714	0.719	0.721	0.722	0.722
Prob_Raw+QCS	0.681	0.712	0.720	0.721	0.722	0.722
Ideal	0.723	0.723	0.723	0.723	0.723	0.723

Fig. 3. Top-1 recall of the first-level retrieval in SIFT1B.

higher values for $w_1, \ldots, w_k$. In the following experiments, we set w_1 to 100 and other weights to 1 when top-1 recall is evaluated; for top-k recall, we set $w_1, \ldots, w_k$ to 10 and other weights to 1.

4.3.1 *First-Level Retrieval*

In this experiment, we observe the effects on: (1) the first-level clusters ranked according to their NN probabilities instead of their distances from the query, and (2) the three

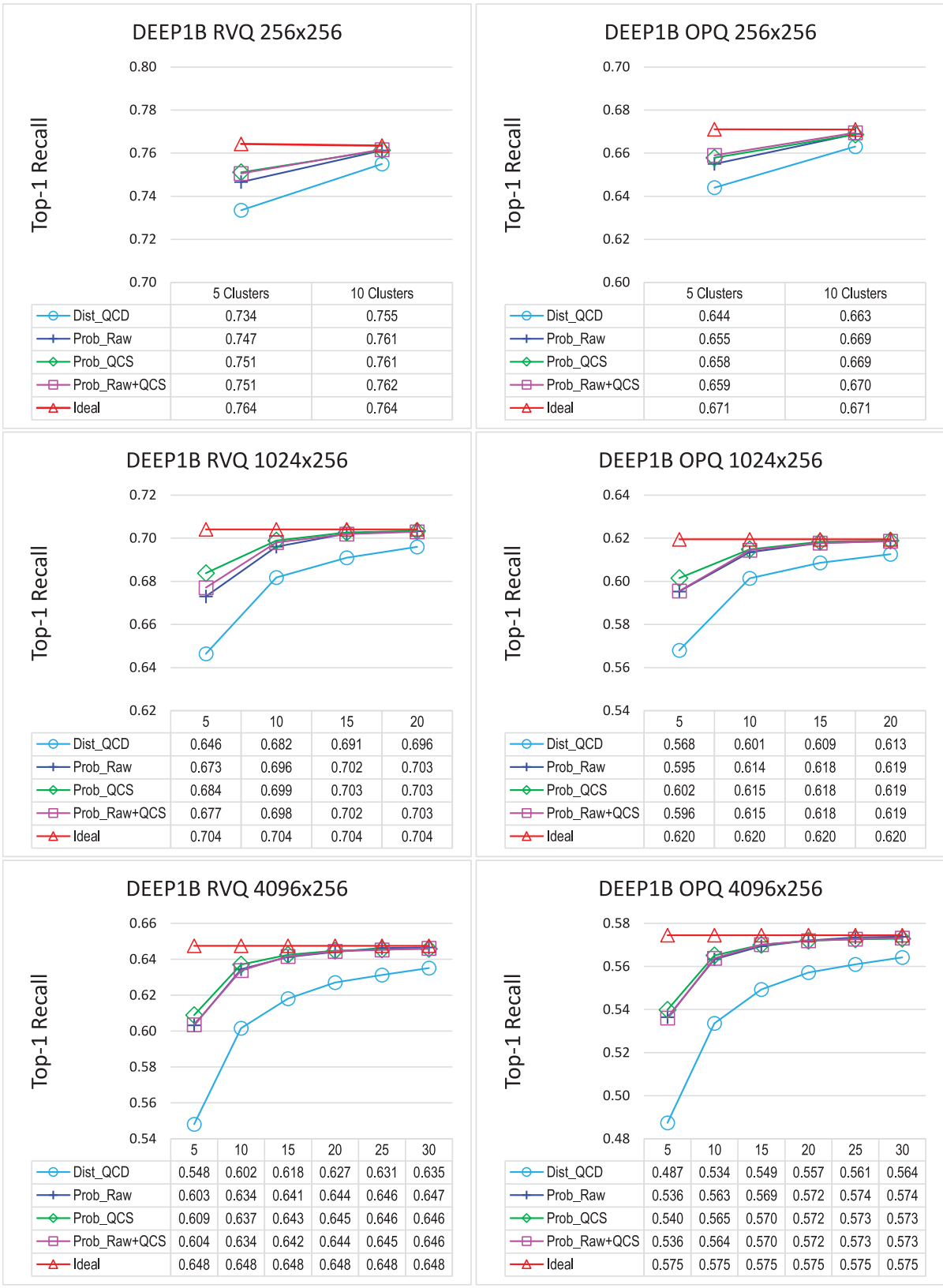


Fig. 4. Top-1 recall of the first-level retrieval in DEEP1B.

query-dependent features used to derive the NN probabilities. To evaluate the performance, we considered the following approaches for comparison.

- *Distance-based ranking by query-centroid distances* (abbreviated as *Dist_QCD*). This approach applies the conventional distance-based ranking to rank the

first-level clusters according to their euclidean distances from the query.

- *Probability-based ranking by raw query data* (*Prob_RAW*). This approach applies the proposed probability-based ranking to rank the first-level clusters according to their probabilities derived from the raw query data.
- *Probability-based ranking by query-centroid similarities* (*Prob_QCS*). This approach is similar to *Prob_RAW* but the probabilities are derived from the similarities between the query and cluster centroids.
- *Probability-based ranking by raw query data and query-centroid similarities* (*Prob_RAW+QCS*). This approach is similar to *Prob_RAW* but the probabilities are derived from the concatenation of the raw query data and query-centroid similarities.
- *Ideal ranking* (*Ideal*). This approach foresees the NN ground truth distribution among clusters. It always select the best first-level clusters that contain the maximum of the NN ground truth.

Figs. 3 and 4 show top-1 recall when retrieving the top-R first-level clusters for SIFT1B and DEEP1B, respectively, where R = 5, 10, 15, 20, 25, and 30. The *Ideal* approach obtained the highest accuracy and served as the upper bound. The *Prob_RAW*, *Prob_QCS*, and *Prob_RAW+QCS* approaches all obtained higher accuracy than the *Dist_QCD* approach, thereby demonstrating that the proposed probability-based ranking generally performed better than the conventional distance-based ranking. These results support our claim that the neighborhood relationships embedded in the index space can be learned by a nonlinear mapping function. The function predicts the NN probabilities, which can be used in a more effective cluster ranking process than the euclidean distances.

Among these probability-based approaches, *Prob_QCS* obtained the best overall accuracy. However, *Prob_RAW +QCS* based on the concatenation of the two query-dependent features did not improve the performance further. We consider that the query-centroid similarity feature provided sufficient information to learn a sufficiently good representation for NN search. Concatenating the raw query data could introduce some noise or redundant information, and thus it had a detrimental influence on the search accuracy.

4.3.2 Second-Level Retrieval

In the next experiment we focus on the effects of quantity estimation and hierarchical learning for the second-level clusters. We selected *Prob_RAW* as the representative to demonstrate the ability of the proposed method. *Prob_RAW* did not have the best performance among the probability-based approaches, but it had the most compact neural network and this facilitated model training and inference.

The selection for the second-level clusters was the main issue to be addressed here. Thus, the following approaches with different selection schemes were implemented.

- *Distance-based ranking by query-centroid distances* (*Dist_QCD*).
- *Distance-based ranking by short-list extraction* (*Dist_SLE*). This approach implements the short-list extraction method [28] based on distance-based ranking.
- *Probability-based ranking by raw query data* (*Prob_RAW*).
- *Probability-based ranking by raw query data with sum normalization* (*Prob_RAW+*γ_{sum}). This approach is based on *Prob_RAW* but calculates S by Eq. (7) to determine the number of the second-level clusters to be retrieved. S is rescaled by the sum normalization function given in Eq. (8) and the second-level clusters are ranked by Eq. (15).
- *Probability-based ranking by raw query data with standard normalization* (*Prob_RAW+*γ_{std}). This approach is similar to *Prob_RAW+*γ_{sum} but S is rescaled by the standard normalization function in Eq. (9).
- *Probability-based ranking by raw query data with exponent normalization* (*Prob_RAW+*γ_{exp}). This approach is similar to *Prob_RAW+*γ_{sum} but S is rescaled by the exponent normalization function in Eq. (10).
- *Hierarchical probability-based ranking by raw query data with standard normalization* (*HProb_RAW+*γ_{sum}). This approach is similar to *Prob_RAW+*γ_{sum} except hierarchical learning is applied. We rank the second-level clusters using Eq. (18) instead of Eq. (15).
- *Hierarchical probability-based ranking by raw query data with standard normalization* (*HProb_RAW+*γ_{std}). This approach is similar to *Prob_RAW+*γ_{std} except hierarchical learning is applied.
- *Hierarchical probability-based ranking by raw query data with exponent normalization* (*HProb_RAW+*γ_{exp}). This approach is similar to *Prob_RAW+*γ_{exp} except hierarchical learning is applied.

Figs. 5 and 6 show top-1 recall along with the candidates retrieved from the second-level clusters for SIFT1B and DEEP1B, respectively. The curves were generated with increasing numbers of the first-level clusters R. From the top-R first-level clusters, these approaches retrieved the same number of second-level clusters but they were selected in different ways. The result shows that, the proposed cluster quantity estimation effectively improved the search accuracy, thereby supporting our idea that the high-ranking cluster should retrieve a large number of candidates in proportion to its NN probability. It makes the probability-based ranking outperforms the distance-based ranking. Furthermore, the proposed hierarchical learning scheme greatly boosted the performance. A surprising result was that the probability-based ranking still performed well at predicting the NN probabilities of subclusters when using only one neural network to characterize the neighborhood relationships in the residual (second-level) space.

The normalization functions γ_{sum}, γ_{std}, and γ_{exp} all had beneficial effects when integrating with the ranking approach *Prob_RAW*. Function γ_{std} obtained slightly better recall than the other two functions. However, when applying hierarchical learning, they did not differ significantly and generally performed well in all cases.

4.3.3 Scalability Analysis

We conduct three experiments to analyze the scalability of the proposed ranking model. More precisely, we investigate the performance on a large size of the input/output layer of the neural network.

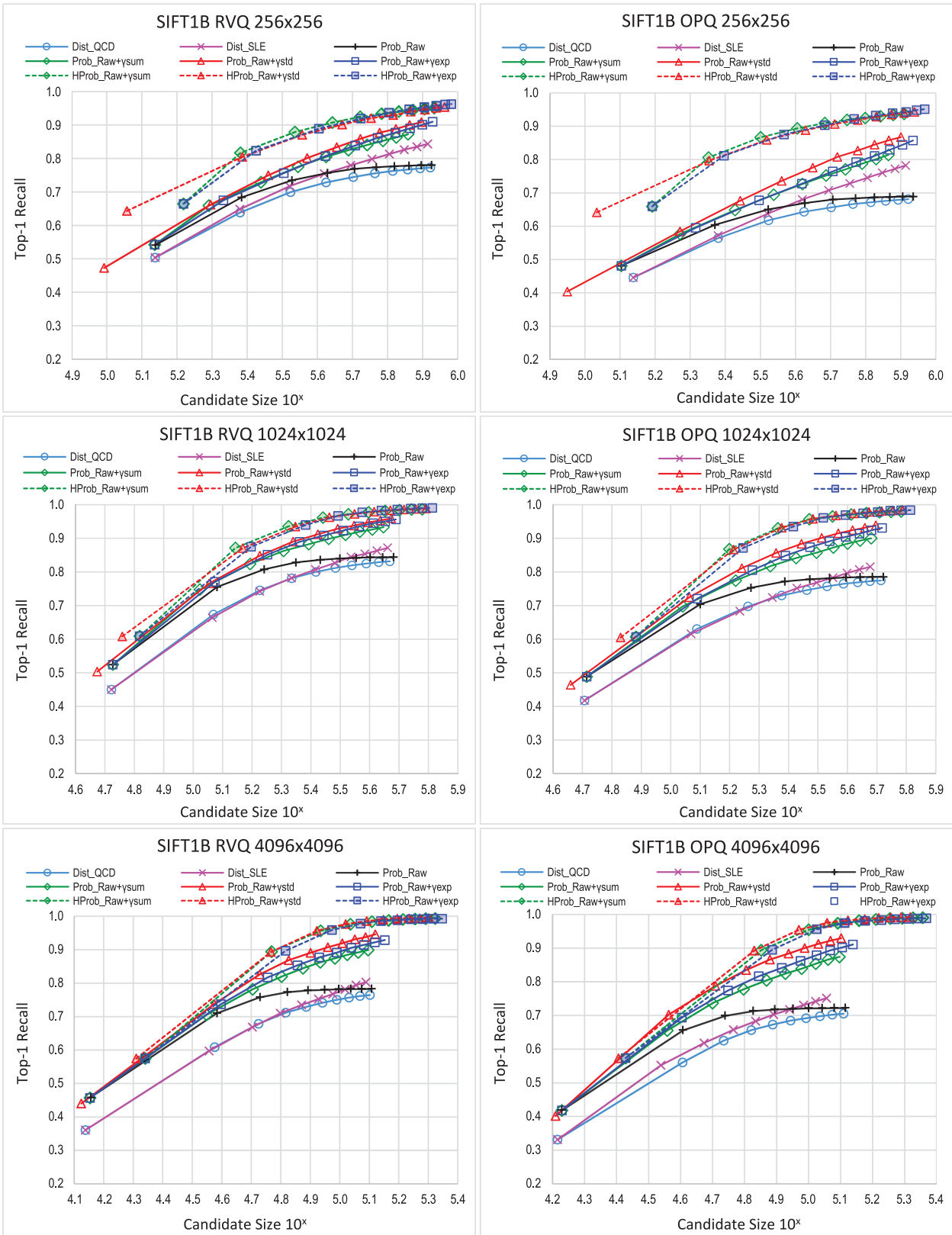


Fig. 5. Top-1 recall of the second-level retrieval in SIFT1B.

In the first experiment, we consider a large output layer size, which means a large number of clusters to be predicted. We prepared the following first-level codebooks with different sizes M = 256, 1024, 4096, and 16384 to produce the SIFT1B RVQ index structures for the first-level cluster retrieval. The number of the second-level clusters N

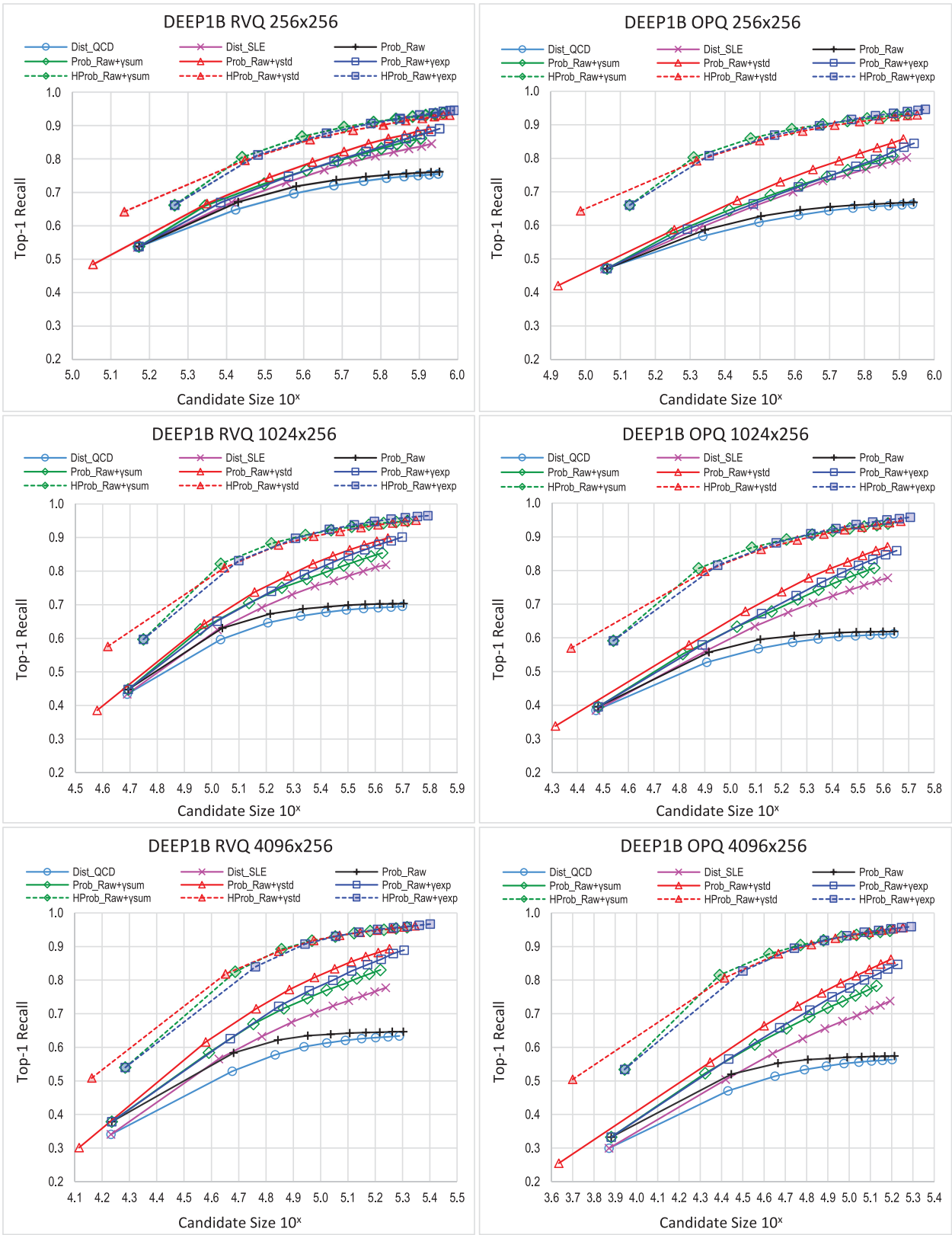


Fig. 6. Top-1 recall of the second-level retrieval in DEEP1B.

was fixed to 256. Table 2 lists the result by using the *Prob_RAW* approach when retrieving the top-R first-level clusters, R = 1, 3, 5, 10, and 15. The runtime overhead mainly comes from probability prediction and partial sorting for M clusters, which takes $O(\tau\lambda^2 + \lambda M)$ and $O(M)$, respectively (see Section 3.3). Hence, the runtime grows linearly with M.

The second experiment is about the input dimensionality. We made a comparison for indexing the PCA-compressed

TABLE 2
Top-1 Recall and Runtime (Milliseconds) of the First-Level Retrieval for Different First-Level Codebooks in SIFT1B RVQ

	1 clu.	3 clu.	5 clu.	10 clu.	15 clu.	time
256×256	0.612	0.637	0.637	0.637	0.637	0.06
1024×256	0.583	0.708	0.721	0.722	0.722	0.09
4096×256	0.644	0.783	0.799	0.804	0.804	0.20
16384×256	0.717	0.851	0.864	0.870	0.871	0.57

TABLE 3
Top-1 Recall and Runtime (Milliseconds) of the Second-Level Retrieval for Indexing PCA-Compressed and Original Data in SIFT1B and DEEP1B RVQ 4096×4096

	1 clu.	3 clu.	5 clu.	10 clu.	15 clu.	time
SIFT1B 32d	0.803	0.963	0.984	0.995	0.996	1.52
SIFT1B 128d	0.779	0.956	0.978	0.990	0.991	1.58
DEEP1B 32d	0.756	0.946	0.975	0.991	0.993	1.52
DEEP1B 96d	0.753	0.943	0.972	0.988	0.992	1.56

TABLE 4
Top-k Recall (k = 1, 10, 100, and 1000) in SIFT1B and DEEP1B RVQ 4096×4096

	Top-1	Top-10	Top-100	Top-1000
SIFT1B	0.996	0.646	0.598	0.506
DEEP1B	0.993	0.620	0.577	0.487

and original data in SIFT1B RVQ 4096×4096 and DEEP1B RVQ 4096×4096 datasets. For the ranking model, we only changed the input layer size of the neural network to fit the compressed/original data, and kept other layer sizes unchanged. Top-1 recall and runtime were evaluated by using the *HProb_RAW*+γ_{std} approach to perform the second-level retrieval within one hundred thousand candidates. As shown in Table 3, the proposed ranking model is scalable for high-dimensional input data. More important, the original data can be compressed by PCA to enjoy a better indexing performance.

Table 4 lists the third experimental result for top-k recall, k = 1, 10, 100, and 1000. These results are yielded by using *HProb_RAW*+γ_{std} in SIFT1B and DEEP1B RVQ 4096×4096.

4.3.4 Comparison to the State-of-the-Art

Finally, we compare the proposed method with the results reported in the literature [24], [28], [41], [44], [47], as shown in Table 5. To make a fair comparison, we report the proposed method using the original data, instead of the PCA-compressed data, in the indexing stage. The first column lists these methods and their index structure configurations. The memory column measures the index structure size (in bytes per data point) and omits the 4-byte data identity and 16-byte PQ code. Each method retrieved within one hundred thousand candidates per query for reranking. The proposed method was evaluated using *HProb_RAW*+γ_{std} in SIFT1B and DEEP1B RVQ 4096×4096.

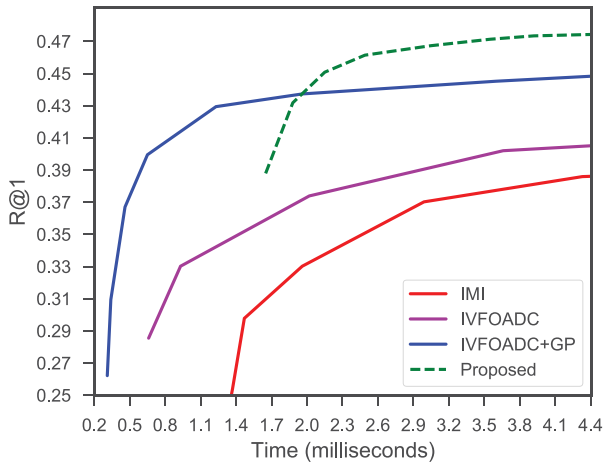


Fig. 7. R@1 versus runtime in DEEP1B.

Our method demonstrated a comparable performance to the state-of-the-arts. In particular, our method is memory-efficient for indexing. It employs the following data structures: an index table, two codebooks, and two neural networks. The two-level index table, which keeps 4096×4096 pointers to inverted lists, takes 128 MB. The first-level and second-level codebooks, each of which stores 4096 128-dimensional real-valued vectors (for SIFT), require 8 MB. The two neural networks used for hierarchical prediction occupy 17 MB. In total we consume close to 160 MB memory to index one billion data points.

Note that in ADC, applying conventional PQ in DEEP1B is less effective than that in SIFT1B due to their different nature; special treatment can be adopted to improve the recall rates in DEEP1B [28]. Fig. 7 plots R@1 versus runtime in DEEP1B to compare the state-of-the-art methods. The runtime for the proposed method started from 1.6 milliseconds, which included the computation overhead for starting to retrieve the first cluster. After 2 milliseconds the proposed method yielded higher recall rates than the other methods.

TABLE 5
Top-1 Recall@**A** (R@1, R@10, and R@100), Runtime (milliseconds), and Memory Consumption (bytes) of the State-of-the-Art Methods

	SIFT1B					DEEP1B				
	R@1	R@10	R@100	time	mem	R@1	R@10	R@100	time	mem
IVFADC (2^{20}) [47]	0.351	0.786	0.918	2.5	0.52	0.405	0.773	0.916	2.5	0.40
IMI ($2^{14} \times 2^{14}$) [44]	0.360	0.792	0.901	5.0	1.34	0.397	0.766	0.909	8.5	1.34
Multi-LOPQ (2^{14}) [24]	0.454	0.862	0.908	19.0	3.22	0.410	0.790	-	20.0	2.68
GNOIMI ($2^{14} \times 2^{14}$) [28]	-	-	-	-	-	0.450	0.810	-	20.0	3.75
IVFOADC+GP (2^{20}) [41]	0.405	0.851	0.957	3.5	2.00	0.452	0.832	0.947	3.3	1.87
Proposed ($2^{12} \times 2^{12}$)	0.457	0.862	0.977	4.7	0.16	0.475	0.785	0.938	4.5	0.16

5 Conclusion

In this study, we proposed a novel ranking model for learning the neighborhood relationships embedded in the index space. The proposed model ranks clusters based on their NN probabilities predicted by the learned neural networks. It can replace the distance-based ranking model and may be integrated with other quantization/search methods to boost their retrieval performance. Our experimental results demonstrated that the proposed model was effective at alleviating the information loss problem during NN search.

Acknowledgments

This work was supported in part by Ministry of Science and Technology, Taiwan, R.O.C. under grant MOST 106-2221-E-415-019-MY3. The authors would like to thank Dmitry Baranchuk for the kindly help with figure reproducing and the reviewers for the thoughtful comments and suggestions.

References

[1] 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, 2017, Art. no. 43.
[2] 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.
[3] A. Arampatzis and G. Kalamatianos, "Suggesting points-of-interest via content-based, collaborative, and hybrid fusion methods in mobile devices," *ACM Trans. Inf. Syst.*, vol. 36, no. 3, 2017, Art. no. 23.
[4] J. Zhang, J. Tang, C. Ma, H. Tong, Y. Jing, J. Li, W. Luyten, and M.-F. Moens, "Fast and flexible top-k similarity search on large networks," *ACM Trans. Inf. Syst.*, vol. 36, no. 2, 2017, Art. no. 13.
[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[6] Z. Huang, B. Hu, H. Cheng, H. T. Shen, H. Liu, and X. Zhou, "Mining near-duplicate graph for cluster-based reranking of web video search results," *ACM Trans. Inf. Syst.*, vol. 28, no. 4, 2010, Art. no. 22.
[7] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.
[8] S. Ercoli, M. Bertini, and A. Del Bimbo, "Compact hash codes for efficient visual descriptors retrieval in large scale databases," *IEEE Trans. Multimedia*, vol. 19, no. 11, pp. 2521–2532, Nov. 2017.
[9] 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.
[10] M. Hu, Y. Yang, F. Shen, N. Xie, and H. T. Shen, "Hashing with angular reconstructive embeddings," *IEEE Trans. Image Process.*, vol. 27, no. 2, pp. 545–555, Feb. 2018.
[11] T. Ge, K. He, and J. Sun, "Graph cuts for supervised binary coding," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 250–264.
[12] K. Li, G.-J. Qi, and K. A. Hua, "Learning label preserving binary codes for multimedia retrieval: A general approach," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 14, no. 1, 2017, Art. no. 2.
[13] F. Shen, Y. Yang, L. Liu, W. Liu, D. Tao, and H. T. Shen, "Asymmetric binary coding for image search," *IEEE Trans. Multimedia*, vol. 19, no. 9, pp. 2022–2032, Sep. 2017.
[14] H.-F. Yang, K. Lin, and C.-S. Chen, "Supervised learning of semantics-preserving hash via deep convolutional neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 2, pp. 437–451, Feb. 2018.
[15] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *CoRR*, vol. abs/1408.2927, 2014, http://arxiv.org/abs/1408.2927
[16] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to hash for indexing big data–a survey," *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.
[17] 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.
[18] H. Jégou, 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.
[19] Y. Matsui, K. Ogaki, T. Yamasaki, and K. Aizawa, "PQK-means: Billion-scale clustering for product-quantized codes," in *Proc. 25th ACM Int. Conf. Multimedia*, 2017, pp. 1725–1733.
[20] D. W. Blalock and J. V. Guttag, "Bolt: Accelerated data mining with fast vector compression," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 727–735.
[21] D. Xu, I. Tsang, and Y. Zhang, "Online product quantization," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 11, pp. 2185–2198, Nov. 2018.
[22] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 2946–2953.
[23] M. Norouzi and D. J. Fleet, "Cartesian k-means," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 3017–3024.
[24] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 2321–2328.
[25] L. Yu, Z. Huang, F. Shen, J. Song, H. T. Shen, and X. Zhou, "Bilinear optimized product quantization for scalable visual content analysis," *IEEE Trans. Image Process.*, vol. 26, no. 10, pp. 5057–5069, Oct. 2017.
[26] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11 259–11 273, 2010.
[27] B. Wei, T. Guan, and J. Yu, "Projected residual vector quantization for ANN search," *IEEE Multimedia*, vol. 21, no. 3, pp. 41–51, Jul.-Sep. 2014.
[28] A. Babenko and V. Lempitsky, "Efficient indexing of billion-scale datasets of deep descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2055–2063.
[29] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 931–938.
[30] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proc. 31st Int. Conf. Int. Conf. Mach. Learn.*, 2014, pp. 838–846.
[31] A. Babenko and V. Lempitsky, "Tree quantization for large-scale similarity search and classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 4240–4248.
[32] A. Babenko and V. Lempitsky, "Annarbor: Approximate nearest neighbors using arborescence coding," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 4885–4893.
[33] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 97–104.
[34] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. IEEE Comput. Society Conf. Comput. Vis. Pattern Recognit.*, 2006, vol. 2, pp. 2161–2168.
[35] 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.
[36] M. E. Houle and M. Nett, "Rank-based similarity search: Reducing the dimensional dependence," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 1, pp. 136–150, Jan. 2015.
[37] S. Liu, J. Shao, and H. Lu, "Generalized residual vector quantization and aggregating tree for large scale search," *IEEE Trans. Multimedia*, vol. 19, no. 8, pp. 1785–1797, Aug. 2017.
[38] W. Dong, M. Charikar, and K. Li, "Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces," in *Proc. 31st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2008, pp. 123–130.
[39] F. André, A.-M. Kermarrec, and N. Le Scouarnec, "Accelerated nearest neighbor search with quick ADC," in *Proc. ACM Int. Conf. Multimedia Retrieval*, 2017, pp. 159–166.
[40] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.
[41] D. Baranchuk, A. Babenko, and Y. Malkov, "Revisiting the inverted indices for billion-scale approximate nearest neighbors," in *Proc. 15th Eur. Conf. Comput. Vis.*, 2018, pp. 209–224.
[42] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognit. Lett.*, vol. 31, no. 11, pp. 1348–1358, 2010.

[43] Y. Xia, K. He, F. Wen, and J. Sun, "Joint inverted indexing," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 3416–3423.
[44] A. Babenko and V. Lempitsky, "The inverted multi-index," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3069–3076.
[45] Y. Matsui, T. Yamasaki, and K. Aizawa, "PQTable: Fast exact asymmetric distance neighbor search for product quantization using hash tables," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1940–1948.
[46] C.-Y. Chiu, Y.-C. Liou, and A. Prayoonwong, "Approximate asymmetric search for binary embedding codes," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 1, 2017, Art. no. 3.
[47] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2011, pp. 861–864.
[48] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 3304–3311.
[49] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 584–599.

Chih-Yi Chiu received the BS degree in information management from National Taiwan University, in 1997, and the MS degree in computer science from National Taiwan University, in 1999, and the PhD degree in computer science from National Tsing Hua University, Taiwan, in 2004. From 2005 to 2009, he was with Academia Sinica. In 2009, he joined National Chiayi University. His current research interests include multimedia information retrieval, data mining, and deep learning.

Amorntip Prayoonwong received the BS degree in business computer from Yonok University, Thailand, in 1995, and the MS degree in computer science from Assumption University, Thailand, in 2004. She is currently working toward the PhD degree in the Computer Science and Information Engineering, National Chiayi University. Her current research interest includes index structures and nearest neighbor search.

Yin-Chih Liao received the BS degree in computer science and information engineering from National Chiayi University, in 2017. His current research interests include multimedia information retrieval and deep learning.