Connecting Compression Spaces with Transformer for Approximate Nearest Neighbor Search

Haokui Zhang[1,2], Buzhou Tang[*2], Wenze Hu[1], and Xiaoyu Wang[1]

[1] Intellifusion, Shenzhen, China
hkzhang1991@mail.nwpu.edu.cn, windsor.hwu@gmail.com, fanghuaxue@gmail.com
[2] Harbin Institute of Technology (Shenzhen), Shenzhen, China
tangbuzhou@hit.edu.cn

Abstract. We propose a generic feature compression method for Approximate Nearest Neighbor Search (ANNS) problems, which speeds up existing ANNS methods in a plug-and-play manner. Specifically, based on transformer, we propose a new network structure to compress the feature into a low dimensional space, and an inhomogeneous neighborhood relationship preserving (INRP) loss that aims to maintain high search accuracy. Specifically, we use multiple compression projections to cast the feature into many low dimensional spaces, and then use transformer to globally optimize these projections such that the features are well compressed following the guidance from our loss function. The loss function is designed to assign high weights on point pairs that are close in original feature space, and keep their distances in projected space. Keeping these distances helps maintain the eventual top-k retrieval accuracy, and down weighting others creates room for feature compression. In experiments, we run our compression method on public datasets, and use the compressed features in graph based, product quantization and scalar quantization based ANNS solutions. Experimental results show that our compression method can significantly improve the efficiency of these methods while preserves or even improves search accuracy, suggesting its broad potential impact on real world applications. Source code is available at https://github.com/hkzhang91/CCST

Keywords: Approximate nearest neighbor search; transformer; neighborhood relationship preserving; compression projections; retrieval

1 Introduction

Approximate nearest neighbor search (ANNS) methods focus on searching for k approximate nearest neighbors from a given database to a given query node q. It is a fundamental technology in information retrieval and is widely used in applications such as search engines and recommendation systems. The common goal of ANNS approaches is to minimize the search latency while maintain a low search accuracy loss on a fixed hardware constraint.

As stated in [1], currently, the two most popular ANNS methods are graph based approaches and quantization based approaches. Many of these methods, such as product quantization (PQ) [2], HNSW [3] and NSG [4] are widely used in real-world applications. Although popular, there are rooms to further improve them. In PQ, the sub vectors are quantized into codewords using a clustering objective, which is a poor proxy to search accuracy. In graph based methods, there are a large number of distance computations in index building time (for instance, indexing complexity NSG graph is $O(kn^{\frac{1+d}{d}} \log n^{\frac{1}{d}} + n \log n)$ [4], where d is data dimension), which in real world applications can last weeks on billion scale datasets. For PQ related methods, introducing the feature compression as an intermediate step avoids direct feature quantization from high dimensional space. Such a two stage quantization strategy usually results in higher accuracy. For graph based methods, computing distance in low dimensional space reduces computational cost linearly, which significantly speeds up indexing.

Existing feature compression methods such as principal components analysis (PCA) [5], Variational Auto-Encoders(VAE) [6] focus on keeping the information in the input features instead of their neighborhood relations, but in ANNS applications the neighborhood structure affects the eventual accuracy more than the locations of the individual data points themselves. As is shown in experiments, directly applying these methods lead to significantly reduced search accuracy, which is not suitable for applications requiring both high speed and high recall.

In this paper, we propose a feature compression method for the approximate nearest neighbor search problem, which aims to retain the local neighborhood relations instead of the fidelity of the reconstructed features. Our method is composed of a compression network which connect compression spaces with transformer (CCST), and an inhomogeneous neighborhood relationship preserving (INRP) loss.

Our CCST is a combination of projection units that projects original features to low dimensions, and transformer units that compose these projections to generate the output feature. The projection unit is initialized as sparse random projection (SRP) [7], which is proved by JL (Johnson–Lindenstrauss) lemma that this projection reserves distances of an increasing number of data points with decreasing dimension reduction ratios. We design our network with multiple projections units, so that each can be used to compress a local neighborhood, and the entire space can be covered in a piece-wise manner by the ensemble of these units. We then use transformers to adaptively combine the output of these random projection units, because the transformer unit has the well known property of globally attending its inputs, which could lead to better global alignment of these projections. In other words, multiple compression spaces are connected to one via transformer. Different from standard transformer based networks, we design an input dependent, non-trivial output token named compression token, which is itself a compressed feature derived from the input feature, and is updated by the transformer layers with skip connections. The compression token is designed to provide an anchor vector to the multi-head attention units so that

transformers properly mix the output of random projection units to generate the output compressed feature.

Our training loss is designed for the purpose of preserving local neighborhood structures. In most ANNS applications, user is mostly interested in the accuracy of the top few nearest neighbors. So, changes in the distances between a query and its far away points will not affect the search result, as long as their distances are still large. Inspired by this property in ANNS problems, we design the INRP loss, which assigns high losses on point pairs whose distances are within a threshold. This loss design allows certain information of the original feature to be discarded while still maintaining high retrieval accuracy.

Our experiments show that our feature compression model can be seamlessly used with popular ANNS methods. While saving 1/2 to 3/4 indexing time, graphs built using our compressed features can improve the recall slightly for HNSW and NSG methods. It also significantly improves recalls of 1@1 and recalls of 1@5 metrics by more than 10.0 percentage points for PQ based methods, and about 1.0 or 2.0 percentage points for a quantization methods tailored to HNSW methods [8].

Our main contributions are summarized as follows.

1. We propose a novel feature vector compression model CCST for ANNS problem. In CCST, traditional projection based compression units and emerging transformer units are jointly used to compress features for ANNS problems. To our knowledge, this is the first attempt to apply transformer to ANNS.
2. We propose an INRP loss that mainly keeps the distances of a point and its close local neighbors. This maintains top k retrieval accuracy and creates space for lossy feature compression.
3. The experiment results show that our proposed method can be used in most ANNS methods to improve efficiency. It speeds up indexing speed to $2\times$ to $4\times$ for graph based methods and it improves recalls significantly for PQ related methods.

2 Related Work

2.1 ANNS methods

Existing ANNS approaches can be divided into four main types: 1) tree-structure based strategies [9], which partitions indexed datapoints into different subspaces based on specific conditions; 2) Locality sensitive hashing (LSH) related methods, which map similar items to the same symbol with a high probability [10]; 3) product quantization (PQ) related approaches decompose the space into a Cartesian product of low dimensional subspaces and quantize each subspace separately [2]; 4) graph based frameworks search on pre-built relative neighborhood graphs (RNG) to find the closest data points to query [3]. Compared with PQ and graph based methods, tree based and LSH based methods need ensembles of trees or hash tables to achieve similar accuracy, which consume considerably more memory and are less frequently used in large scale ANNS problems.

Product quantization based approach reduces the index memory cost by holding quantized codes and speeds up distance computation using online or offline computed distance look up tables. When combined with inverted index to further reduce the number of distance computations, PQ becomes a strong baseline for modern ANNS systems. Compared with quantization based approach, which partitions original feature space to generate shorter features, our method directly maps the original features into a new space. These two method are orthogonal. As is shown in experiments, the new space can be further quantized by PQ, which generates even better speed and accuracy trade-offs than using PQ method alone.

Graph based algorithms usually construct a navigable graph over database, and searches along the edges of this built graph to find the closest points to query. The most recent works in this category are NSG [4], HNSW [3], Disk-ANN [1] and HM-ANN [11]. NSG builds a relatively sparse indexing graph to reduce memory usage and improve search speed. HNSW employs hierarchical graph structure to reduce query latency. Disk-ANN and HM-ANN focus on reducing memory overhead via adopting quantization and heterogeneous memory. Besides, Douze et al. proposed Link and Code (L&C), which takes HNSW as basic ANNS framework and replaces full precision vectors with refined quantiztion codes [8]. Compared with brute force, these graph based methods visit 1/1000 or even less of the indexed points in each search, significantly reduce the number of needed distance computations. However, most of these methods suffer from the problem of high indexing time, as constructing a navigable graph over database has high time complexity. For instance, $O(n \log n)$ for HNSW and $O(kn^{\frac{1+d}{d}} \log n^{\frac{1}{d}} + n \log n)$ for NSG. In real world application, building a navigable graph over a billion-scale database with 30+ threads costs several days or even weeks.

2.2 Dimension reduction / compression

In early stage, almost all dimension reduction methods are based on mathematical theory. PCA [5] and independent components analysis (ICA) [12] are proposed to transform vectors into lower dimensional space, where the most information of high dimension vectors are retained. Random projection [13][7] project data into low dimensional space, while approximately preserving structure information in original feature space. The efficiency of random projection methods is guaranteed by Johnson-Lindenstrauss (JL) lemma [14]. This lemma states that data points in sufficiently high dimensional space may be projected into suitable low dimensional space while approximately preserving the distances between the points in original space.

In 2006, Hinton and Salakhutdiov proposed to compress high dimensional data via neural network [15], and several recent works on dimension reduction have shifted their methods to neural networks. For example, VAE maps data to low dimensional space and forces the encoded features to follow a multivariate Gaussian distribution [6].

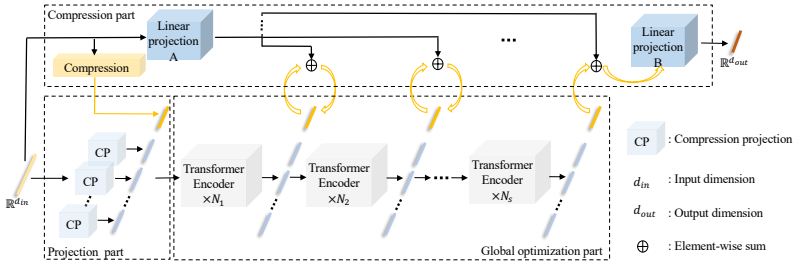

Fig. 1. Overview of the CCST model. The proposed model consists of three major parts, including compression part, projection part and global optimization part. The projection part projects input vectors into multiple subspaces and outputs a sequence of sub vectors. Global optimization part optimizes the sequence of sub vectors from global perspective and summarizes all optimized information into compression token. Compression part provides initial the compression token, and interacts with global optimization part to generate the final compression result.

Very recently, Sablayrolles et al. [16] proposed an end-to-end quantization model for similarity search, where a compression network named catalyst is designed and trained to compress vectors from d_{in}-dimensional input space to the hypersphere of a d_{out}-dimensional space, where $d_{in} > d_{out}$. Our proposed CCST is close to this work, as it involves an ANNS oriented loss and a learnable network to compress feature vectors. Compared with catalyst, our training objective is easier to implement as it does not involve offline exhaustive search for positive and negative pairs. Also, we design a deep compression model for the ANNS problem, where [16] uses multi-layer perceptrons to demonstrate their entropy maximization idea.

2.3 Transformer

Vaswani et al. first proposed the transformer architecture for the task of language modelling in [17]. Inspired by the promising performance of transformer, several works started to introduce transformer into vision tasks and proposed a new type of models. Dosovitskiy et.al proposed ViT, in which the input image is cropped into a sequence of patches to meet the input format requirement of transformer [18]. Later, Touvron et al. proposed to use knowledge distillation to overcome the difficulties of training ViT models [19]. Liu et al proposed a hierarchical transformer, where representation is computed within shifted windows [20] to reduce computation cost. Very recently, Graham et al. mixed CNN and transformer in their LeVit model, which significantly outperforms previous CNNs and ViT models with respect to the speed/accuracy tradeoff [21].

The transformer part of our CCST model is inspired by the works above. We have made several modifications to better fit transformer with our applications, which are detailed in section *global optimization part*.

3 The Proposed Approach

In this section, we present our proposed method in detail. We first introduce the architecture of CCST. Then we elaborate on our INRP loss function.

3.1 CCST

As illustrated in Fig. 1, the proposed CCST consists of three major parts, including projection part, global optimization part and compression part.

The projection part casts the input feature vector $x \in \mathbb{R}^{d_{in}}$ to n different low dimensional spaces. The output of this part is a sequence of low dimensional vectors $\left[p^1(x), p^2(x), \cdots p^n(x)\right]$, where $p^i(x) \in \mathbb{R}^{d_{out}}, i = 1, \cdots, n$. d_{in} and d_{out} denote feature dimensions before and after compression, respectively. In addition, note that $[p^1(x), p^2(x), \cdots, p^n(x)]$ constructs a n by d_{out} matrix instead of a $n \cdot d_{out}$ dimensional vector. It is a sequence of tokens as the input to transformers instead of a concatenated vector. $p^i()$ is a compression projection function. $p^i(x) = W^i x$, where $W^i \in \mathbb{R}^{d_{in} \times d_{out}}$. The global optimization part takes $\left[cp(x), p^1(x), p^2(x), \cdots p^n(x)\right]$ as input and globally optimizes all sub vectors step by step. The global optimization part then summarizes all optimized sub vectors into a compression token $cp(x)$. The compression part is responsible for initializing the compression token, interacting with global optimization part in each step and further processes the information in compression token to generate the final compression result $f(x) \in \mathbb{R}^{d_{out}}$.

Projection part Our goal is compressing feature vectors into a low dimensional space, where the data neighborhood relation in original space is preserved. It seems that random projections satisfy our requirement and there are already some classical random projection methods. Unfortunately, according to JL lemma [14], the projection error of using single random projection function satisfies:

$$(1-\epsilon)||x_i - x_j||_2^2 \leq ||p(x_i) - p(x_j)||_2^2 \leq (1+\epsilon)||x_i - x_j||_2^2 \tag{1}$$

where x_i and x_j represent two data points in high dimensional space and $p(x_i)$ and $p(x_j)$ are corresponding projection data points in low dimensional space. $p()$ is the projection function. ϵ satisfies equation $d_{out} > \frac{4\ln(m)}{\epsilon^2/2 - \epsilon^3/3}$, where $0 < \epsilon < 1$ and m represents data size. Random projections are inapplicable in our case. For example, if compressing 960 dimensional vectors in GIST1M dataset to 480 dimensions, we have $0.63 < \epsilon < 1$. In GIST1M, the distance of a query to its nearest neighbor and to its hundredth neighbor are probably 1.177 and 1.5615, respectively. After projection, with the minimum ϵ, the distance of a query to its nearest neighbor and to its hundredth neighbor fall into the scope of [0.7160, 1.5027] and the scope of [0.9498, 1.9936], respectively. These two scopes have a considerable overlap, which may disturb neighborhood relationship and lead to a lower search accuracy. We can draw a consistent conclusion from our experimental results.

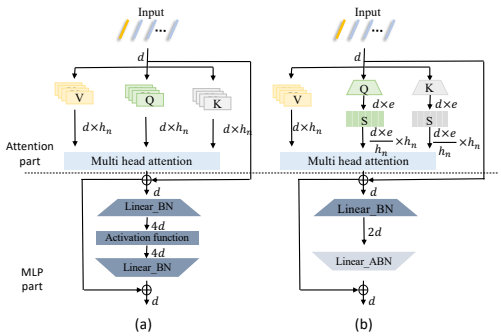

Fig. 2. Different modules. (a) Transformer module of ViT [18]. (b) Transformer module proposed in this paper. By adopting lightweight designs, transformer module in (b) has fewer parameters than that in (a).

Existing random projection methods do not address our problem directly, but they give us some inspirations. Here, we propose to use n different projections to boost compression accuracy. Specifically, we initialize the n projection matrices in $p^i(x) = W_i x$ $(i = 1, 2, \cdots, n)$ on input $x \in \mathbb{R}^{d_{in}}$ to generate a sequence of features $\left[p^1(x), p^2(x), \cdots p^n(x)\right]$. Following [7], the elements in W_i are randomly drawn from:

$$\begin{cases} -\sqrt{\frac{s}{d_{out}}} & \text{with probability} \quad 1/2s, \\ 0 & \text{with probability } 1 - 1/s \\ \sqrt{\frac{s}{d_{out}}} & \text{with probability} \quad 1/2s \end{cases} \tag{2}$$

where $s = \sqrt{d_i n}$. Different from traditional random projections, projection matrices used in here will be further optimized by our INRP loss.

Global optimization part The projection part outputs a sequence of low dimensional vectors, which brings in the core problem of this global optimization part, that is how to generate the result compressed feature using these features from different sub spaces.

Here, as the problem that we need to solve is exactly what transformer model is good at, we tailor a transformer model to overcome the core problem. One biggest advantage of transformer is it good at capturing and taking use of relationships between different tokens to get a global optimization result. When we treat features from different sub spaces as different tokens, our core problem becomes optimizing a sequence of tokens from the global perspective. So, based on ViT [18] structure, we propose a new transformer model for feature compression.

Based on the characteristics of our compression problem, we have made four major modifications to the original ViT structure, including two structural modifications and two lightweight designs.

Structural modifications ensure the new proposed transformer structure is aligned with our problem. These two modifications are:

1. Discard position embedding. In transformer models for processing sentences or images, position embedding and relative position embedding are widely used, as the order of words and the positions or relative positions of images patches contain important information. However, in our case, the order of random projections is fixed. Naturally, position embedding is discarded in our transformer model. Interestingly, when we add position embedding into our transformer, the search accuracy drops. We conjecture this is because randomly initialized position coding disturbs our model.
2. Add compression token. In our transformer model, an extra token named compression token is used. Structurally, this design is similar to the transformer model in ViT, where an extra token named classification token is also employed. Compression token and classification token are added for different purposes and they have different functions. Compression token provides a base reference for optimizing and summarizing information from different projection spaces and it also works as a bridge between compression part and global optimization part. Classification token is used as a placeholder to perform classification. Different from classification token, which is a learned constant, our compression token is derived from input feature vector. Initializing compression token with random numbers is not in accordance with its role of working as base reference.

Lightweight designs are adopted to reduce the number of learnable parameters. Fig. 2 shows lightweight designs for MLP and attention parts are:

1. In MLP part, we decrease the expansion ratio (changes among widths of the adjacent linear mapping layers) from 4 to 2 to reduce the number of parameters by half, as shown in the bottom half of Fig. 2 (b). In addition, the basic computing unit in our model is Linear_ABN, which, in data processing order, consists of a linear mapping layer, activation function and batch normalization. As pointed out in [22], placing batch normalization layer before ReLU leads to the conv layer updated in a suboptimal way due to the nonnegative responses of activation function. So we design our unit following the order of conv→activation→bn. This order is also recommended in [23].
2. Attention part has three intermediate variables, which are represented as Value (V), Query (Q) and Key (K). The output of head i is calculated as:

$$\text{head}_i(Q, K, V) = \text{softmax}(\frac{QW_i^Q \cdot KW_i^{K^T}}{\sqrt{d}})VW_i^V \tag{3}$$

where W_i^Q, W_i^K and W_i^V are three projection matrices. As in Eqn. 3, V stores main information, Q and K are used to generate attention weights. In ViT, Q, K and V are all the same in each head. Inspired by [17], we decrease the dimension of Q and K for reducing parameters of W_i^K and W_i^V.

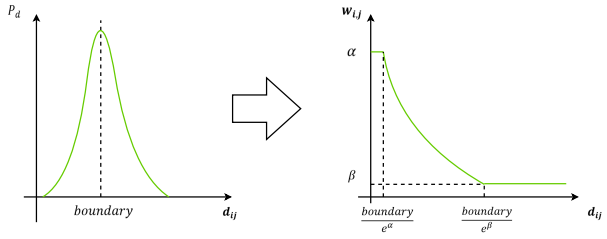

Fig. 3. The weight curve of INRP loss. d_{ij} is Euclidean distance between x_i and x_j.

Structurally, as shown in the middle part of Fig. 1, global optimization part consists of s stages, each of which has N_i transformer encoders. From stage 1 to $s-1$, projected vector from compression part is added to compression token at the end of each stage. Finally, $cp(x)$ is from the output of last stage is fed back to compression part to generate the final compression result.

Compression part The compression part is responsible for providing initial compression token, interacting with global optimization part in each step and generating the final compression result. Corresponding to three responsibilities three learnable modules are constructed. As shown in the top part of Fig. 1, three modules are a compression module and two linear projection modules.

The compression module is made up of a single Linear_ABN module. Each linear projection module has a linear mapping function. Compression module takes $x \in \mathbb{R}^{d_{in}}$ as input and outputs $cp(x) \in \mathbb{R}^{d_{out}}$. Linear projection A maps input $x \in \mathbb{R}^{d_{in}}$ to d_{out}-dimensional vector, which is added to compression tokens of global optimization part. Linear projection B takes compression token from the last stage of global optimization part as input and output the final compression result.

3.2 INRP loss

To preserve neighborhood structures, a straightforward way is training the network to keep distances between all possible point pairs. However, the goal of ANNS is searching for top k approximate nearest neighbors to a query point, which generally are points that are close to query point in the original space. In other words, focusing on keeping distances between close pairs is more important, as these pairs affect search result the most. In [24], Guo et al. presented a similar idea, which is proposed for quantization instead of compression. Inspired by this, we design our INRP loss, the formula of which is:

$$loss = \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} w_{ij} \cdot \|\|f(x_i) - f(x_j)\|_2 - \|x_i - x_j\|_2\|_2 \tag{4}$$

where x_i and x_j are a pair of any two nodes from original space. m denotes data size. $f(x_i)$ and $f(x_j)$ are compression results. w_{ij} is weight for this pair and it is calculated as:

$$w_{ij} = \min(\alpha, \max(\beta, -\ln(\frac{d_{ij}}{boundary}))) \tag{5}$$

where $d_{ij} = ||x_i - x_j||_2$. α and β are two hyper-parameters, which are set to 2.0 and 0.01 respectively. *boundary* denotes the average distance between any two nodes in original space. The curve is shown in Fig. 3. In implementation, we use all pairs inside a mini-batch to approximate Eqn.4, which significantly simplifies computation.

4 Experiments

4.1 Datasets and implementation details

Datasest We carry out experiments on two million-scale benchmark datasets GIST1M and Deep1M [3]. GIST1M consists of 1 million of 960-dimensional hand-crafted feature vectors, which are extracted from images with GIST descriptors [25]. Deep1M contains 1 million of 256-dimensional deep feature vectors, which are extracted by using GoogleNet [26].
Training setting Following [27][28], we take database as training set. We train our model for 2400 epochs with the Adamw optimzier [29], where the initial learning rate and batch size are set to 1e-4 and 1024 respectively. We use poly learning rate policy with power of 0.9 to adjust the learning rate for every epoch. Just like other deep learning models, training CCST is time consuming. With a single RTX2080Ti GPU, training costs about half a day. However in practice, the trained model will be used to process huge amount of data. In Bigann-1B, indexing the 1 billion data saves 54 hours, which already pays off.
Platforms We implement our CSST using Pytorch and conduct ANNS experiments based on Faiss[4].

4.2 Speeding up indexing for graph-based methods

Here, we focus on using our proposed CSST to speed up indexing of graph-based methods. Two most popular graph-based methods HNSW [3] and NSG [4] are employed as baselines. We construct all HNSW indices with M=48, efConstruction=512, and perform search with efSearch=100 on GIST1M and efSearch=200 on Deep1M. Following [4][1], we build all NSG indices using R=60, L=70 and C=500 and initialize K-NN graphs with NN-descent. On GIST1M, we initialize K-NN graph using GK=400, L=400, iter=12, S=15 and R=100. On Deep1M, we initialize K-NN graph using GK=200, L=200, iter=10, S=12 and R=100.

[3] Downloaded from https://www.cse.cuhk.edu.hk/systems/hash/gqr/datasets.html
[4] https://github.com/facebookresearch/faiss/releases/tag/v1.7.1

Table 1. Experiments of speeding up indexing for graph based methods. C_F represents the feature compression factor. IND and SS denote indexing time and search speed. q/s is query per second. Experiments are performed on a server with two Xeon Gold 5218 CPUs. We build indices with 32 threads and perform search with a single thread. All the speed numbers are averaged from 20 cold runs to rule out random factors. Comp_t is encoding time. Here, a single RTX2080Ti GPU is used for accelerating compressing process and the GPU memory usage is restricted to about 1GB.

C_F	Methods										Comp_t (s)	GPU (G)
	HNSW					NSG						
	IND (s)	SS (q/s)	Recall 1@1	Recall 1@10	Recall 100@100	IND (s)	SS (q/s)	Recall 1@1	Recall 1@10	Recall 100@100		
GIST1M												
1	733	182	97.40	100.00	94.39	1118	179	97.40	100.00	93.97	-	-
2	454	184	97.70	100.00	94.55	778	180	98.00	99.90	94.16	21.43	1.19
4	**245**	186	**98.00**	100.00	**95.27**	**645**	182	**98.30**	100.00	**94.85**	10.00	1.17
Deep1M												
1	300	861	99.50	100.00	95.23	505	820	99.7	100.00	94.53	-	-
2	144	870	99.50	100.00	95.46	246	822	99.6	100.00	94.69	5.15	1.09
4	**100**	874	**99.80**	100.00	**95.60**	**209**	842	**99.6**	100.00	**94.92**	4.64	1.06

Table 2. Experiments of speeding up indexing for real world database and billion-scale database. Deepfeat25M is a subset of our internal 1 billion 512-d dataset.

Datasets	C_factor	indexing time(h)	Recall 1@1	Compressing time (h)	GPU usage(G)
Deepfeat25M	1	15.1	95.4	-	-
	2	9.5	95.1	0.18	1.32
Bigann-1B	1	106	92.6	-	-
	2	51	92.8	1.01	1.24

We conduct three groups of experiments with different compression ratios. In all three groups, full-dimensional vectors are used to search nearest neighbors. Full-dimensional vectors, feature vectors compressed with a factor of 2 and 4 are respectively used in indexing. Experiment results listed in Table 1 show that using compressed features triples indexing speed for HNSW and doubles indexing speed for NSG. Taking experiments based on HNSW as an example, using compressed features reduce indexing time from 733 seconds to 454 seconds and 245 seconds for GIST1M dataset and decreases indexing time from 300 seconds to 144 and 100 seconds for Deep1M. Results on NSG show similar trends.

Interestingly, using compression feature vectors slightly improves search accuracy. For instance, on GIST1M, using 4×-compressed vectors improves recall of 1@1 to 98.00% and improves recall of 100@100 to 95.60, 0.5 and 0.37 percentage points higher that that of baseline respectively. We conjecture this is due to that using compressed feature vectors to build index introduces some extra links

Table 3. Fusion experiments. Bytes denotes the size of quantized feacture vectors. Comp. and Quant. represent compression and quantization methods, respectively. Following default settings in their source code, we conduct experiments on L&C [8], except that the M of coder is set to 8 (This gets highest accuracy.). For PQ experiments, we adopt IVFADC (nlist=8) to avoid exhaustive search.

Datasets	Bytes	Comp.	ANNS	Speed (q/s)	Recall 1@1	1@5	1@50
GIST1M	60	-	PQ	115	23.0	45.2	79.7
	60	CSST	PQ	117	39.5(16.5↑)	72.8(27.6↑)	97.3(17.6↑)
	60	-	L&C	1536	31.4	48.9	54.2
	60	CSST	L&C	1527	30.7(0.7↓)	50.2(1.3↑)	55.1(0.9↑)
	30	-	PQ	246	15.2	31.0	62.4
	30	CSST	PQ	251	20.9(5.7↑)	43.4(12.4↑)	79.7(17.3↑)
	30	-	L&C	2000	23.6	43.0	53.9
	30	CSST	L&C	1992	23.9(0.3↑)	43.9(0.9↑)	54.3(0.4↑)
Deep1M	32	-	PQ	236	32.3	67.5	95.9
	32	CSST	PQ	240	53.3(21.0↑)	88.5(21.0↑)	100.0(4.1↑)
	32	-	L&C	2000	47.9	72.3	76.4
	32	CSST	L&C	2020	48.7(0.9↑)	74.1(1.8↑)	78.3(1.9↑)
	16	-	PQ	520	16.8	38.6	73.6
	16	Catalyst	PQ	545	19.8(3.0↑)	47.9(9.3↑)	82.5(8.9↑)
	16	CSST	PQ	537	32.5(15.7↑)	65.6(27.0↑)	94.4(20.8↑)
	16	-	L&C	2677	28.8	55.6	71.6
	16	CSST	L&C	2676	29.2(0.4↑)	56.9(1.3↑)	73.7(2.1↑)

and these links improves search accuracy, just like extra links selected by select neighbors heuristic improves the accuracy of HNSW [3].

Note that the speedup is scalable to larger datasets. As shown in Table 2, for Bigann-1B[5] (1 billion 128-d hand drafted features), using our features reduces indexing time from 106 hours to 51 hours; On our internal dataset Deepfeat25M, comparison results shows similar trend.

4.3 Improving accuracy and speed for PQ related methods

Besides graph-based methods, PQ related methods are also very popular in real world applications. As quantization and dimension compression are orthogonal at the method level, we conjecture that our proposed CSST can be applied on PQ related methods to achieve higher accuracy. In this section, we conduct fusion experiments to verify this point. Specifically, we fuse our proposed CSST with the classical PQ [2] and most recent proposed L&C [8]. In addition, as the inspirational method catalyst [16] is close to our method in training networks to compress feature dimension, we compare our proposed model with catalyst on Deep1M, the dataset which is used in both this paper and their paper. Experiment results are listed in Table 3.

[5] https://dl.fbaipublicfiles.com/billion-scale-ann-benchmarks/bigann/base.1B.u8bin

Table 4. Ablation study on lightweight designs on GIST1M (d_{in}=960). C_F, k/s and GPU denote compression factor, thousand feature vectors per second and the peak GPU memory usage during encoding. Queries are batch-processed, where batch size is set to 512 or 1024 according to GPU GPU memory usage. # param. is the number of learnable parameters in model. GPU adopted here is RTX2080Ti.

C_F	d_{out}	Light designs	# param. (M)	HNSW		encoding info		
				R1@1	R100@100	time (s)	speed (k/s)	GPU(G)
2	480	Y	10.4	97.70	94.55	21.43	46.66	1.19
2	480	N	15.9	97.70	94.57	35.21	28.40	1.37
4	240	Y	4.7	98.00	95.27	10.00	100.00	1.17
4	240	N	6.1	97.98	95.27	14.12	70.82	1.28

Table 5. Ablation study on random projection initialization (RP init) and the proposed INPR loss, using GIST1M dataset.

Settings		None	RP init	INPR loss	RP init and INPR loss
Recall	1@1	76.1	77.9	78.5	80.1
	1@5	94.3	96.1	97.2	98.4

Experiment results show that using compressed feature learned by our proposed CSST improves both search accuracy and speed for PQ related methods. On Deep1M, in experiments of quantizing feature vectors to 32 bytes, combining CSST with PQ improves recalls 1@1, 1@5 and 1@50 by 21.0, 21.0 and 4.1 percentage points respectively. Our proposed CSST also brings more improvement than Catalyst. In experiments of coding input vectors with 16 bytes, using Catalyst improves recalls 1@1 and 1@5 by 3.0 and 9.3 percentage points and our proposed CSST improves recalls by 11.7 and 27 percentage points.

Using CSST also improves search accuracy for L&C, but the improvements CSST brings in here are less than that in PQ. This may result from that 2-level residual codec already optimized PQ quantized codes once and there is less room for improvement. Overall, CSST still improves search accuracy for L&C, while keeping high search speed.

4.4 Ablation study

In this section, we conduct ablation study experiments on lightweight designs, random projection initialization and the proposed INPR loss. Experimental results are listed in Tables 4 and 5 . From results in Table 4, we can see that lightweight designs improve encoding speed and save GPU usage without sacrificing accuracy. Results in Table 5 show that using random projection units to initialize projection matrices and adopting the proposed INPR loss are beneficial for final accuracy. The best accuracy is achieved by employing both.

Table 6. Comparison experiments. Compression factor = 4.

Methods	GIST1M						Deep1M					
	HNSW			Brute force			HNSW			Brute force		
	1@1	1@5	1@10	1@1	1@5	1@10	1@1	1@5	1@10	1@1	1@5	1@10
SRP	24.9	49.4	60.7	24.8	49.7	60.9	17.3	35.6	44.4	17.3	35.6	44.4
MLP	47.9	73.8	84.2	47.9	73.8	84.1	48.3	77.6	89.1	48.4	77.6	88.9
VAE	49.2	77.0	86.0	49.3	76.8	85.8	50.0	81.9	90.2	50.0	81.9	90.2
Catalyst		-	-	-	-	-	57.9	89.6	92.1	57.9	89.6	92.1
CSST	**80.1**	**98.4**	**99.8**	**80.9**	**98.3**	**99.7**	**67.3**	**94.9**	**98.9**	**67.3**	**94.9**	**98.9**

4.5 Comparison with other compression methods

Previous experiments have shown that using compression feature vectors learned by our proposed CSST speeds up indexing without sacrificing accuracy. Besides our CSST, there are other compression methods. In this section, we compare the proposed models with other methods to evaluate its efficiency.

Here, we employed five comparison methods, including one traditional methods and four network based learning methods. Table 6 presents comparison results. Compared with other four compression methods, our proposed CSST achieves the highest accuracy. For GIST1M, using CSST and HNSW achieves 80.1% recall 1@1, 30.9 percentage points higher that that of VAE. As is analysed in the section *the proposed method*, using single sparse random projection harms search accuracy seriously. Both VAE is classical and powerful compression method, but it focus on keeping information of the input feature instead of keeping the neighborhood structure, which is not aligned with the requirement of ANNS. They are outperformed by our proposed CSST. On Deep1M, the proposed CSST also achieves better performance, even compared with the most recent proposed method Catalyst.

5 Discussion

In this work, we have proposed a generic feature compression method for ANNS problem. The proposed method consists of a compression network (CSST) which combines traditional projection function and transformer model, and an inhomogeneous neighborhood relationship preserving (INRP) loss which is aligned with the characteristic of ANNS. The proposed method can be generalized to most ANNS methods. It speeds up indexing speed to 2× to 4 × its original speed without hurting accuracy for graph based methods. It improves recalls by several or even a dozen percentage points for PQ related methods.

Acknowledgements

B. Tang's participation was in part supported by the National Natural Science Foundations of China (U1813215)

References

1. Subramanya, S.J., Kadekodi, R., Krishaswamy, R., Simhadri, H.V.: Diskann: Fast accurate billion-point nearest neighbor search on a single node. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. (2019) 13766–13776
2. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence **33**(1) (2010) 117–128
3. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE transactions on pattern analysis and machine intelligence **42**(4) (2018) 824–836
4. Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. arXiv preprint arXiv:1707.00143 (2017)
5. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemometrics and intelligent laboratory systems **2**(1-3) (1987) 37–52
6. Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., Carin, L.: Variational autoencoder for deep learning of images, labels and captions. Advances in neural information processing systems **29** (2016) 2352–2360
7. Li, P., Hastie, T.J., Church, K.W.: Very sparse random projections. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. (2006) 287–296
8. Douze, M., Sablayrolles, A., Jégou, H.: Link and code: Fast indexing with graphs and compact regression codes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2018) 3646–3654
9. Silpa-Anan, C., Hartley, R.: Optimised kd-trees for fast image descriptor matching. In: 2008 IEEE Conference on Computer Vision and Pattern Recognition, IEEE (2008) 1–8
10. Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: Proceedings of the forty-seventh annual ACM symposium on Theory of computing. (2015) 793–801
11. Ren, J., Zhang, M., Li, D.: Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory. Advances in Neural Information Processing Systems (2020)
12. Hyvärinen, A., Oja, E.: Independent component analysis: algorithms and applications. Neural networks **13**(4-5) (2000) 411–430
13. Achlioptas, D.: Database-friendly random projections: Johnson-lindenstrauss with binary coins. Journal of computer and System Sciences **66**(4) (2003) 671–687
14. Johnson, W.B., Lindenstrauss, J.: Extensions of lipschitz mappings into a hilbert space 26. Contemporary mathematics **26** (1984)
15. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science **313**(5786) (2006) 504–507
16. Sablayrolles, A., Douze, M., Schmid, C., Jégou, H.: Spreading vectors for similarity search. In: ICLR. (2019)
17. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. (2017) 5998–6008
18. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)

19. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. In: International Conference on Machine Learning, PMLR (2021) 10347–10357
20. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. arXiv preprint arXiv:2103.14030 (2021)
21. Graham, B., El-Nouby, A., Touvron, H., Stock, P., Joulin, A., Jégou, H., Douze, M.: Levit: a vision transformer in convnet's clothing for faster inference. arXiv preprint arXiv:2104.01136 (2021)
22. Chen, G., Chen, P., Shi, Y., Hsieh, C.Y., Liao, B., Zhang, S.: Rethinking the usage of batch normalization and dropout in the training of deep neural networks. arXiv preprint arXiv:1905.05928 (2019)
23. Zhuang, B., Shen, C., Tan, M., Liu, L., Reid, I.: Structured binary neural networks for accurate image classification and semantic segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. (2019) 413–422
24. Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., Kumar, S.: Accelerating large-scale inference with anisotropic vector quantization. In: International Conference on Machine Learning, PMLR (2020) 3887–3896
25. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. International journal of computer vision **42**(3) (2001) 145–175
26. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012) 1097–1105
27. Muja, M., Lowe, D.G.: Scalable nearest neighbor algorithms for high dimensional data. IEEE transactions on pattern analysis and machine intelligence **36**(11) (2014) 2227–2240
28. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE transactions on pattern analysis and machine intelligence **35**(12) (2012) 2916–2929
29. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)