Using Autoencoders to improve Nearest Neighbor Search on large Datasets*

Igor Buyanov, Vasiliy Yadrintsev, and Ilya Sochenkov

FRC CSC RAS

Abstract. In this work, we explore the applicability of autoencoders as a vector compressor in the pipeline of approximate nearest neighbor search. We conduct extensive tests with several autoencoders and indices on several large-scale datasets. The results show that while none of the combinations of autoencoders and index can completely outperform the pure solutions, it might be useful in some cases. We also find some empirical connections with the optimal hidden layer dimension and intrinsic dimensionality of the datasets.[1]

Keywords: Approximate neighborhood search · Autoencoder · Large scale dataset

1 Introduction

In today's information age, the volume of data available on the internet is growing exponentially. This growth has created a need for effective methods of information retrieval that can handle the sheer amount of data available. Information retrieval (IR) is the process of searching for and retrieving information relevant to a user's needs. It involves both retrieving and filtering large amounts of data to present only the most relevant results to the user. This is especially crucial in fields such as academic research, where the accuracy and relevance of retrieved information can make a significant impact on the success of a project. The exponential growth of data has made it increasingly difficult to find relevant information amidst the sea of irrelevant data.

One of the most active areas of research in information retrieval is the problem of approximate nearest neighbor (ANN) search. This problem arises in a wide range of applications, from recommendation systems to computer vision, where the goal is to find the nearest neighbors to a query point in a large database of vectors. Exact nearest neighbor search algorithms can be computationally expensive and impractical for large-scale datasets, leading researchers to investigate approximate algorithms that can efficiently return an approximate solution with reasonable accuracy. These methods typically involve building a data structure, such as a tree or graph, to efficiently prune the search space and avoid examining

* Supported by organization Foundation for Assistance to Small Innovative Enterprises.

[1] Email for correspondence: buyanov.igor.o@yandex.ru

all possible candidates. Another possible direction is to explore various vector compression methods.

Specifically, the problem with large-scale datasets arises from the fact that the common vector dimension ranges from one hundred to a thousand. This leads to big storage space demands as well as RAM to operate with them. Although graphical processing units allow fast computations over high-dimension matrices, it would be much better to find a way to compress the vector representation to save resources.

In this work, we investigate autoencoder networks as a compressor for nearest neighbor search. Probably, they seem to be irrelevant due to their high computation cost. Nevertheless, we decide to explore what benefits autoencoders might bring to us in the role of vector compressor in the neighbor search pipeline.

As a contribution of this paper, we test our hypothesis of using neural autoencoders as vector compression methods by conducting an extensive test of various ANN methods with several autoencoders on six datasets. We also found out that there exists a compression boundary crossing that causes quick performance decay independent of autoencoder type on some search algorithms that shows some quality level before this boundary, we experimentally bound it with intrinsic dimension of the dataset.

2 Related work

The first search algorithms were tree-based methods, such as KD-trees [3] and Ball trees [14], partitioning the data into a tree structure for efficient searching. These methods have been widely used for ANN search in low- to moderate-dimensional spaces, where the data can be efficiently partitioned into a tree structure. Several variants of tree-based methods have been proposed to improve their performance, including the use of randomized KD-trees and hierarchical clustering. These methods are efficient and effective for lower-dimensional data.

As was mentioned, there are a lot of studios about compression methods. One popular method is product quantization (PQ) [8]. It involves dividing the data into subspaces and then independently quantizing each subspace. This method has been shown to be effective for high-dimensional data and has been used in applications such as image retrieval and speech recognition.

Another big branch of algorithms is graph-based. For example, navigable small world (NSW) [1] graphs are a type of graph-based method for ANN search that uses a small-world network structure to efficiently navigate the data. NSW graphs maintain a balance between local and global connections, which allows them to efficiently navigate the data for ANN search. This method has been shown to be effective for high-dimensional data and has been used in applications such as image retrieval and text search. Hierarchical navigable small world graphs (HNSW) [12] are an extension of NSW graphs that use a hierarchical structure to further improve the efficiency of ANN search. HNSW graphs divide the data into multiple levels, each of which contains a different graph structure. This allows for efficient navigation of the data at different scales, improving the overall

efficiency of ANN search. HNSW graphs have been shown to be effective for high-dimensional data and have been used in applications such as image retrieval and recommendation systems.

Navigable spread-out graphs (NSG) [5] is another type of graph-based method for ANN search that uses a spread-out network structure to efficiently navigate the data. NSG graphs maintain a balance between exploration and exploitation of the data, which allows them to efficiently search for nearest neighbors in high-dimensional spaces. This method has been shown to be effective for a wide range of applications, including image retrieval, text search, and recommendation systems.

Another line of work which is adjacent to ours in some way is to make embeddings that would be especially good for ANN search. For example, in this work [16], authors propose a learnable embedding indexing layer that composes of several techniques. With presented specific loss regularization term, this layer can be plugged in any deep retrieval model that allows jointly learn embeddings and its index.

3 Problem formulation

3.1 Nearest neighbor search

Nearest neighbor search is a fundamental problem in many machine learning and data mining applications, where the goal is to find the k nearest neighbors of a given query point in a large dataset. Formally, let $X = \{x_1, x_2, ..., x_n\}$ be a dataset of n d-dimensional points, and let q be a query point in d-dimensional space. The k nearest neighbors of q in X are defined as the k points x_i in X that minimize the distance function $dist(q, x_i)$:

$$min_{i=1,2,..,k} dist(q, x_i) \tag{1}$$

where $dist(q, x_i)$ is a distance function that measures the similarity or dissimilarity between the query point q and the database point x_i, which is the Euclidean metric in our case.

The brute-force approach of computing the distances between the query point and every point in the dataset has a time complexity of $O(nd)$, which is impractical for large datasets. Approximate nearest neighbor (ANN) algorithms aim to find an approximate set of k nearest neighbors that are close to the true nearest neighbors, while significantly reducing the computational cost:

$$dist(q, x_i) \leq (1 + \epsilon) dist(q, x_i^*) \tag{2}$$

where ϵ is a degree of the relaxation and x_i^* are the approximately closest examples.

The accuracy of ANN algorithms is typically measured in terms of recall, which is the proportion of true nearest neighbors that are found by the algorithm. Let $NN(q)$ denote the set of true k nearest neighbors of q in X. The recall $R@k$

of an ANN algorithm that returns a set of k points $S(q)$ for a given query point q is defined as:

$$R@k = \frac{|S(q) \cap NN(q)}{|NN(q)|} \tag{3}$$

where $|S(q) \cap NN(q)|$ is the cardinality of the intersection of $S(q)$ and $NN(q)$, and $|NN(q)|$ is the cardinality of $NN(q)$.

The efficiency of ANN algorithms is measured in terms of the query time, which is the time required to find the k nearest neighbors for a given query point. The goal of ANN algorithms is to achieve a good balance between accuracy and efficiency, by trading off between the quality of the approximate solution and the computational cost of finding it.

3.2 Using autoencoders for dataset compression

Neural autoencoders [15] is a type of neural network that is used for unsupervised learning. They consist of an encoder network and decoder networks, and the goal of the network is to reconstruct its input at its output. The encoder network maps the input data to a lower-dimensional latent space, while the decoder network maps the latent space back to the input space. We hypothesize that we can use the encoder to efficiently reduce the dimensionality of vectorized datasets.

Formally, let $\mathbf{x}$ be the input data and $\mathbf{h}$ be the latent representation in the encoder network. The encoder network maps $\mathbf{x}$ to $\mathbf{h}$ using a function f as $\mathbf{h} = f(\mathbf{x})$. Similarly, let $\mathbf{y}$ be the output data and $\mathbf{g}$ be the function that maps $\mathbf{h}$ back to $\mathbf{y}$ in the decoder network, so we have $\mathbf{y} = \mathbf{h}$. The goal of the autoencoder is to minimize the reconstruction error, which is the difference between the input $\mathbf{x}$ and the output $\mathbf{y}$. One common way to measure the reconstruction error is the mean-squared error (MSE):

$$L(x, y) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{y}_i)^2 \tag{4}$$

where n is the number of data points in the input.

The main property that should have an autoencoder is to preserve the relations between points while transforming the vector space. In our experiments, we use several modifications of the described autoencoder, which we will refer to as the **Vanilla autoencoder**, that we believe preserves the point relations better than the Vanilla one.

We start with **The Neighborhood Reconstructing Autoencoders** [11] the idea of which is to employ an approximation of the decoder function with the help of local graphs that captures the local geometry of the data distribution. That will help to make the autoencoder more robust to the overfitting and connectivity issues. This method implies building a fully connected neighborhood graph of the data that will be used in the loss function

$$L = \sum_{i} \sum_{x \in N(x_i)} ||x - f_\theta(g_\phi(x); g_\phi(x_i))|| \tag{5}$$

where $N(x_i)$ is the set of neighborhood points of x_i and $f_\theta(.; g_\phi(x_i))$ is the approximation of the decoder $f_\phi(x)$ about the encoded point $g_\phi(x)$.

Another modification is **Deep Clustering with Convolutional Autoencoders** (DCEC) [7] where the Vanilla architecture was modified by adding a clustering layer after the encoder. Adding this layer is expected to learn clusters of the data which we believe allows doing more fine-grained separation of the vector space leading to better performance of non-graph search algorithms. Mathematically, the clustering layer represents cluster centers as trainable weights and maps each hidden point z_i into a soft label q_i that is calculated as follows:

$$q_{ij} = \frac{(1 + ||z_i - \mu_i||^2)^{(-1)}}{\sum_j (1 + ||z_j - \mu_j||^2)^{(-1)}} \tag{6}$$

The cluster loss is defined as

$$L_c = KL(P||Q) \tag{7}$$

with P as target distribution. The authors propose the heuristic target distribution (see eq. 8) but generally, it can be any distribution that (1) can produce soft labels, (2) improve the cluster purity (3) pay attention to a high confident point and (4) normalize the contribution of centroids to prevent distortion of the embedding space. Finally, the clustering loss is added to a reconstruction loss with a hyperparameter γ that controls its influence

$$L = L_{rec} + \gamma L_c \tag{8}$$

Last but not least is **Hyperbolic Antoencoder** [13] where authors propose an autoencoder that works in a hyperbolic space. They use the Poincare Ball model that is defined as $B^n = \{x \in R^n : ||x||\}$ with the Riemannian metric tensor. The feature of this autoencoder lies in the distance metric between two points (see eq. 2) that makes some distances that would be close in Euclidean space exponentially large in hyperbolic space. That allows us to effectively model complex networks and tree-like structures.

4 Experiments

4.1 The datasets

To make our test broad enough, we conduct our experiments on six datasets, the summary statistics of which are in Table 1. Their descriptions are as follows:

- SIFT-Small Dataset [8]: This is a smaller version of the SIFT-1M dataset and consists of 10,000 SIFT descriptors extracted from a set of images.
- SIFT-1M Dataset: This dataset consists of 1 million SIFT descriptors extracted from a set of images and is commonly used as a benchmark for evaluating the performance of ANN search algorithms.

Table 1. The summary statistics of datasets. BPS stands for Bytes per Component.

Name	Dimension	Vec cnt	BPS	Query vec cnt	Learn vec cnt	Orig size
siftsmall	128	10,000	4	100	25,000	5,120,000
sift1m	128	1,000,000	4	10000	100,000	512,000,000
gist	960	1,000,000	4	1000	500,000	3,840,000,000
sift1b	128	1,000,000,000	1	10000	100,000,000	128,000,000,000
OID small	512	1,012,239	4	1012	101,223	2,073,065,472
OID	512	8,390,600	4	83906	843,480	17,183,948,800
wiki small	1024	576,940	4	576	57,694	2,363,146,240
wiki	1024	87,817,400	4	87783	1,317,261	359,700,070,400

- SIFT-1B Dataset: This is an even larger dataset than SIFT-1M, consisting of 1 billion SIFT descriptors extracted from a set of images. It is also commonly used as a benchmark for evaluating the performance of ANN search algorithms.
- GIST Dataset [8]: This is a dataset of 1 million GIST descriptors. Another commonly used benchmark for ANN search algorithms.
- Wiki-LASER Dataset: This is the dataset that we created from Wikipedia articles. We split each text into sentences and encoded them with language agnostic sentence representations (LASER) technique [6].
- Open Images Dataset [10]: This is a large-scale dataset of images that contains over 9 million images with more than 30 million bounding boxes. The dataset is designed to facilitate object detection, classification, and visual relationship detection tasks. We use ResNet18 to get embeddings from images that were normalized and transformed in one way.

4.2 Test setting

We extensively test various pipelines that we compose of separate flags and components. We organize the results of only perspective combinations in tables for each particular dataset. Due to space limitations, we place them in Appendix A. Moreover, we have to divide tables into two parts. The first part consists of results and the second part describes some variables. The rows can be matched by the ID column. We use implementations of various indices from Faiss library [9]. We can divide our test case into two groups: the first group is where we use one autoencoder and then feed output vectors to a Faiss index and quantization combination, and the second group is where only Faiss components are used. As indices, we use Flat, inverted file index with product quantization (IVFPQ), HNSW, and NSG indices (index). As autoencoders (enc) we use all described variants: Vanilla, DCEC, HyperAE, NRAE. In addition, we experiment with **input embedding and output normalization (norm inp vect norm embs and norm out vect)**, **the hidden dimension (hidden dim)** of autoencoders, and **setting hidden layer (set l hidden)** dimension equals to the input dimension in autoencoders. They are represented as columns in the second table. If the column is not presented, that means that it has the same value in each

cell. By default, the absence means that action was not taken. Also, we report constants such as the batch size being 8, the number of train epochs being 5, nprobe parameter being 20. We compute several metrics such as different variants of recall that we encode as $n - Rm$ where n is the number of true nearest neighbors and m is the top relevant vectors returned by the index. We also compute **compression ratio (cr)**, **space-saving (ss)**, **on-disk index size (index size)**, **one vector search time (o-v-s)**, **overall search time (s)**, and **finally train and index time (t_i)**. All experiments were done in two stages. At first, we select a small subset of the dataset in order to find parameter combinations that work best. Next, we run several selected setups on the entire dataset and the results we have provided.

4.3 Choice of autoencoder dimension

An important question is how to choose the dimension of the hidden layer, which is equivalent to a compression rate. Unfortunately, there is no clear way to determine the best-hidden layer dimension that wouldn't hurt the representation while compressing the vector space as soon as possible.

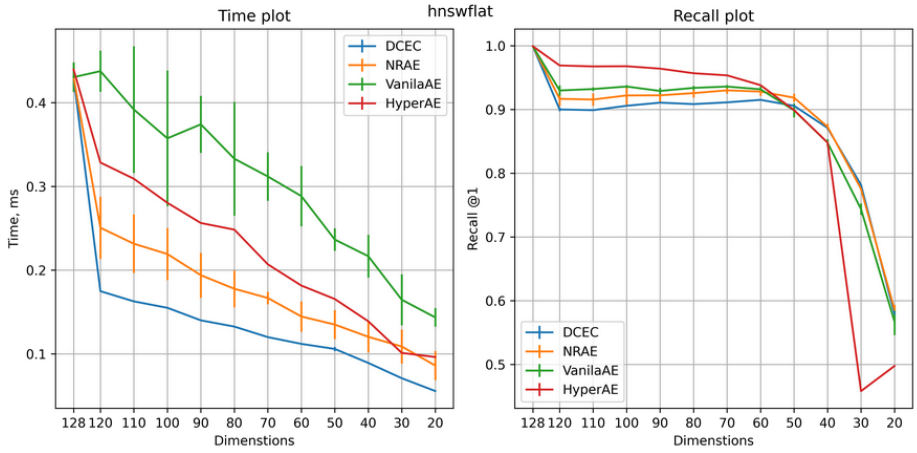

Fig. 1. Test HNSW with autoencoders. The left plot is an inference time. The right plot is a recall.

During our preliminary study, we found out that there exists a compression boundary crossing that causes quick performance decay independent of autoencoder type on some search algorithms that shows some quality level before this boundary, for example, HNSW on Fig 1. The study of this phenomenon leads to the conclusion that this boundary is an intrinsic dimension, which is defined as a dimension of the manifold to which the embedding space belongs [4]. Al-

ternatively, it can be thought of as a minimal number of dimensions that can represent the data.

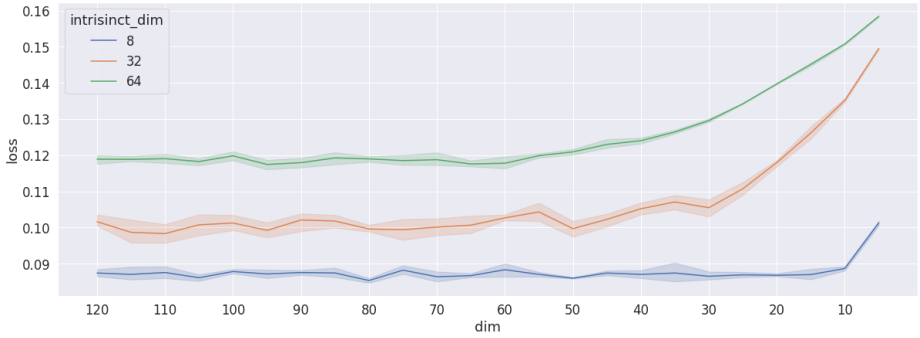

Fig. 2. The test of the Vanilla autoencoder loss behavior with several intrinsic dimensions of synthetic datasets.

Table 2. The results of intrinsic dimensionality estimation algorithm test on SIFT1M.

Alg. Name	Dimension	Time, s
Expected Simplex Skewness algorithm	25.4422	9324.320
Dimensionality from Angle and Norm Concentration algorithm	nan	265.773
Correlation Dimension	9.99273	420.920
Fisher Separability algorithm	3.11696	304.551
k Nearest Neighbors	3	695.634
Principal component analysis	10	0.880
Manifold-Adaptive Dimension Estimation algorithm	21.9273	1956.410
Minimum Neighbor Distance—Maximum Likelihood	1	255.691
Maximum Likelihood Estimation	0	256.540
Method Of Moments	19.5008	253.178
Tight Local Estimator	19.1451	288.200
Two Nearest Neighbors	10.4751	831.291

We experiment with synthetically generated data with the known intrinsic dimension with the "Nonlinear manifold" mode. We use the scikit-dimension package[2] for a generation. Next, we train the Vanilla autoencoder on the train set and calculate the loss on the test set with the hidden layer ranging from source dimension to 10. As intrinsic dimensions, we test 8, 32, and 64. In Fig 2 we can see that loss starts increasing after the hidden layer becomes less than the intrinsic dimension. We also repeat this experiment with real data, in particular, with SIFT1M and Open Images Dataset. We found out that the

loss of the autoencoder starts increasing near the dimension when search quality starts decreasing.

As we experimentally convince that there is an optimal dimension to compress to, we examine all algorithms for intrinsic dimensionality estimation of SIFT1M that are implemented in the scikit-dimension package. Based on our empirical study, we know that the intrinsic dimension of SIFT1M is about 60. Unfortunately, all algorithms fail to provide a close to 60 value, see Table 2. In addition, some algorithms are very time-consuming. We leave the ways for automatic optimal dimension selection for future work.

5 Results

As we can clearly see, the graph-based search methods NSG and HNSW show near-perfect results in terms of all variants of recall for all datasets (Tables 3, 5, 7). The disadvantages of these methods are that they consume vectors as is and moreover add some memory overhead which can be seen from compression ratio and space-saving metrics. They are also fast at the inference stage and have medium speed on indexing.

Speaking of autoencoders, we see that, unfortunately, none of the combinations, where they are presented, show results that completely outperform the pure Faiss pipeline. So we can conclude that compressing the vector space autoencoders disturbs it too much. It leads that local relationship also disturbs, so points that were neighbors in the source space could be far away in a compressed space. Comparing only autoencoders, we can see that one autoencoder works better than others depending on the dataset, but we can highlight that HyperAE works better in many cases. In contrast, NRAE doesn't show a valuable result in any setting.

We also see that for Wiki and Open Images datasets, the pipelines show identical results in tern of 1-R@, so we can reduce the comparison to 10-R@. Here we can highlight that combination of Vanilla AE with Flat index is comparable with HNSW but provides a double reduction of size. Though, the inference time with Vanilla AE is much slower.

6 Conclusion

In this paper, we investigate the applicability of neural autoencoders as a vector compressor component in the neighborhood search pipeline. We consider the Vanilla autoencoder and its variants that better preserve space geometry while compressing with different index techniques.

We found that autoencoders weren't able to compress the source vector space in such a way that the resulting space accurately preserves the local relationship between vectors, though some of them do it better than others. During the main work, we also found some connections between the lower-bound compression of autoencoders and intrinsic dimensionality estimation. We show that the loss of the autoencoder could be a great signal that an intrinsic dimension has been

reached, whereas none of the tested estimation algorithms can handle such a high dimension.

Experiments have shown that by using encoders in some cases, it is possible to save a vector of a smaller dimension with confidence that the selected quality metric retains the same level. Thus, it can give an advantage, for example, in the problem of storing compressed vectors.

In future work, we will continue to search for the autoencoder that disturbs the space minimally while compressing the data as much as possible. We also will try to come up with how to automate the choice of autoencoder dimensionality.

Acknowledgements We thanks Foundation for Assistance to Small Innovative Enterprises for funding this work (Agreement №12GUKodIIS12-D7/72692).

References

1. Alexander, P., Malkov, Y., Andrey, L., Krylov, V.: Approximate nearest neighbor search small world approach (2011)
2. Bac, J., Mirkes, E.M., Gorban, A.N., Tyukin, I.Y., Zinovyev, A.Y.: Scikit-dimension: A python package for intrinsic dimension estimation. Entropy **23** (2021)
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**, 509–517 (1975)
4. Erba, V., Gherardi, M., Rotondo, P.: Intrinsic dimension estimation for locally undersampled data. Scientific Reports **9** (2019)
5. Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. Proc. VLDB Endow. **12**, 461–474 (2017)
6. Gong, H., Schwenk, H.: Multimodal and multilingual embeddings for large-scale speech mining. In: Neural Information Processing Systems (2021)
7. Guo, X., Liu, X., Zhu, E., Yin, J.: Deep clustering with convolutional autoencoders. In: International Conference on Neural Information Processing (2017)
8. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**, 117–128 (2011)
9. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with gpus. IEEE Transactions on Big Data **7**, 535–547 (2017)
10. Kuznetsova, A., Rom, H., Alldrin, N.G., Uijlings, J.R.R., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., Duerig, T., Ferrari, V.: The open images dataset v4. International Journal of Computer Vision **128**, 1956–1981 (2018)
11. Lee, Y., Kwon, H., Park, F.C.: Neighborhood reconstructing autoencoders. In: Neural Information Processing Systems (2021)
12. 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**, 824–836 (2016)
13. Mirvakhabova, L., Frolov, E., Khrulkov, V., Oseledets, I., Tuzhilin, A.: Performance of hyperbolic geometry models on top-n recommendation tasks. Proceedings of the 14th ACM Conference on Recommender Systems (2020)
14. Omohundro, S.M.: Five balltree construction algorithms (2009)

15. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation (1986)
16. Zhang, H., Shen, H., Qiu, Y., Jiang, Y., Wang, S., Xu, S., Xiao, Y., Long, B., Yang, W.: Joint learning of deep retrieval model and product quantization based embedding index. Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (2021)

7 Appendix A

Table 3. SIFT1M main results. The first six columns are the recall results which formatted according to the Formula 3. For the rest columns, an explanation can be found in 4.2 in bold.

id	1-R100	1-R10	1-R1	10-R100	10-R10	100-R100	index_size	cr	ss	o-v s	s	t_i
0	0.56	0.33	0.16	0.38	0.15	0.18	256000045	2.0	0.5	0.0282	58.2	80.57
1	0.56	0.33	0.16	0.38	0.15	0.17	528129506	0.97	-0.03	0.0058	0.67	98.56
2	1.0	0.93	0.57	0.98	0.62	0.65	256000045	2.0	0.5	0.0259	57.75	289.5
3	1.0	0.94	0.58	0.98	0.63	0.67	256000045	2.0	0.5	0.0257	57.88	299.66
4	1.0	0.93	0.56	0.97	0.61	0.65	256000045	2.0	0.5	0.0283	57.87	108.23
5	0.98	0.82	0.41	0.92	0.49	0.56	24082612	21.26	0.95	0.0192	12.76	131.78
6	1.0	0.92	0.56	0.97	0.61	0.65	72133300	7.1	0.86	0.0257	17.92	190.7
7	1.0	0.93	0.56	0.97	0.61	0.65	256000045	2.0	0.5	0.0283	57.87	108.23
8	1.0	0.93	0.56	0.97	0.61	0.65	256000045	2.0	0.5	0.0283	57.87	108.23
9	0.99	0.87	0.44	0.96	0.54	0.62	24164532	21.19	0.95	0.0097	4.65	28.87
10	1.0	1.0	0.81	1.0	0.87	0.89	72264372	7.09	0.86	0.0099	4.75	99.95
11	1.0	1.0	0.81	1.0	0.85	0.85	72663732	7.05	0.86	0.0069	1.79	108.84
12	1.0	1.0	0.9	1.0	0.86	0.66	784129506	0.65	-0.53	0.007	0.74	31.54
13	1.0	1.0	0.99	1.0	0.99	0.94	596221684	0.86	-0.16	0.007	1.04	114.66

Table 4. SIFT1M parameter variations. Hidden dim is 64. The description of the parameters can be found in 4.2.

id	index	norm embs	encoder
0	Flat	false	dcec
1	HNSW32	false	dcec
2	Flat	false	hyperae
3	Flat	true	hyperae
4	Flat	false	vanae
5	IVF64,PQ16	false	vanae
6	IVF256,PQ64	false	vanae
7	Flat	false	vanae
8	Flat	false	vanae
9	IVF64,PQ16	false	-
10	IVF256,PQ64	false	-
11	IVF1024,PQ64	false	-
12	HNSW32	false	-
13	NSG32	false	-

Table 5. GIST main results. The first six columns are the recall results which formatted according to the Formula 3. For the rest columns, an explanation can be found in 4.2 in bold.

	1-R100	1-R10	1-R1	10-R100	10-R10	100-R100	index_size	cr	ss	o-v s	s	t_i
0	0.5	0.23	0.09	0.31	0.09	0.12	130465972	29.43	0.97	0.0167	1.58	860.27
1	0.0	0.0	0.0	0.0	0.0	0.0	1232129506	3.12	0.68	0.0061	0.58	2,111.7
2	0.0	0.0	0.0	0.0	0.0	0.0	1232129506	3.12	0.68	0.0063	0.59	2,100.57
3	0.0	0.0	0.0	0.0	0.0	0.0	1232129506	3.12	0.68	0.0061	0.58	2,047.03
4	0.0	0.0	0.0	0.0	0.0	0.0	1232129506	3.12	0.68	0.0072	0.57	1,969.21
5	0.07	0.04	0.03	0.03	0.01	0.02	2192129506	1.75	0.43	0.006	0.59	2,189.44
6	0.06	0.04	0.02	0.03	0.01	0.02	130465972	29.43	0.97	0.0179	1.68	2,086.03
7	0.06	0.04	0.03	0.03	0.01	0.02	250465972	15.33	0.93	0.029	3.08	2,112.66
8	0.5	0.23	0.09	0.31	0.09	0.12	130465972	29.43	0.97	0.0167	1.58	860.27
9	0.5	0.23	0.09	0.31	0.09	0.12	130465972	29.43	0.97	0.0167	1.58	860.27
10	0.5	0.23	0.09	0.31	0.09	0.12	130465972	29.43	0.97	0.0167	1.58	860.27
11	0.98	0.85	0.42	0.94	0.52	0.59	129968308	29.55	0.97	0.0155	1.47	190.0
12	1.0	0.98	0.63	1.0	0.69	0.69	252923572	15.18	0.93	0.0114	1.15	454.85
13	1.0	0.99	0.67	1.0	0.6	0.43	4112129506	0.93	-0.07	0.0067	0.68	390.93
14	1.0	0.98	0.63	1.0	0.69	0.69	252923572	15.18	0.93	0.0114	1.15	454.85

Table 6. GIST parameter variations.The description of the parameters can be found in 4.2.

id	index	norm embs	enc	hidden dim	norm inp vect	norm out vect	set l hidden
0	IVF1024,PQ120x8	false	dcec	480	false	false	true
1	HNSW32	false	hyperae	240	false	false	false
2	HNSW32	false	hyperae	240	false	false	true
3	HNSW32	true	hyperae	240	true	true	false
4	HNSW32	true	hyperae	240	true	true	true
5	HNSW32	false	vanae	480	false	false	true
6	IVF1024,PQ120x8	false	vanae	480	false	false	true
7	IVF1024,PQ240x8	false	vanae	480	false	false	true
8	IVF1024,PQ120x8	false	dcec	480	false	false	true
9	IVF1024,PQ120x8	false	dcec	480	false	false	true
10	IVF1024,PQ120x8	false	dcec	480	false	false	true
11	IVF256,PQ120x8	false	-	-	-	-	-
12	IVF1024,PQ240x8	false	-	-	-	-	-
13	HNSW32	false	-	-	-	-	-
14	IVF1024,PQ240x8	false	-	-	-	-	-

Table 7. Open Image Dataset main results. The first six columns are the recall results which formatted according to the Formula 3. For the rest columns, an explanation can be found in 4.2 in bold.

	1-R100	1-R10	1-R1	10-R100	10-R10	100-R100	index_size	cr	ss	o-v s	s	t_i
0	0.99	0.99	0.98	0.83	0.54	0.45	10937995050	1.57	0.36	0.0147	4.79	4,096.41
1	0.99	0.99	0.98	0.83	0.54	0.45	10937995050	1.57	0.36	0.0157	4.83	4,111.58
2	1.0	1.0	1.0	0.8	0.49	0.47	609867876	28.18	0.96	0.0163	75.98	3,462.03
3	1.0	1.0	1.0	0.8	0.49	0.47	609867876	28.18	0.96	0.0163	75.8	3,484.63
4	1.0	1.0	1.0	0.81	0.51	0.48	612096100	28.07	0.96	0.0163	76.52	4,180.76
5	1.0	1.0	1.0	0.81	0.5	0.48	612096100	28.07	0.96	0.0165	76.81	4,178.86
6	0.99	0.99	0.98	0.83	0.54	0.45	10937995050	1.57	0.36	0.0147	4.79	4,096.41
7	0.99	0.99	0.98	0.83	0.54	0.45	10937995050	1.57	0.36	0.0157	4.83	4,111.58
8	1.0	0.99	0.92	0.99	0.79	0.58	19579512618	0.88	-0.14	0.0171	8.2	1,731.1
9	1.0	1.0	1.0	0.95	0.58	0.57	616552548	27.87	0.96	0.009	20.54	1,487.1
10	1.0	0.99	0.92	0.99	0.79	0.58	19579512618	0.88	-0.14	0.0171	8.2	1,731.1

Table 8. Open Image Dataset parameter variations. The description of the parameters can be found in 4.2.

id	index	enc	hidden dim	set l hidden
0	HNSW32	hyperae	256	false
1	HNSW32	hyperae	256	true
2	IVF4096,PQ64	hyperae	128	false
3	IVF4096,PQ64	hyperae	128	true
4	IVF4096,PQ64	hyperae	256	false
5	IVF4096,PQ64	hyperae	256	true
6	HNSW32	hyperae	256	false
7	HNSW32	hyperae	256	true
8	HNSW32	-	-	-
9	IVF4096,PQ64	-	-	-
10	HNSW32	-	-	-

Table 9. Wiki-LASER main results. The first six columns are the recall results which formatted according to the Formula 3. For the rest columns, an explanation can be found in 4.2 in bold.

	1-R100	1-R10	1-R1	10-R100	10-R10	100-R100	index_size	cr	ss	o-v s	s	t_i
0	0.99	0.98	0.97	0.44	0.34	0.24	6340197076	56.73	0.98	0.0412	268.86	22,424.1
1	0.99	0.98	0.97	0.33	0.3	0.19	6340197076	56.73	0.98	0.0427	263.97	21,576.6
2	0.99	0.98	0.97	0.44	0.34	0.24	6340197076	56.73	0.98	0.0412	268.86	22,424.1
3	0.99	0.98	0.97	0.87	0.58	0.52	6391315156	56.28	0.98	0.024	103.61	62,943.41

Table 10. Wiki-LASER parameter variations. Hidden dim is 256. The description of the parameters can be found in 4.2.

id	index	enc	set 1 hidden
0	IVF16384,PQ64	dcec	true
1	IVF16384,PQ64	dcec	false
2	IVF16384,PQ64	dcec	true
3	IVF16384,PQ64	-	-

8 Reviewer remarks

Reviewer 1:

1. Need to add authors and their affiliations.
   (a) Added.
2. Need to number the formulas.
   (a) Added.
3. In the formal representation of the loss function in the description of Neighborhood Reconstructing Autoencoders (page 4), in the formula, the authors used f_theta, but in the description below, f with a tilde.
   (a) Fixed.
4. The q formula (page 5, Deep Clustering with Convolutional Autoencoders) needs to fix brackets.
   (a) Fixed.
5. In the results section, will be better to reveal the names of abbreviations - cs and ss metrics.
   (a) Fixed.

Reviewer 2:

1. Formally (by mathematical notation), the authors consider the problem of exact search for the nearest neighbor, which does not correspond to the further presentation.
   (a) Added clarification.
2. Bibliographic references are also inaccurately formatted.
   (a) Don't understand in which way. We used officially provided latex template, and only we did is a collection of bibliographies as BibTeX file and insertion of references in right places by latex command.
3. All abbreviations must be explicitly spelled out (IVEPQ, Table 2).
   (a) Done.
4. The authors mention the new Wiki-LASER dataset, but refer to it differently in the tables.
   (a) Fixed.
5. Reference to Table 1 in the text is indicated as Table 4.1
   (a) Fixed

Reviewer 3:

1. The experiment results are hard to read. You provided many big tables without descriptive captions or any markup.
   (a) Added references where the description of the column headers can be found.
2. Plots are not monotonic over dimension. This raises questions about the correctness of the results.
   (a) We use the fact that the number of autoencoders show extreme decreasing and not the behavior of each autoencoder in isolation. It's hard to imagine that all four autoencoders show almost the same behavior in the same region due to the error. The same goes for the loss behavior experiment. We hypothesize that the fluctuations can be attributed to the stochastic nature of the learning process, where in some cases the updated weights can locally be better than on the previous iteration.