Applied Intelligence (2021) 51:493–505
https://doi.org/10.1007/s10489-020-01797-y

Autoencoder-based unsupervised clustering and hashing

Bolin Zhang[1] · **Jiangbo Qian**[1]

Published online: 19 August 2020

Abstract
Faced with a large amount of data and high-dimensional data information in a database, the existing exact nearest neighbor retrieval methods cannot obtain ideal retrieval results within an acceptable retrieval time. Therefore, researchers have begun to focus on approximate nearest neighbor retrieval. Recently, the hashing-based approximate nearest neighbor retrieval method has attracted increasing attention because of its small storage space and high retrieval efficiency. The development of neural networks has also promoted progress in hash learning. However, these methods are mostly supervised. In practical applications, annotating large amounts of data is a very time-consuming and laborious task. Furthermore, efficiently using a large amount of unlabeled data for hash learning is challenging. In this paper, we create a new autoencoder variant to efficiently capture the features of high-dimensional data, and propose an unsupervised deep hashing method for large-scale data retrieval, named as Autoencoder-based Unsupervised Clustering and Hashing (AUCH). By constructing a hashing layer as a hidden layer of the autoencoder, hash learning is performed together with unsupervised clustering by minimizing the overall loss. AUCH can unify unsupervised clustering and retrieval tasks into a single learning model. In addition, the method can use a deep neural network to simultaneously learn feature representations, hashing functions and cluster assignments. Experimental results on standard datasets indicate that AUCH achieves competitive results compared to state-of-the-art models for retrieval and clustering tasks.

Keywords Information retrieval · Unsupervised hashing · Deep learning · Clustering

1 Introduction

In the era of big data, content-based multimedia data retrieval has become increasingly difficult. Simultaneously, efficient indexing and search algorithms are receiving increasing attention [6, 8, 29, 37, 55, 58]. However, when the amount of data is very large, the speed of exact nearest neighbor searches will drop dramatically. To balance retrieval performance and computational cost, approximate nearest neighbor (ANN) [1] approach has begun to gain considerable attention. The representative ANN solutions include two kinds of methods: tree-based [18] and hashing-based methods [16, 34, 55]. From an application perspective, images or texts can often be represented by high-dimensional features, such as SIFT-based bag-of-words features [35] and deep features. It is well known that traditional tree-based methods have high feature space dimensions, and in many cases, their performance has been theoretically shown to be comparable to that of linear scans [45]. Therefore, hashing-based large-scale ANN search methods have become a focus of research.

Hashing, which is a method of transforming high-dimensional feature vectors into compact and informative binary codes using mapping functions, has achieved remarkable success in quickly retrieving data [15, 19]. Recently, the rapid development in convolutional neural networks (CNNs) has contributed to significant progress in ANN research [30–32, 50]. In particular, compared to supervised hashing methods, since unsupervised hashing methods do not require labeled training data, they have received increasing attention, which has broadened their application prospects. Stacked restricted Boltzmann machines (RBMs) were first used to encode hash codes for unsupervised hashing [31, 42]. However, the shortcomings of high complexity and the need for pretraining make RBMs fundamentally difficult to implement. More recently, many studies have made remarkable achievements in hash learning using deep

✉ Jiangbo Qian
qianjiangbo@nbu.edu.cn

Bolin Zhang
onlyoufxx@qq.com

[1] Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, China

learning frameworks, especially generative adversarial networks [2, 36, 44, 55]. However, most of these methods can perform hash learning only for retrieval tasks, which are relatively simple. As we know, combining similar data can greatly improve the efficiency of data retrieval, especially in big data systems. Hence, our goal is to unify unsupervised clustering and retrieval tasks into a single learning model without degradation of performance.

In this paper, we propose a novel unsupervised deep hashing method, named Autoencoder-based Unsupervised Clustering and Hashing (AUCH). Specifically, we construct the hashing function as a latent layer with K units between the encoder and decoder in an autoencoder. We first pretrain a stacked denoising autoencoder and then perform feature representation extraction, hashing learning and cluster assignment simultaneously based on an overall loss function. Finally, we remove the decoder and keep the encoder. By feeding visual data into the network, we can obtain compact hash codes that preserve the structural similarity between input data. The main contributions are outlined as follows:

- AUCH is an unsupervised hashing approach that makes full use of the characteristics of autoencoders, unifies clustering and retrieval tasks in a single learning model, and jointly learns feature representations, hashing functions and clustering assignments from input data.
- For the retrieval and clustering tasks, we design an overall loss function. By optimizing this loss function, we can obtain efficient feature representations, compact hash codes and outstanding clustering performance.
- Extensive experiments on general datasets, including several image datasets, verify that AUCH can achieve better performance than other unsupervised hashing and clustering methods.

2 Related work

2.1 Hash learning

Based on whether the process of hash learning focuses on the data themselves, hashing methods can be divided into data-independent hashing methods and data-dependent hashing methods. At the same time, based on whether deep learning is employed for hash learning, hashing methods can be divided into traditional unsupervised hashing methods and deep unsupervised hashing methods.

2.1.1 Data-Independent hashing

The seminal LSH method presented in [1] represents groundbreaking work on locality-sensitive hashing; it maps similar data to similar hash codes in Hamming space. LSH usually requires longer hash bits to achieve better retrieval performance since it is a data-independent method. By contrast, data-dependent hashing methods [6, 15] can learn the distribution of the data themselves, and usually, shorter hash codes are needed to achieve better performance. In practice, data-dependent hashing is more valuable.

2.1.2 Data-Dependent hashing

For data-dependent hashing methods, several studies [16, 54] have been performed in a supervised manner, which usually relies on label information. The process of hash learning is guided by extracting the semantic information of the labels, which can usually lead to good retrieval performance. However, manually annotating data is a time-consuming and labor-intensive task, which prevents these methods from being widely used. To improve the utilization of unlabeled data, the study of unsupervised hashing methods is important.

2.1.3 Traditional unsupervised hashing methods

Most traditional unsupervised hashing methods usually consist of two independent processes: feature learning and hash learning. Representative unsupervised traditional hashing methods include spectral hashing (SH) [49], anchor graph hashing (AGH) [33], iterative quantization (ITQ) [15], and stochastic generative hashing (SGH) [6]. In SH, spectral analysis is first performed on the original high-dimensional dataset, and the constraints are then relaxed to convert the problem into a dimension reduction problem for the Laplacian feature graph. SH introduces the concept of feature functions to address the problem of data outside the training dataset. Inspired by SH, AGH addresses the same optimization problem as SH but offers a novel process for solving the objective function, breaking away from the assumption that the data were sampled from a multidimensional uniform distribution to obtain the prior hypothesis. Using an approximate adjacency matrix instead of the adjacency matrix reduces the time complexity and gives the method broader applicability. In ITQ, PCA-based dimension reduction is first performed on the dataset, and the rotation matrix with the smallest quantization error is then found to obtain the binary code of the feature vector corresponding to the optimal rotation matrix. SGH is essentially a generation method. It uses the principle of the minimum description length to map data to compact hash codes while retainings the structural similarity of the original data. The learned hash codes can in turn, be used to regenerate the input data. However, these methods have an obvious weakness. They cannot learn hash codes and features at the same time, which prevents their further development in practice.

2.1.4 Deep unsupervised hashing methods

Recently, the rapid development of deep neural networks has led to significant developments in various areas, such as computer vision and pattern recognition [59]. Deep unsupervised hashing methods have also been proposed, which adopt deep architectures to extract features and perform hash mapping. Representative unsupervised deep hashing methods include deep binary descriptors (DeepBit) [32], semantic structure-based unsupervised deep hashing (SSDH) [55], binary generative adversarial networks (BGANs) [44], deep hashing (DH) [36] and DeepQuan [2]. DeepBit jointly performs feature learning and hash learning, and the learned hash codes preserve the similarity between the visual data. DH utilizes a deep neural network to seek multiple hierarchical nonlinear transformations to learn binary codes so that the nonlinear relationship among the samples can be well exploited. SSDH is based on the empirical study of deep feature statistics and the construction of a semantic structure to guide hash learning. In a BGAN, the input noise variable of a generative adversarial network (GAN) is restricted to be binary, and conditions are imposed on the features of each image. A BGAN can learn to obtain a hash code for each image and generate an image similar to the original image. Moreover, the DeepQuan model is essentially a deep autoencoder network, including an encoder to generate efficient hash codes and a decoder to reconstruct visual data.

2.2 Clustering algorithms

In recent years, the advent of the era of big data has led to the rapid development of machine learning technology. Cluster analysis, as a commonly used method in traditional machine learning algorithms, is widely favored due to its practicality, simplicity and efficiency. It has been successfully applied in many fields [3, 13, 14, 24, 39, 43, 46, 52, 56, 57, 60], such as document clustering, market segmentation, image segmentation, and feature learning. Clustering is also an important concept in data mining, where its core purpose is to find valuable information hidden in data objects. Recently, multiview clustering and subspace clustering methods have also undergone rapid development. Representative methods include GBS-KO [47], L3E-M2VC [61], and LRLER [9]. GBS-KO is based on the clustering of multiview data and an extension of the graph-based approach to multiview clustering. L3E-M2VC is a new multitask multiview clustering algorithm proposed to overcome certain limitations in real-world applications. LRLER is a locally linear embedding low-rank representation model proposed for the subspace clustering of data that achieves superior performance.

3 Proposed AUCH method

Specifically, our goal is to design an unsupervised hashing algorithm that exploits an autoencoder, which can efficiently capture the features of high-dimensional data, to obtain compact hash codes. First, let $\boldsymbol{X} = \{\boldsymbol{x_i}\}_{i=1}^{N}$ denote N items of visual data. For the hash codes to preserve the structural similarity of the input data, we need to learn a mapping function $F : \boldsymbol{X} \rightarrow \{-1, +1\}^{K \times N}$. This function maps visual data to their K-bit hash codes $\boldsymbol{B} = \{\boldsymbol{b_i}\} \in \{-1, +1\}^{K \times N}$, which reflects the structural similarity between the input data. Similar data are mapped to the same or similar hash codes.

3.1 Autoencoder with a hashing layer

The architecture of AUCH is illustrated in Fig. 1. The fundamental structure of AUCH is essentially a stacked denoising autoencoder.

The denoising autoencoder is based on an autoencoder. To prevent overfitting problems, noise is added to the input data (in the input layer of the network), which makes the learned autoencoder more robust. As a result, the generalization ability of the model is enhanced. The network consists of two parts: an encoder function $\boldsymbol{h_i^{(H)}} = E(\boldsymbol{x_i})$, and a decoder function $\boldsymbol{x_i}' = D(\boldsymbol{h_i^{(H)}})$ that produces a reconstruction. The denoising autoencoder minimizes the following objective function:

$$L = \sum_{i=1}^{N} \left\| \boldsymbol{x_i} - D(E(\tilde{\boldsymbol{x}}_{\boldsymbol{i}})) \right\|_2^2 \tag{1}$$

where $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$ represents the data $\boldsymbol{x_i}$ after noise has been added in a specific way.

The original autoencoder has eight fully connected layers. The first four layers (F_{1-4}) form the encoder and the last four layers (F_{5-8}) form the decoder. The ReLU function is used as the activation function for the network because it largely solves the vanishing gradient problem of the BP algorithm when optimizing deep neural networks. To incorporate hash learning into the autoencoder, we add a latent layer $\boldsymbol{H}$ (i.e., the hashing layer) with K units after layer F_4 (i.e., the last layer of the encoder) as illustrated in Fig. 1. This latent layer is fully connected to F_4 and uses tanh units, meaning that the activation values are between -1 and 1.

3.2 Pretraining a stacked denoising autoencoder

Given a sample $\boldsymbol{x_i}$, by feeding it into the network, we can obtain the output of F_4, denoted by $\boldsymbol{a_i^{(4)}} \in R^d$, where d is the number of neurons in F_4. Let $\boldsymbol{W^{(H)}} \in R^{d \times K}$ denote the weights between F_4 and the hashing layer, and

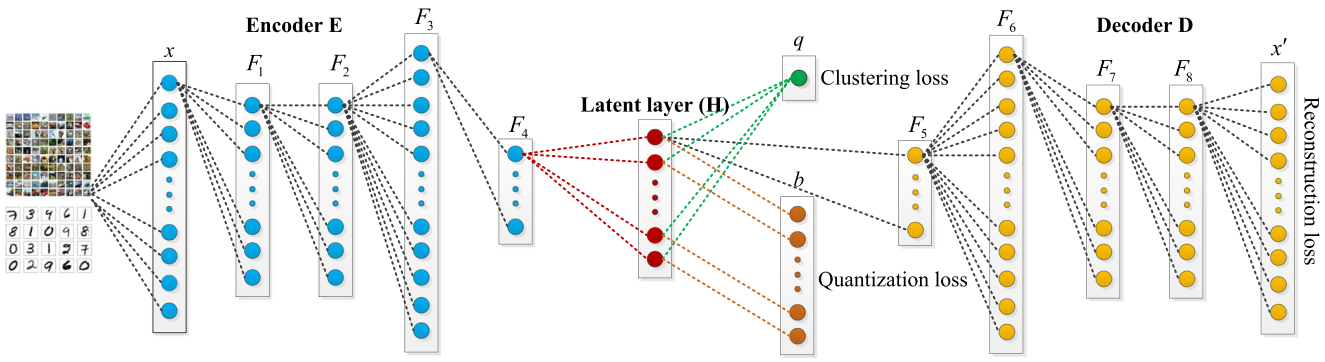


Fig. 1 An overview of our proposed Autoencoder-Based Unsupervised Clustering and Hashing (AUCH) method. AUCH performs hash learning by means of an additional hashing layer with K units, which is equivalent to hash mapping, between the encoder and decoder of an autoencoder. By optimizing an objective function defined over the clustering loss, reconstruction loss and quantization loss for the hash codes, AUCH jointly learns feature representations, hashing functions and clustering assignments from input data. The learned hash codes can preserve the structural relationship of visual data and enable efficient image retrieval. The discriminative features extracted can also enable superior clustering performance

let $\boldsymbol{b}^{(H)}$ denote the bias of the hashing layer. Then, the activation values of the units in $\boldsymbol{H}$ can be computed as $\boldsymbol{h}_i^{(H)} = \tanh(\boldsymbol{a}_i^{(4)}\boldsymbol{W}^{(H)} + \boldsymbol{b}^{(H)})$, where $\boldsymbol{h}_i^{(H)}$ is a K-dimensional vector and tanh(.) is the hyperbolic tangent function, defined as $\tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$, with z being a real value. The hash code generation function is given by

$$\boldsymbol{b_i} = sgn(tanh(\boldsymbol{a}_i^{(4)}\boldsymbol{W}^{(H)} + \boldsymbol{b}^{(H)})) \tag{2}$$

where $sgn(v) = -1$ if $v < 0$ and +1 otherwise; for a matrix or vector, $sgn(.)$ is applied in an elementwise manner.

Before clustering, we first pretrain a stacked denoising autoencoder for efficient feature extraction. After pretraining, all input data are embedded into the hashing space by the encoder mapping $\boldsymbol{h}_i^{(H)} = E(\boldsymbol{x_i})$. We then apply the k-means algorithm to $\{\boldsymbol{h}_i^{(H)}\}_{i=1}^N$ to obtain C initial cluster centers $\{\boldsymbol{\mu}_j\}_{j=1}^C$.

4 Overall loss

The dataset $\boldsymbol{X}$ consists of N samples, and each sample $\boldsymbol{x_i} \in R^e$ can be represented by an e-dimensional vector. There are a total of C cluster centers, where the value of C is known a priori. After the features of each $\boldsymbol{x_i}$ are clustered, the assigned cluster indices are represented by $s_i \in \{1, 2, ..., C\}$. Let us define the encoder mapping $E : \boldsymbol{x_i} \rightarrow \boldsymbol{h}_i^{(H)}$ and the decoder mapping $D : \boldsymbol{h}_i^{(H)} \rightarrow \boldsymbol{x_i}'$, where $\boldsymbol{x_i}'$ represents the reconstruction of $\boldsymbol{x_i}$.

For the clustering and hashing tasks, we need to find a good encoder mapping E to map the data $\boldsymbol{X} = \{\boldsymbol{x_i}\}_{i=1}^N$ to the hashing space $\{\boldsymbol{h}_i^{(H)}\}_{i=1}^N$. With regard to this, the overall loss function consists of three parts, namely, a clustering loss, a reconstruction loss and a quantization loss. The reconstruction loss guarantees that the autoencoder can efficiently learn representations of the data in an unsupervised manner. The clustering loss mainly operates on the hashing space to disperse the embedded points. The quantization loss causes the outputs of the hashing layer neurons to be close to either +1 or -1 to avoid the introduction of unnecessary errors when these outputs are quantized into binary codes. The loss function is defined as

$$L = \gamma L_c + \alpha L_r + \beta L_h \tag{3}$$

where L_c, L_r and L_h are the clustering loss, reconstruction loss and hashing quantization loss, respectively, and $\gamma > 0$, $\alpha > 0$ and $\beta > 0$ determine the relative importance of the clustering loss, reconstruction loss and quantization loss.

4.1 Clustering loss

The clustering loss is defined as

$$L_c = KL(P \,\|\, Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{4}$$

where KL is the KullbackLeibler divergence, which measures the degree of matching between the two probability distributions P and Q. The greater the difference between these two distributions is, the larger the KL divergence. We use Student's t-distribution to measure the distribution of soft labels, defined as Q. Student's t-distribution, proposed in [38], is used to measure the similarity between an embedded hashing point $\boldsymbol{h}_i^{(H)}$ and its cluster center $\boldsymbol{\mu}_j$. This similarity is expressed as

$$q_{ij} = \frac{\left(1 + \left\|\boldsymbol{h}_i^{(H)} - \boldsymbol{\mu}_j\right\|^2\right)^{-1}}{\sum_j \left(1 + \left\|\boldsymbol{h}_i^{(H)} - \boldsymbol{\mu}_j\right\|^2\right)^{-1}} \tag{5}$$

P is the target distribution generated by Q. p_{ij} in (4) is defined as

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_j (q_{ij}^2 / \sum_i q_{ij})} \tag{6}$$

We can see that the target distribution P is generated by Q, so we train the network in a self-training manner [40]. The clustering loss was proposed in [52].

4.2 Reconstruction loss

Following [17], after pretraining, we replace the stacked denoising autoencoder with an undercomplete autoencoder because the hashing space features of clean data are needed to perform clustering, meaning that it is unnecessary to add noise to the input data. Simultaneously, we must ensure that the decoder is unaffected and apply the clustering loss to the hashing space. In this way, the autoencoder can finally learn the most significant features of the data. The reconstruction loss is essentially the mean squared error (MSE):

$$L_r = \sum_{i=1}^{N} \left\| \boldsymbol{x_i} - D(\boldsymbol{h_i^{(H)}}) \right\|^2 \tag{7}$$

with $\boldsymbol{h_i^{(H)}} = E(\boldsymbol{x_i})$, where E and D are the encoder and decoder mappings, respectively.

4.3 Quantization loss

The hash codes need to preserve the similarity of the original data. In addition, it is necessary to ensure that the outputs of the hashing layer are close to +1 or -1 to reduce the quantization loss when transforming the hashing features into hash codes. Let $h_{i,k}^{(H)} (k = 1, ..., K)$ be the k-th element of the output vector $\boldsymbol{h_i^{(H)}}$ from the hashing layer. Because $h_{i,k}^{(H)}$ has already been activated by a tanh function, it is a float value and within the range of [-1, +1]. To make the output value as close as possible to the quantized value +1 or -1 and reduce the error loss caused by the quantization process, the quantization loss function is expressed as follows:

$$L_h = -\frac{1}{K} \sum_{i=1}^{N} \left\| \boldsymbol{h_i^{(H)}} \right\|^2 \tag{8}$$

with $\boldsymbol{h_i^{(H)}} = E(\boldsymbol{x_i})$, where E is the encoder mapping.

5 Optimization

During the training process, there are three kinds of parameters that need to be updated: the autoencoder's weights, the cluster centers and the target distribution. Minibatch stochastic gradient decent (SGD) and backpropagation are used in the optimization of (3).

5.1 Updating the autoencoder's weights and cluster centers

First, the target distribution P needs to remain constant, and the learning rate λ and a mini batch with m samples are given simultaneously. The cluster center $\boldsymbol{\mu_j}$ is updated as follows:

$$\boldsymbol{\mu_j} = \boldsymbol{\mu_j} - \frac{\lambda}{m} \frac{\partial L_c}{\partial \boldsymbol{\mu_j}} \tag{9}$$

where $\frac{\partial L_c}{\partial \boldsymbol{\mu_j}}$ is the gradient of L_c with respect to the cluster center $\boldsymbol{\mu_j}$, which can be computed as

$$\frac{\partial L_c}{\partial \boldsymbol{\mu_j}} = 2 \sum_{i=1}^{m} \left(1 + \left\| \boldsymbol{h_i^{(H)}} - \boldsymbol{\mu_j} \right\|^2 \right)^{-1} (q_{ij} - p_{ij})(\boldsymbol{h_i^{(H)}} - \boldsymbol{\mu_j}) \tag{10}$$

The gradient of L_c with respect to the embedded hashing point $\boldsymbol{h_i^{(H)}}$, denoted by $\frac{\partial L_c}{\partial \boldsymbol{h_i^{(H)}}}$, is computed as follows:

$$\frac{\partial L_c}{\partial \boldsymbol{h_i^{(H)}}} = 2 \sum_{j=1}^{C} \left(1 + \left\| \boldsymbol{h_i^{(H)}} - \boldsymbol{\mu_j} \right\|^2 \right)^{-1} (p_{ij} - q_{ij})(\boldsymbol{h_i^{(H)}} - \boldsymbol{\mu_j}) \tag{11}$$

The decoder's weights are updated as follows:

$$\boldsymbol{W'} = \boldsymbol{W'} - \frac{\lambda}{m} \sum_{i=1}^{m} \frac{\partial L_r}{\partial \boldsymbol{W'}} \tag{12}$$

The encoder's weights are updated as follows:

$$\boldsymbol{W} = \boldsymbol{W} - \frac{\lambda}{m} \sum_{i=1}^{m} \left(\gamma \frac{\partial L_r}{\partial \boldsymbol{W}} + \alpha \frac{\partial L_c}{\partial \boldsymbol{W}} + \beta \frac{\partial L_h}{\partial \boldsymbol{W}} \right) \tag{13}$$

5.2 Updating the target distribution

If we update the target distribution P in every iteration, this may lead to instability during the training process. In practice, we use all embedded hashing points to update P every T iterations. See (5) and (6) for the update rules. Once the target distribution P has been updated, the labels assigned to $\boldsymbol{x_i}$ will be updated accordingly as follows:

$$s_i = \arg\max_j q_{ij} \tag{14}$$

where q_{ij} is computed using (5). After two consecutive updates of the target distribution, if the percentage change in the label assignments for the image is less than a threshold ϕ, we will terminate training.

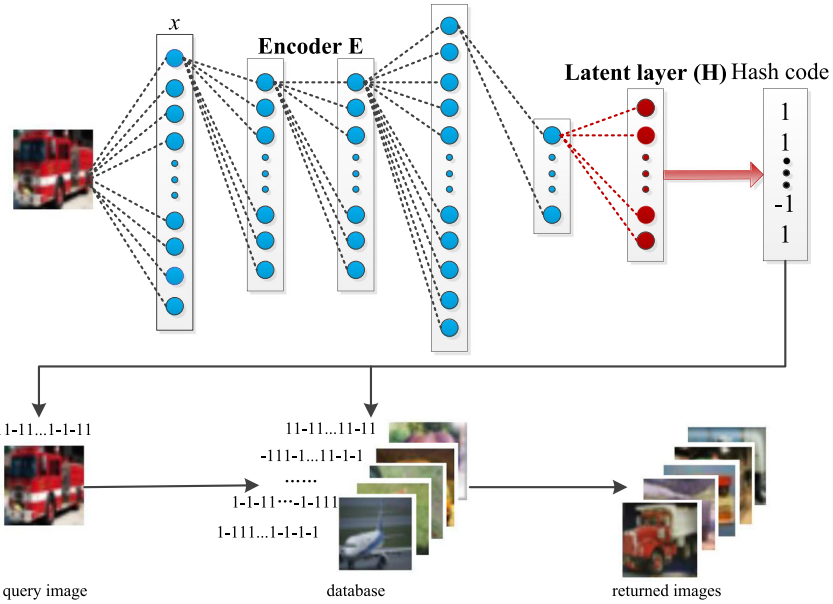


Fig. 2 Generation of hash codes for retrieval. After training, the query image and the images in the database are fed to the network, and the corresponding hash codes are obtained by quantizing the output of the hashing layer. For image retrieval, we calculate the Hamming distance between the hash codes of the query image and each image in the database and then return the images with the smallest Hamming distances as retrieval results

5.3 Generating binary codes

Figure 2 shows how to obtain the hash code of an image and use that hash code to retrieve similar images from a database. After training the autoencoder, we remove the decoder and keep the encoder. First, we send an image to the network, obtain the output of the hashing layer, and then quantize the output to obtain the hash code via (2). By calculating the Hamming distance between the hash codes of the query image and each data in the database, we select images from the database with small Hamming distances as the retrieval results. Our method is summarized in Algorithm 1.

Algorithm 1 AUCH.

Input: Input data: $\boldsymbol{X}$; Hashing bits: K; Number of clusters: C; Target distribution update interval: T; Termination threshold: ϕ; Maximum number of iterations: $MaxIter$.

Output: Autoencoder weights $\boldsymbol{W}$ and $\boldsymbol{W'}$; Hash codes $\boldsymbol{B}$; Cluster centers $\boldsymbol{\mu}$ and labels $\boldsymbol{s}$.

Initialize $\boldsymbol{\mu}$, $\boldsymbol{W'}$ and $\boldsymbol{W}$ as described in Section 3.2.
for $iter \in \{0, 1, ..., MaxIter\}$ **do**
  if $iter\%T == 0$ **then**
    Compute all embedded hashing points $\{\boldsymbol{h}_i^{(H)} = f_{\boldsymbol{W}}(\boldsymbol{x}_i)\}_{i=1}^{N}$.
    Update P using (5), (6) and $\{\boldsymbol{h}_i^{(H)}\}_{i=1}^{N}$.
    Save last label assignment:$\boldsymbol{s}_{old} = \boldsymbol{s}$.
    Compute new label assignments $\boldsymbol{s}$ via (14).
    if $sum(\boldsymbol{s}_{old} \neq \boldsymbol{s})/N < \phi$ **then**
      Terminate training.
    end if
  end if
  Choose a batch of samples $\boldsymbol{S} \in \boldsymbol{X}$.
  Update $\boldsymbol{\mu}$, $\boldsymbol{W'}$ and $\boldsymbol{W}$ via (10), (12) and (13).
end for
Remove the decoder and generate binary codes $\boldsymbol{B}$ via (2).

6 Experiments

To evaluate the performance of AUCH, we performed several experiments involving image retrieval and clustering tasks on multiple datasets.

6.1 Datasets

We compare our method with other unsupervised hashing methods for the image retrieval task on MNIST [28] and CIFAR-10 [26]. Furthermore, we compare the performance of AUCH and other clustering methods for the clustering task on the MNIST, CIFAR-10, USPS[23], STL-10 [5] and Fashion-MNIST [51] datasets. Table 1 shows the descriptions of all datasets used.

Table 1 Dataset descriptions

Dataset	#Samples	#Classes	#Dimensions
MNIST	70,000	10	1×28×28
USPS	9,298	10	1×16×16
CIFAR-10	60,000	10	3×32×32
STL-10	13,000	10	3×96×96
FASHION-10	70,000	10	1×28×28

Table 2 Optimal hyperparameters on different datasets

Coefficients	γ	α	β
MNIST	10^{-2}	1	10^{-4}
CIFAR-10-GIST	10^{-5}	1	10^{-5}
USPS	10^{-1}	1	10^{-2}
STL-10	10^{-3}	1	10^{-4}
FASHION-MNIST	10^{-2}	1	10^{-5}

6.2 Implementation details

Specifically, when performing experiments on the CIFAR-10 dataset, we used the methods considered for comparison to extract 512-D GIST [41] features of the raw images as inputs for training. These methods include ITQ [15], KMH [20], Spherical [22], SH [49], PCAH [48], LSH [7], AGH [33], DH [12, 36], UH-BNN [10], DeepQuan [2], and KNNH [21]. Let CIFAR-10-GIST denote the CIFAR-10 dataset after feature extraction. Then, following [2], we randomly selected 1,000 images (100 images per class) as the query image set, and the remaining 59,000 images as the training set.

The autoencoder is essentially composed of two fully connected multilayer perceptrons (MLPs), i.e., an encoder and a decoder. The dimensions of the encoder were d-500-500-2000-10-h for all datasets, where d and h are the dimensions of the input layer and the hashing layer, respectively. The decoder network is a mirror of the encoder. For the input and output layers, we do not use an activation function. The activation function used in the hashing layer is the tanh activation function. The other layers are activated with the nonlinear ReLU function. The autoencoder pretraining settings were the same as those in [52]. After pretraining, the batch size was set to 256 for all datasets. The coefficients of the clustering loss, reconstruction loss and hashing loss were set differently for different datasets. We first set $\alpha = 1$ and determined γ and β through a grid search of $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ by comparing the retrieval performance. Table 2 shows the values of γ, α and β corresponding to the optimal retrieval performance as determined by comparing the mean average precision (mAP) on the different datasets. We divided each dataset into three parts. The first part was the training set used to train the model. The second part was used to determine the optimal hyperparameters of the model by calculating and comparing the mAPs of models with different hyperparameters. The third part was the test set, which was used to evaluate the performance of the model for comparison with other algorithms. For the MNIST, CIFAR-10-GIST and FASHION-MNIST datasets, we used the Adam optimizer [25] with an initial learning rate of $\lambda = 0.001$, $\theta_1 = 0.9$, and $\theta_2 = 0.999$. For the other datasets, we use SGD with a learning rate of $\lambda = 0.1$ and a momentum of $\theta = 0.99$. The update intervals T were 140, 30, 100, 30, and 140 iterations for MNIST, USPS, CIFAR-10-GIST, STL-10 and FASHION-MNIST, respectively. We used Python and Keras [4] to write our code and ran the algorithm on a machine with one GeForce RTX 2080 Ti GPU.

Table 3 Comparison of the average retrieval results (mAP score, Precision@N with $N = 1000$, and Hamming look-up results with a Hamming radius of $r = 2$) on MNIST

Method	Hamming ranking (mAP@All, %)			Precision (%) @ sample = 1000			Precision (%) @ r = 2	
	16	32	64	16	32	64	16	32
PCA-ITQ	41.18	43.82	45.37	66.39	74.04	77.42	65.73	73.14
KMH	32.12	33.29	35.78	60.43	67.19	72.65	61.88	68.85
Spherical	25.81	30.77	34.75	49.48	61.27	69.85	51.71	64.26
SH	26.64	25.72	24.10	56.29	64.29	61.98	57.52	65.31
PCAH	27.33	24.85	21.47	56.56	59.99	57.97	36.36	65.54
LSH	20.88	25.83	31.71	37.77	50.16	61.73	25.10	55.61
AGH	39.92	33.39	28.64	70.75	73.93	74.79	64.69	74.64
DH	43.14	44.97	46.74	67.89	74.72	78.63	66.10	73.29
UH-BNN	45.38	47.21	–	–	–	–	–	–
DeepQuan	60.30	55.50	52.54	-	–	–	–	–
KNNH	47.33	53.25	56.03	67.95	75.89	79.04	71.82	69.08
ClusterGAN	–	88.70	89.93	–	90.83	91.60	–	–
AUCH	**84.61**	**92.43**	**92.14**	**85.91**	**93.67**	**93.91**	**79.92**	**94.12**

The best performance results are displayed in boldface

Table 4 Comparison of the average retrieval results (mAP score, Precision@N with $N = 1000$ and Hamming look-up results with a Hamming radius of $r = 2$) on CIFAR-10-GIST

Method	Hamming ranking (mAP@1000, %)			Precision (%) @ sample = 1000			Precision (%) @ r = 2	
	16	32	64	16	32	64	16	32
PCA-ITQ	15.67	16.20	16.64	22.64	25.30	**27.90**	22.60	14.99
KMH	13.59	13.93	14.46	20.28	21.97	22.80	22.08	5.72
Spherical	13.98	14.58	15.38	20.13	22.33	25.19	20.96	12.50
SH	12.55	12.42	12.56	18.83	19.72	20.16	18.52	20.60
PCAH	12.91	12.60	12.10	18.89	19.35	18.73	21.29	2.68
LSH	12.55	13.76	15.07	16.21	19.10	22.25	16.73	7.07
AGH	13.64	13.61	13.54	22.61	23.28	25.48	21.25	24.53
DH	16.17	16.62	16.96	23.79	26.00	27.70	23.33	15.77
UH-BNN	17.83	18.52	–	–	–	–	–	–
DeepQuan	18.19	18.20	18.74	–	–	–	–	–
KNNH	17.32	18.76	19.54	22.52	25.48	27.08	23.36	15.05
AUCH	**25.65**	**28.53**	**27.54**	**24.49**	**27.24**	25.88	**26.04**	**39.69**

The best performance results are displayed in boldface

6.3 Image retrieval

6.3.1 Alternative models

To evaluate the effectiveness of AUCH for the image retrieval task, we performed experiments on the MNIST and CIFAR-10-GIST datasets and compared the retrieval performance with that of several state-of-the-art unsupervised hashing methods, including ITQ, KMH, Spherical, SH, PCAH, LSH, AGH, DH, UH-BNN, DeepQuan, KNNH, ClusterGAN, and UDHSCA [11]. Note that because 512-D GIST features were not used for training in the ClusterGAN paper, our experimental results on CIFAR-10-GIST are not compared with those of ClusterGAN.

6.3.2 Evaluation metrics

To evaluate the retrieval performance of AUCH compared to the other methods, we rely on three popular metrics used to evaluate retrieval: the mean average precision (mAP) , the precision for the top N samples and the Hamming look-up result when the Hamming radius is set to r. The mAP is actually the area enclosed by the PR curve and the coordinate axis. The larger the value of the mAP is, the better the overall performance of the hashing algorithm. The second metric is calculated as the percentage of true neighbors among the top N retrieved samples. The premise of the last metric is that the precision of a failed search is zero; this metric measures the precision for all points in the buckets that fall within a Hamming radius of $r = 2$.

6.3.3 Performance comparison

Tables 3 and 4 show that our AUCH method significantly outperforms the other state-of-the-art hashing methods in general on the MNIST and CIFAR-10-GIST datasets, except that when the number of hash bits is 64, the precision for the top 1000 samples of AUCH is slightly lower than that

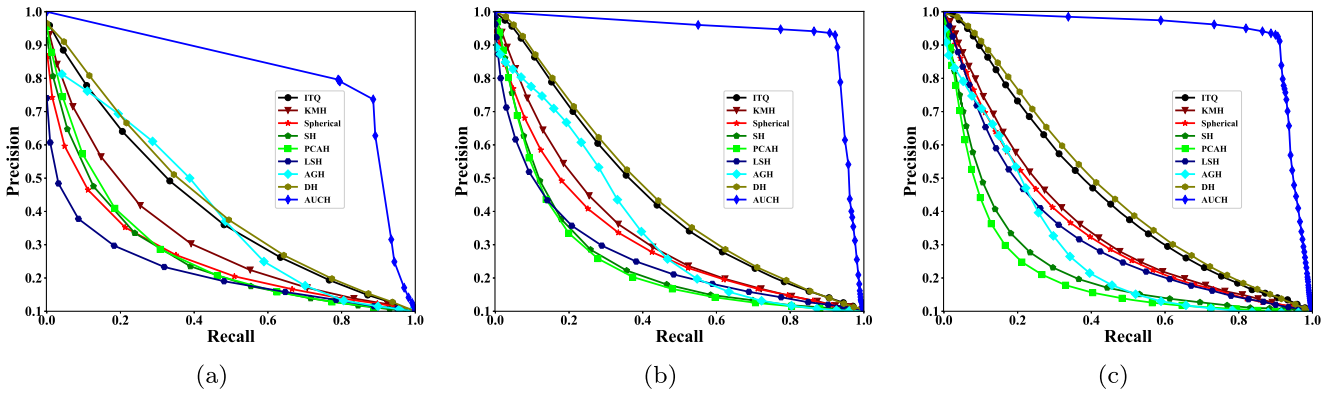


Fig. 3 Recall vs. precision curves on the MNIST dataset for unsupervised hashing methods at 16, 32 and 64 bits: **a** 16 bits. **b** 32 bits. **c** 64 bits

Table 5 Retrieval mAP@1000 results on MNIST. The notation XXX-VGGF is used to denote hashing methods built on top of the features produced by the pretrained VGG-F network

Method	12 bits	24 bits	32 bits	48 bits
ITQ-VGGF	0.407	0.478	0.487	0.506
SH-VGGF	0.301	0.304	0.296	0.287
LSH-VGGF	0.176	0.191	0.220	0.305
ITQ	0.404	0.442	0.447	0.460
SH	0.270	0.278	0.260	0.254
LSH	0.162	0.236	0.222	0.276
UDHSCA	0.552	0.540	0.521	0.518
AUCH	**0.741**	**0.911**	**0.934**	**0.947**

The best performance results are displayed in boldface

of PCA-ITQ on the CIFAR-10-GIST dataset. In other cases, whether the number of hash bits is 16, 32 or 64, AUCH is superior to the other methods in terms of all three indicators. Figure 3 shows the PR curve on the MNIST dataset for the unsupervised hashing methods at 16, 32 and 64 bits. As seen from this figure, AUCH is significantly better than the other methods. It should be noted that for the MNIST dataset, we computed the mAP value on the whole database, whereas for the CIFAR-10-GIST dataset, the map was computed only on the top 1000 returned samples (as done in [12]). These two calculation methods are denoted by mAP@All and mAP@1000, respectively.

In addition, to ensure the integrity and breadth of the experiment, we further evaluated the retrieval performance of AUCH on MNIST for different numbers of bits. Table 5 shows the retrieval mAP@1000 obtained with AUCH on the MNIST dataset and compares it with those of other state-of-the-art methods, including UDHSCA [11], ITQ, SH, and LSH. The results of the compared methods were taken from [11]. AUCH also significantly outperforms other unsupervised hashing methods on these hash codes.

In short, we believe that because AUCH captures efficient feature representations of the data, the results are greatly improved.

6.4 Ablation study and parameter sensitivity analysis

We also performed an ablation study to investigate the effect of each component of the overall loss on the retrieval task. We evaluated the retrieval performance across γL_c, $\gamma L_c + \alpha L_r$, $\gamma L_c + \beta L_h$ and $\gamma L_c + \alpha L_r + \beta L_h$. We changed the loss components one at a time, measuring the difference in mAP@1000 on the CIFAR-10 dataset and in mAP@All on the other datasets, with the number of hash bits set to 32 (see Table 6). Note that γL_c is required during the training process, which guarantees that AUCH can normally perform unsupervised clustering to guide hash learning. When we keep only γL_c or $\gamma L_c + \beta L_h$, this means that after pretraining, we remove the decoder and train only the encoder to minimize the loss. If αL_r participates in the training process, this means that we retain the decoder after pretraining and train the entire autoencoder.

From Table 6, the first observation is that all of the terms contribute to improving the results. Moreover, when we keep only γL_c for training, AUCH can already achieve reasonably satisfactory retrieval performance on all five datasets. When we add βL_h or αL_r, the retrieval performance is improved. αL_r has a greater impact on the mAP than βL_h does, especially on the MNIST dataset. This shows that the clustering loss is the dominant part, and the quantization loss or reconstruction loss can only provide some improvement. In addition, L_r can guide AUCH to extract better features than L_h. AUCH achieves the best retrieval performance on all datasets with the application of all three partial losses.

Figure 4 shows the effects of varying the values of the hyperparameters γ and β. The experiments were conducted by varying one parameter at a time while keeping the others fixed and setting α to 1. Note that the fixed hyperparameters were set to their optimal values and the number of hash bits was 32. To achieve competitive results, the settings for γ and β are different on the different datasets. For example, when we fix α, β and set the value of γ to approximately 10^{-2}, AUCH can achieve the best retrieval performance on MNIST, while the optimal γ is approximately 10^{-1} on USPS .

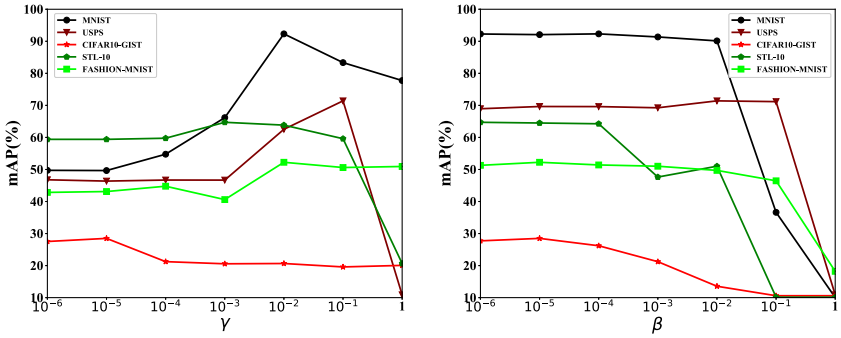
6.5 Image clustering

6.5.1 Alternative models

To evaluate the effectiveness of AUCH for the image clustering task, we performed experiments on the MNIST,

Table 6 Ablation study results in terms of the mean average precision (mAP, %) on different datasets

Dateset	MNIST	CIFAR-10-GIST	USPS	STL-10	FASHION-10
γL_c	73.23	19.26	66.34	63.21	48.33
$\gamma L_c + \beta L_h$	77.05	20.88	68.30	63.45	49.71
$\gamma L_c + \alpha L_r$	91.74	26.85	70.26	64.65	51.61
$\gamma L_c + \alpha L_r + \beta L_h$	92.43	28.53	71.41	64.72	52.24



(a) mAP@32 bits vs γ (b) mAP@32 bits vs β

Fig. 4 Parameter sensitivity analysis

USPS, CIFAR-10, STL-10 and FASHION-MNIST datasets, which are designed only for clustering, to compare the clustering performance of AUCH with that of several state-of-the-art unsupervised clustering methods, including K-means [24], SC-Ncut [43], SC-LS [3], AC-PIC [60], SEC [39], LDMGI [57], NMF-D [46], DEC [52], JULE [56], DEPICT [14] and ClusterGAN.

6.5.2 Evaluation metrics

To evaluate the clustering performance of AUCH in comparison to the other methods, we use two popular metrics for clustering: the accuracy (ACC) and the normalized mutual information (NMI). We need to find the best mapping between the predicted clusters and the true labels to calculate the ACC [27]. The NMI measures the similarity between two data points with the same label. The lowest similarity value is normalized to 0, and the highest is normalized to 1 [53].

6.5.3 Performance comparison

Table 7 shows the clustering performance comparison with the other clustering methods, all of which except ClusterGAN are designed only for clustering, on the five real datasets. From Table 7, we can see that AUCH is superior to most other state-of-the-art clustering methods on the MNIST, USPS and CIFAR-10 datasets in terms of the ACC and NMI metrics. It is more straightforward to look at the confusion matrix. Figure 5 presents the confusion matrix of AUCH after clustering on MNIST with 32 bits. We can also see that AUCH generally divides MNIST into 10 categories. Simultaneously, on the STL-10 and FASHION-MNIST datasets, AUCH is obviously superior to the other methods. Especially on the STL-10 dataset, AUCH shows greatly improved clustering performance compared with the other methods. In summary, this experiment demonstrates that AUCH can extract discriminative features from images and achieve superior clustering performance.

Table 7 Clustering performance comparison with other clustering methods, all of which except ClusterGAN are designed only for clustering, on five real datasets

Method	MNIST		USPS		CIFAR-10		STL-10		FASHION-10	
	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC
K-means	0.500	0.534	0.450	0.460	0.102	0.239	0.209	0.284	0.512	0.474
SC-Ncut	0.411	0.327	0.675	0.314	–	–	–	–	0.575	0.508
SC-LS	0.706	0.714	0.681	0.659	0.114	0.258	0.105	0.168	0.497	0.496
AC-PIC	0.017	0.115	0.840	0.855	0.118	0.264	0.235	0.329	–	–
SEC	0.779	0.804	0.511	0.544	0.107	0.249	0.245	0.307	–	–
LDMGI	0.802	0.842	0.563	0.580	0.109	0.253	0.260	0.331	–	–
NMF-D	0.152	0.175	0.287	0.382	–	–	–	–	–	–
DEC	0.816	0.844	0.586	0.619	0.267	0.312	0.284	0.359	0.546	0.518
JULE	0.913	0.964	0.913	0.950	0.194	0.275	0.204	0.288	0.608	0.563
DEPICT	0.917	**0.965**	0.927	0.964	0.274	0.326	0.303	0.371	0.392	0.392
ClusterGAN	**0.921**	0.964	**0.931**	**0.970**	**0.323**	**0.412**	0.335	0.423	–	–
AUCH	0.909	0.960	0.787	0.775	0.188	0.318	**0.709**	**0.734**	**0.634**	**0.617**

The highest values of all metrics on the datasets are shown in bold

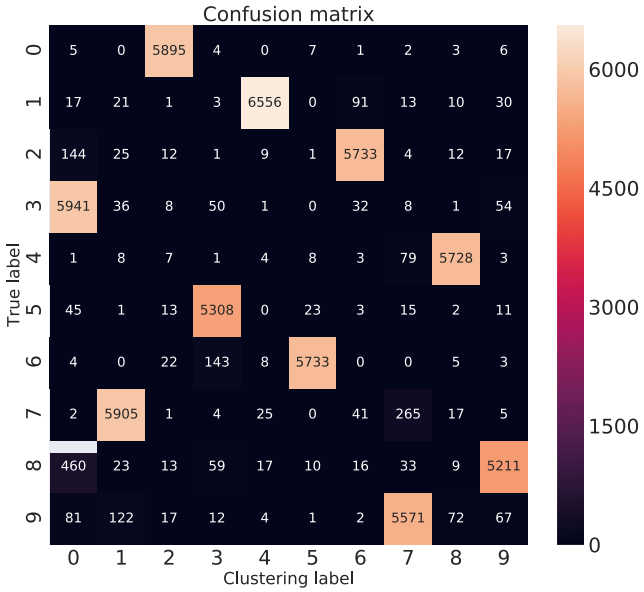


Fig. 5 Confusion matrix of the true labels and clustering labels on MNIST

7 Conclusion

We have presented an unsupervised deep hashing method, AUCH, that preserves the structural relationship between visual data. AUCH performs hash learning by means of an additional hashing layer, which is equivalent to hash mapping, between the encoder and decoder of an autoencoder. By optimizing an objective function defined over the clustering loss, reconstruction loss and quantization loss for hash codes, AUCH jointly learns feature representations, hashing functions and clustering assignments from input data. Experimental results obtained on general datasets show that AUCH achieves superior retrieval performance and produces promising clustering results.

Acknowledgements This work was supported in part by Zhejiang NSF Grant No. LZ20F020001 and No. LY20F020009, China NSF Grants No. 61472194, No. 61572266, Ningbo NSF Grant No. 2019A610085 as well as programs sponsored by K.C. Wong Magna Fund in Ningbo University. (Corresponding author: Jiangbo Qian.)

References

1. Andoni A, Indyk P (2006) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: 2006 47Th annual IEEE symposium on foundations of computer science (FOCS'06), IEEE, pp 459–468
2. Chen J, Cheung WK, Wang A (2018) Learning deep unsupervised binary codes for image retrieval
3. Chen X, Cai D (2011) Large scale spectral clustering with landmark-based representation. In: Twenty-fifth AAAI conference on artificial intelligence
4. Chollet F (2015) Keras documentation. keras io
5. Coates A, Ng A, Lee H (2011) An analysis of single-layer networks in unsupervised feature learning. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp 215–223
6. Dai B, Guo R, Kumar S, He N, Song L (2017) Stochastic generative hashing. In: Proceedings of the 34th international conference on machine learning-Volume 70, JMLR. org, pp 913–922
7. Datar M, Immorlica N, Indyk P, Mirrokni VS (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the twentieth annual symposium on Computational geometry, ACM, pp 253–262
8. Deng C, Chen Z, Liu X, Gao X, Tao D (2018) Triplet-based deep hashing network for cross-modal retrieval. IEEE Trans Image Process 27(8):3893–3903
9. Deng T, Ye D, Ma R, Fujita H, Xiong L (2020) Low-rank local tangent space embedding for subspace clustering. Inf Sci 508:1–21
10. Do TT, Doan AD, Cheung NM (2016) Learning to hash with binary deep neural network. In: European conference on computer vision, Springer, pp 219–234
11. En S, Crémilleux B, Jurie F (2017) Unsupervised deep hashing with stacked convolutional autoencoders. In: 2017 IEEE International conference on image processing (ICIP), IEEE, pp 3420–3424
12. Erin Liong V, Lu J, Wang G, Moulin P, Zhou J (2015) Deep hashing for compact binary codes learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2475–2483
13. Ghasedi K, Wang X, Deng C, Huang H (2019) Balanced self-paced learning for generative adversarial clustering network. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4391–4400
14. Ghasedi Dizaji K, Herandi A, Deng C, Cai W, Huang H (2017) Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In: Proceedings of the IEEE international conference on computer vision, pp 5736–5745
15. Gong Y, Lazebnik S, Gordo A, Perronnin F (2012) Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. IEEE Trans Patt Anal Mach Intel 35(12):2916–2929
16. Gui J, Liu T, Sun Z, Tao D, Tan T (2017) Fast supervised discrete hashing. IEEE Trans Patt Anal Mach Intel 40(2):490–496
17. Guo X, Gao L, Liu X, Yin J (2017) Improved deep embedded clustering with local structure preservation. In: IJCAI, pp 1753–1759
18. Guttman A (1984) R-trees: a dynamic index structure for spatial searching, vol 14 ACM
19. Har-Peled S, Indyk P, Motwani R (2012) Approximate nearest neighbor: Towards removing the curse of dimensionality. Theory of Computing 8(1):321–350
20. He K, Wen F, Sun J (2013) K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2938–2945
21. He X, Wang P, Cheng J (2019) K-nearest neighbors hashing. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2839–2848
22. Heo JP, Lee Y, He J, Chang SF, Yoon SE (2012) Spherical hashing. In: 2012 IEEE Conference on computer vision and pattern recognition, IEEE, pp 2957–2964
23. Hull JJ (1994) A database for handwritten text recognition research. IEEE Trans Patt Anal Mach Intel 16(5):550–554
24. Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY (2002) An efficient k-means clustering algorithm: Analysis and implementation. IEEE Transactions on Pattern Analysis & Machine Intelligence 24(7): 881–892

25. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980
26. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images. Tech. rep., Citeseer
27. Kuhn HW (1955) The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1-2):83–97
28. LeCun Y, Bottou L, Bengio Y, Haffner P et al (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
29. Li C, Deng C, Li N, Liu W, Gao X, Tao D (2018) Self-supervised adversarial hashing networks for cross-modal retrieval. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4242–4251
30. Lin J, Morere O, Chandrasekhar V, Veillard A, Goh H (2015) Deephash: Getting regularization, depth and fine-tuning right. arXiv:1501.04711
31. Lin J, Morere O, Petta J, Chandrasekhar V, Veillard A (2016) Tiny descriptors for image retrieval with unsupervised triplet hashing. In: 2016 Data compression conference (DCC), IEEE, pp 397–406
32. Lin K, Lu J, Chen CS, Zhou J (2016) Learning compact binary descriptors with unsupervised deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1183–1192
33. Liu W, Wang J, Kumar S, Chang SF (2011) Hashing with graphs
34. Liu X, He J, Deng C, Lang B (2014) Collaborative hashing. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2139–2146
35. Lowe DG (2004) Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2):91–110
36. Lu J, Liong VE, Zhou J (2017) Deep hashing for scalable image search. IEEE Trans Image Process 26(5):2352–2367
37. Ma C, Gong C, Gu Y, Yang J, Feng D (2018) Shiss: Supervised hashing with informative set selection. Pattern Recogn Lett 107:105–113
38. Maaten LVD, Hinton G (2008) Visualizing data using t-sne. Journal of Machine Learning Research 9(Nov):2579–2605
39. Nie F, Zeng Z, Tsang IW, Xu D, Zhang C (2011) Spectral embedded clustering: a framework for in-sample and out-of-sample spectral clustering. IEEE Transactions on Neural Networks 22(11): 1796–1808
40. Nigam K, Ghani R (2000) Analyzing the effectiveness and applicability of co-training. In: Cikm, vol 5, pp 3
41. Oliva A, Torralba A (2001) Modeling the shape of the scene: a holistic representation of the spatial envelope. International Journal of Computer Vision 42(3):145–175
42. Salakhutdinov R, Hinton G (2009) Semantic hashing. Int J Approx Reason 50(7):969–978
43. Shi J, Malik J (2000) Normalized cuts and image segmentation. Departmental Papers (CIS), pp 107
44. Song J, He T, Gao L, Xu X, Hanjalic A, Shen HT (2018) Binary generative adversarial networks for image retrieval. In: Thirty-second AAAI conference on artificial intelligence
45. Toth CD, O'Rourke J, Goodman JE (2017) Handbook of discrete and computational geometry. Chapman and hall/CRC
46. Trigeorgis G, Bousmalis K, Zafeiriou S, Schuller B (2014) A deep semi-nmf model for learning hidden representations. In: International conference on machine learning, pp 1692–1700
47. Wang H, Yang Y, Liu B, Fujita H (2019) A study of graph-based system for multi-view clustering. Knowl-Based Syst 163:1009–1019
48. Wang J, Kumar S, Chang SF (2012) Semi-supervised hashing for large-scale search. IEEE Trans Pattern Anal Mach Intel 34(12):2393–2406
49. Weiss Y, Torralba A, Fergus R (2009) Spectral hashing. Advances in neural information processing systems, pp 1753–1760
50. Xia R, Pan Y, Lai H, Liu C, Yan S (2014) Supervised hashing for image retrieval via image representation learning. In: Twenty-eighth AAAI conference on artificial intelligence
51. Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747
52. Xie J, Girshick R, Farhadi A (2016) Unsupervised deep embedding for clustering analysis. In: International conference on machine learning, pp 478–487
53. Xu W, Liu X, Gong Y (2003) Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, ACM, pp 267–273
54. Yang E, Deng C, Li C, Liu W, Li J, Tao D (2018) Shared predictive cross-modal deep quantization. IEEE Transactions on Neural Networks and Learning Systems 29(11):5292–5303
55. Yang E, Deng C, Liu T, Liu W, Tao D (2018) Semantic structure-based unsupervised deep hashing. In: IJCAI, pp 1064–1070
56. Yang J, Parikh D, Batra D (2016) Joint unsupervised learning of deep representations and image clusters. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5147–5156
57. Yang Y, Xu D, Nie F, Yan S, Zhuang Y (2010) Image clustering using local discriminant models and global integration. IEEE Trans Image Process 19(10):2761–2773
58. You S, Xu C, Wang Y, Xu C, Tao D (2017) Privileged multi-label learning. arXiv:1701.07194
59. You S, Xu C, Xu C, Tao D (2017) Learning from multiple teacher networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 1285–1294
60. Zhang W, Zhao D, Wang X (2013) Agglomerative clustering via maximum incremental path integral. Pattern Recogn 46(11):3056–3065
61. Zhang Y, Yang Y, Li T, Fujita H (2019) A multitask multiview clustering algorithm in heterogeneous situations based on lle and le. Knowl-Based Syst 163:776–786

Bolin Zhang is a graduate student of Faculty of Electrical Engineering and Computer Science at Ningbo University. His research interests mainly include information retrieval, deep learning, and computer vision.

Jiangbo Qian received his Ph.D. degree in Computer Science from Southeast University (China) in 2006. He is currently a Professor in the Faculty of Electrical Engineering and Computer Science at Ningbo University, China. He was a visiting scholar in the Department of Computer and Information Science at The University of Michigan, USA. His research interests include big data management, deep learning, information retrieval, cloud computing, multidimensional indexing, Bloom filter, and hardware/software co-design.