Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Improved deep convolutional embedded clustering with re-selectable sample training

Hu Lu[a,c], Chao Chen[a,*], Hui Wei[b], Zhongchen Ma[a], Ke Jiang[a], Yingquan Wang[a]

[a] Computer Science and Communication Engineering, JiangSu University, JiangSu 212013, China
[b] School of Computer Science, Fudan University, ShangHai, China
[c] Jiangsu Province Big Data Ubiquitous Perception and Intelligent Agricultural Application Engineering Research Center, China

A R T I C L E I N F O

Article history:
Received 26 January 2021
Revised 10 February 2022
Accepted 24 February 2022
Available online 2 March 2022

Keywords:
Unsupervised clustering
Deep embedded clustering
Autoencoder
Reliable samples

A B S T R A C T

The deep clustering algorithm can learn the latent features of the embedded subspace, and further realize the clustering of samples in the feature space. The existing deep clustering algorithms mostly integrate neural networks and traditional clustering algorithms. However, for sample sets with many noise points, the effect of the clustering remains unsatisfactory. To address this issue, we propose an improved deep convolutional embedded clustering algorithm using reliable samples (IDCEC) in this paper. The algorithm first uses the convolutional autoencoder to extract features and cluster the samples. Then we select reliable samples with pseudo-labels and pass them to the convolutional neural network for training to get a better clustering model. We construct a new loss function for backpropagation training and implement an unsupervised deep clustering method. To verify the performance of the method proposed in this paper, we conducted experimental tests on standard data sets such as MNIST and USPS. Experimental results show that our method has better performance compared to traditional clustering algorithms and the state-of-the-art deep clustering algorithm under four clustering metrics.

1. Introduction

The clustering algorithm is an important research algorithm in the unsupervised field. Compared with supervised algorithms, clustering can directly divide samples into different clusters by measuring the similarity among samples.The purpose of clustering is to ensure that samples in the same cluster have a high degree of similarity, and samples between different clusters have a low degree of similarity. Traditional clustering algorithms such as *k*-means [1], DBSCAN [2], and spectral clustering [3,4], hierarchical clustering [5,6] have their own advantages in clustering effects. However, when faced with the features of high-dimensional space, traditional clustering algorithms often result in the algorithm having high time complexity due to the similarity measurement problem, which can even reduce the clustering accuracy. In addition, the feature extraction of samples also plays a vital role in the clustering effect.

Nowadays, many studies have been dedicated to deep clustering algorithms, for example, combining traditional clustering algorithms with autoencoder models [7–9]. Since the autoencoder itself is also a kind of unsupervised algorithm, it does not require the participation of sample labels when combined with the clustering algorithm. With the addition of the autoencoder, the deep clustering algorithm can solve the problems caused by excessive dimensionality through nonlinear dimensionality reduction of the encoder. At the same time, the neural network can be used to extract the latent features of the sample, and then the sample can be clustered into different clusters with the traditional clustering algorithms. This not only solves the problems caused by the traditional clustering algorithms when the data set is too large and the sample dimension is too high [10], but it is also conducive to the model's extraction of sample features [11,12]. However, when the data set contains a large number of noisy samples or samples that are difficult to distinguish, the results of the above-mentioned deep clustering algorithms are often affected to a certain extent.

Aiming to address these issues, this paper proposes a new deep clustering algorithm model. The model first uses a convolutional autoencoder to extract features from the samples, then classifies the samples through clustering and assigns pseudo labels to each sample. Thereafter, according to the similarity between the sample and the center point of each class, a sample with high similarity is selected as the reliable sample, and the selected sample with the pseudo label is used to participate in the training of the convolutional neural network. The trained convolutional neural network has the function of the classification. The model not only uses a

* Corresponding author.
E-mail address: michealcchao@163.com (C. Chen).

https://doi.org/10.1016/j.patcog.2022.108611
0031-3203/© 2022 Elsevier Ltd. All rights reserved.

convolutional layer in the design of autoencoder, but also embeds a pooling layer and an up-sampling layer to further extract the potential features of the sample. A new loss function for backpropagation is designed, and finally, a deep clustering method is realized.

The main contributions of this paper can be summarized as follows:

(1) This paper proposes an improved deep convolutional embedded clustering algorithm using reliable samples (IDCEC). After the feature extraction and clustering are performed by the autoencoder, reliable samples with pseudo labels are screened out, and a model with a better classification effect is trained using these reliable samples. This effectively alleviates the impact of noise samples on clustering and improves the accuracy of model clustering.
(2) This paper designs our own deep clustering model structure and corresponding loss function. The model structure uses the convolutional autoencoder with max pooling layers, upsampling layers and a clustering layer with soft label allocation capabilities. According to the reconstruction loss, clustering loss, weight and bias regularization constraints of the deep clustering model, a new loss function is proposed which improves the feature extraction ability of the model and the final clustering effect.
(3) A large number of experiments show that the algorithm proposed in this paper demonstrates good performance with four large image datasets. The experimental results also prove the effectiveness of the algorithm.

We will first introduce related work in Section 2. Then in Section 3, IDCEC proposed in this work will be introduced in detail. We will list specific experiments in Section 4. Finally, Section 5 is our conclusion.

2. Related work

2.1. Clustering

Clustering is an algorithm that divides similar samples into the same class or cluster. During clustering, the parameters of a category are unimportant, and it is necessary simply to group similar samples together according to some metric [13,14]. Among the traditional clustering algorithms, the k-means algorithm is one of the most commonly used. The algorithm alternately adjusts the samples in each cluster and the cluster centers according to the distance between the samples and the cluster centers. In addition, Shi [15] and Chen [16] use improved spectral clustering algorithms to achieve sample clustering with large data sets. Taking into account the importance of different attributes of different categories, Lu [17,18] et al. proposed a multi-kernel fuzzy clustering algorithm. However, when the sample dimension is too high, the traditional clustering algorithm is often not good in terms of clustering effect.

2.2. Autoencoder

With the development of deep learning, many deep neural networks have been proposed one after another [19]. An autoencoder is an unsupervised algorithm neural network model composed of an encoder and a decoder. The traditional autoencoder model optimizes the parameters of the entire network by calculating the reconstruction loss. Variants of the basic autoencoder include the denoising autoencoder [20], the convolutional autoencoder [8,21], the variational autoencoder [22,23], and the sparse autoencoder [24]. In the deep clustering model, the autoencoder not only performs the task of dimensionality reduction, but also can further extract the deep features of the sample. The encoder layer converts the sample X in the source data space into a latent low-dimensional feature space Z through nonlinear dimensionality reduction. Then, the decoder layer transforms Z into $\widehat{X}$ with the same dimension as the source data space, which is $f_\theta : X \to Z \to \widehat{X}$. The optimization goal of the autoencoder is to restore the original data as much as possible after the compressed features are decoded, so that the encoder can better learn the latent features of the original data.

2.3. Deep autoencoder clustering

The deep clustering model is a model that combines the traditional clustering algorithm with the deep model and can achieve higher accuracy in the clustering effect. Since clustering is an unsupervised learning algorithm, deep clustering models often use autoencoders to assist traditional clustering algorithms to achieve clustering. For example, in the Deep Embedded Clustering (DEC) algorithm [25] and the Improved Deep Embedded Clustering (IDEC) algorithm [26], an ordinary autoencoder is used for pre-training to obtain a data representation of the original data. This is non-linearly mapped to the latent feature space, then the k-means algorithm is used to cluster the sample features to get the initial cluster center, then the network is fine-tuned through the clustering layer that it designed until the convergence criterion is met. In order to better extract the hierarchical features of the image, Guo [27] et al. proposed the Deep Convolutional Embedded Clustering (DCEC) algorithm, which improved the structure of the autoencoder and replaced the original part of the fully connected layer with the convolutional layer and the deconvolutional layer. This paper also designs its own deep clustering model based on this idea. In 2019, Zhang [28] et al. performed feature extraction and clustering at the same time, introduced entropy-weighted fuzzy k-means, and added regular terms for weights and biases to prevent overfitting. In order to obtain better clustering results, people often pay attention to the feature representation of subspaces [29]. For example, Ji [30] et al. introduced a novel self-expression layer between the encoder and the decoder Z_i through backpropagation to learn the pairwise affinity between all data points. He [31] et al. proposed the Neural Collaborative Subspace Clustering (NCSC) algorithm, which represents subspace clustering as a classification problem. The neural model consists of two modules, one for classification and one for similarity learning. These two modules cooperate with each other in the learning process. In addition, graph clustering is also often used in deep clustering. For example, Bo [32] et al. in the Structural Deep Clustering Network (SDCN) proposed combining autoencoders with graph matching, which can add structured information among samples by establishing a graph structure during clustering. Elie [33] and Wang [34] et al. used attention networks to capture the importance of neighboring nodes to the target node, and encoded the topological structure and node content in the graph into a compact representation, then trained an internal product decoder to reproduce the graph structure.

3. Proposed method

The algorithm model proposed in this paper is divided into two primary modules. The first module extracts features through the convolutional autoencoder and the corresponding loss function. Then, the clustering of samples is realized through the clustering layer. The second module selects reliable samples with pseudo-labels to participate in the training of the convolutional neural network according to the results of the clustering. This operation can effectively filter out unreliable pseudo-label samples, so that the trained model can achieve better classification results. The whole network structure is illustrated in Fig. 1.

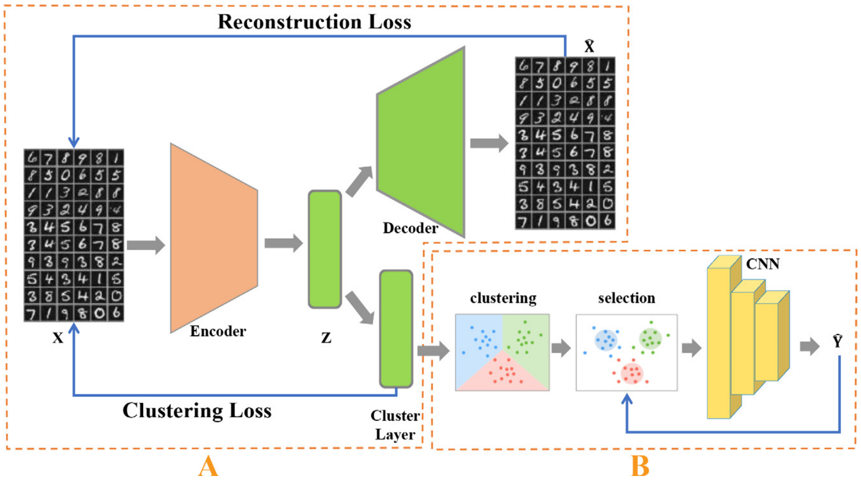

Fig. 1. The framework of our proposed IDCEC. We first input the sample X into the autoencoder to obtain the reconstructed sample $\widehat{X}$, which then cooperates with the clustering layer to initially cluster the samples. Reliable samples are selected from the pseudo-labeled samples to participate in the training of the convolutional neural network, and finally, the convolutional neural network is used to classify the samples.

3.1. Deep clustering model based on convolutional autoencoder

As shown in part A in Fig. 1, the deep clustering model based on the convolutional autoencoder is mainly composed of an autoencoder and a clustering layer. The sample passes through the encoder layer of the autoencoder to obtain the embedded layer feature Z. Subsequently, the decoder layer is used to decode Z, and the decoded image is compared with the original image to form the reconstruction loss. The feature Z is then used as the input of the clustering layer and the corresponding soft label assigned to each sample, and the KL divergence is defined as the clustering loss to narrow the difference between the distribution of soft labels and the custom target distribution. In addition, the weight and bias L2 regularization constraints are added to the design of the loss function to prevent the model from overfitting. Therefore, the total loss function L of the model proposed in this paper can be defined as:

$$L = L_r + \alpha L_c + \beta L_w \tag{1}$$

where L_r represents the reconstruction loss, L_c represents the clustering loss, L_w is the regularization constraint of weight and bias, and α and β represent the weight of the corresponding sub-loss function.

The autoencoder proposed in this paper uses a convolutional autoencoder structure. The encoder layer structure of the autoencoder model can be expressed as $z_i = Encoder(x_i)$, and the decoder layer can be expressed as $\widehat{x}_i = Decoder(z_i)$. Therefore, the reconstruction loss function L_r of the autoencoder can be expressed as:

$$L_r = \frac{1}{2}\sum_{i=1}^{n} \|x_i - \widehat{x}_i\|_2^2 \tag{2}$$

In addition, the model also combines the clustering layer with the encoder layer of the autoencoder and uses the t distribution as the kernel to measure the similarity between the sample points and the centroid, which is used to fine-tune the clustering center. The probability that sample ith belongs to class jth can be expressed as:

$$q_{ij} = \frac{(1 + \frac{\|z_i - u_j\|^2}{\tau})^{-\frac{\tau+1}{2}}}{\sum_{j'}(1 + \frac{\|z_i - u_{j'}\|^2}{\tau})^{-\frac{\tau+1}{2}}} \tag{3}$$

where $z_i = Encoder(x_i) \in Z$, which represents the features extracted by the encoder layer of sample x_i, $\{u_j\}_{j=1}^{k}$ represents the cluster center of the jth cluster, and τ represents the distribution of student-t degree of freedom. The value of τ is set to 1 in the experiment. In order to optimize the clustering centers, the model defines an auxiliary target probability distribution q_{ij} to measure the probability that sample x_i belongs to class jth and then uses the KL divergence as the clustering loss to narrow the distance between the original distribution and the target distribution. Therefore, the clustering loss function L_c is defined as:

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

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_{j'}(q_{ij'}^2 / \sum_i q_{ij'})} \tag{5}$$

where q_{ij} represents the original probability distribution calculated by the clustering layer and p_{ij} represents the target probability distribution.

The deep clustering model based on the convolutional autoencoder proposed in this paper not only uses the reconstruction loss and clustering loss in the loss function, but also adds the regular term constraints of weight and bias, namely:

$$L_w = \frac{1}{2}\sum_l \{\|w^l\|_2^2 + \|b^l\|_2^2\} \tag{6}$$

The regular term can help the hidden layer to effectively learn the feature expression of the sample even when the number of neurons is large. In summary, the total loss function L of the model is specifically expressed as:

$$\begin{aligned} L &= L_r + \alpha L_c + \beta L_w \\ &= \tfrac{1}{2}\textstyle\sum_{i=1}^{n} \|x_i - \widehat{x}_i\|_2^2 + \alpha \sum_i \sum_j p_{ij} \cdot \log \frac{p_{ij}}{q_{ij}} \\ &\quad + \beta \tfrac{1}{2} \textstyle\sum_l \{\|w^l\|_2^2 + \|b^l\|_2^2\} \end{aligned} \tag{7}$$

In formula (7), L_r represents the reconstruction loss, L_c represents the clustering loss, L_w represents the regular constraint term, and α and β represent the weight of the corresponding sub-loss function.

In model design, in order to give the features learned by the model better robustness, a small amount of random noise is added to the input samples. Then, the noisy samples are passed into the autoencoder for encoding and decoding, and the reconstruction loss is calculated. The model also adds the max-pooling layer and the up-sampling layer on the basis of the DCEC [27], so that the filter moving step length of the convolutional layer can be reduced to 1, thereby obtaining more detailed features (Fig. 2).

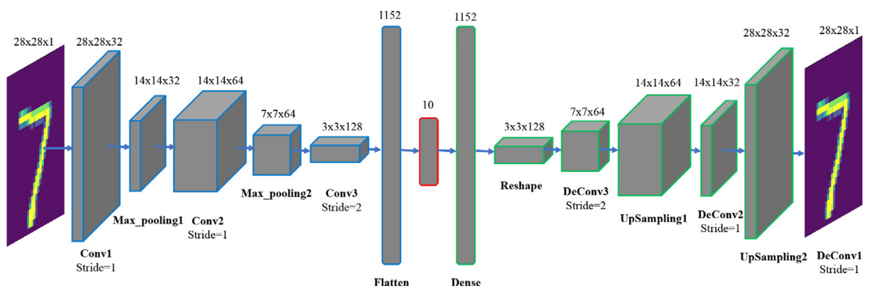

Fig. 2. Convolutional autoencoder model structure. The encoder layer of the model is composed of a convolutional layer and a pooling layer, and then the features are flattened through the Flatten layer, and the fully connected layer is used to continue to extract the features. In the decoder layer, the dimension is resized to the size of the Conv3 layer through the fully connected layer, and then the upsampling layer and deconvolution layer are used to restore the original image size. The entire network can be directly trained in an end-to-end format.

3.2. Network optimization

The convolutional autoencoder uses the loss function and the corresponding optimization method to perform backpropagation to update the parameters of the network layer, and it can then extract the latent features of each sample.

The model obtains the initial cluster center u_j by performing k-means clustering on the sample feature Z extracted from the encoder layer. After that, the cluster center u_j can be updated iteratively according to the cluster loss value. The calculation formula of u_j is as follows:

$$u_j = u_j - \frac{\eta}{m}\sum_{i=1}^{m}\frac{\partial L_c}{\partial u_j} \tag{8}$$

In formula (8), η is the learning rate and m is the number of samples in each batch during small batch training.

In the process of network optimization, the target distribution P is fixed, and the values of the embedding point z_i and the cluster center u_i can be updated according to the clustering loss L_c:

$$\frac{\partial L_c}{\partial z_i} = 2\cdot\sum_{j=1}^{K}(1+\|z_i-u_j\|^2)^{-1}(p_{ij}-q_{ij})(z_i-u_j) \tag{9}$$

$$\frac{\partial L_c}{\partial u_j} = 2\cdot\sum_{i=1}^{n}(1+\|z_i-u_j\|^2)^{-1}(q_{ij}-p_{ij})(z_i-u_j) \tag{10}$$

For the autoencoder, the forward propagation process of the network can be expressed as:

$$a^l = \sigma(z^l) = \sigma(w^l\cdot a^{l-1}+b^l) \tag{11}$$

where $\sigma(\cdot)$ represents the activation function. In backpropagation, the weight and bias of the encoder layer are affected by L_r, L_c, and L_w. Therefore, the gradient of the last L layer in the encoder layer can be expressed as:

$$\delta^L = \frac{\partial L}{\partial z^L} = \frac{\partial(L_r+\alpha L_c+\beta L_w)}{\partial a^L}\odot\sigma'(z^L) \tag{12}$$

where $\odot$ represents the Hadamard product and $\sigma'(\cdot)$ represents the derivative of the activation function. When $l=2,3,\ldots,L-1$, the gradient of the output layer z^l, weight w_e and bias b_e can be calculated according to formula (13) and (14).

$$\delta^l = (\frac{\partial z^{l+1}}{\partial z^l})^T\delta^{l+1} = (w^{l+1})^T\delta^{l+1}\odot\sigma'(z^l) \tag{13}$$

$$\begin{cases}\dfrac{\partial L}{\partial w_e^l} = \delta^l(a^{l-1})^T\\ \dfrac{\partial L}{\partial b_e^l} = \delta^l\end{cases} \tag{14}$$

Therefore, the weight w_e^l and bias b_e^l of the lth layer of the encoder layer can be updated according to formula (15), where m is the number of samples in each mini-batch and η is the learning rate.

$$\begin{cases} w_e^l = w_e^l - \frac{\eta}{m}\cdot\frac{\partial L}{\partial w_e^l}\\ \quad = w_e^l - \frac{\eta}{m}\sum_{i=1}^{m}(\frac{\partial L_r}{\partial w_e^l}+\alpha\frac{\partial L_c}{\partial w_e^l}+\beta\frac{\partial L_w}{\partial w_e^l})\\ b_e^l = b_e^l - \frac{\eta}{m}\cdot\frac{\partial L}{\partial b_e^l}\\ \quad = b_e^l - \frac{\eta}{m}\sum_{i=1}^{m}(\frac{\partial L_r}{\partial b_e^l}+\alpha\frac{\partial L_c}{\partial b_e^l}+\beta\frac{\partial L_w}{\partial b_e^l})\end{cases} \tag{15}$$

Since the clustering layer is connected to the encoder layer and does not affect the parameter update of the decoder layer, the weight and bias of the decoder layer are only affected by L_r, L_w.

After the above optimization, the weight and bias of the model can be updated, and the clustering layer can also calculate a new cluster center $\{u_i\}_{i=1}^{K}$ in each iteration. When the model reaches the maximum number of iterations, the model stops training.

3.3. Retrain the model with reliable samples

In actual prediction, feature z_i of sample x_i can be extracted directly through the encoder layer of the convolutional autoencoder in Section 3.1. The feature z_i is passed into the clustering layer to obtain the probability q_{ij} that the sample x_i belongs to the jth category. The category with the largest probability is the label s_i of the sample, that is, $s_i = \arg\max_j q_{ij}$. However, when the model encounters samples are difficult to distinguish or there are many noise samples, the recognition effect is not ideal. Because the deep clustering model takes all samples as input and initializes the cluster centers with the k-means clustering algorithm, the calculated cluster centers are greatly disturbed by the noise points in the class. Based on this phenomenon, this paper proposes a sample reselection strategy based on the deep clustering model. The specific process is shown in Fig. 3.

We put all samples X into the encoder layer of the convolutional autoencoder to extract features, and then use k-means clustering to obtain pseudo labels S of all samples. In order to ensure the reliability of the pseudo-labels, samples that are closer to the center point in each cluster are selected from all samples to participate in the retraining of the model. In this way, most unreliable pseudo-label samples can be effectively filtered out. In the experiment, if the distance between the sample x_i point and the class center is more than the threshold λ, the sample x_i will be selected as a reliable sample ($v_i = 1$); otherwise, the sample will be judged as noise ($v_i = 0$) and the distance D calculated as follows:

$$D(x_i, center_j) = \frac{\phi(x_i;\theta)}{\|\phi(x_i;\theta)\|^2}\cdot\frac{center_j}{\|center_j\|^2} \tag{16}$$

where $\phi(x_i;\theta)$ is the feature of the sample x_i, that is, $\phi(x_i;\theta) = Encoder(x_i)$, and $center_j$ is the center of the jth category.

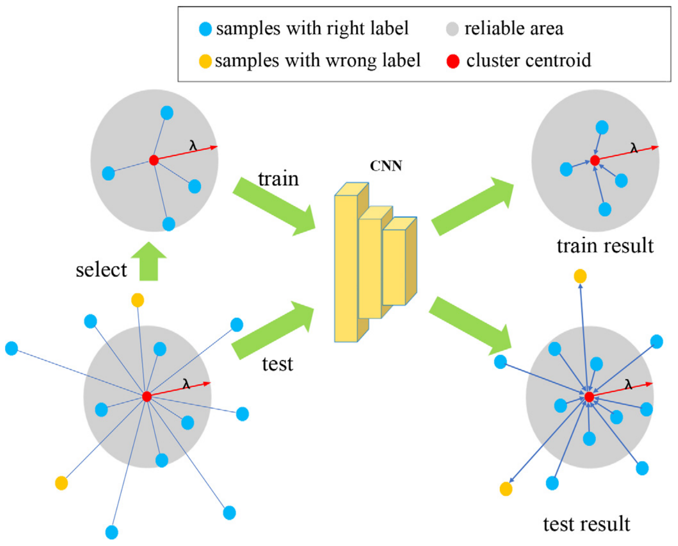

Fig. 3. Flow chart of sample retraining. First, we select samples with higher reliability and pass them to the CNN network for training, which can reduce the impact of false label assignment errors on the model. The trained network can more accurately extract the features of each category sample.

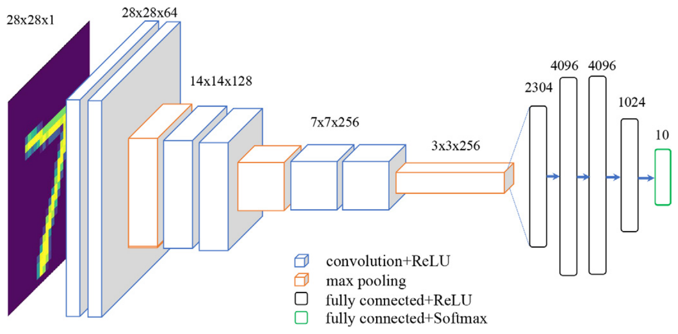

Fig. 4. CNN model structure diagram.

Considering that a certain class of samples may not be found when selecting reliable samples, the algorithm proposed in this paper does not simply average all the reliable sample features in the class as the center point of the class. Instead, the sample point closest to the center point is selected as the center of the class on this basis, so as to ensure that each class contains at least one reliable sample point. Therefore, the calculation method of the class center $center_j$ is:

$$center_j = \arg\min_i \|\phi(x_i;\theta) - u_j\|^2 \quad (17)$$

where u_j represents the cluster center of the jth category obtained by the k-means algorithm.

Although the class label obtained by the selected reliable sample is still the pseudo label assigned by the k-means algorithm, the overall reliability of the pseudo label of the sample filtered by the selection rule is higher. Then, we use the selected reliable samples and their pseudo-labels to train the convolutional neural network. By removing data from the sample that is noise, the model can better learn the salient features of each type of sample, so as to achieve a better classification effect. Finally, when testing the model, we can directly put all the samples into the CNN model that we built to predict the category to which they belong. The structure of the CNN model is shown in Fig. 4.

The details of the IDCEC algorithm is shown in Algorithm 1.

Algorithm 1 Improved deep convolutional embedded clustering using reliable samples (IDCEC).

```
Input: input data:$X$; Number of cluster:$K$; Target distribute update
    interval:$T$; Maximum iterations:$MaxIter$; reliability threshold:$\lambda$.
Output: Autoencoder; Cluster center $u$; labels $s$ and CNN.
 1: Initialize $u$ by $k$-means, autoencoder's weight.
 2: while $iter \le MaxIter$ do
 3:     if $iter\%T == 0$ then
 4:         Get all data's feature $\{z_i = Encoder(x_i)\}_{i=1}^n$.
 5:         Compute $P, Q$ via (3),(5).
 6:         Save last labels assignment: $S_{old} = S$.
 7:         Compute samples' labels assignment $S$ by $P$.
 8:         if $sum(S \neq S_{old})$ then
 9:             Stop Training.
10:         end if
11:     end if
12:     Choose a batch of sample $X_{batch} \in X$.
13:     Update $u$, autoencoder's weight via (8–15) on $X_{batch}$.
14: end while
15: Calculate the distance $\{d_i\}_{i=1}^n$ between $x_i$ and center point via
    (16).
16: Mark $x_i$ as reliable Sample if $d_i \ge \lambda$. Randomly initialize CNN's
    weight.
17: while not convergence do
18:     Update CNN's weight with reliable Sample
19: end while
20: Input $X$ into the trained CNN model to obtain the prediction
    labels of all samples.
21: return the predicted labels of all samples.
```

4. Experiments

4.1. DataSets

In order to verify the effectiveness of the algorithm proposed in this paper and its general applicability to various data sets, four types of frequently used image data sets were selected as test objects in the experiment. Since the model belongs to an unsupervised clustering algorithm, in the experiment, the real label does not participate in the training of the model, and the training set and test set of the data are used together as the unlabeled sample set of the model.

1) MNIST-full: The handwritten digits are divided into 10 categories with 70,000 samples, including 60,000 training samples and 10,000 testing samples. These images have been standardized in size and are located in the center of the image. The size of each image is 28×28 pixels.
2) MNIST-test: This data set is part of the test set of MNIST-full, with a total of 10,000 handwritten digits.
3) USPS: The data set is a set of handwritten digits of the United States Postal Service. There are 9298 handwritten digit images divided into 10 categories. The training samples have 7291 samples and the testing samples have 2007 samples. Each sample is a 16×16 pixel grayscale image, and the grayscale value has been normalized.
4) YTF: YTF is a face recognition data set from YouTube. We choose the first 41 subjects of YTF dataset which contains 10,000 samples and the size of the face image is $55 \times 55 \times 3$ pixels.

Visualization of partial samples from the four datasets is shown in Fig. 5.

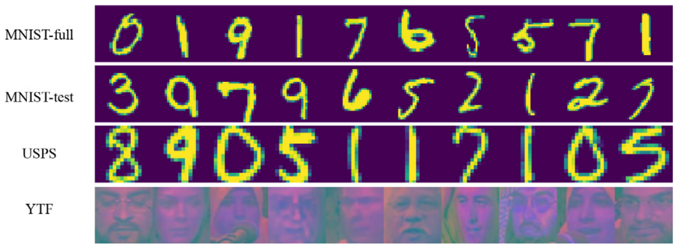

Fig. 5. Data enhancement.

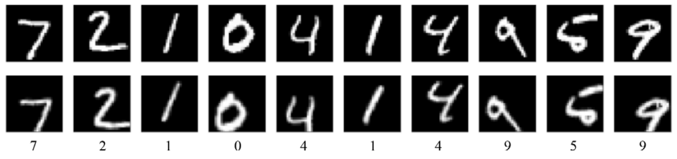

Fig. 6. Data enhancement.

4.2. Evaluation metric

In order to evaluate the results of the clustering algorithm, we used four indicators to measure the clustering effect of the model: cluster accuracy (ACC), normalized mutual information (NMI), adjusted rand index (ARI), macro F1-score (F1).

ACC is used to measure the similarity between the predicted label and the true label. ACC is defined as:

$$ACC = \frac{\sum_{i=1}^{n} \delta(y_i, map(\widehat{y}_i))}{n} \tag{18}$$

where y_i and $\widehat{y}_i$ are, respectively, the real label and predicted label corresponding to sample x_i, n is the total number of samples, and δ is the indicator function, $\delta(x,y) = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{otherwise} \end{cases}$.

NMI is also a measure of the similarity between the clustering results and the real situation of the data set. NMI is defined as:

$$NMI = \frac{2MI(y,\hat{y})}{H(y) + H(\hat{y})} \tag{19}$$

where $MI(y,\hat{y})$ represents the mutual information between the predicted label and the true label. $H(y)$ represents the information entropy of y.

If there is a category label, the clustering result can also calculate the precision and recall rate like classification and also calculate ARI and F1. The calculation formula of ARI and F1 is as follows:

$$ARI = \frac{RI - E(RI)}{max(RI) - E(RI)} \tag{20}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{21}$$

where RI stands for Rand index, defined as $RI = \frac{(TP+TN)}{(TP+FP+TN+FN)}$; precision stands for precision rate, defined as $precision = \frac{TP}{TP+FP}$; and recall stands for recall rate, defined as $recall = \frac{TP}{(TP+FN)}$.

4.3. Implementation details

In the first part of the model, first, the input samples are data-enhanced; that is, the random changes of the images are used to generate new pictures (such as translation, scaling, rotation, etc.; flipping cannot be used). As shown in Fig. 6, the first line is the original image, and the second line is the image with data enhancement. Data enhancement can prevent the model from overfitting and enhance the generalizability of the model. The weight parameter α of the clustering loss in the loss function is 1e−1, and the weight parameter β of the weight and bias regularization is 1e−6. The optimizer uses the Adam optimization method. The number of neurons in the middle layer is the number of categories corresponding to the data set. A total of six convolutional layers are used in the model, and the activation function corresponding to each layer is the LeakyRelu.

The second part of the model mainly includes the selection of reliable samples and the training of the CNN network. The selected distance threshold parameter λ of the reliable sample is 0.65. During training, the CNN network uses the Adam optimizer with a learning rate of 1e−4 for network optimization, and the loss function is cross-entropy loss.

Table 1
Comparison of clustering performance on MNIST-full dataset.

Metric	ACC	NMI	ARI	F1
k-means [1]	0.532	0.499	0.365	0.536
SC-Ncut [15]	0.68	0.758	0.625	0.629
SC-LS [16]	0.714	0.706	–	–
DEC [25]	0.847	0.791	0.749	0.846
IDEC [26]	0.881	0.867	0.850	0.886
DCEC [27]	0.89	0.885	0.851	0.889
DASC [35]	0.801	0.784	–	–
SDCN[32]	0.893	0.895	0.859	0.892
K-DAE [36]	0.860	0.820	0.880	–
DSSEC [37]	0.877	0.857	0.823	–
IDCEC	**0.948**	**0.906**	**0.889**	**0.947**

Table 2
Comparison of clustering performance on MNIST-test dataset.

Metric	ACC	NMI	ARI	F1
k-means [1]	0.542	0.500	0.378	0.541
SC-Ncut [15]	0.667	0.712	0.563	0.661
SC-LS [16]	0.740	0.756	–	–
DEC [25]	0.776	0.724	0.660	0.772
IDEC [26]	0.791	0.736	0.666	0.792
DCEC [27]	0.852	0.809	0.773	0.850
DASC[35]	0.804	0.780	-	–
SDCN [32]	0.802	0.738	0.686	0.801
IDCEC	**0.923**	**0.853**	**0.851**	**0.922**

4.4. Baseline methods

In the comparative experiment, the algorithm proposed in this paper has been compared with traditional clustering algorithms, such as: *k*-means [1], spectral clustering with normalized cuts (SC-Ncut) [15], large-scale spectral clustering (SC-LS) [16]. In addition, our method is also compared with the deep clustering algorithms in recent years, such as: Deep Embedded Clustering (DEC) [25], Improved Deep Embedded Clustering (IDEC) [26], Deep Convolutional Embedded Clustering (DCEC) [27], Deep adversarial subspace clustering (DASC) [35], Structural Deep Clustering Network (SDCN) [32], K-autoencoders deep clustering (K-DAE) [36], Deep Stacked Sparse Embedded Clustering (DSSEC) [37].

4.5. Experiment results

In this paper, the above algorithm was randomly run five times on four data sets and the average value was calculated as the final result, the error range is no more than 3%. If the algorithm for comparison had not been tested on the dataset, we used the hyperparameters mentioned in the paper to obtain the results. If the code was not published or could not be tested on this dataset, we use a minus sign (−) to replace the corresponding result.

Tables 1–4 shows that the algorithm proposed in this article has good performance on the MNIST-full, MNIST-test, USPS, and YTF

Table 3
Comparison of clustering performance on USPS dataset.

Metric	ACC	NMI	ARI	F1
k-means [1]	0.668	0.626	0.545	0.647
SC-Ncut [15]	0.656	0.796	0.650	0.643
SC-LS [16]	0.746	0.755	–	–
DEC [25]	0.733	0.706	0.637	0.718
IDEC [26]	0.762	0.756	0.679	0.746
DCEC [27]	0.790	0.826	0.764	0.794
SDCN [32]	0.781	0.795	0.718	0.770
K-DAE [36]	0.800	0.710	0.770	–
DSSEC [37]	0.784	0.811	0.736	–
IDCEC	**0.812**	**0.858**	**0.772**	**0.798**

Table 4
Comparison of clustering performance on YTF dataset.

Metric	ACC	NMI	ARI	F1
k-means [1]	0.606	0.776	0.576	0.532
SC-Ncut [15]	0.607	0.787	0.478	0.467
SC-LS [16]	0.544	0.759	–	–
DEC [25]	0.379	0.575	0.292	0.357
IDEC [26]	0.381	0.574	0.290	0.357
DCEC [27]	0.592	0.722	0.541	0.515
SDCN [32]	0.400	0.609	0.345	0.360
IDCEC	**0.632**	**0.793**	**0.595**	**0.567**

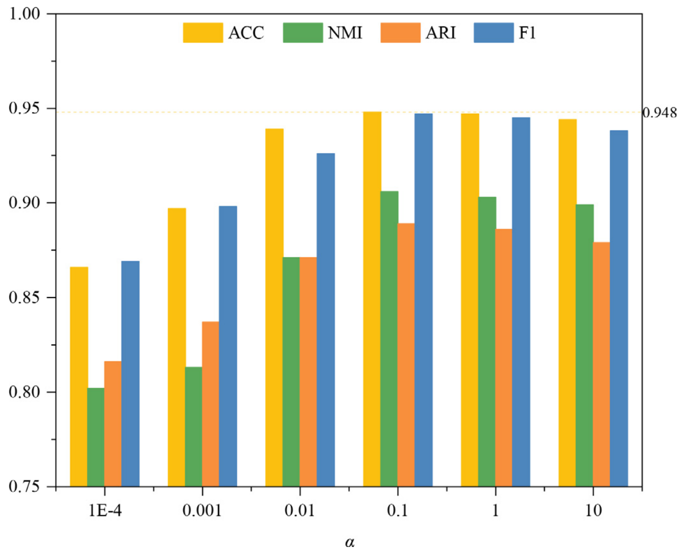

Fig. 7. Selection of the balance coefficient α on the MNIST-full dataset.

data sets, and all evaluation indicators are better than the methods compared in the experiment. The indicator ACC reached 94.8%, 92.3%, 81.2%, and 63.2% in each data set.

4.6. Parameter analysis

For the total loss function in formula (1), the value of α refers to the paper [27], and 0.1 is the most appropriate. We also discussed this parameter on the MNIST-full dataset. The value of α is selected from $\{10^{-4}, 10^{-3}, \ldots, 10^{1}\}$, and the results are shown in Fig. 7.

This paper carried out several experimental analyses on the β parameter, and the experimental results are shown in Fig. 8, which shows that the value of β is 1e−6, which has the best effect. When the value of β is large, the L2 regular term of weight and bias has a greater impact on the total loss function, thereby reducing the impact of reconstruction loss and clustering loss on the model. When the value of β is small, the constraints on the weight and bias terms are weakened.

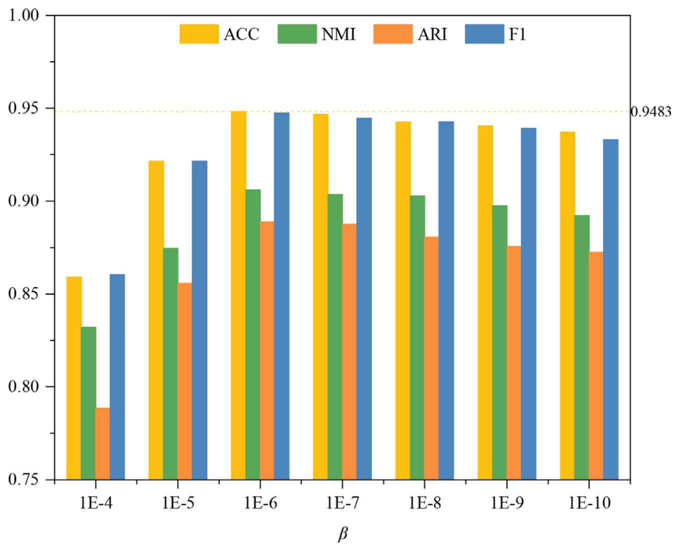

Fig. 8. Selection of the balance coefficient β on the MNIST-full dataset.

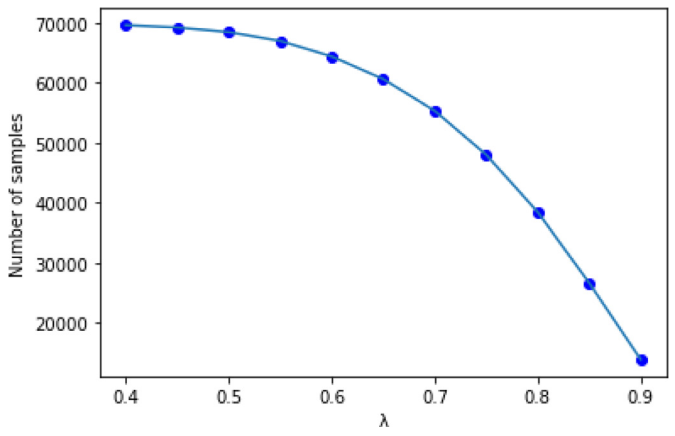

Fig. 9. The influence of the value of the parameter λ on the number of reliable samples in the MNIST-full dataset.

The choice of reliable samples is related to the distance threshold parameter λ. In order to find a suitable value of λ, we try to take the value of λ in the experiment. Since the features of the sample and the center point have been normalized and the dot product is used to calculate the distance, the distance L from each sample to the cluster center is between 0 and 1. In the experiment, λ takes a value every 0.05 in [0.4, 0.9]. Taking the MNIST-full dataset as an example, the experimental results are shown in Figs. 9 and 10.

According to Fig. 9, as the distance threshold λ becomes larger, the number of reliable samples that meet the conditions gradually decreases. Combining the four evaluation indicators in Fig. 10, shows that the distance threshold λ is more appropriate between 0.55 and 0.75. When the value of λ is too low, the distance constraint is relaxed, and a large number of samples can be selected. That is, when $\lambda = 0$, all samples are selected, and the retraining of the model loses its meaning. When the value of λ is too high, a large number of samples are not selected. Only some samples that are close to the center point are selected as reliable samples. In this case, it is easy to cause too few sample points and overfitting of the model. In the design of the CNN model, considering that only reliable samples are selected to participate in the training of the CNN network, the model must prevent problems caused by overfitting, so a dropout layer with a drop rate of 0.5 is added

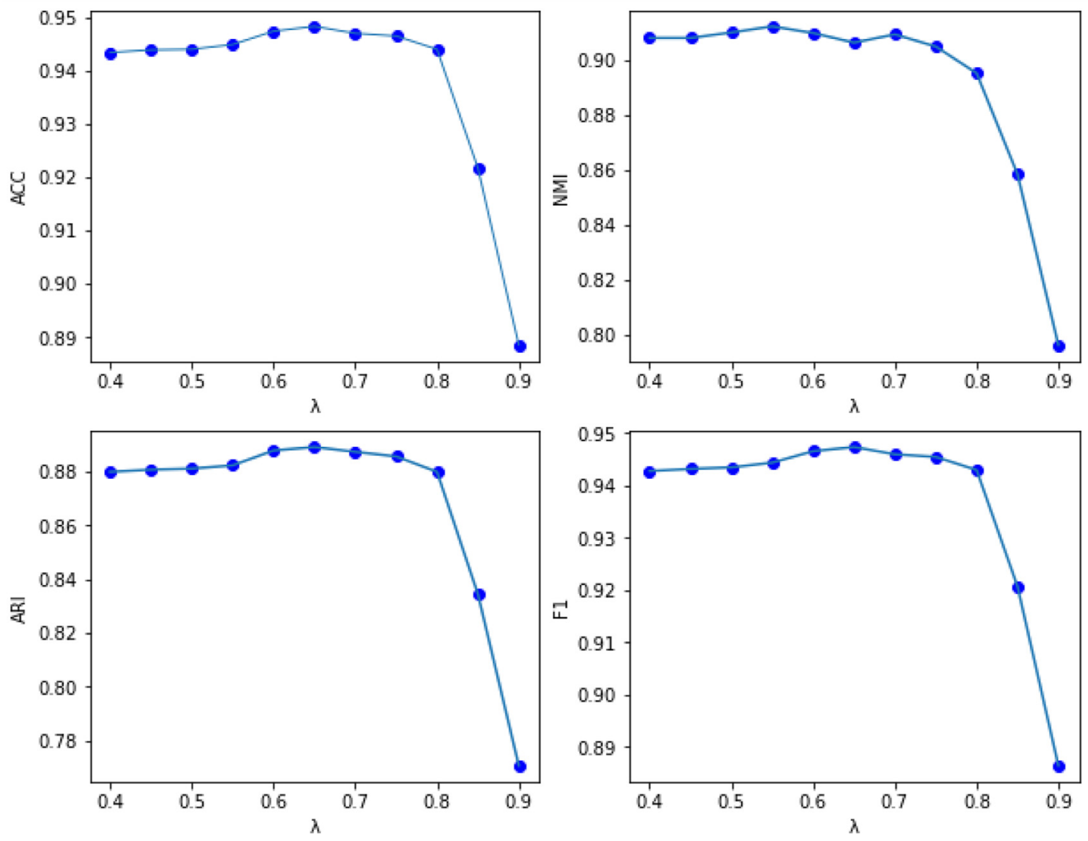

Fig. 10. The influence of the value of λ on each indicator in the MNIST-full dataset.

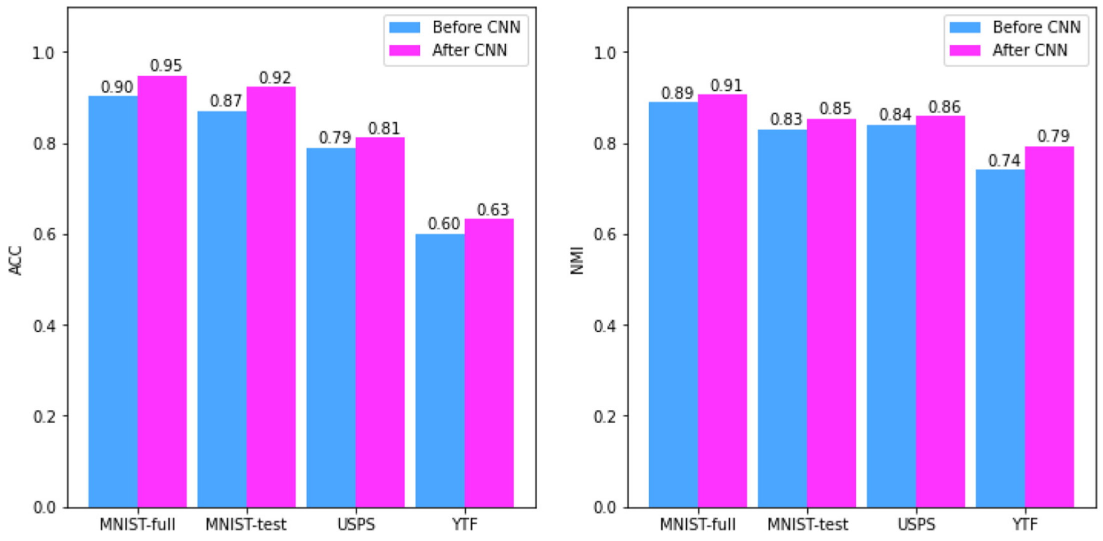

Fig. 11. Comparison of the results of ACC and NMI in each data set before and after CNN network training.

before the last fully connected layer of the CNN network, and the input sample is randomly rotated and scaled to increase its complexity.

4.7. Evaluation of re-selectable samples training

In order to verify the effect of the sample reselection on the entire algorithm, this paper compares the effect before sample reselection and the effect obtained by training the CNN network after selecting a reliable sample. The experimental results are shown in Fig. 11.

Fig. 11 shows by selecting samples with higher reliability and using them as the input of the CNN network for training, the model can obtain better classification results.

4.8. Clustering visualization in module A and module B

Module A mainly uses a convolutional autoencoder with a clustering layer for feature extraction and clustering. Module B takes module A as the upstream task and selects samples from the pseudo-labels obtained by clustering. Then it uses reliable samples to conduct self-supervised training. Finally, the whole samples are

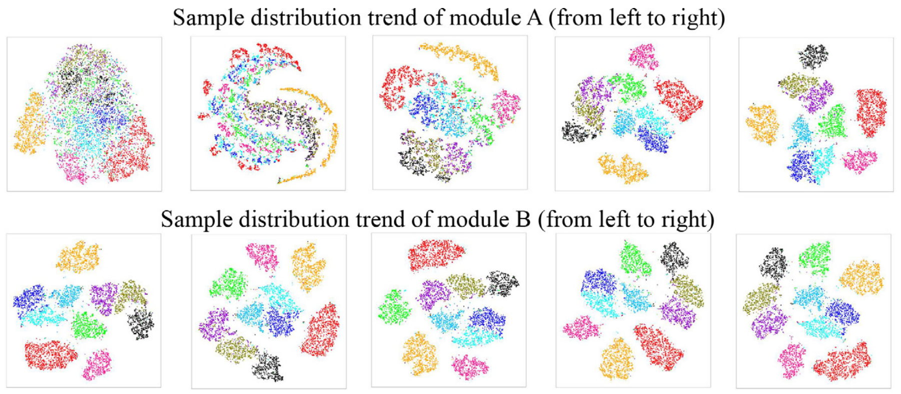

Fig. 12. Visualize the distribution of samples on the USPS dataset in module A and modele B

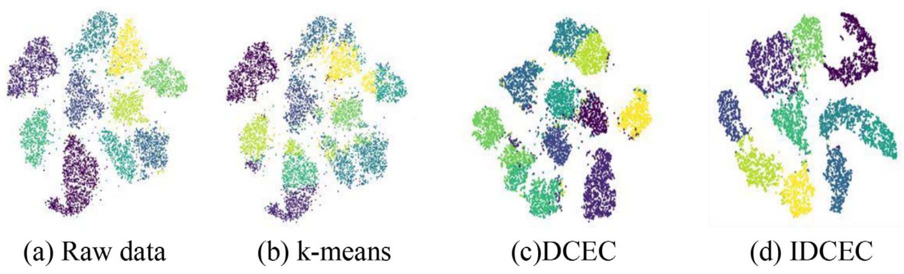

Fig. 13. Visualize the distribution of samples with different algorithms on the USPS dataset.

input for testing, and the final result is obtained. In this section, we visualized the sample distribution of these two modules respectively to observe how the sample distribution changes in the process of model training. We reduced the dimension of the extracted sample features and then visualized the overall distribution of our samples in the two-dimensional coordinate system. We use different colors to distinguish different sample categories. As shown in Fig. 12, the first line is the sample visualization in the training process of module A, and the second line is the sample visualization in the training process of module B.

4.9. Visualizing the discriminative distribution of samples with different algorithms.

In order to clearly show the distribution of samples in the feature space and the clustering effect of the algorithm proposed in this paper, we use the t-SNE algorithm to reduce the feature dimension of the sample to two dimensions, and then visualize it, as shown in Fig. 13. Fig. 13(a) shows the data distribution in the original feature space. Fig. 13(b) shows the sample distribution after *k*-means clustering. Fig. 13(c) shows the sample distribution after clustering with the DCEC model. Fig. 13(d) shows the sample distribution after clustering with the model proposed in this paper. It can be seen that the IDCEC model has a better division of each sample, and the clustering effect is better.

5. Conclusion

This paper proposes an improved deep convolutional embedded clustering algorithm using reliable samples. The algorithm combines the convolutional autoencoder model with a clustering layer and the sample reselection mechanism. After the autoencoder model is trained, each sample is assigned a pseudo-label through the clustering layer. The samples in each cluster that are close to the center are then selected and recorded as reliable samples and allowed to participate in the training of the CNN model. The model also demonstrated good test performance on four image data sets, which also confirmed the effectiveness of the algorithm. Our work is mainly based on the idea of self-supervised learning, and the clustering results obtained by upstream tasks can improve the model's performance. The retraining idea of reliable samples in this paper can be applied to other deep clustering models, but its disadvantages are obvious. The model will be affected by upstream tasks. Therefore, we should select a model with higher accuracy to pre-train the baseline when selecting it. In the future, we intend to apply the deep clustering model proposed in this paper to a wider range of realistic data sets. For example, we can apply the model to medical image processing and other applications [38,39].

Declaration of Competing Interest

Authors declare that they have no conflict of interest.

Acknowledgments

This work was supported by the Postgraduate Research & Practice Innovation Program of Jiangsu Province (Project no. SJCX20_1416).

References

[1] J. MacQueen, et al., Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, Oakland, CA, USA, 1967, pp. 281–297.
[2] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: Kdd, vol. 96, 1996, pp. 226–231.
[3] A. Ng, M. Jordan, Y. Weiss, On spectral clustering: analysis and an algorithm, Adv. Neural Inf. Process. Syst. 14 (2001) 849–856.
[4] X. Yang, C. Deng, X. Liu, F. Nie, New l 2, 1-norm relaxation of multi-way graph cut for clustering, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[5] S.C. Johnson, Hierarchical clustering schemes, Psychometrika 32 (3) (1967) 241–254.
[6] H. Koga, T. Ishibashi, T. Watanabe, Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing, Knowl. Inf. Syst. 12 (1) (2007) 25–53.
[7] S. Affeldt, L. Labiod, M. Nadif, Spectral clustering via ensemble deep autoencoder learning (SC-EDAE), Pattern Recognit. 108 (2020) 107522.

[8] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: International Conference on Artificial Neural Networks, Springer, 2011, pp. 52–59.
[9] F. Li, H. Qiao, B. Zhang, Discriminatively boosted image clustering with fully convolutional auto-encoders, Pattern Recognit. 83 (2018) 161–173.
[10] X.-J. Shen, S.-X. Liu, B.-K. Bao, C.-H. Pan, Z.-J. Zha, J. Fan, A generalized least--squares approach regularized with graph embedding for dimensionality reduction, Pattern Recognit. 98 (2020) 107023.
[11] Q. Ji, Y. Sun, J. Gao, Y. Hu, B. Yin, Nonlinear subspace clustering via adaptive graph regularized autoencoder, IEEE Access 7 (2019) 74122–74133.
[12] Q. Feng, L. Chen, C.P. Chen, L. Guo, Deep fuzzy clustering a representation learning approach, IEEE Trans. Fuzzy Syst. 28 (7) (2020) 1420–1433.
[13] M. Filippone, F. Camastra, F. Masulli, S. Rovetta, A survey of kernel and spectral methods for clustering, Pattern Recognit. 41 (1) (2008) 176–190.
[14] S. Vega-Pons, J. Ruiz-Shulcloper, A survey of clustering ensemble algorithms, Int. J. Pattern Recognit. Artif. Intell. 25 (03) (2011) 337–372.
[15] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 22 (8) (2000) 888–905.
[16] X. Chen, D. Cai, Large scale spectral clustering with landmark-based representation, in: Twenty-fifth AAAI Conference on Artificial Intelligence, Citeseer, 2011.
[17] H. Lu, Y. Song, H. Wei, Multiple-kernel combination fuzzy clustering for community detection, Soft Comput. 24 (18) (2020) 14157–14165.
[18] H. Lu, S. Liu, H. Wei, J. Tu, Multi-kernel fuzzy clustering based on auto-encoder for fMRI functional network, Expert Syst. Appl. 159 (2020) 113513.
[19] J. Sun, X. He, M. Wu, X. Wu, J. Shen, B. Lu, Detection of tomato organs based on convolutional neural network under the overlap and occlusion backgrounds, Mach. Vis. Appl. 31 (2020) 1–13.
[20] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 1096–1103.
[21] S. Ryu, H. Choi, H. Lee, H. Kim, Convolutional autoencoder based feature extraction and clustering for customer load analysis, IEEE Trans. Power Syst. 35 (2) (2019) 1048–1060.
[22] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, arXiv preprint arXiv:1312.6114(2013).
[23] K.-L. Lim, X. Jiang, C. Yi, Deep clustering with variational autoencoder, IEEE Signal Process. Lett. 27 (2020) 231–235.
[24] C. Tao, H. Pan, Y. Li, Z. Zou, Unsupervised spectral–spatial feature learning with stacked sparse autoencoder for hyperspectral imagery classification, IEEE Geosci. Remote Sens. Lett. 12 (12) (2015) 2438–2442.
[25] J. Xie, R. Girshick, A. Farhadi, Unsupervised deep embedding for clustering analysis, in: International Conference on Machine Learning, 2016, pp. 478–487.
[26] X. Guo, L. Gao, X. Liu, J. Yin, Improved deep embedded clustering with local structure preservation, in: IJCAI, 2017, pp. 1753–1759.
[27] X. Guo, X. Liu, E. Zhu, J. Yin, Deep clustering with convolutional autoencoders, in: International Conference on Neural Information Processing, Springer, 2017, pp. 373–382.
[28] R. Zhang, X. Li, H. Zhang, F. Nie, Deep fuzzy *k*-means with adaptive loss and entropy regularization, IEEE Trans. Fuzzy Syst. 28 (11) (2019) 2814–2824.
[29] Q. Yin, J. Zhang, S. Wu, H. Li, Multi-view clustering via joint feature selection and partially constrained cluster label learning, Pattern Recognit. 93 (2019) 380–391.
[30] P. Ji, T. Zhang, H. Li, M. Salzmann, I. Reid, Deep subspace clustering networks, Adv. Neural Inf. Process. Syst. 30 (2017) 24–33.
[31] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, 2017, pp. 173–182.
[32] D. Bo, X. Wang, C. Shi, M. Zhu, E. Lu, P. Cui, Structural deep clustering network, in: Proceedings of The Web Conference 2020, 2020, pp. 1400–1410.
[33] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, D. Cremers, Clustering with deep learning: taxonomy and new methods, arXiv preprint arXiv:1801.07648(2018).
[34] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, C. Zhang, Attributed graph clustering: a deep attentional embedding approach, in: International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence, 2019.
[35] P. Zhou, Y. Hou, J. Feng, Deep adversarial subspace clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1596–1604.
[36] Y. Opochinsky, S.E. Chazan, S. Gannot, J. Goldberger, K-autoencoders deep clustering, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2020, pp. 4037–4041.
[37] J. Cai, S. Wang, W. Guo, Unsupervised embedded feature learning for deep clustering with stacked sparse auto-encoder, Expert Syst. Appl. 186 (2021) 115729.
[38] X. Geng, H. Lu, J. Sun, Network structural transformation-based community detection with autoencoder, Symmetry 12 (6) (2020) 944.
[39] H. Lu, S. Liu, H. Wei, C. Chen, X. Geng, Deep multi-kernel auto-encoder network for clustering brain functional connectivity data, Neural Netw. 135 (2021) 148–157.

Hu Lu received the Ph.D. degree from Fudan University in 2013. He was a postdoctoral fellow at the School of software, Fudan University from 2013 to 2015. He was a visiting scholar with the Department of Computer and Information Sciences, Temple University, USA in 2016. He is currently with the School of Computer Science and Communication Engineering, Jiangsu University, Jiangsu, China, where he serves as an Associate Professor. His main research interests include computer vision, machine learning, deep learning and brain cognitive function.

Chao Chen received the B.E. degree from Huaiyin Institute of Technology, Jiangsu, China. He is currently working toward the M.Eng. degree with the School of Computer Science and Communication Engineering, Jiangsu University, Jiangsu, China. His research interests include unsupervised learning and deep learning.

Hui Wei received the Ph.D. degree from the Department of Computer Science, Beijing University of Aeronautics and Astronautics, Beijing, China, in 1998.He was a Post-Doctoral Fellow with the Department of Computer Science and the Institute of Artificial Intelligence, Zhejiang University, Zhejiang,China, from 1998 to 2000. Since 2000, he has been with the Department of Computer Science and Engineering, Fudan University, Shanghai, China. Hiscurrent research interests include artificial intelligence and cognitive science.

Zhongchen Ma received his B.S. degree in Information and Computing Science from Qingdao Agricultural University in 2012. In 2015 and 2019, he respectively completed his M.S. and Ph.D. degree in computer science and technique at Nanjing University of Aeronautics and Astronautics. He is currently working at Jiangsu University. His research interests include pattern recognition and machine learning.

Ke Jiang received the B.E. degree from Jiangsu University Jingjiang College, Jiangsu, China. He is currently working toward the M.Eng. degree with the School of Computer Science and Communication Engineering, Jiangsu University, Jiangsu, China. His research interests include computer vision and deep learning.

YingQuan Wang received the B.S. degree from Anhui University, Anhui, China, in 2017. He is currently working toward the MA.Sc degree with the School of Computer Science and Communication Engineering, Jiangsu University, Jiangsu, China. His research interests include video understanding and person re-identification.