Exploring Clustering and Dimensionality Reduction in Data Analysis

Zikun Zhuang
Southern University of Science and Technology
Shenzhen, China
12110418@mail.sustech.edu.cn

***Abstract*—This project investigates the application of clustering techniques and dimensionality reduction methods in data analysis. Implementing K-means and Soft K-means algorithms in Python using Numpy, we evaluate their performance on a diverse dataset with seven input features and three distinct labels. The project explores clustering effectiveness without label information and introduces non-local split-and-merge moves to enhance algorithmic accuracy. Additionally, dimensionality reduction techniques, PCA and Linear Autoencoder, are implemented to showcase their versatility.**

***Index Terms*—clustering, K-means, Soft K-means, dimensionality reduction, PCA, linear auto encoder, data analysis, non-local split-and-merge moves**

I. Introduction

This project focuses on developing machine learning models for clustering and dimensionality reduction tasks and evaluating their performance.

For the clustering task, KMeans and Soft KMeans algorithm models are implemented using Python and Numpy, with the seed dataset serving as the basis for evaluating their performance. Horizontal comparison of model accuracy is conducted by assessing clustering results against true labels. Optimization techniques, including refining the initialization method for cluster centers, are employed to enhance the efficiency of clustering.

In the dimensionality reduction task, Principal Component Analysis (PCA) and Linear Autoencoder models are constructed using Python and Numpy. Evaluation involves comparing dimensionality reduction and reconstruction results for both RGB color images and grayscale images. The similarity between reconstructed and original images serves as a performance indicator. Additionally, due to constraints in Python and Numpy's data handling capabilities, the Linear Autoencoder is tested using images from the Fashion MNIST dataset.

Optimization of the Linear Autoencoder model involves adjusting the dimensions (number of neurons) of each hidden layer, aiming for dimensionality reduction with minimal components while achieving close recovery of original images.

II. Problem formulation

The project addresses several key challenges inherent in data analysis:

- Clustering Analysis: Evaluate the performance of K-means and Soft K-means clustering algorithms on datasets with diverse characteristics. Assess their effectiveness in uncovering patterns without using label information.
- Impact of Cluster Number: Investigate the consequences of setting K=10 in both K-means and Soft K-means. Analyze the algorithms' ability to handle an increased number of clusters and differentiate finer distinctions within the dataset.
- Algorithm Enhancement: Modify clustering algorithms by incorporating non-local split-and-merge moves. Observe the impact of these enhancements on clustering accuracy, particularly with a higher number of clusters.
- Dimension Reduction: Implement PCA and Linear Autoencoder to extract essential features and reconstruct original data. Showcase the potential of these methods in simplifying data representation across diverse datasets.

The project aims to provide insights into the strengths and limitations of clustering and dimensionality reduction techniques, offering valuable implications for a broad range of data analysis scenarios.

III. Method and algorithms

A. Methods

This project employs methods such as model construction, dataset testing, control variables, and cross-model comparisons to develop suitable clustering and data dimensionality reduction models. The models are optimized through testing on various datasets.

The KMeans and SoftKMeans clustering models are compared through clustering analysis on the wheat seed dataset and color images. Non-local split-and-merge moves are introduced to enhance the model, and the performance is further evaluated by increasing the number of clusters in the wheat seed dataset.

For the data dimensionality reduction model, linear autoencoders are trained on Google's Fashion MNIST dataset (comprising 28*28 grayscale images) and the CIFAR-10 dataset (comprising 32*32 color images). The original images are analyzed and compared to assess the model performance. Additionally, PCA models are applied to extract features from these two datasets. The reconstructed images from PCA are compared with those from linear autoencoders to analyze the performance of these two distinct models.

B. Algorithms

*1) **KMeans**:* The KMeans algorithm achieves clustering by dividing data into different categories through the initialization of centroids, centroid movements, and cluster updates. The loss function for KMeans can be defined as the sum of the Euclidean distances between each clustered data point and its cluster center:

$$\min_{\{\mathbf{m}\},\{\mathbf{r}\}} J(\{\mathbf{m}\},\{\mathbf{r}\}) = \min_{\{\mathbf{m}\},\{\mathbf{r}\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2 \tag{1}$$

Here, $\mathbf{x} \in \mathbb{R}^{n\times d}$ represents n samples of d-dimensional input. $\mathbf{m} \in \mathbb{R}^{k\times d}$ is the cluster center, and k is the number of clusters. $r_k^{(n)} \in \{0,1\}$ is the identifier for sample classification, using one-hot encoding, where 1 indicates that the data point belongs to the k-th cluster center.

The algorithm iterates to minimize the loss function, and the specific steps are as follows:

1) **Initialize Cluster Centers:** For a given dataset, set k cluster centers and initialize them through methods such as random number generation.
2) **Cluster:** Determine the category to which each data point belongs by selecting the nearest cluster center based on the distance between each data point and the cluster centers.

$$\hat{k}^{(n)} = \arg\min_k ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2 \tag{2}$$

and

$$r_k^{(n)} = 1 \leftrightarrow \hat{k}^{(n)} = k \tag{3}$$

3) **Move Cluster Centers:** After the clustering step, move the cluster centers to the center of the corresponding clustered data points.

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{\sum_n r_k^{(n)}} \tag{4}$$

4) **Update:** After moving the cluster centers, repeat steps 2 and 3 until the cluster centers no longer change or reach the maximum number of iterations.

*2) **Soft KMeans**:* To introduce Soft KMeans and allow the model to capture more data information during the clustering process, the clustering identifier $r_k^{(n)}$ is adjusted at each clustering update using the following expression:

$$r_k^{(n)} = \frac{\exp[-\beta||\mathbf{m}_k - \mathbf{x}^{(n)}||^2]}{\sum_j \exp[-\beta||\mathbf{m}_j - \mathbf{x}^{(n)}||^2]} \tag{5}$$

The process of updating cluster centers remains unchanged.

In this algorithm, the hyperparameter β is used to adjust the degree of soft assignment. A smaller β results in a smoother soft assignment, while a larger β leads to a more concentrated soft assignment, tending toward a hard assignment.

*3) **KMeans++**:* To minimize the risk of overlapping cluster centers and getting trapped in local optima, the KMeans++ method is introduced for initializing the cluster centers of both KMeans and SoftKMeans. [1]

The initialization process for K-Means++ centers is as follows:

1) Randomly select the first center from the data points.
2) For each data point, calculate its shortest distance to the already selected centers (i.e., the distance to the nearest center).
3) Select the next center according to the probability distribution of these distances. Points that are farther away have a higher probability of being chosen as the next center.
4) Repeat steps 2 and 3 to select the remaining centers until k initial centers are chosen.

This way, K-Means++ ensures a widespread distribution of initial center points, making it more likely for the algorithm to find the global optimum and reducing the risk of converging to a local optimum.

*4) **Non-Local Split-and-Merge Moves**:* In the K-Means algorithm, the choice of initial cluster centers can significantly impact the final clustering results. The introduction of non-local split-and-merge moves in the iterative process of K-Means provides a dynamic mechanism for updating cluster centers, addressing the challenges associated with initial center selection and enhancing clustering accuracy.

Specifically, the role of non-local split-and-merge moves is as follows:

1) **Splitting:** During the iterations of K-Means, if a cluster exhibits uneven data distribution or contains multiple subclusters, non-local split-and-merge moves attempt to further split the cluster into smaller subclusters, aiming to better capture the intrinsic structure of the data.
2) **Merging:** Conversely, if a cluster has sparse data distribution or unnecessary subdivisions occur during the clustering process, non-local split-and-merge moves can attempt to merge similar subclusters, reducing unnecessary details and enhancing the overall performance of clustering.

Through these dynamic split-and-merge operations, non-local split-and-merge moves assist the K-Means algorithm in adapting better to diverse data distributions, improving the effectiveness and robustness of clustering.

*5) **PCA**:* The Principal Component Analysis (PCA) algorithm aims to achieve dimensionality reduction by extracting the principal components of the input data. This algorithm effectively compresses the dataset, and the compressed dataset can be used to reconstruct data similar to the original dataset.

Data Compression:

For any input data $\mathbf{x} \in \mathbb{R}^d$, the compressed reconstruction can be approximated as

$$\mathbf{x} \approx U\mathbf{z} + \mathbf{a} \tag{6}$$

where $U \in \mathbb{R}^{d\times m}$ is the projection matrix, $\mathbf{z} \in \mathbb{R}^m$ is the compressed data, and $\mathbf{a} \in \mathbb{R}^d$ is the central offset compensation.

To find the projection matrix U, construct the covariance matrix $C \in \mathbb{R}^{d\times d}$

$$C = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}^{(n)} - \bar{\mathbf{x}})(\mathbf{x}^{(n)} - \bar{\mathbf{x}})^T \tag{7}$$

where M largest eigenvalues correspond to the sought principal components. Thus, the covariance matrix can be expressed as

$$C = U\Sigma U^T \approx U_{1:M}\Sigma_{1:M}U_{1:M}^T \tag{8}$$

where U is the required projection matrix composed of the eigenvectors corresponding to the M largest eigenvalues. Additionally, the matrix U is unitary

$$U^TU = UU^T = 1 \tag{9}$$

Σ is a diagonal matrix composed of the eigenvalues of the covariance matrix.

After obtaining the projection matrix U, the original data $\mathbf{x}$ can be projected to obtain the compressed data $\mathbf{z}$.

$$\mathbf{z} = U^T\mathbf{x} \tag{10}$$

PCA satisfies the properties of maximizing variance (green points) and minimizing error (red points), i.e., the distances between the projected points $\widetilde{\mathbf{x}}_n$ along the direction of $\mathbf{u}$ are maximized, and the distance between the projected data $\widetilde{\mathbf{x}}_n$ and the original data $\mathbf{x}_n$ is minimized. **Data Reconstruction:**

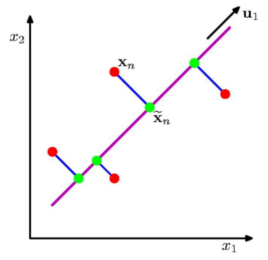

Fig. 1. Two Derivations of PCA

After completing the dimensionality reduction with PCA, the data can be reconstructed based on the dimensionality reduction process, recovering data $\widetilde{\mathbf{x}}_n$ that is approximately similar to the original data. The reconstruction process aims to minimize the reconstruction error function:

$$J(\mathbf{u}, \mathbf{z}, \mathbf{b}) = \sum_n ||\mathbf{x}^{(n)} - \widetilde{\mathbf{x}}^{(n)}||^2 \tag{11}$$

where

$$\widetilde{\mathbf{x}}^{(n)} = \sum_{j=1}^{M} z_j^{(n)}\mathbf{u}_j + \sum_{j=M+1}^{D} b_j\mathbf{u}_j \tag{12}$$

The values of $z_j^{(n)}$ and b_j that minimize the reconstruction function are given by:

$$z_j^{(n)} = \mathbf{u}_j^T\mathbf{x}^{(n)} \tag{13}$$

$$b_j = \bar{\mathbf{x}}^T\mathbf{u}_j \tag{14}$$

The process of obtaining $z_j^{(n)}$ is the inverse of the projection process, and b_j serves as the central offset compensation.

*6) **Linear Auto Encoder:*** Linear autoencoders, similar to Multi-Layer Perceptrons (MLP), take input data as their own output, leveraging neural networks to learn and extract features from the data. They have the capability to compress the data into a lower-dimensional space and reconstruct data that is almost identical to the original. Linear autoencoders can be

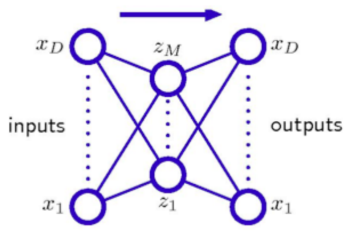

Fig. 2. Linear Auto Encoder Neural Network Structure

divided into two parts: the encoder, responsible for reducing the data to a lower dimension, and the decoder, which reconstructs the data from the reduced dimension. After training, the data in the middle layer, with the lowest dimension, serves as the desired reduced-dimensional data.

The key distinction between linear autoencoders and MLP models lies in the output layer, which does not require an activation function, resulting in a linear output:

$$\mathbf{y} = \mathbf{w}_0 + \mathbf{w}^T\mathbf{h} \tag{15}$$

IV. Experiment results and analysis

A. *KMeans and Soft KMeans*

Performance evaluation of KMeans and Soft KMeans is conducted on the wheat seed dataset, which contains 210 data points, each with 7 features. The dataset has three labels, hence the number of clusters is set to $K = 3$.

Comparing the true labels of the dataset with the clustering results of both models, the accuracy of the KMeans model is 71.03%, while the clustering accuracy of the Soft KMeans model (with hyperparameter $\beta = 10$) is 71.67%. The performance of the two models is similar. This improvement may be attributed to the introduction of the KMeans++ initialization method, which significantly enhances the clustering performance of both models, preventing them from getting stuck in local optima and narrowing the gap in clustering effectiveness. However, it is noticeable that Soft KMeans outperforms KMeans, likely due to Soft KMeans incorporating more data information, thereby avoiding overly absolute clustering results.

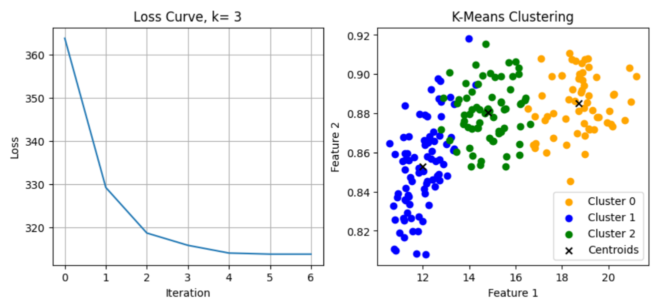

Fig. 3. KMeans clustering results with K=3

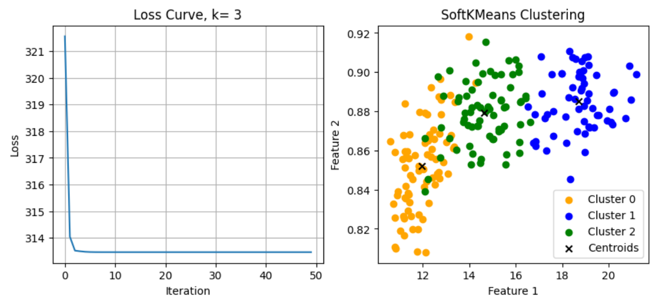

Fig. 4. Soft KMeans clustering results with K=3

Additionally, by setting $K = 10$, clustering is re-performed using the KMeans and SoftKMeans (with hyperparameter $\beta = 10$) models. The final loss function of the KMeans model decreases to 201.25, while the loss function of the SoftKMeans model decreases to 190.67. It is evident that Soft KMeans performs better than KMeans when the number of clusters is increased. This is attributed to Soft KMeans acquiring more data information, allowing it to adapt more effectively to a higher number of clusters. Building upon $K = 10$, the

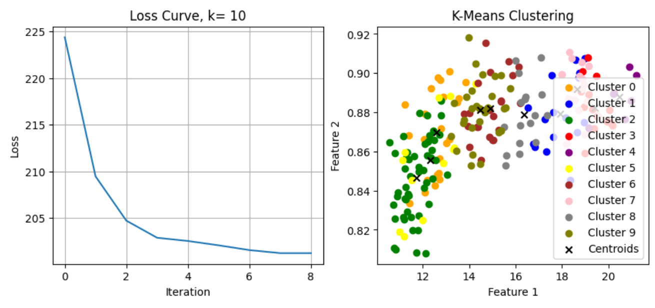

Fig. 5. KMeans clustering results with K=10

introduction of non-local split-and-merge moves enhances the adaptive capability of the KMeans model. By setting a threshold for split and merge, the data is clustered again using both the KMeans and Soft KMeans models. Selecting the optimal split-and-merge thresholds for each model, it is observed that, after clustering, both models return to 3 clusters, achieving accuracies identical to those when $K = 3$ (KMeans: 71.03%, SoftKMeans: 71.67%). This indicates that non-local split-and-merge moves effectively enhance the model's adaptability. The

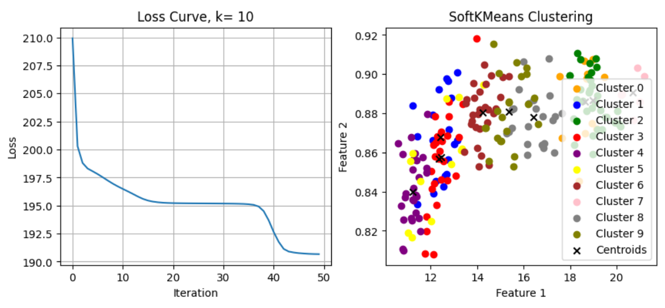

Fig. 6. Soft KMeans clustering results with K=10

models can still perform well when faced with an unknown number of clusters. Additionally, observing the loss function

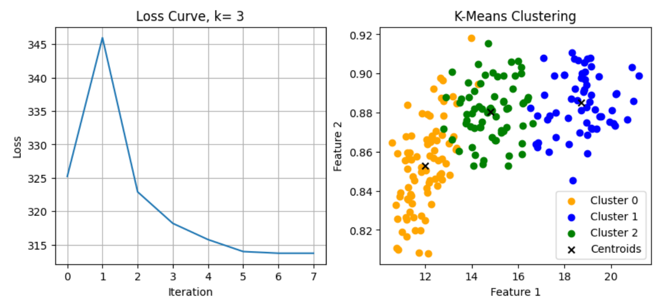

Fig. 7. NLSM KMeans clustering results with initial K=10

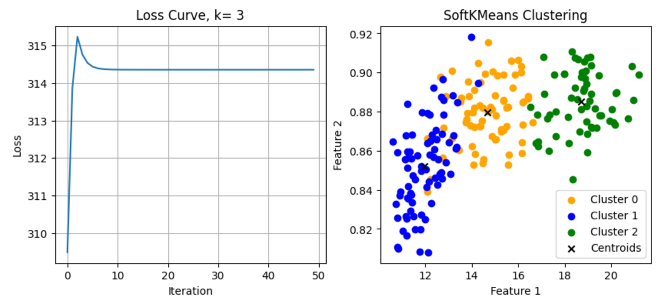

Fig. 8. NLSM Soft KMeans clustering results with initial K=10

curve reveals an initial increase followed by a decrease. This pattern is attributed to the merging of clusters during the iterative process of clustering. Afterward, the number of clusters stabilizes, and the loss begins to gradually decrease.

B. *PCA*

In this project, PCA is employed for feature extraction from both color and grayscale images. Color images consist of three features corresponding to the RGB channels of each pixel. PCA is utilized to extract the most significant color features, reducing the data to one dimension. On the other hand, grayscale images, characterized by pixel values, undergo PCA for extracting primary pixel features to compress the pixel information.

*1) **Processing Color Images:*** Setting the number of principal components to 1, feature extraction is performed on color images from the Google CIFAR-10 dataset. The most significant color feature is extracted and compared with the original image. Through the comparison, it is observed that although the reconstructed image consists of a single color, it still preserves almost all the shape features of the original image. This demonstrates that PCA is effective in dimensionality reduction and reconstruction of images while retaining their key features. Increasing the number of principal components

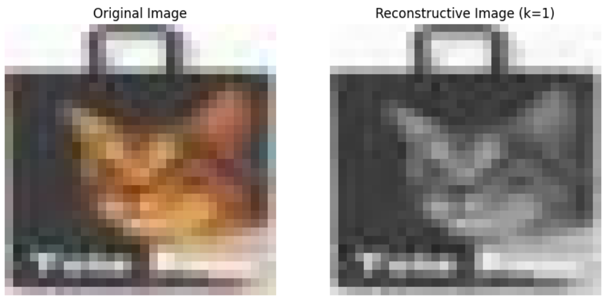

Fig. 9. PCA processing color image with k=1

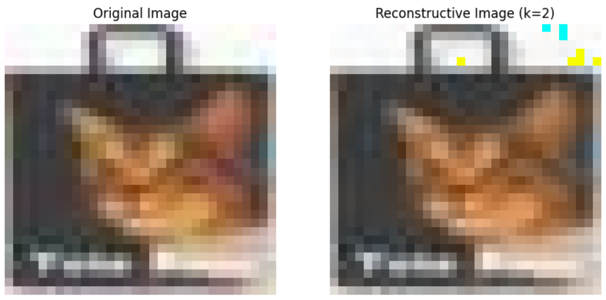

Fig. 10. PCA processing color image with k=2

to 2, the dimensionality-reduced and reconstructed image closely resembles the original image.

*2) **Processing Grayscale Images:*** In contrast to color images, the processing of grayscale images involves extracting pixel features for dimensionality reduction and reconstruction. Using a 640x640 grayscale image, where column pixels are considered the primary features, with the number of features set to 40, PCA is employed to compress and reconstruct the image. Comparing the original and reconstructed images reveals that, despite retaining only 6.25% of pixel information, the grayscale image reconstructed is still consistent with the primary features of the original image.

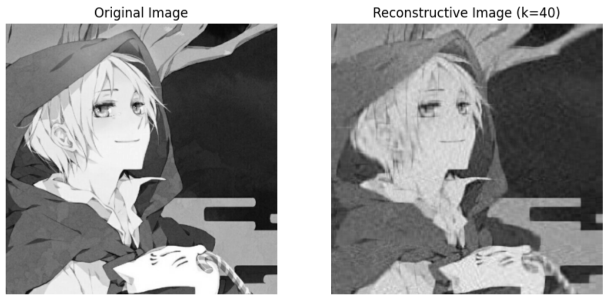

Fig. 11. PCA processing grayscale image with k=40

*C. **Linear Autoencoder***

Similar to PCA, linear autoencoders are trained on both color and grayscale images. The middle hidden layer is extracted as the compressed data, and data reconstruction is achieved through the latter part of the neural network.

For grayscale images, the Google Fashion MNIST dataset is used, consisting of 28x28 pixel grayscale images. The neural network is configured with a hidden layer structure of [20,10,20] (the middle 10-dimensional hidden layer serves as the compressed data). With a learning rate of $\lambda = 0.8$, the grayscale image is compressed to one dimension (784 pixel features in total). After 30 iterations, the loss function converges to 0. Comparing the input and reconstructed images reveals that using only 1.2% of the data, the linear autoencoder can reconstruct images identical to the original, showcasing its excellent feature extraction capability in dimensionality reduction tasks.

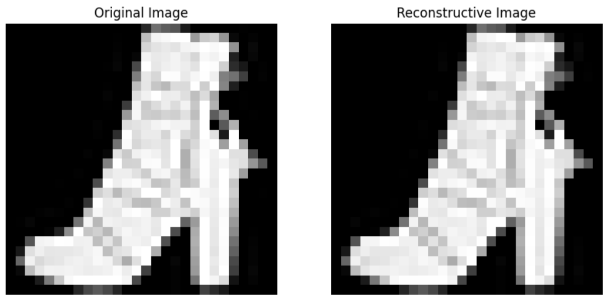

Fig. 12. Linear AutoEncoder processing grayscale image

Color images, using the same images as PCA, have dimensions of 32x32x3. The hidden layer structure is set to [20,10,20], with a learning rate of $\lambda = 0.8$. The color image is compressed to one dimension (3072 pixel features in total) and passed through the linear autoencoder. After 9 iterations, the reconstructed image becomes similar to the original, and after 30 iterations, the loss function converges to 0. The reconstructed image is consistent with the original, demonstrating the effectiveness of the linear autoencoder in compressing color images while retaining their essential features.

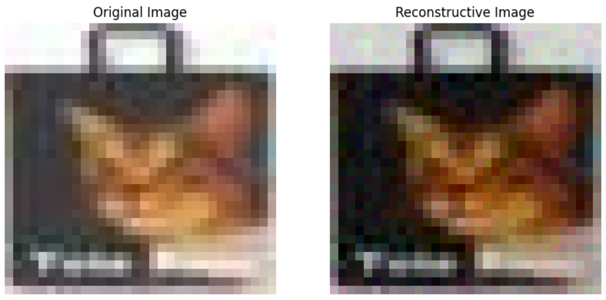

Fig. 13. Linear AutoEncoder processing color image(Epoch=9)

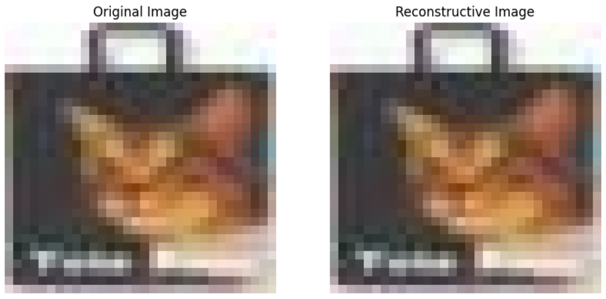

Fig. 14. Linear AutoEncoder processing color image(Epoch=30)

Simultaneously, comparing the dimensionality-reduced and reconstructed results from PCA with those from the linear autoencoder reveals that the linear autoencoder preserves almost all the original features, effectively compressing and restoring the images. In contrast, PCA, even after extracting 1 to 2 color features, exhibits some feature loss, demonstrating comparatively weaker performance compared to the linear autoencoder.

D. Clustering with SoftKMeans

To further explore the clustering capability of the SoftKMeans model, a comparison is made with PCA data reduction. The same color images as those used in the PCA model (dimension: 640x640x3) are input into the SoftKMeans model for clustering, with the number of clusters set to 10. Comparing the clustering results with the original images reveals the successful preservation of the main features of the original images.

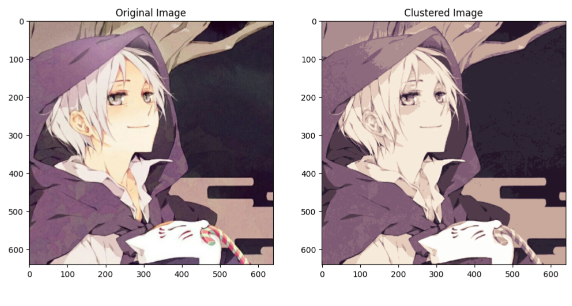

Fig. 15. Clustering color imiage with Soft KMeans(K=10)

V. Conclusion and Future Problems

*1) **Conclusion**:* This project explored machine learning models for clustering and dimensionality reduction and reconstruction tasks, including KMeans, Soft KMeans, PCA, and linear autoencoders.

In terms of clustering effectiveness, KMeans exhibited weaker clustering performance compared to Soft KMeans, attributed to Soft KMeans's ability to capture more data information. Importantly, the utilization of KMeans++ for initializing cluster centers significantly improved the performance of both KMeans and Soft KMeans clustering models. Additionally, the incorporation of non-local split-and-merge moves enhanced the stability and adaptability of these models, playing a crucial role in handling data with an unknown number of clusters.

Regarding dimensionality reduction, both PCA and linear autoencoders demonstrated excellent performance. PCA is a simpler and faster algorithm, providing relatively good feature extraction with minimal computational resources. On the other hand, linear autoencoders, leveraging their strong learning capability, excelled in the task of dimensionality reduction and reconstruction. The reconstructed images were consistent with the original images.

*2) **Future Problems**:*

1) **Local Optima Issues in KMeans and Soft KMeans Algorithms:**To prevent these clustering models from getting stuck in local optima, besides employing the KMeans++ method for initializing cluster centers, additional algorithms such as random forests could be explored. This project focused solely on KMeans++, leaving room for investigating other approaches to mitigate local optima problems.
2) **Optimization of KMeans and Soft KMeans Algorithms:** The implementation of these clustering models in Python for this project resulted in reduced efficiency. Future improvements could involve optimizing the clustering process and centroid updates by avoiding loops and utilizing matrix expressions.
3) **Optimization Challenges in Linear Autoencoder Algorithm:** The linear autoencoder implemented in this project using Python and NumPy has limitations when dealing with large datasets. It can only handle images with lower pixel counts, imposing significant constraints. Future optimizations might include refining the algorithm logic or implementing the algorithm in other programming languages, such as C++, to enhance its scalability.

References

[1] Arthur, D., & Vassilvitskii, S. (2007, January). K-means++: The advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (pp. 1027-1035).