Project
Image Clustering

Pattern Recognition

Name	ID
Bassant Mohamed	20221376715
Malak Mahmoud Aref	20221445867
Nureen Ehab Barakat	20221465124
Zainab Mohamed Abdallah	20221310251

Faculty of Computers and Data Sciences,

Alexandria University.

May 2023

Table of Contents

Introduction

Image clustering is a crucial but challenging task. Technically, image clustering is the process of grouping images into clusters such that the images within the same clusters are similar to each other, while those in different clusters are dissimilar. We have a collection of images, and our program will cluster the images based on their color and shape together using our own training dataset without using external trained model or library. The goal of our program is to count how many images in each cluster, and show the clustered images in each cluster.

Dataset

We defined our dataset by a collection of shapes (circle, rectangle, and triangle) with 3 different colors (red, blue, black). Also, we duplicate and rotated some of the shapes and colors in order to have a large dataset to make it easier for the model to learn from the large dataset.

Implementation

Import libraries

Will import all the needed libraries we will use in the project

- numpy → is a library for numerical computing in Python.
- os → the OS module in Python provides functions for interacting with the operating system.
- cv2 → is an open-source computer vision and machine learning software library.
- matplotlib → is a popular library to plot graphs in Python.

Note on OS library: ‘os’ is a Python module that provides a way to interact with the operating system. When we ‘import os’ in our Python program, we gain access to a set of functions and constants that allow us to perform various tasks related to the operating system, such as:

- Navigate the file system (‘os.chdir()’, ‘os.listdir()’, ‘os.path.join()’, etc.)
- Interact with processes (‘os.system()’, ‘os.spawn()’, etc.)
- Manage environment variables (‘os.environ’, ‘os.getenv()’, etc.)
- Manipulate file and directory permissions (‘os.chmod()’, ‘os.chown()’, etc.)

And much more!

In short, ‘import os’ adds a wide range of functionality related to the operating system to your Python program, allowing you to perform tasks that would otherwise be difficult or impossible to accomplish with pure Python code.

Data preprocessing

This process is commonly used as a preprocessing step for machine learning tasks that involve image classification or recognition. Define the preprocess_image function that takes an input image and resize the input image to a fixed size of 100x100 pixels using the 'cv2.resize' function from the OpenCV library. Then convert the resized image to grayscale using the 'cv2.cvtColor' function from OpenCV. Apply a threshold to the grayscale image to create a binary image using the 'cv2.threshold' function from OpenCV. This converts all pixel values below 127 to 0 and all pixel values above 127 to 255. Finally, flatten the binary image into a 1D numpy array using the 'numpy.flatten' function. This creates a long 1D array where each element represents a pixel in the flattened image. Then return the flattened image as the output of the 'preprocess_image' function.

Initialize Centroids

This function randomly selects 'k' data points from the input data to serve as the initial centroids for the clustering algorithm. It does this by first generating an array of random indices using 'numpy.random.choice', and then selecting the corresponding data points from the input data array. The resulting centroids are returned as an array.

Assign Labels

This function assigns each data point in the input data to one of the centroids based on their distance to each centroid. It calculates the Euclidean distance between each data point and each centroid using the 'sqrt', 'sum', and 'square' functions, and stores the resulting distances in a 2D array called 'distances'. It then assigns each data point to the centroid with the smallest distance using the 'numpy.argmin' function along the first axis (for each data point). The resulting labels are returned as a 1D array.

Update Centroids

The purpose of this function is to update the positions of the centroids based on the mean of the data points that are currently assigned to each centroid. This is done iteratively in the k-means algorithm until the centroids stop moving significantly. The function takes three arguments: 'data', 'labels', and 'k'.

Data is a 2D numpy array of shape '(n_samples, n_features)' that contains the data points to be clustered.

Labels is a 1D numpy array of shape ‘(n_samples)’ that contains the current cluster assignments for each data point. 1

K is an integer that specifies the number of clusters.

The function then initializes an empty numpy array called ‘centroids’ with a shape of ‘(k, n_features)’. This array will be used to store the updated positions of the centroids. The function then loops through each centroid index ‘i’ from 0 to ‘k-1’. For each centroid, it selects all the data points that are currently assigned to that centroid using boolean indexing with ‘labels == i’. It calculates the mean of the selected data points along the first axis (for each feature) using the ‘numpy.mean’ function with ‘axis=0’. The resulting mean is then assigned to the corresponding row of the ‘centroids’ array.

After looping through all ‘k’ centroids, the function returns the updated ‘centroids’ array. Overall, the ‘update_centroids’ function is a crucial step in the k-means algorithm as it updates the positions of the centroids based on the current cluster assignments. This allows the algorithm to iteratively refine the cluster assignments and improve the clustering results.

K-means Clustering

This function performs K-means clustering on a given dataset. The function takes three input parameters: ‘data’, which is the dataset to be clustered, ‘k’, which specifies the number of clusters to be formed, and ‘max_iterations’, which specifies the maximum number of iterations the algorithm should run.

Initially, the ‘initialize_centroids’ function is called with ‘k’ and ‘data’ parameters to choose the initial centroids randomly from the data. Then, the algorithm runs for ‘max_iterations’ iterations.

In each iteration, the ‘assign_labels’ function is called with ‘data’ and `centroids` to assign each data point to a cluster represented by its nearest centroid.

Then, the ‘update_centroids’ function is called with ‘data’, ‘labels’, and ‘k’ to calculate the new centroid coordinates for each cluster.

If the new centroids are equal to the old centroids, the algorithm will break from the loop and return the final cluster labels and centroid coordinates as output. Finally, the function returns the clusters labels and final centroid coordinates.

Main

The main() function is defined, which serves as the entry point of the script. A folder path is assigned to the variable folder_path. This variable specifies the directory where the images are located.

An empty list called images is created to store the image data.

The script iterates over each file in the specified folder using os.listdir(folder_path). **os.listdir()** method in python is used to get the list of all files and directories in the specified directory. If we don't specify any directory, then list of files and directories in the current working directory will be returned.

For each file, the script checks if it ends with either '.jpg' or '.png' extension. If the condition is true, cv2.imread() is used to read the image file. os.path.join(folder_path, filename) constructs the full path of the image file by joining the folder path and the filename. The cv2.imread() function returns a numpy array representing the image if it is successfully read, or None if there was an error in reading the image. If the image is successfully loaded (not None), it is appended to the images list.

The script then applies a preprocessing function called **preprocess_image()** to each image in the images list using a list comprehension. The resulting preprocessed images are stored in the preprocessed_images list. The preprocessed_images list is converted to a NumPy array called data.

A variable k is set to 9, representing the desired number of clusters for the k-means algorithm. We choose the k to be equal 9 beacause we has 3 different and 3 different colors. The k-means algorithm is applied to the data using a function called **k_means_clustering()**, which returns the cluster labels and centroids.

The unique labels and their counts are computed using NumPy's np.unique() function with the return_counts=True argument. They are assigned to the variables unique_labels and counts, respectively. A loop is performed to print the number of images in each cluster by iterating over unique_labels and counts simultaneously.

Another loop iterates over each label from 0 to k-1.

For each label, a list called cluster_images is created using list comprehension to filter images that belong to the current cluster label. The number of columns, num_columns, is determined based on the length of cluster_images.

If num_columns is greater than 0, a figure and axes are created using **plt.subplots()**, with len(cluster_images) as the number of columns and a fixed figure size of 10x10. The axes are flattened using **np.ravel()** for easier iteration.

A loop is performed over the flattened axes and the corresponding images from cluster_images are displayed using **imshow()** after converting them from BGR to RGB using **cv2.cvtColor()**. The axes properties are set to turn off the axis and provide a title for each image.

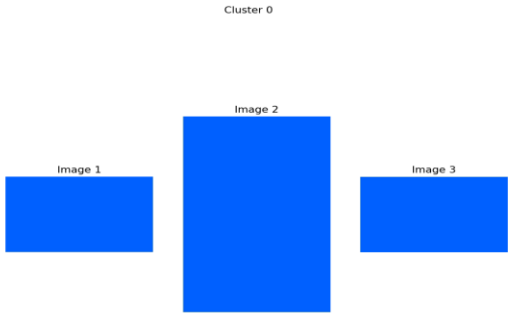
The title of the figure is set to indicate the cluster label. The figure is displayed using **plt.show()**.

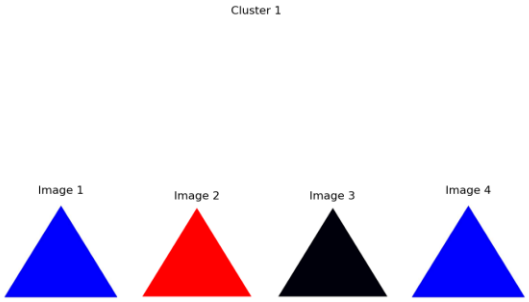
If num_columns is 0, meaning no images belong to the current cluster, a message is printed indicating that the number of columns must be a positive integer.

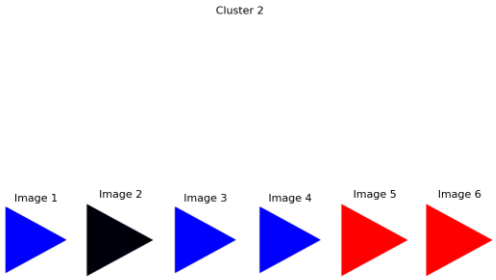
Finally, outside the main() function, there is an if statement checking if the script is being run as the main program. If it is, the main() function is called.

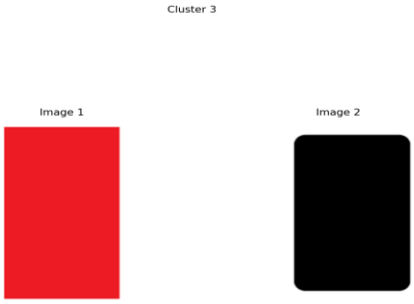
Sample Run

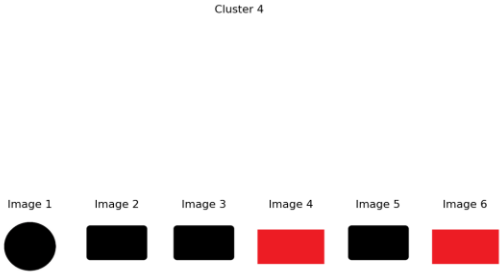
First Trial

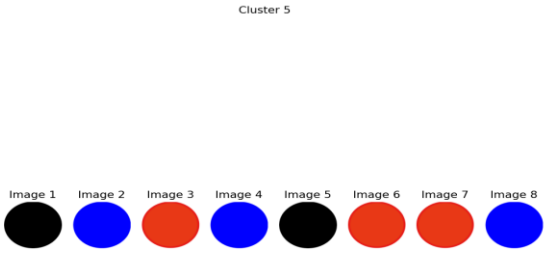


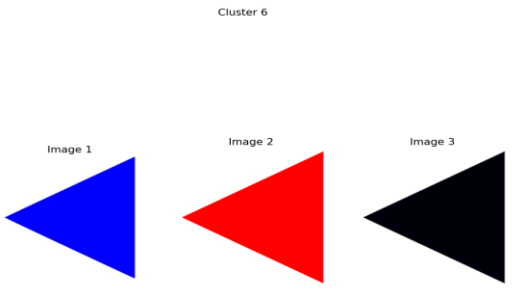


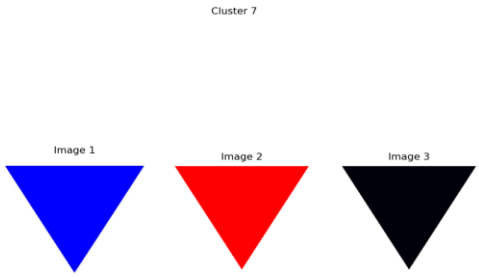


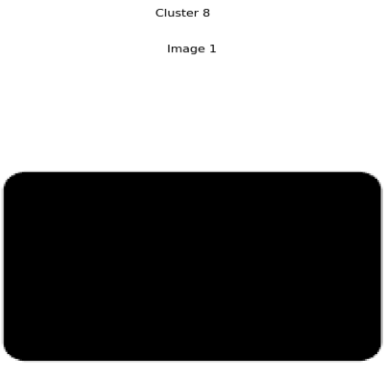













```
Cluster 0: 3 images
Cluster 1: 4 images
Cluster 2: 6 images
Cluster 3: 2 images
Cluster 4: 6 images
Cluster 5: 8 images
Cluster 6: 3 images
Cluster 7: 3 images
Cluster 8: 1 images
```

Second Trial

RuntimeWarning: Mean of empty slice.

return _methods._mean(a, axis=axis, dtype=dtype,

RuntimeWarning: invalid value encountered in divide

ret = um.true_divide(

```
Cluster 1: 36 images
Number of columns must be a positive integer.
```

Sometimes this error happened due to that the model cluster all the images in the same cluster which is cluster 1, so it gives this error in the rest of the clutsers because the number of columns is equal to zero because there is no image to show.

If num_columns is 0, meaning no images belong to the current cluster, a message is printed indicating that the number of columns must be a positive integer.

Conclusion

To improve the performance of the model, we should consider the following steps:

1. **Increase the training data:** Gather more diverse and representative images for training. Having a larger dataset can help the model learn better patterns and generalize well to unseen data.
2. **Data augmentation:** Apply data augmentation techniques to artificially increase the size and diversity of your training dataset. Techniques like random rotation, scaling, cropping, and flipping can introduce variations and help the model learn invariant features.
3. **Feature engineering:** Explore different feature representations that can capture relevant information for our task. For image data, you can try techniques like extracting Histogram of Oriented Gradients (HOG) features, Local Binary Patterns (LBP), or using pre-trained convolutional neural networks (CNN) as feature extractors.
4. **Hyperparameter tuning:** Experiment with different values for hyperparameters such as the number of clusters (k), learning rate, regularization parameters, and convergence criteria. You can use techniques like grid search or random search to systematically explore the hyperparameter space and find better configurations.
5. **Ensemble methods:** Combine multiple models to form an ensemble. Each model in the ensemble can have different initializations or hyperparameters, and their predictions can be aggregated to make the final decision. Ensemble methods often result in improved performance compared to individual models.
6. **Cross-validation:** Perform cross-validation to evaluate the performance of your model on different subsets of the data. This helps in assessing the model's generalization ability and can guide you in making improvements.
7. **Error analysis:** Analyze the errors made by your model and identify the common patterns or difficult cases. This analysis can provide insights into the limitations of your current approach and guide you towards addressing those specific challenges.
8. **Transfer learning:** If your dataset is small, consider using pre-trained models on larger datasets or similar tasks as a starting point. You can fine-tune these models on your dataset or use them as feature extractors.

Training a model is an iterative process. It often requires trying different approaches, evaluating their performance, and refining the model based on the insights gained from the evaluation.