Computer Vision (CSL7360)

Programming Assignment-2

Sushant Ravva (B21CS084)

Links of the Google Colab Code Notebook

Solution

Introduction

In this report, we present a comparison between two clustering techniques, Ratio-Cut and K-means, for image segmentation. Image segmentation is a crucial task in computer vision and image processing, aiming to partition an image into meaningful regions or segments. The goal is to group together pixels that share similar characteristics.

Clustering Techniques

Ratio-Cut Clustering

Ratio-Cut clustering is a spectral clustering technique that partitions a graph into disjoint sets by minimizing the ratio-cut criterion. The ratio-cut criterion measures the dissimilarity between different clusters based on the ratio of the number of edges between clusters to the total number of edges within clusters.

K-means Clustering

K-means clustering is a popular partitioning clustering algorithm that aims to partition data into K clusters by minimizing the within-cluster sum of squares. It

iteratively assigns data points to the nearest cluster centroid and updates the centroids based on the mean of the assigned points.

K-Means Implementation

Visualising the Original Images

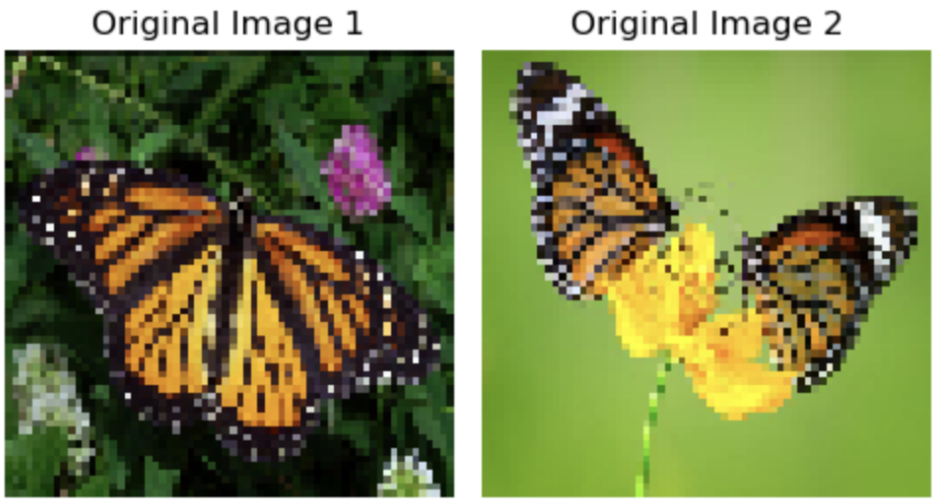


Image Preprocessing

The `resize_images` function takes a list of images as input and resizes each image to the specified dimensions (64×64 pixels). It returns a list of resized images.

The `load_imgs` function loads images from the specified folder path. It iterates through the files in the folder, reads each image using OpenCV, converts it to RGB format, and appends it to a list of images. The function returns the list of loaded images.

Resized images are visualized using Matplotlib's `imshow` function. Subplots are set up to display original images side by side.

K-Means (K = 3)

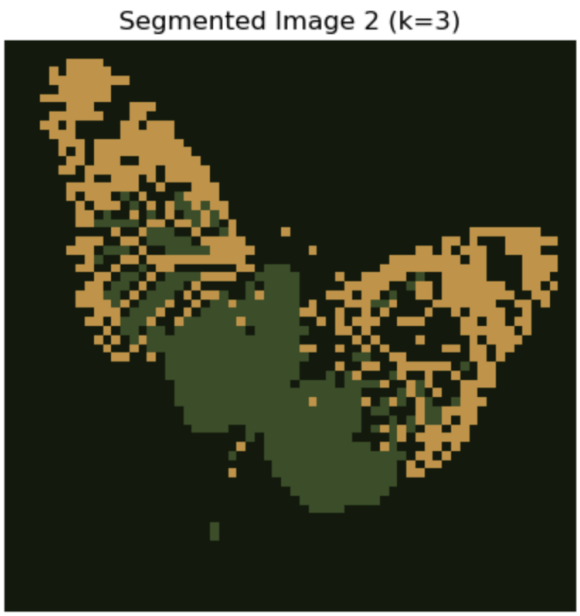
- Initialize the K-means algorithm with the number of clusters, k=3.

- Fit the K-means algorithm to the flattened pixel values of the first image (`all_pixels`).
- Obtain the cluster centers representing the dominant colors in the image.
- Display the colors of the clusters as color swatches.

- Assign each pixel to its corresponding cluster based on the K-means labels.
- Reconstruct the segmented image by replacing each pixel with the color of its assigned cluster.
- Visualize the segmented image.
- Follow the same above steps for the segmentation of the second image with a change of `all_pixels2` instead of `all_pixels`.

Results

Segmented Image 1 (K-Means → k = 3)

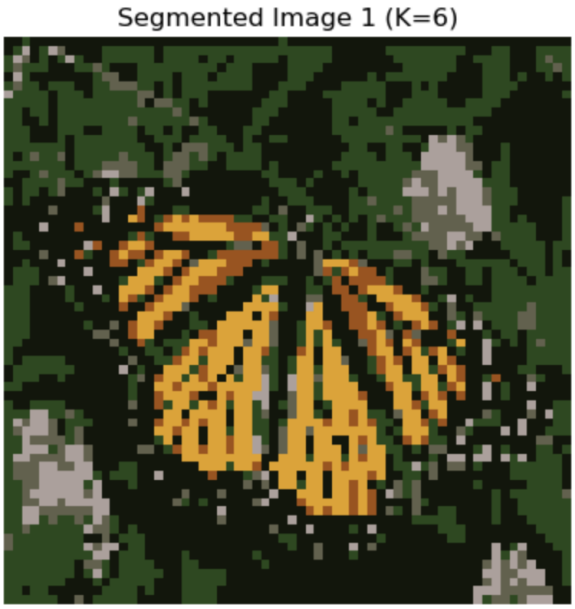


Segmented Image 2 (K-Means → k = 3)

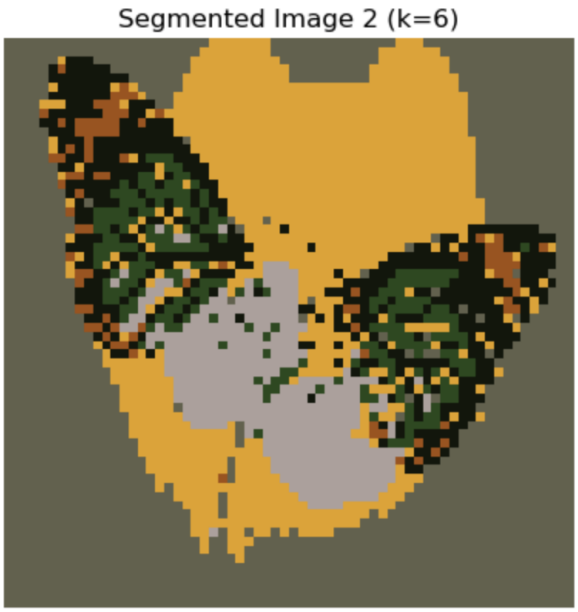
K-Means (K = 6)

- Do the same above steps with a change of `k=6` instead of `k=3`.

Results



Segmented Image 1 (K-Means → k = 6)



Segmented Image 2 (K-Means → k = 6)

Spectral Clustering Algorithm Implementation

- **Affinity Matrix:** Compute pairwise distances between flattened pixels and construct the affinity matrix using a Gaussian kernel.
- **Degree Matrix:** Calculate the degree matrix from the affinity matrix.

- **Laplacian Matrix:** Obtain the Laplacian matrix by subtracting the affinity matrix from the degree matrix.
- **Eigenvalue Decomposition:** Compute the eigenvectors of the Laplacian matrix and select the k eigenvectors corresponding to the smallest eigenvalues.
- **K-means Clustering:** Perform K-means clustering on the selected eigenvectors to obtain the final segmentation.

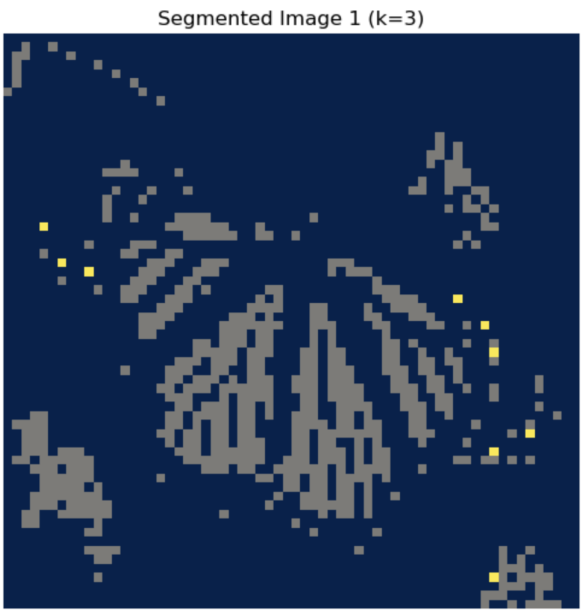
Conversion to Grayscale

RGB images are converted to grayscale using the provided `rgb2gray` function. Grayscale images are preferred for spectral clustering as they simplify the computation and focus on intensity variations rather than color.

The remaining preprocessing steps are same as done in the K-means algorithm.

Results (k=3)

Spectral clustering effectively segments the images into regions based on intensity similarities. The segmented images reveal distinct regions corresponding to different clusters, providing insights into the composition and structure of the original images.

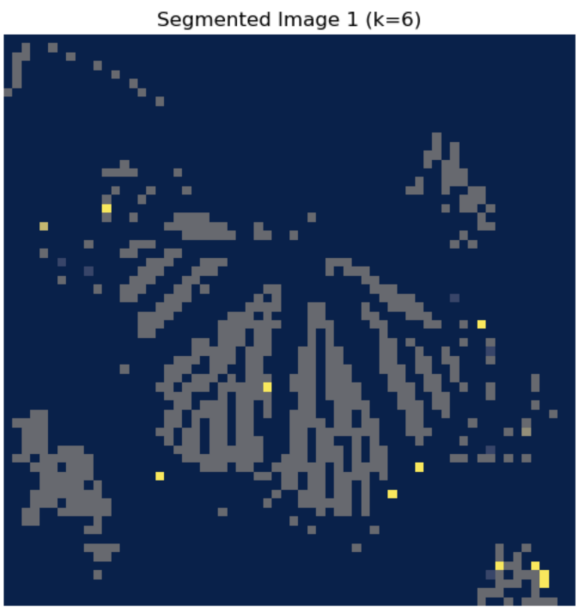


Segmented Image 1 (Ratio-Cut → k = 3)

Segmented Image 2 (Ratio-Cut → k = 3)

Results (k=6)

Display the resulting segmented images from the Ratio-Cut method with `k=6`. The images should reveal more distinct regions compared to when `k=3` was used, providing more detailed insights into the composition and structure of the original images.



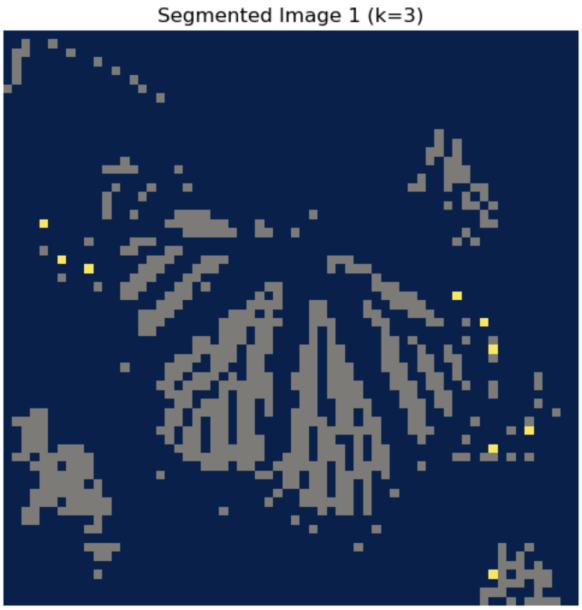
Segmented Image 1 (Ratio-Cut → k = 6)

Segmented Image 2 (Ratio-Cut → k = 6)

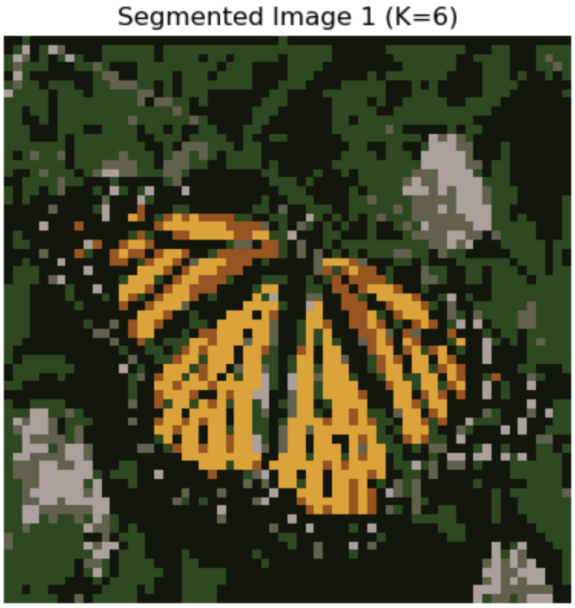
Comparisions

Image 1

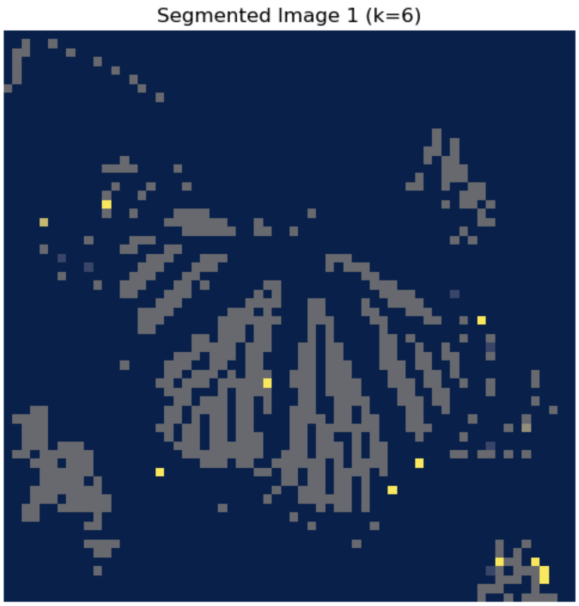
K-Means (k=3)



Ratio-Cut (k=3)

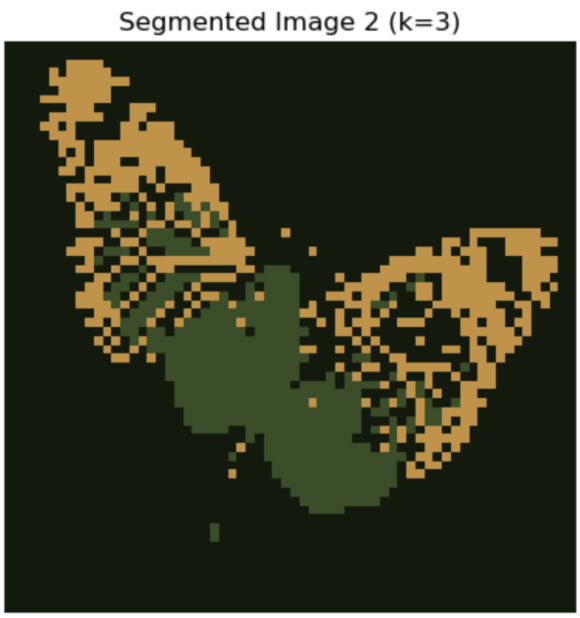


K-Means (k=6)

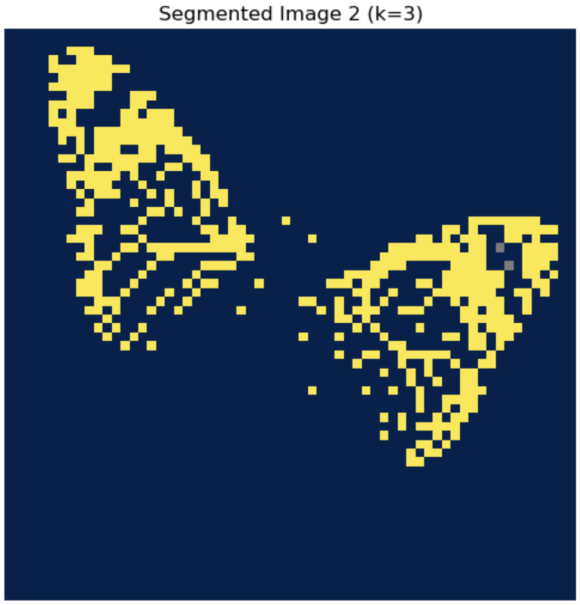


Ratio-Cut (k=6)

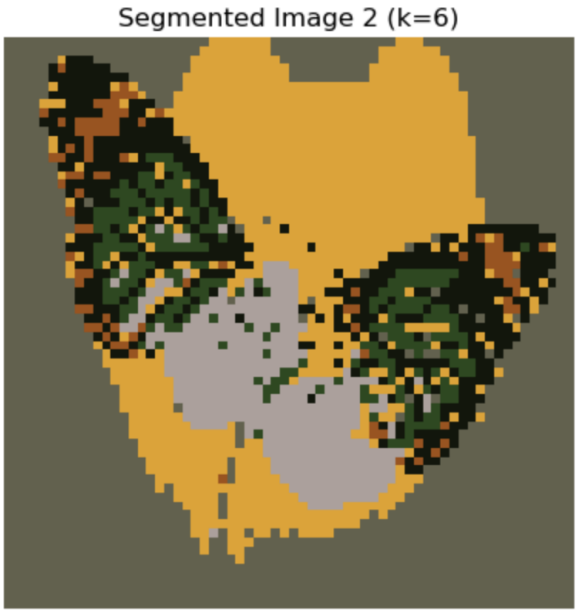
Image 2



K-Means (k=3)



Ratio-Cut (k=3)



K-Means (k=6)

Ratio-Cut (k=6)