Task 4
Thresolding& Segmentation

Team members:

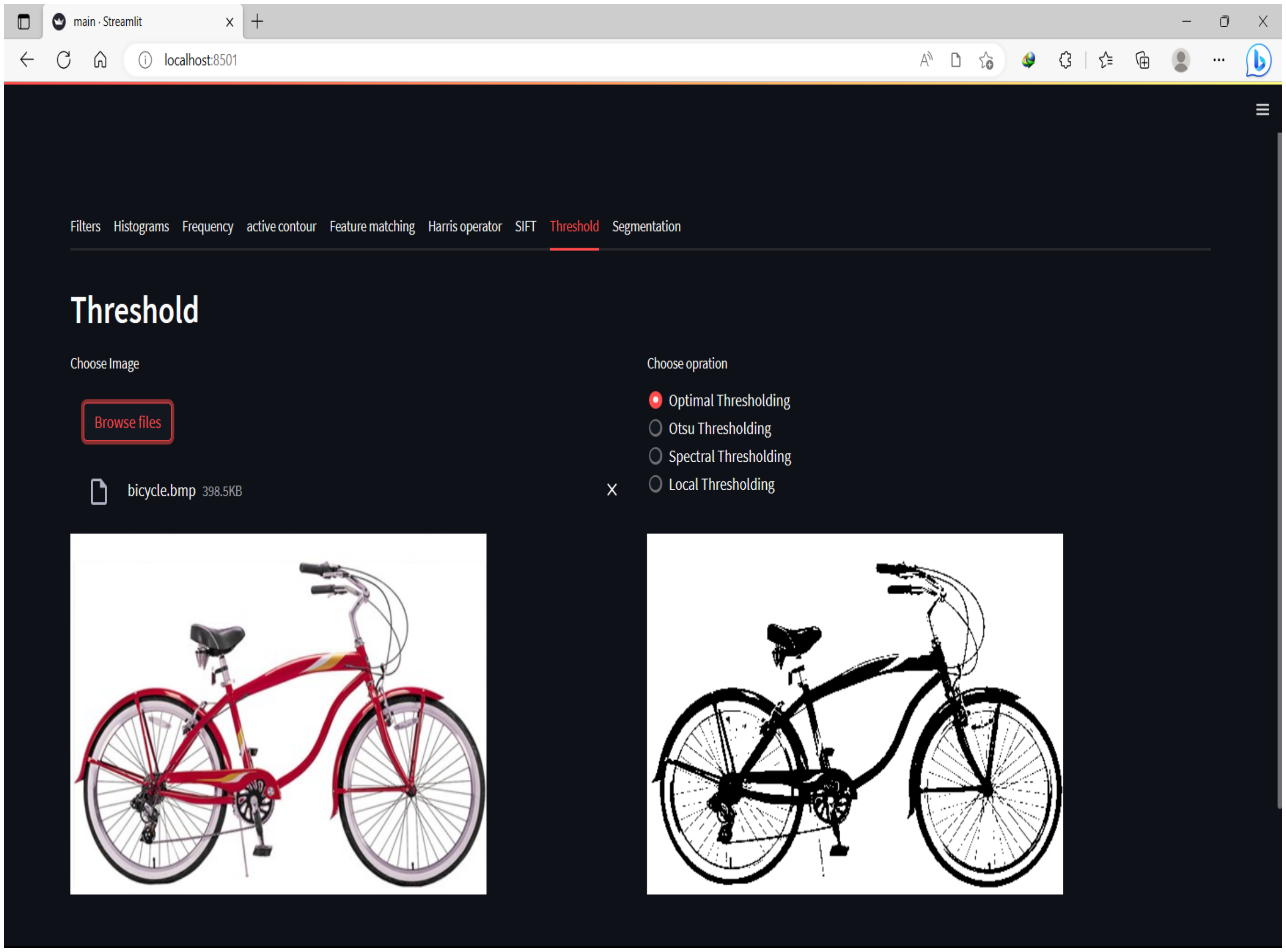
NAME	SECTION	BN
Abram Gad	1	1
Mena Safwat	2	44
Mahmoud Hamdy	2	24
Mariam Hossam Eldin	2	31
Mariam Mohammed Mahmoud	2	34

Contents:

GUI:

Tab1:

The user can browse for an image and choose between four options, the first one is applying optimal thresholding, the second is applying Otsu thresholding, the third is applying Spectral thresholding, the final option is applying Local thresholding



Threshold

Choose Image

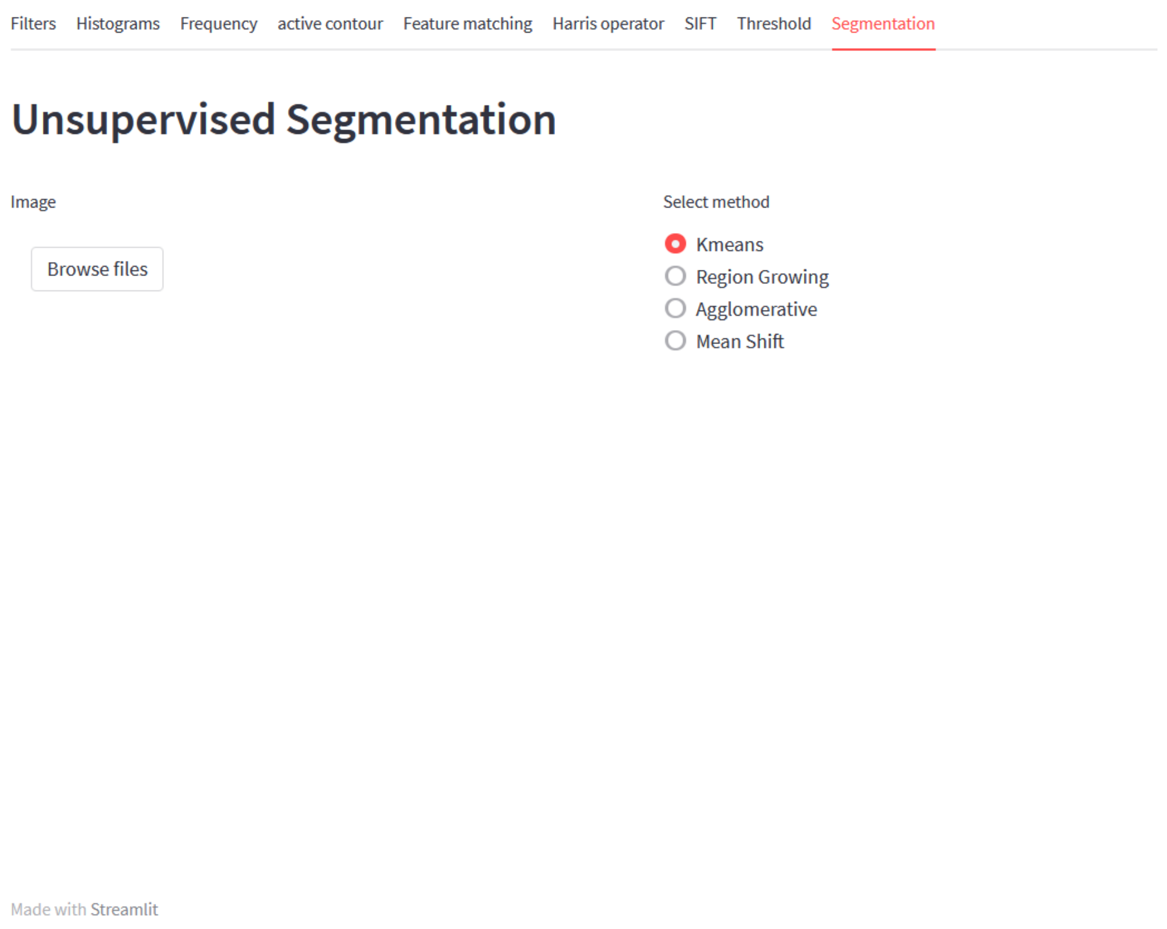
Browse files

bicycle.bmp 398.5KB

Choose opration

[✓ Optimal Thresholding]
Otsu Thresholding
Spectral Thresholding
Local Thresholding

Tab2:
The user can browse for an image and choose one of the four segmentation Algorithms (Kmeans, Region Growing, Agglomerative, Mean Shift)



Unsupervised Segmentation

Image

Select method

[✓ Kmeans] Region Growing Agglomerative Mean Shift

Code Functionalities:

1. Optimal Thresholding.
2. Otsu Thresholding.
3. Spectral Thresholding.
4. Local Thresholding.
5. Kmeans Segmentation
6. Region Growing Segmentation
7. Agglomerative Segmentation
8. Mean Shift Segmentation

Thresholding Tabb:

Note: All of the following thresholding functions are provided in both local and global ways and their outputs are quite similar. We've done this by dividing the image into some preset squares and applying the functions to these squares individually.

After choosing image you need to choose the operation

First operation: Optimal Thresholding

Optimal Thresholding function

Parameters:

- input grayscale image as a NumPy array
- optional tolerance value for convergence

Algorithm:

1. Initialize the threshold value to the midpoint of the intensity range of the image. The intensity range is typically from 0 to 255 for 8-bit grayscale images.

2. Divide the image into two regions based on the threshold value: a foreground region containing pixels with intensities greater than the threshold, and a background region containing pixels with intensities less than or equal to the threshold.

3. Compute the average intensity of pixels in each region. This gives us an estimate of the mean intensity of the foreground and background regions.

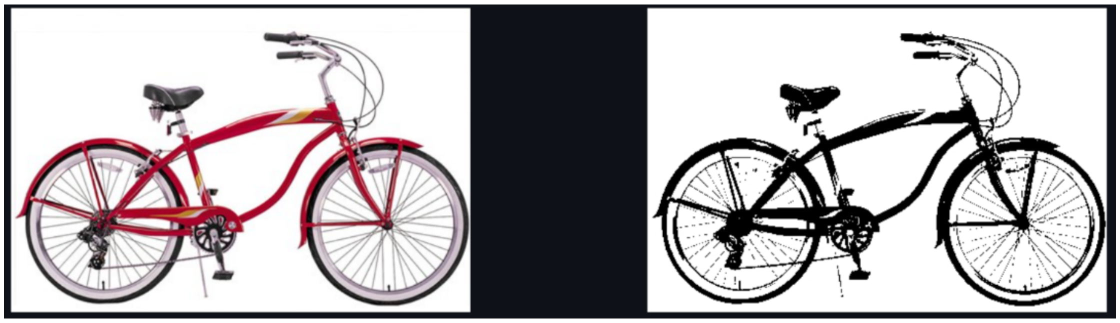
4. Compute a new threshold value as the average of the two region intensities. This new threshold value is a better estimate of the true threshold value that separates the foreground from the background.

5. If the new threshold value is different from the previous threshold value, repeat steps 2-4 until convergence. Convergence is typically achieved when the new threshold value is within a certain tolerance of the previous threshold value.

6. Apply the final threshold value to the image to obtain a binary image. Pixels with intensities greater than the threshold are set to 1

(foreground), and pixels with intensities less than or equal to the threshold are set to 0 (background).

Output:



Second operation: Otsu Thresholding

Otsu Thresholding function

Parameters:

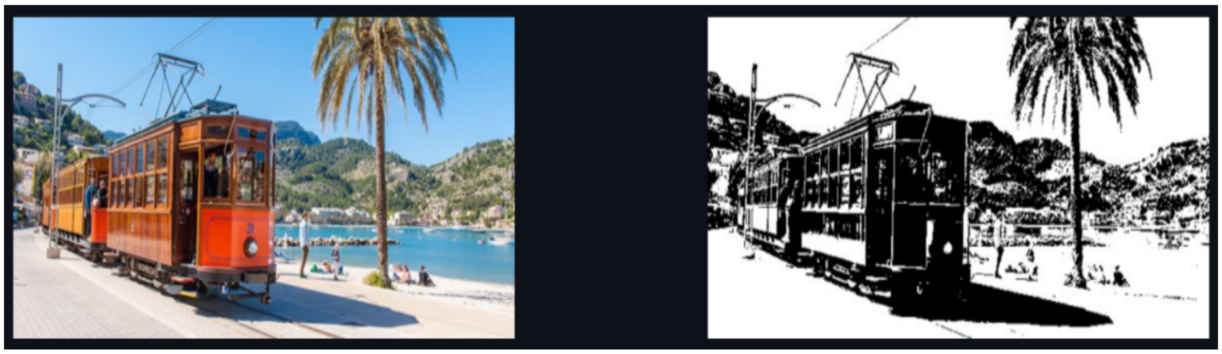
- input grayscale image as a NumPy array.

Algorithm:

1. Compute the histogram of image intensities using numpy.histogram. The ravel method is used to flatten the image into a 1D array, and we specify 256 bins and a range of intensities from 0 to 255.
2. Compute the probabilities of each intensity level by dividing the histogram by the total number of pixels in the image.
3. Compute the cumulative sums of the probabilities and intensity values using numpy.cumsum.
4. Compute the global mean intensity value as the last element of the cumulative sum of intensities.
5. Compute the between-class variance for all possible threshold values using the Otsu formula:
6. ``sigma_b_squared = (global_mean * omega - mu) ** 2 / (omega * (1 - omega))`
7. where omega is the cumulative sum of probabilities up to the current threshold value, and mu is the cumulative sum of intensities up to the current threshold value.
8. Find the threshold value that maximizes the between-class variance by finding the index of the maximum value in the sigma_b_squared array and using that index to obtain the corresponding threshold value from the bins array.
9. Apply the threshold to the image by creating a binary image where pixels with intensities greater than the threshold are set to 1, and

pixels with intensities less than or equal to the threshold are set to 0. We initialize a binary image with the same dimensions as the input image using numpy.zeros_like, and then set the values of the binary image using boolean indexing.

Output:



Third operation: Spectral Thresholding

Spectral Thresholding function

Parameters:

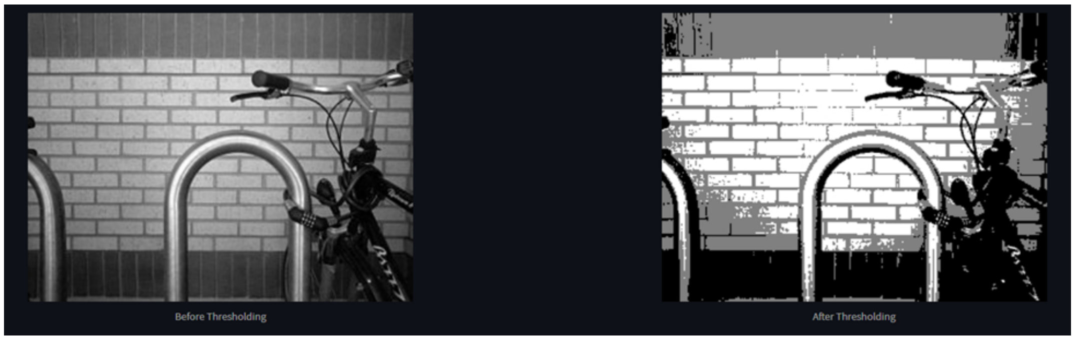
- input grayscale image as a NumPy array.

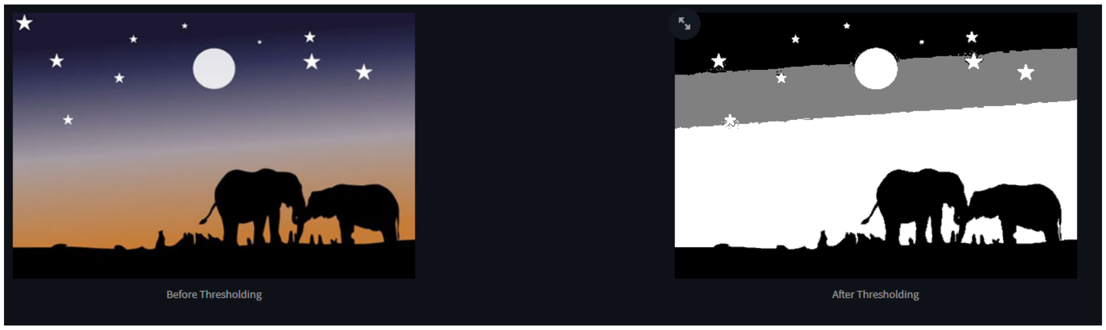
Algorithm:

1. Checks whether the input image is a gray scale image or not.
2. Then compute the image's histogram and get the cdf, Then we get the image's mean.
3. We then initialize our variables
4. After that we loop over all possible thresholds and then get the threshold with max variance between the modes
5. Then we get the weight and the mean of the 3 classes(The regions around the optimal threshold) we defined
6. Get the variance through this equation sigma^2 = w_0 (mean$_0$ – mean$_G$) + w_1 (mean$_1$ – mean$_G$) + w_2 (mean$_2$ – mean$_G$).
7. We then compare this variance to the one we calculated above. If it's greater we update the variance value with the maximum variance value and also update our 2 thresholds.

8. Then we threshold the regions less than optimal_low by 0 and the regions greater than optimal_low and smaller than optimal_high by 128 and the regions greater than optimal_high by 255.

Output:





4th operation: Local Thresholding

Local Thresholding function

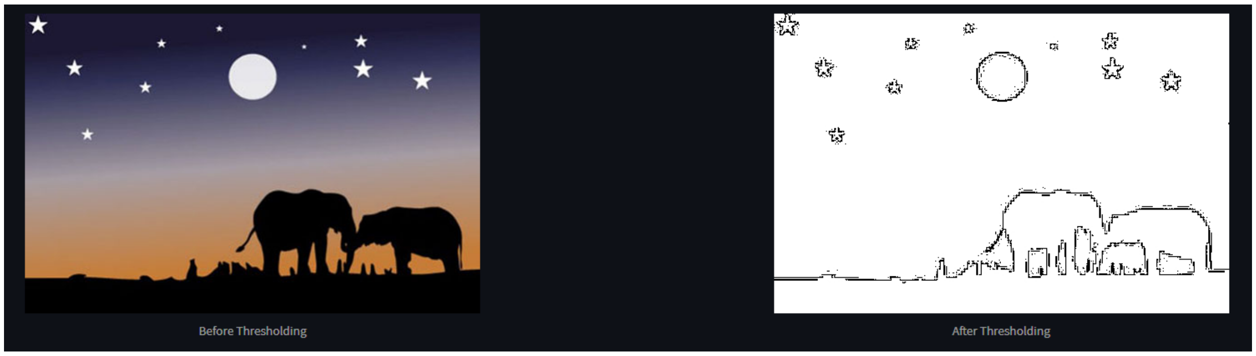
Parameters:

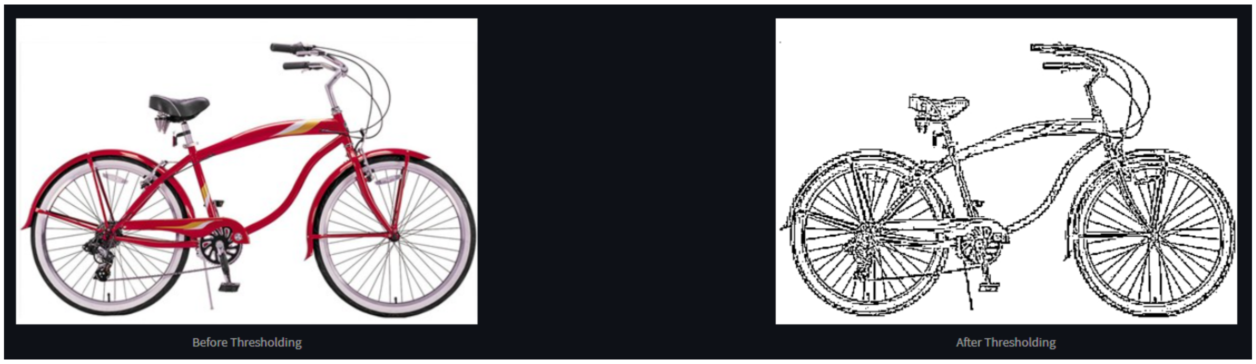
- input grayscale image as a NumPy array.

Algorithm:

1. This method divides the image into smaller windows(squares) with a set size and at each window we call the global thresholding method independently.
2. Then Global Thresholding takes the image's part and sets the threshold as the mean and then threshold it as the pixels intensities larger than the mean they're set to 255 and the pixels less than the mean are set to 0.

Output:


Before Thresholding
After Thresholding


Before Thresholding
After Thresholding

Segmentation Tab

I. KMeans Segmentation

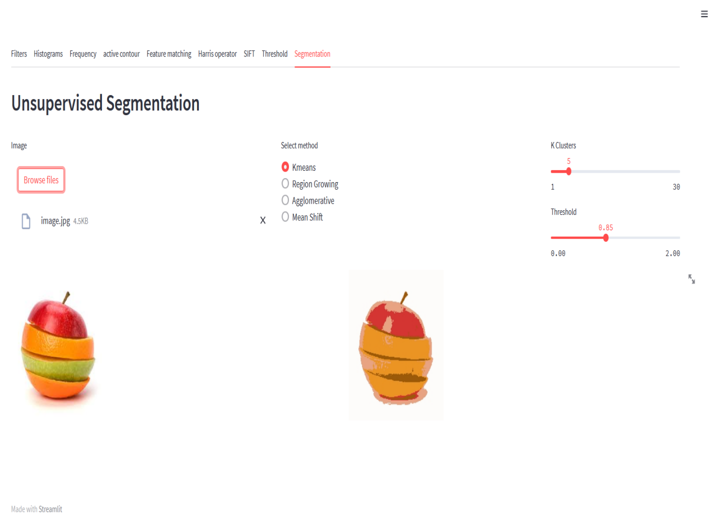
Parameters

- img: the input image to be segmented.
- K: the number of clusters to be generated.
- threshold: the threshold for the algorithm to stop iterating. The default value is 0.85.
- max_iter: the maximum number of iterations for the algorithm to stop. The default value is 100.

Algorithm

1. Convert the input image from BGR to RGB color space due to the use of cv2.
2. Reshape the image into a 2D array of pixels and 3 color values (RGB).
3. Convert the pixel values to float type, check for NaN values in the array, and set them to a small value.
4. Initialize the cluster centroids randomly.
5. Assign each pixel to the nearest centroid using Euclidean distance.
6. Update the cluster centroids by taking the mean of the pixel values assigned to each cluster.
7. Check for convergence by computing the difference between the old and new centroids. Stop the algorithm if the difference is below the convergence threshold or the maximum number of iterations is reached.
8. Convert the cluster centroids into 8-bit values and use them to segment the input image.
9. Reshape the segmented data into the original image dimensions.
10. Return the segmented image.

Output



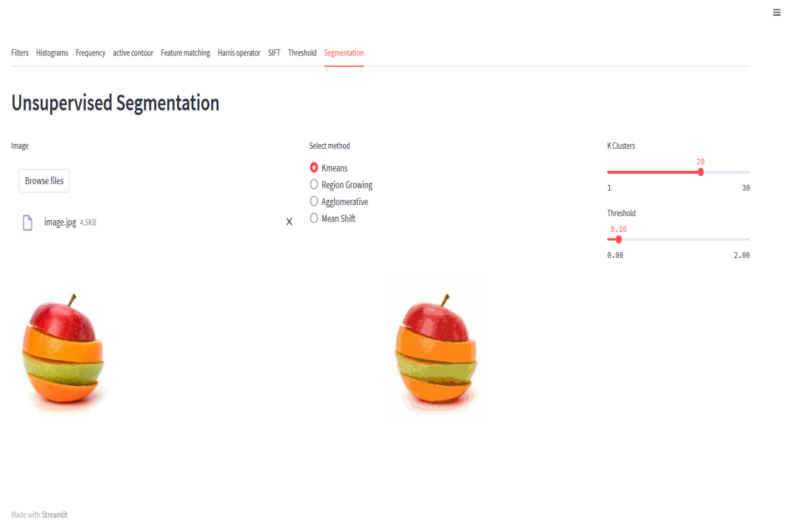
Unsupervised Segmentation

Image: image.jpg 4.5KB

Select method: [✓ Kmeans] Region Growing Agglomerative Mean Shift

K Clusters: 5 (1 – 30)

Threshold: 0.85 (0.00 – 2.00)



Unsupervised Segmentation

Image: image.jpg 4.5KB

Select method: [✓ Kmeans] Region Growing Agglomerative Mean Shift

K Clusters: 20 (1 – 30)

Threshold: 0.16 (0.00 – 2.00)

II. Region Growing Segmentation

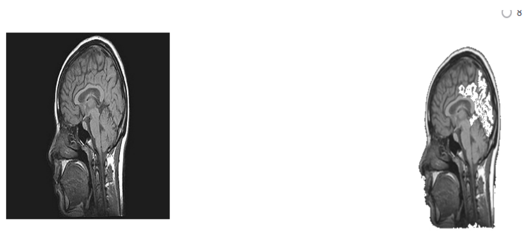
Parameters

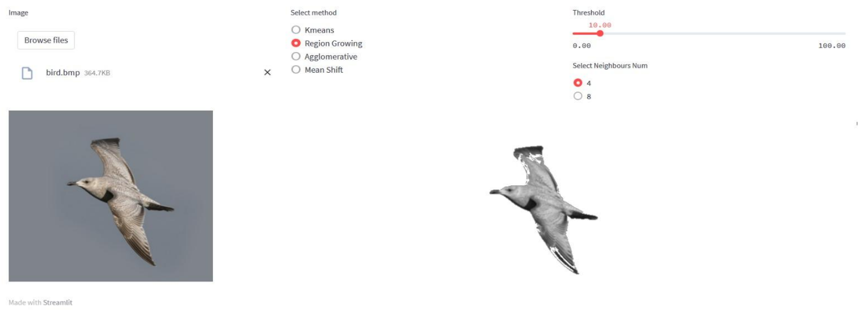
- Image: the input image to be segmented.
- Threshold: the threshold for the algorithm to stop iterating. The default value is 0.85.
- Number of neigbhours: 8 or 4 neifghbours to be checked

Algorithm

1. Reading the image by cv2
2. Resizing image if needed ,if image's is too big (above a certain thresholds) it would resize it for faster computation.
3. Applying gaussian smoothing filter
4. initializing all of the image's variables with null , except for threshold and number of neighbors, both are the users's inputs.
5. Setting the seeds by choosing peaks from an image histogram or u can choose it randomly
6. Start with the first seed , Check the neighbors and them only if they are similar to seed .
7. Keep on looping until the segmentation difference reaches the threshold ,while updating the region contour.
8. repeat the steps 5,6 for each seed.
9. merge the original image and the result of segmentation
10. return the segmented image

Output





Image

Browse files

bird.bmp 364.7KB

Select method

- Kmeans
- [✓ Region Growing]
- Agglomerative
- Mean Shift

Threshold

10.00

0.00 100.00

Select Neighbours Num

- [✓ 4]
- 8

Made with Streamlit

III. Agglomerative Segmentation

Algorithm

1- Initialize the cluster labels for each data point as a separate cluster.

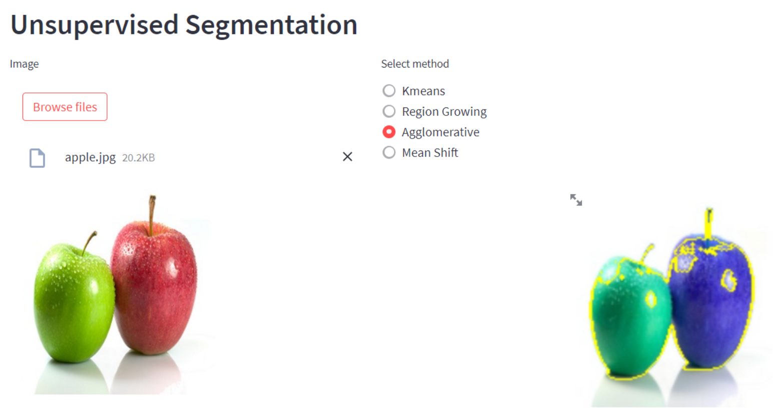
2- Compute the pairwise distance between all data points.

3- While the number of clusters is greater than the desired number of clusters:

1. Find the two closest clusters based on the chosen linkage method (e.g., single linkage, complete linkage, etc.).
2. Merge the two closest clusters into a single cluster.
3. Update the pairwise distance matrix to reflect the new distances between the merged cluster and the remaining clusters.
4. Update the cluster labels for all data points to reflect the new cluster assignments.

4- Return the cluster labels for each data point.

output image



Unsupervised Segmentation

Image

Browse files

apple.jpg 20.2KB

Select method

Kmeans

Region Growing

[✓ Agglomerative]

Mean Shift

IV. Mean Shift Segmentation

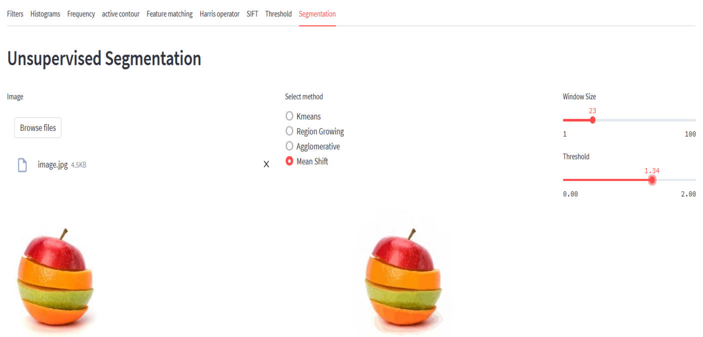
Parameters

- img: input image to be segmented
- window: window size used for the mean shift, the default value is 70
- threshold: threshold value used to determine if a pixel belongs to the same cluster as the current mean, the default value is 1.0

Algorithm

1. Define a function euclidean_distance used to calculate the Euclidean distance between two points.
2. Get the number of rows, columns, and channels of the input image.
3. Create an empty segmented image with the same shape as the input image.
4. Create an empty feature space with the same number of rows and columns as the input image, and 5 columns for the RGB color values and x,y position of each pixel.
5. Fill the feature space with the RGB color values and the x,y positions of each pixel in the input image.
6. Loop while there are still pixels in the feature space:
    a. Select a random row from the feature space and assign it as the current mean.
    b. Find all the pixels in the feature space that are within the window around the current mean.
    c. Calculate the mean color and position of the pixels within the window.
    d. Calculate the Euclidean distance between the mean color/position and the current mean.
    e. If the Euclidean distance is less than the threshold, assign the mean color to all the pixels within the window and remove them from the feature space.
    f. If the Euclidean distance is greater than or equal to the threshold, update the current mean to the mean color/position of the pixels within the window.
7. Return the segmented image in RGB format.

Output



Unsupervised Segmentation

Image

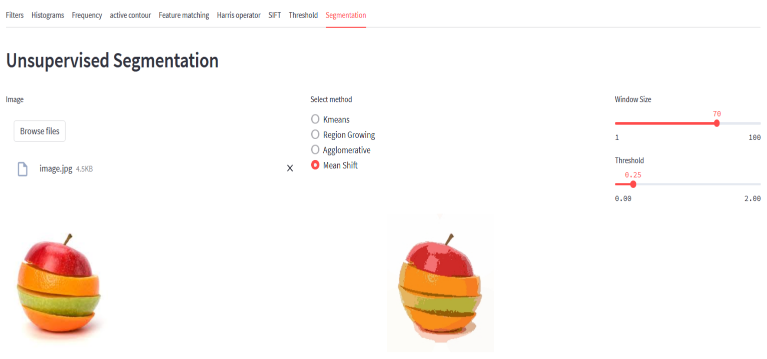
image.jpg 4.5KB

Select method

( ) Kmeans
( ) Region Growing
( ) Agglomerative
[✓ Mean Shift]

Window Size: 23 (1 – 100)

Threshold: 1.34 (0.00 – 2.00)



Unsupervised Segmentation

Image

image.jpg 4.5KB

Select method

( ) Kmeans
( ) Region Growing
( ) Agglomerative
[✓ Mean Shift]

Window Size: 70 (1 – 100)

Threshold: 0.25 (0.00 – 2.00)