

# ***Task -4-***

## ***Computer vision***

*Under supervision of Dr. Ahmed M. Badawi*

*Team -21-*

<b><i>Name</i></b>	<b><i>Section</i></b>	<b><i>B.N</i></b>
<i>Amira Mohamed Abd El-Fattah</i>	<i>1</i>	<i>15</i>
<i>Doha Eid</i>	<i>1</i>	<i>49</i>
<i>Maha Medhat Fathy</i>	<i>2</i>	<i>38</i>
<i>Mayar Ehab Mohamed</i>	<i>2</i>	<i>41</i>
<i>Mariam Mohamed Ezzat</i>	<i>2</i>	<i>33</i>

## **Overview:**

This project is an educational desktop application that performs many functions on uploaded images and visualizes the effect of each of these functions by displaying the output and input images to show the difference between them all on one page.

All functions in our application are written using C++ language without using any built-in functions of the OpenCV library.

## **Thresholding:**

### ➤ **Optimal thresholding:**

Optimal thresholding is a technique used in image processing and computer vision to classify pixels or regions of an image based on their intensity or color values. It involves selecting a threshold value that maximizes a specific quality criterion, such as accuracy or precision, and can be achieved through methods like Otsu's method, adaptive thresholding, or iterative thresholding. Optimal thresholding is important for improving the accuracy and efficiency of applications such as object detection, image segmentation, and edge detection.

### ➤ **Otsu thresholding:**

Otsu's method is a technique for optimal thresholding in image processing and computer vision. It is a histogram-based method that computes the threshold value that minimizes the intra-class variance of the two classes (foreground and background) defined by the threshold. This is achieved by computing the between-class variance for each possible threshold value and choosing the threshold value that maximizes the between-class variance. Otsu's method is particularly effective for images with bimodal intensity distributions and is widely used in applications such as image segmentation, object detection, and feature extraction. However, it may not work well in images with non-uniform lighting or complex backgrounds, in which case other thresholding techniques may be more appropriate.

### ➤ **Spectral thresholding:**

Spectral thresholding is a method that can be used to perform Otsu thresholding on multi-channel images, such as color or hyperspectral images. The idea behind spectral thresholding is to apply Otsu's method to each channel of the image separately, and then combine the resulting binary images into a single binary image.

## **Segmentation:**

### ➤ **K-means:**

K-means segmentation is a technique used in image processing to group similar pixels into K clusters based on their color or intensity values. The resulting segmentation divides the image into K regions, each with a distinct color or intensity. K-means segmentation can be a powerful and effective way to segment an image based on color or intensity similarities, with applications in various fields such as object recognition, image retrieval, and video processing. However, it may not work well with images that have complex textures or structures, in which case more advanced segmentation techniques may be more appropriate.

### ➤ **Region growing:**

Region-growing segmentation is a technique used in image processing to partition an image into regions based on their similarity. The idea behind region growing is to start with a seed pixel or region and iteratively add neighboring pixels or regions that satisfy certain similarity criteria, such as color or intensity similarity, texture similarity, or edge continuity. Region-growing segmentation can be a powerful and flexible way to segment an image based on local similarities, with applications in various fields such as medical imaging, remote sensing, and computer vision. However, it can be sensitive to the choice of seed points or regions and may not work well with images that have complex or ambiguous structures, in which case more advanced segmentation techniques may be more appropriate.

### ➤ **Agglomerative:**

Agglomerative segmentation is a technique used in image processing to partition an image into a hierarchy of regions based on their similarity. The idea behind agglomerative segmentation is to start with each pixel as a separate region and iteratively merge neighboring regions that satisfy certain similarity criteria, until all the pixels belong to a single region. Agglomerative segmentation is a powerful and flexible way to segment an image based on local similarities, with applications in various fields such as medical imaging, remote sensing, and computer vision. However, it can be computationally expensive and may not work well with images that have non-uniform or complex backgrounds, in which case more advanced segmentation techniques may be more appropriate.

➤ **Mean shift:**

Mean shift segmentation is a non-parametric clustering algorithm used in image processing and computer vision to segment an image into regions based on its color or intensity values. The algorithm works by iteratively shifting each pixel towards the mean color or intensity value of its neighboring pixels until convergence is reached, resulting in a set of clusters. Mean shift segmentation is a powerful and flexible way to segment an image based on local similarities, with applications in various fields such as object recognition, image retrieval, and video processing. However, it can be computationally expensive and may not work well with images that have non-uniform or complex backgrounds, in which case more advanced segmentation techniques may be more appropriate

**Implemented algorithms steps:**

**Optimal thresholding:**

1. The function "optimal\_thresholding" takes an input image "image" of type "Mat" and converts it to grayscale using the OpenCV function "cvtColor".
2. The function initializes two threshold values: "threshold\_previous" and "threshold\_next". The initial value of "threshold\_next" is calculated as the average of the pixel values at the four corners of the image.
3. The function declares two empty vectors "background" and "foreground" to store the pixel values that belong to the background and foreground, respectively.
4. The function enters a do-while loop that iteratively segments the image into background and foreground based on the current threshold value. For each pixel in the image, if its grayscale value is greater than the threshold value, the pixel value is added to the "foreground" vector. Otherwise, the pixel value is added to the "background" vector.
5. The function then calculates the mean pixel values of the background and foreground vectors.
6. The function updates the threshold value as the average of the two means, and stores the previous threshold value in "threshold\_previous".
7. The function clears the "background" and "foreground" vectors.
8. The loop continues until the difference between the previous and current threshold values falls below a threshold of 0.02.
9. Finally, the function applies the resulting optimal threshold value to the grayscale image using the thresholding function (that is apply global thresholding) to obtain the binary threshold image.
10. The function returns the threshold image "threshold\_img" of type "Mat" as output.

### **Otsu thresholding:**

1. The function "otsu\_thresholding" takes an input image "image" of type "Mat" and converts it to grayscale using the OpenCV function "cvtColor".
2. The function computes the histogram of the grayscale image using the "histo" function (to calculate the histogram of the image).
3. The function initializes the maximum between-class variance "max\_sigma\_b" to zero and the threshold value "threshold" to zero. It also calculates the total number of pixels in the image "size".
4. The function enters a loop over the histogram values, where it calculates the between-class variance for each possible threshold value.
5. For each iteration of the loop, the function calculates the within-class variances of the background and foreground pixels, as well as their respective weights. It then calculates the between-class variance as the product of the weights and the squared difference between the means of the background and foreground pixels.
6. The function updates the threshold value "threshold" as the value that maximizes the between-class variance.
7. Finally, the function applies the resulting optimal threshold value to the grayscale image using the "thresholding" function to obtain the binary thresholded image.
8. The function returns the thresholded image "threshold\_image" of type "Mat" as output.

### **Spectral thresholding:**

1. The function "spectral\_thresholding" takes an input image "image" of type "Mat" and converts it to grayscale using the OpenCV function "cvtColor".
2. The function calculates the total number of pixels in the image "size" and computes the histogram of the grayscale image using the "histo" function (which is not shown in this code snippet).
3. The function initializes two iterators "it\_first\_threshold" and "it\_second\_threshold" to the second and fourth records of the histogram, respectively. It also initializes pointers to the last four records of the histogram.
4. The function enters a nested loop over the histogram values, where it calculates the between-class variance for each possible pair of threshold values.
5. For each iteration of the loop, the function calculates the within-class variances and means of the three clusters, as well as the global mean. It then calculates the between-class variance as the sum of the products of the cluster weights and the squared differences between the cluster means and the global mean.
6. The function updates the threshold values "first\_threshold" and "second\_threshold" as the values that maximize the between-class variance.
7. Finally, the function applies the resulting optimal threshold values to the grayscale image using the "double\_thresholding" function (which is not shown in this code snippet) to obtain the binary thresholded image.
8. The function returns the thresholded image "threshold\_image" of type "Mat" as output.

### **Spectral local thresholding:**

1. The function "spectral\_localThresholding" takes an input image "image" of type "Mat".
2. The function obtains the number of rows and columns of the input image "image".
3. The function defines four rectangular regions of equal size, each covering a quarter of the input image using the "Rect" constructor.
4. The function extracts the four regions of the input image using the OpenCV "Mat" operator "()" and applies the "spectral\_thresholding" function to each region to obtain the corresponding thresholded image.
5. The function creates a copy of the input image "image" named "thresholded\_image" using the OpenCV "clone" function.
6. The function converts the "thresholded\_image" to grayscale using the OpenCV function "cvtColor".
7. The function copies the thresholded regions obtained in step 4 back into the corresponding regions of the "thresholded\_image" using the OpenCV "copyTo" function.
8. Finally, the function returns the thresholded image "thresholded\_image" of type "Mat" as output.

### **K-means:**

1. Convert the input image to a vector of pixels, where each pixel is represented as a 3D vector of color values.
2. Choose the number of clusters (K) to generate.
3. Randomly initialize K cluster centroids from the pixel vector.
4. For each pixel in the image, assign it to the closest centroid based on Euclidean distance.
5. Compute the mean color of each cluster.
6. Update each centroid by setting it to the mean color of its assigned pixels.
7. Repeat steps 4-6 until convergence or maximum iterations.
8. Assign colors to each cluster based on its centroid color.
9. Create a new image with the same size and type as the input image.
10. For each pixel in the input image, replace it with the color of the nearest cluster centroid.
11. Return the segmented image.

**Note** -> that the K-means clustering algorithm aims to minimize the sum of the squared distances between each pixel and its assigned centroid. The algorithm iteratively refines the centroids by updating them to the mean color of their assigned pixels. The result is a compressed representation of the input image, where each pixel is replaced with the color of the nearest cluster centroid.

### **Region growing:**

1. Create a queue to store the pixels to be processed.
2. Set the seed pixel to 255 in the mask (which will be used to store the labeled pixels).
3. Add the seed pixel to the queue.
4. Get the height and width of the image.
5. Loop until the queue is empty.
6. Get the next pixel from the queue.
7. Loop through the neighboring pixels of the current pixel (including the pixel itself).
8. Check if the neighboring pixel is within the bounds of the image. If not, continue to the next neighboring pixel.
9. Check if the neighboring pixel has already been labeled in the mask. If so, continue to the next neighboring pixel.
10. Check if the difference in intensity between the current pixel and the neighboring pixel is below the threshold. If not, continue to the next neighboring pixel.
11. Set the neighboring pixel to 255 in the mask.
12. Add the neighboring pixel to the queue.
13. Repeat steps 6-12 until the queue is empty.
14. The resulting mask contains the labeled pixels of the region.

### **Agglomerative:**

1. The function "calculateDistance" calculates the Euclidean distance between two pixel values in a 3-channel color space.
2. The function "mergeClusters" merges two clusters by taking the average of each channel for the two clusters.
3. The function "agglomerativeSegmentation" takes an input image "input" and the desired number of clusters "numberOfClusters".
4. The function converts the input image to the Luv color space using the OpenCV "cvtColor" function.
5. The function initializes the clusters by adding each pixel to the clusters vector.
6. The function performs agglomerative clustering by iteratively merging the two closest clusters until the desired number of clusters is reached. The two closest clusters are found by computing the pairwise distances between all clusters and selecting the pair with the smallest distance.
7. The function assigns a label to each pixel based on the closest cluster it belongs to. The labels are stored in a separate matrix "labels" of type "CV\_32S".
8. The function creates a segmented image by assigning the color of the closest cluster to each pixel.
9. The function converts the segmented image back to the BGR color space using the OpenCV "cvtColor" function.
10. Finally, the function returns the segmented image as output.



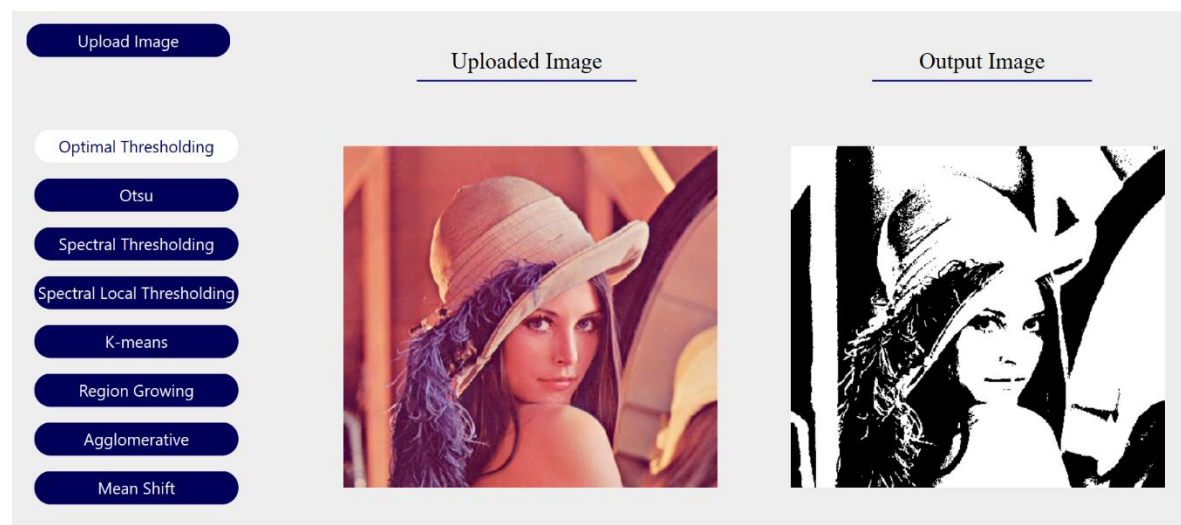
### **Mean shift:**

1. The Mean Shift algorithm starts with an input image and a bandwidth value (or kernel size) that determines the size of the region to be considered around each pixel.
2. For each pixel in the image, a search window is defined centered on the pixel and with a size equal to the bandwidth.
3. The algorithm then computes the mean shift vector for each pixel in the search window. The mean shift vector is calculated as the weighted average of the pixel values inside the search window, with the weights being determined by a kernel function.
4. The algorithm then updates the position of the pixel by shifting it towards the direction of the mean shift vector. This process is repeated until convergence, which is typically achieved when the pixel moves by a small amount or when a maximum number of iterations is reached.
5. Once all pixels have been shifted to their respective convergence points, the algorithm assigns a label to each pixel based on the convergence point it reached. Pixels that converge to the same point are assigned the same label.
6. Finally, the algorithm applies a post-processing step to merge adjacent regions with the same label and remove small isolated regions.
7. The output of the Mean Shift algorithm is a segmented image, where each pixel belongs to a region with a unique label.

### **Output of algorithms:**

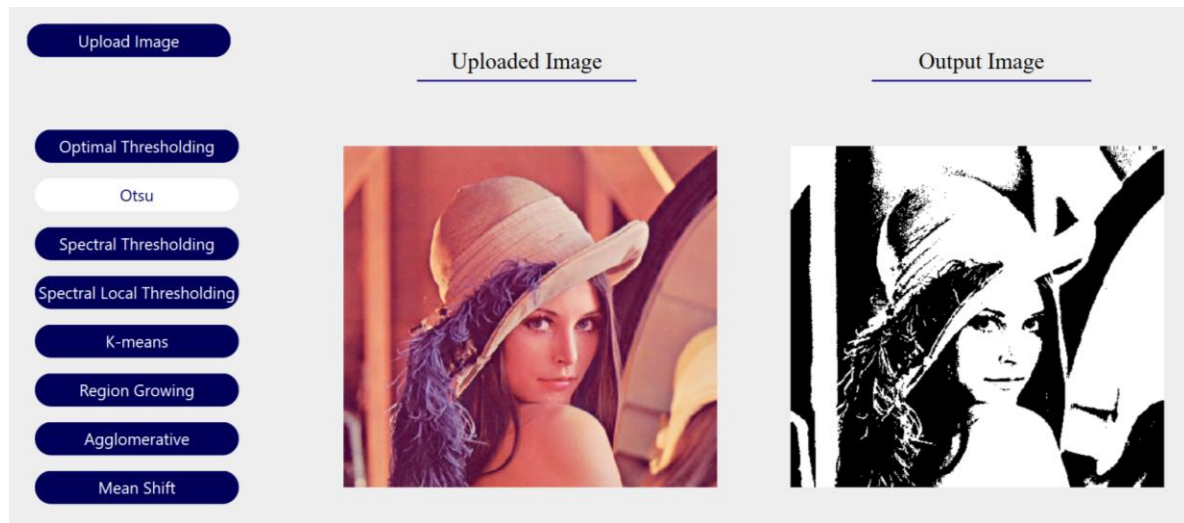
#### **Thresholding:**

#### **Optimal thresholding:**

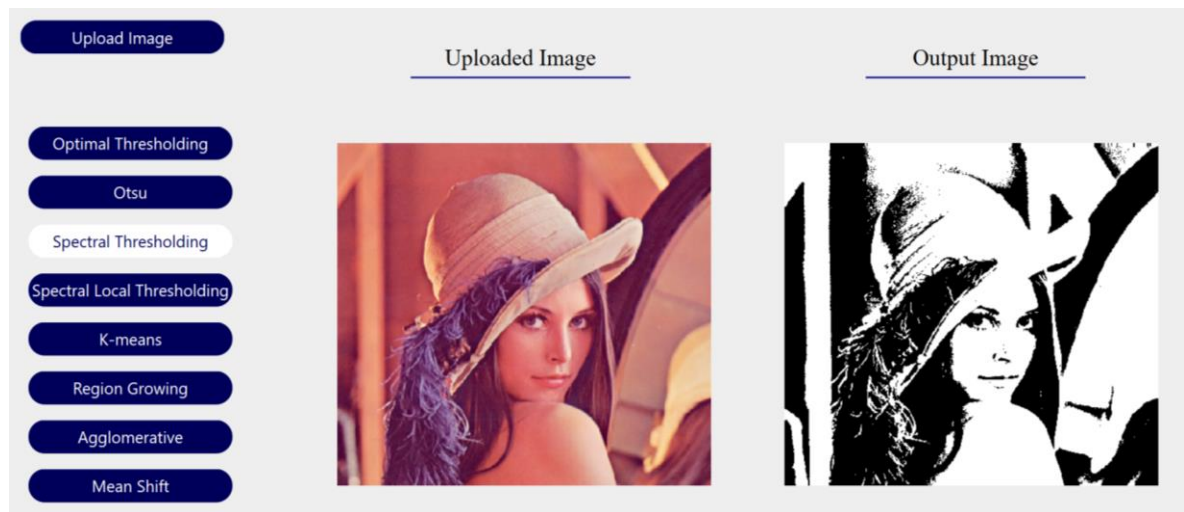




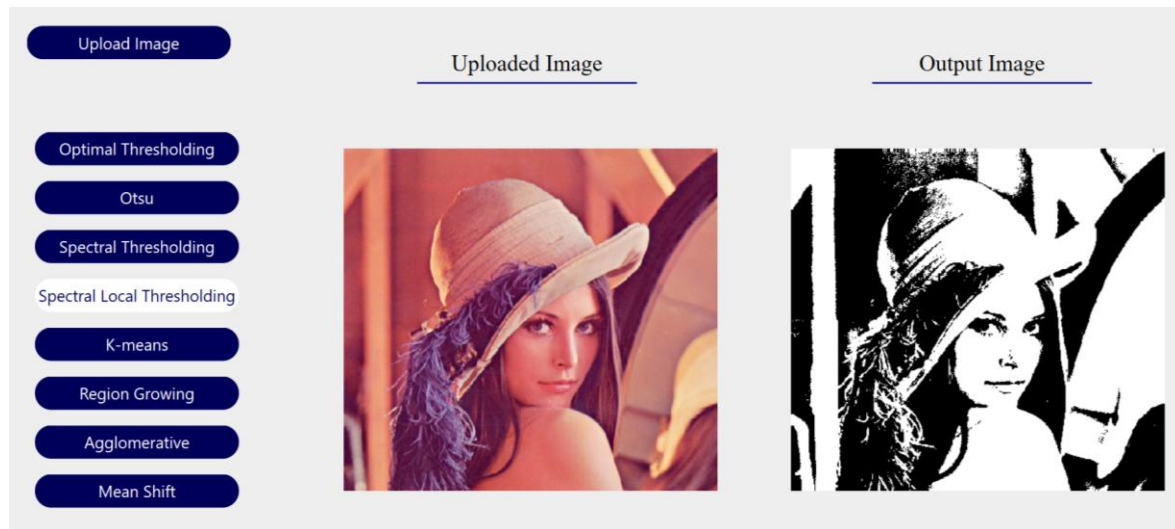
## ***Otsu thresholding:***



## ***Spectral thresholding:***

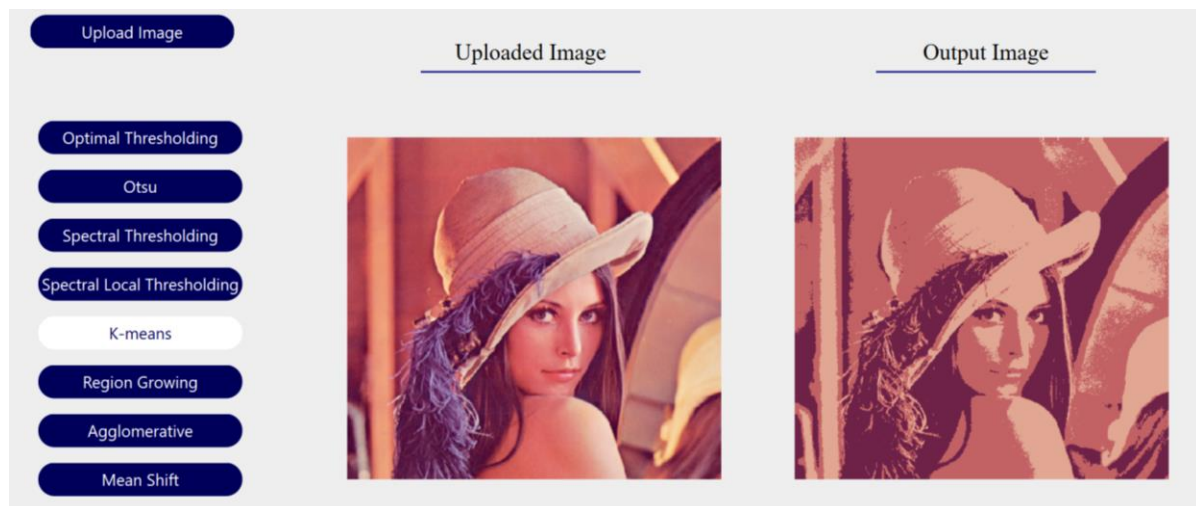


## **Spectral thresholding (using local thresholding):**



## **Segmentation:**

### **K-means:**



## **Region growing:**

Upload Image

Optimal Thresholding

Otsu

Spectral Thresholding

Spectral Local Thresholding

K-means


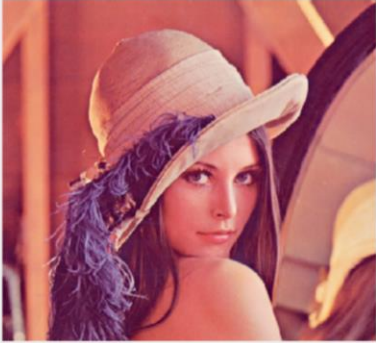
Region Growing

Agglomerative

Mean Shift

Uploaded Image

Output Image



## **Agglomerative:**

Upload Image

Optimal Thresholding

Otsu

Spectral Thresholding

Spectral Local Thresholding

K-means



Region Growing

Agglomerative

Mean Shift

Uploaded Image

Output Image



## **Mean shift:**

Upload Image

Optimal Thresholding

Otsu

Spectral Thresholding

Spectral Local Thresholding


K-means

Region Growing

Agglomerative

Mean Shift

Uploaded Image



Output Image

