Faculty of Engineering
Systems & Biomedical Engineering Department

Computer Vision-Task 4

A Third Year Systems & Biomedical Engineering Task Report

Done by Team 5

Name	Section	BN
Ibrahim Mohamed	1	2
Omnia Sayed	1	14
Marina Nasser	2	12
Mahmoud Yaser	2	30
Maye Khaled	2	40

Submitted in Partial Fulfilment of the Requirements for

Computer Vision (SBE3230)

Submitted to:

Eng. Peter Emad

Eng. Laila Abbas

May 2023

1. Optimal Thresholding

- **Algorithm**
  The getOptimalThreshold function takes the input image, which is a cv::Mat object representing the input image to be thresholded,max_iterations and threshold that control the maximum number of iterations and the convergence threshold for the iterative algorithm, respectively.

  The iterative optimal thresholding algorithm starts with an initial threshold value of 128 and iteratively computes the mean intensity values of the foreground and background regions, and then updates the threshold value as the average of these means. The algorithm continues to iterate until either the change in the threshold value between iterations is less than the convergence threshold or the maximum number of iterations is reached.

- **Output Sample**

Input Image

Output Image

2. Otsu’s method

- **Algorithm**

The otsu function takes a single argument which is input image, which is a cv::Mat object representing the input image to be thresholded.

The purpose of this function is to determine the optimal threshold value for the input image using a thresholding algorithm. In the implementation of this function, these are the followed steps:

1. Convert the input image to grayscale
2. Compute a histogram of the image pixel values

3. Calculate the between-class variance for each possible threshold value:
   The between-class variance is a measure of the separation between the pixel values in the two classes (foreground and background) that are created when the image is thresholded at a given threshold value. The threshold value with the highest between-class variance is selected as the optimal threshold value.
4. Get the optimal threshold value.
5. Create an output image.
6. Perform thresholding with the optimal threshold value.

- **Output Sample**

Input Image

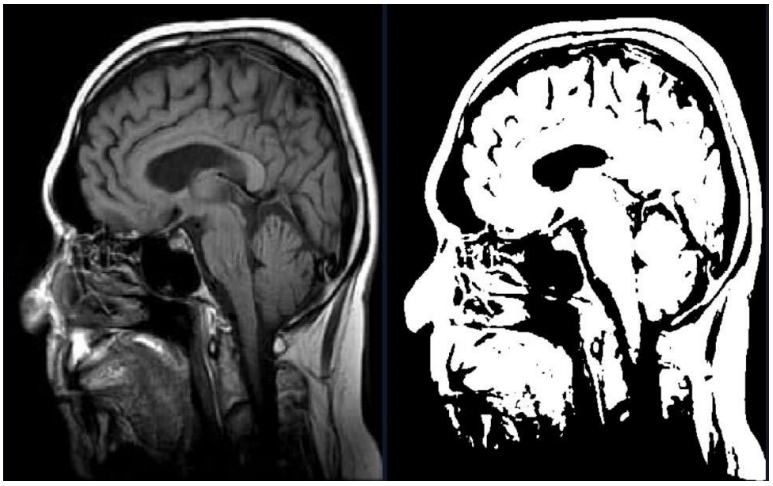
Output Image

3. Global Thresholding

The global thresholding algorithm assigns a binary value to each pixel based on whether its intensity is above or below a global threshold value.

- **Algorithm**
  1. Create a new output image with the same size as the input image.
  2. Loop through each pixel in the input image.
  3. For each pixel, compare its intensity with the global threshold value.
  4. If the intensity is greater than or equal to the threshold value, assign a value of 255 to the corresponding pixel in the output image. Otherwise, assign a value of 0.
  5. Return the output image.

- **Output Sample**



4. Local Thresholding

The local thresholding algorithm assigns a threshold value to each pixel based on the mean and standard deviation of the surrounding pixels within a block, and then assigns a binary value to the pixel based on whether its intensity is above or below the local threshold value.

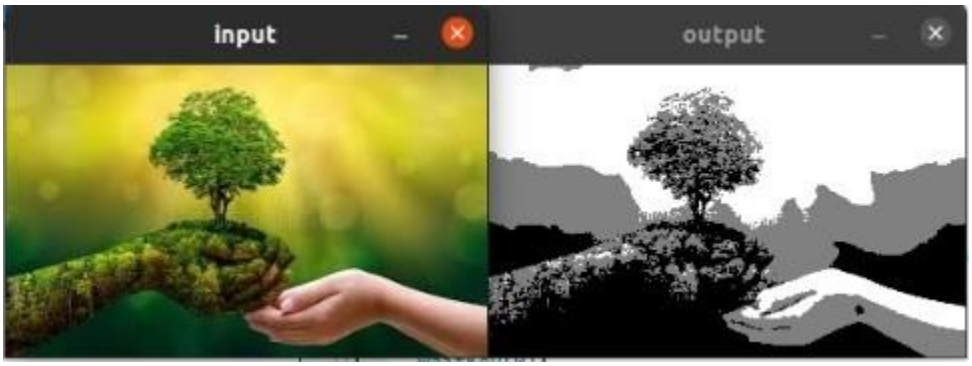
- **Algorithm**
  1. Create a new output image with the same size as the input image.
  2. Pad the input image with zeros to handle edge cases.
  3. Loop through each pixel in the input image.
  4. For each pixel, define a square block of pixels centered at the current pixel.
  5. Calculate the mean and standard deviation of the intensities of the pixels within the block.
  6. Compute a local threshold value based on the mean and standard deviation of the block.
  7. Compare the intensity of the current pixel with the local threshold value.
  8. If the intensity is greater than or equal to the local threshold value, assign a value of 255 to the corresponding pixel in the output image. Otherwise, assign a value of 0.
  9. Return the output image.

- **Output Sample**

5. K-means

- **Algorithm**
  1. Initialize the KmeansSegmentation constructor with no data input.
  2. Define EuclideanDistance function to compute distance between data points and centroids.
  3. Define KmeansDrivingFunction as main function to perform K-means clustering on image.
  4. Convert input image to data points matrix.
  5. Initialize labels and centroids matrices based on data points and number of clusters.
  6. Randomly initialize centroids matrix.
  7. Perform K-means clustering with KmeansClustering function.
  8. Reshape labels matrix to match image dimensions.
  9. Convert label values to colors.
  10. Define UpdateCentroids function to update centroids.
  11. Define KmeansClustering function to perform K-means clustering algorithm.
  12. Initialize previous centroids matrix to -infinity and iterator to 0.
  13. Loop until maximum iterations are reached or centroids stop changing.
  14. Save current centroids to previous centroids matrix.
  15. Assign each data point to closest centroid using Euclidean distance.
  16. Update centroids using UpdateCentroids function.
  17. Increment iterator.

- **Output Sample**



6. Region Growing

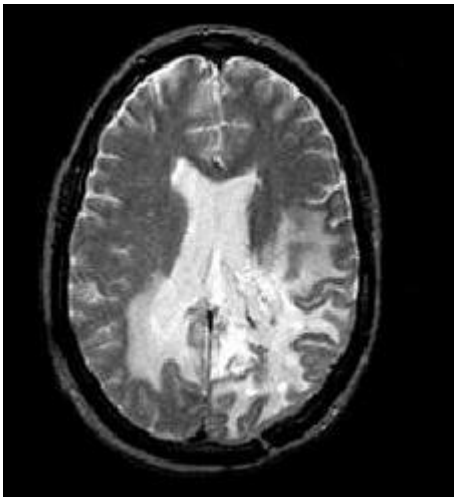
- **Algorithm**
  The code starts by defining a struct Pixel to represent a pixel in the image, which consists of its row and column coordinates. Then, a function isPixelInBounds is defined to check if a pixel is within the bounds of the image.

  The main function of the code loads an input image and converts it to grayscale. Then, it defines the seed pixel coordinates (seedRow and seedCol) and the threshold value (threshold) for the region-growing algorithm.

  The regionGrowing function implements the region-growing algorithm by taking the input image, seed pixel coordinates, and threshold value as input parameters and returns the segmented output image. The function first initializes an output image of the same size as the input image with all pixels set to 0. Then, it creates a queue pixelQueue to store the pixels to be processed, adds the seed pixel to the queue, and marks it as processed in the output image.

  The function then defines the neighborhood of a pixel as the eight adjacent pixels, including the diagonals, and processes the pixels in the queue. For each pixel in the queue, the function processes its neighbors and checks if they meet the criteria for adding them to the region or not (by checking if the difference between pixels' intensities is larger or smaller than the threshold set). If a neighbor pixel meets the criteria, it is added to the queue and marked as processed in the output image.

- **Output Sample**



Input Image

Output image at row=80, col=80, threshold=10

7. Agglomerative

- **Algorithm**

1. Initialize the clusters:
    - Create an empty vector to store the clusters.
    - Traverse the input image and add each pixel as an individual cluster to the vector.
2. Perform agglomerative clustering based on stopping criteria:
    - While the number of clusters is greater than the desired number of clusters:
        - Find the two closest clusters:
        - Traverse all pairs of clusters and calculate the distance between them using the calculateDistance() function.
        - Keep track of the minimum distance and the indices of the two clusters with the minimum distance.
    - Merge the two clusters:
    - Update the first cluster with the mean of the two clusters using the mergeClusters() function.
    - Delete the second cluster from the vector.
3. Assign labels to the pixels:
    - Create a new matrix to store the labels of the pixels.
    - Traverse the input image and for each pixel:
    - Find the cluster with the minimum distance to the pixel using the calculateDistance() function.

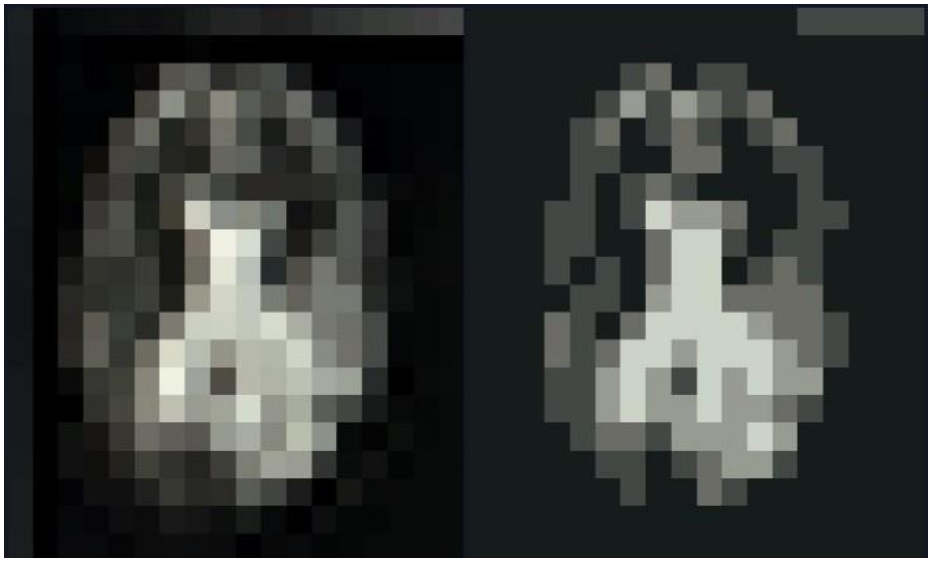
• Assign the index of the cluster as the label of the pixel.

4.Visualize the segmentation:

• Create a new image to store the segmented output.
• Traverse the input image and for each pixel:
• Get the cluster index of the pixel from the label matrix.
• Get the color of the corresponding cluster from the clusters vector.
• Assign the color to the corresponding pixel in the output image.
• Convert the output image from Luv color space to BGR color space.

5. Return the segmented image.

- **Output Sample**



8. Mean Shift

Mean shift segmentation works by iteratively shifting points towards the direction of the highest density in the data space. It begins by randomly selecting points as initial cluster centers, and then computes the mean of the points within a given radius, which becomes the new center. This process is repeated until convergence, with points eventually settling into local maxima of the density function. The resulting clusters are then determined by assigning points to their nearest center.

- **Algorithm**
  1. Read the input image and convert it to the Luv color space.
  2. For each pixel in the image, calculate the spatial window around it based on a distance bandwidth.

3. Apply mean shift algorithm to each pixel within the spatial window until convergence.
4. The convergence is checked by comparing the color distance between the current pixel and the previous one.
5. The output image is obtained by assigning each pixel to its corresponding converged mode.

- **Output Sample**