IMAGE THRESHOLDING & CLUSTERING

Assignment 4

TEAM 1

- HABIBA FATAHALLA
- RAWAN MOHAMMED FEKRY
- SARA AYMAN EL-WATANY
- MARIAM WAEL

- Eng. Peter Emad
- Eng. Laila Abbaas

Introduction:

To implement the algorithms in the assignment from scratch we used C++ and Qt for GUI, to be able to obtain the same results as the algorithms in Open cv built in library.

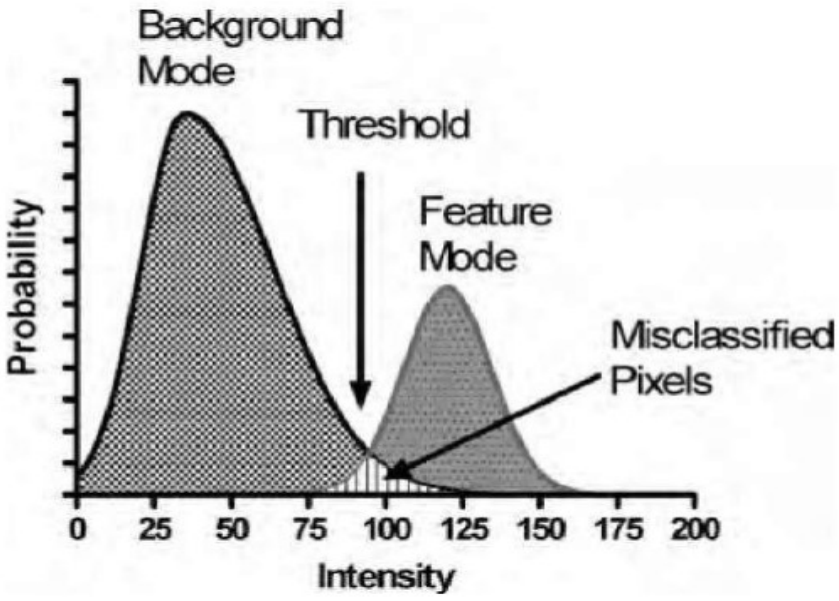
To read and show the images we used open cv methods, and we also used it to test our output to see if it’s accurate or not compared to the built in algorithms.

Thresholding:

There are two broad categories of thresholding: global thresholding and local thresholding

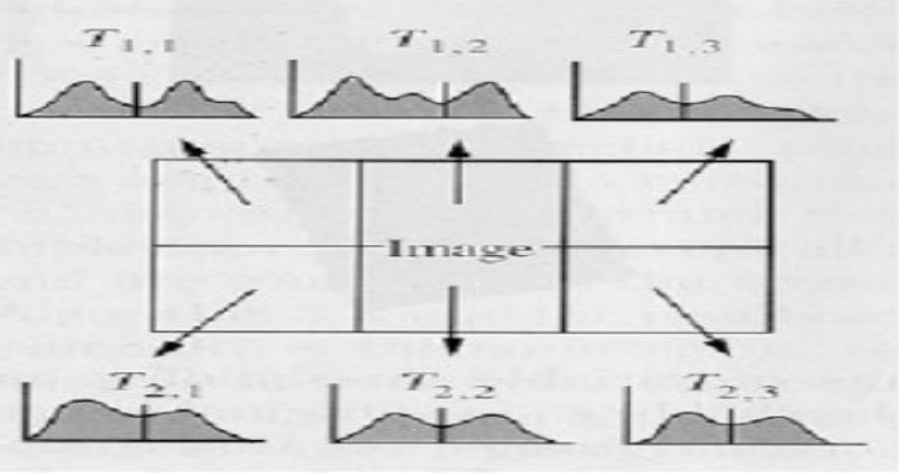
1-Global thresholding:

- Global thresholding is a fundamental technique in computer vision that is used to segment images into foreground and background regions based on a threshold value.
- The threshold value is a pixel intensity value that separates the image into two regions, where all pixels with intensities above the threshold value are classified as foreground and all pixels with intensities below the threshold value are classified as background.



2-Local thresholding:

- Local thresholding is a technique that is used to segment images into foreground and background regions based on locally adaptive threshold values.
- Unlike global thresholding, which applies the same threshold value to all pixels in the image, local thresholding applies different threshold values to different regions of the image.
- The block size determines the size of the regions that the image is divided into. A larger block size will divide the image into larger regions, whereas a smaller block size will divide the image into smaller regions.
- The appropriate block size for a given image depends on the characteristics of the image and the desired outcome. A larger block size may be appropriate for images with large and relatively uniform regions, while a smaller block size may be better for images with fine details or regions with significant variations in brightness.
- This approach is particularly useful in images with uneven illumination or low contrast, where a global threshold value may not be effective.



Calculating thresholding value techniques:

- **Optimal thresholding:**

  - Optimal thresholding algorithms seek to find a threshold value that best separates the foreground and background regions in an image based on some criterion.
  - One simple approach to optimal thresholding involves **using the means of two clusters to compute the threshold value**.
  - In this approach, the image is first segmented into two clusters based on their intensity values. The clusters are typically defined by a threshold value that separates the foreground and background regions. Once the clusters have been identified, their means are calculated.
  - The threshold value is then calculated as the average of the two cluster means, using the formula:

$$T = \frac{Mean_{\text{foreground}} + Mean_{\text{background}}}{2}$$

  - The idea behind this approach is that the optimal threshold value should be one that maximizes the separation between the two clusters. By choosing the threshold value as the average of the two cluster means, we are effectively selecting a threshold that bisects the distance between the two means, which can be a good estimate for the optimal threshold.

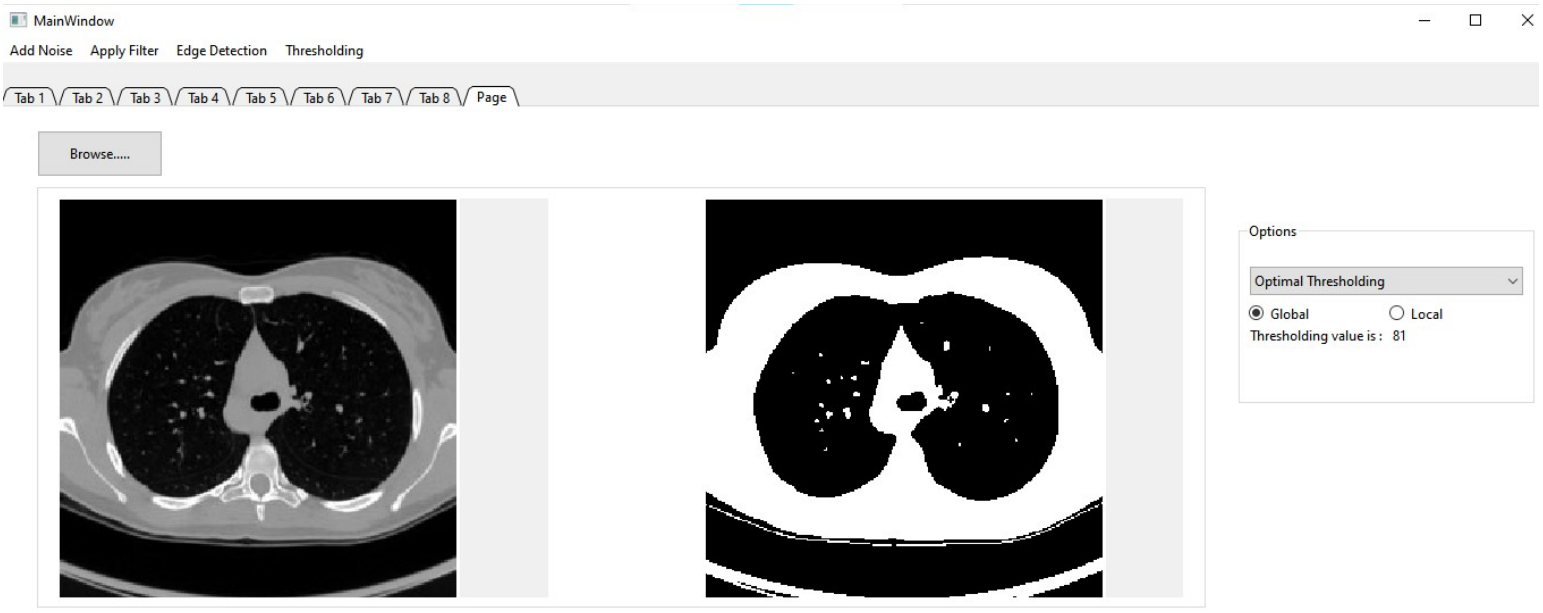
Our Algorithm:

1. Initialize two clusters: one containing the four corner pixels of the image, and the other containing the remaining pixels.
2. Compute the initial threshold as the average intensity value of the two clusters.
3. Iterate until convergence:
   - Assign each pixel in the image to one of the two clusters based on the current threshold value.
   - Compute a new threshold value as the average of the intensity values of the pixels in each cluster.
   - . Check for convergence by comparing the new threshold value to the previous threshold value. If the difference is small enough, terminate the algorithm and return the optimal threshold.
4. Return the optimal threshold value.

Output Examples:

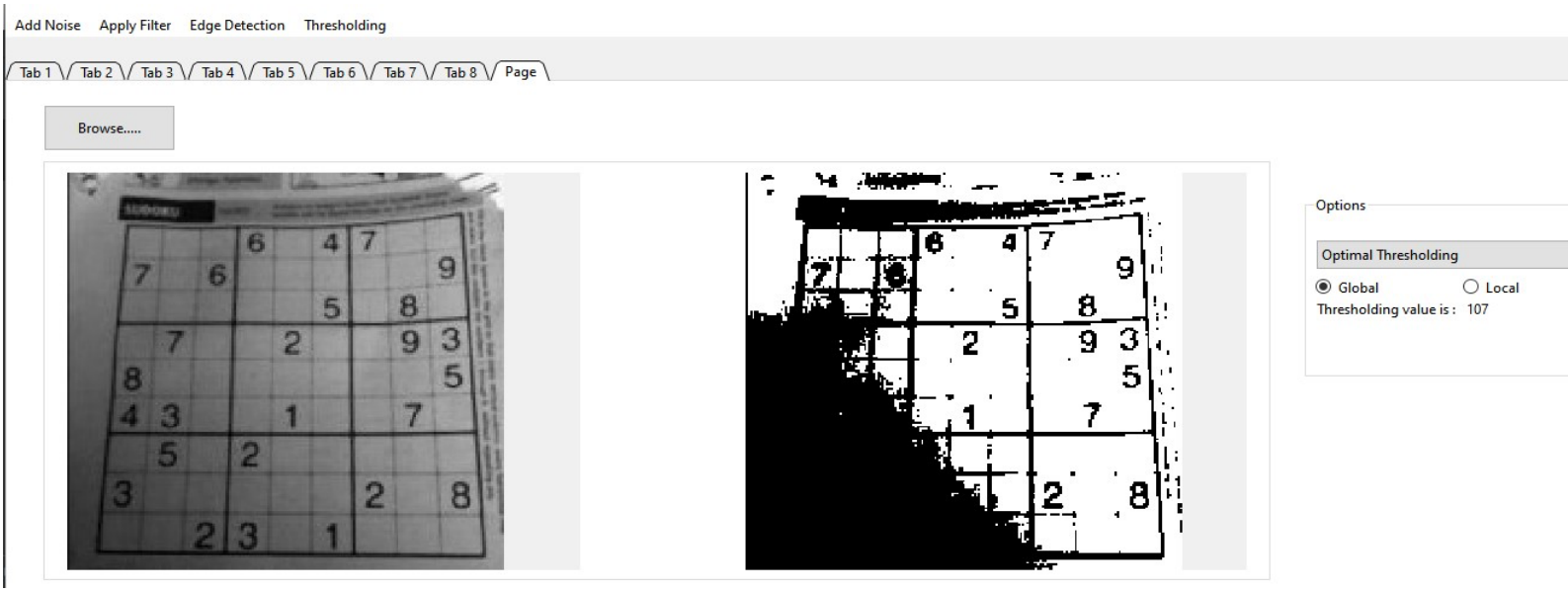
1-Optimal global thresholding:

1.1 Applying optimal global thresholding on even illumination image:



1.2 Applying optimal global thresholding on uneven illumination image:

Global thresholding is not suitable in this case



2-Optimal local thresholding:

Note:

Output is dependent on the Changing of the box size.

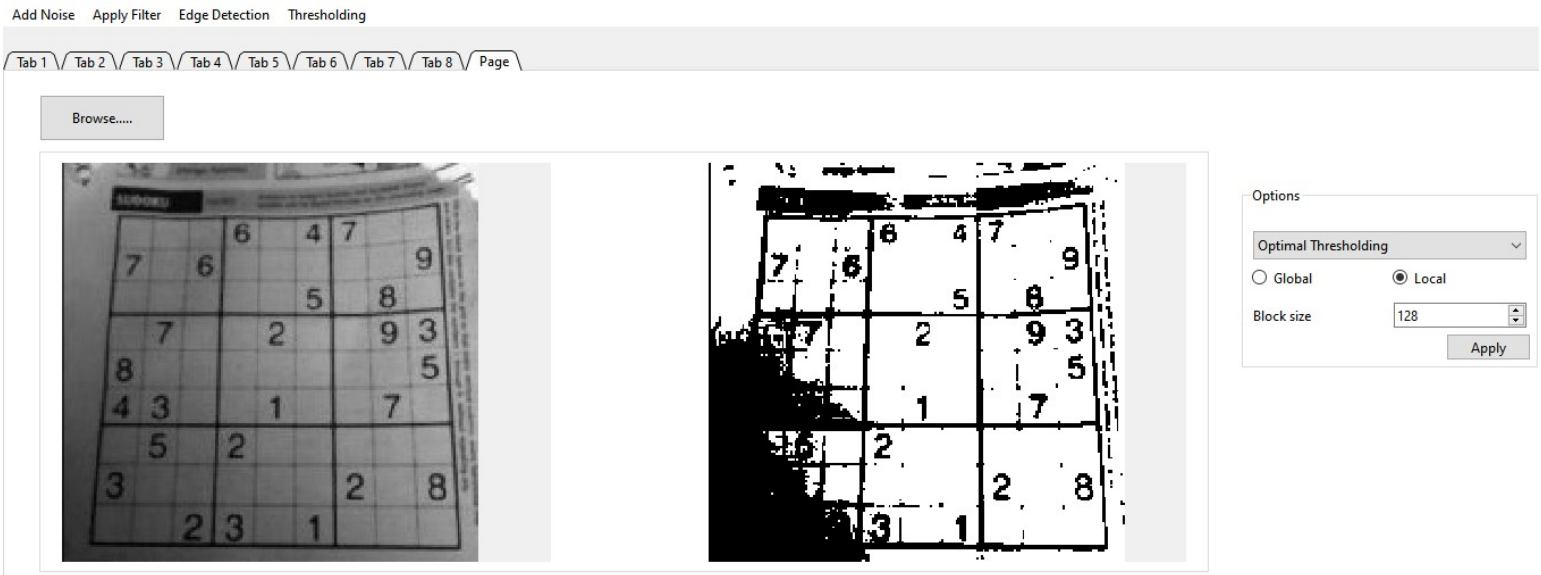
2.1 Applying optimal local thresholding on even illumination image:

- **With big block size:** output will be almost accurate.

2.2 Applying optimal local thresholding on uneven illumination image:

With small block size:

With big block size:



We choose to use global or local thresholding depending on image illumination and contrast.

Note:

This approach is limited to images with only two distinct intensity regions and may not work well for images with more complex intensity distributions. In such cases, more advanced algorithms like Otsu's method may be more appropriate for optimal threshold selection.

- ## OTSU'S thresholding:

  - The Otsu algorithm, also known as the maximum variance method, is a simple yet powerful image thresholding technique that automatically determines the optimal threshold value to separate an image into foreground and background pixels. It was proposed by Nobuyuki Otsu in 1979 and has since become one of the most widely used algorithms in computer vision and image processing.
  - The basic idea behind the Otsu algorithm is to find a threshold value that maximizes the variance between the foreground and background pixels, while minimizing the variance within each of these regions. The variance is a measure of how spread out the pixel values is within each region**. By maximizing the variance between the two regions**, we can find a threshold value that effectively separates them.

- The Otsu algorithm is widely used in a variety of applications such as image segmentation, object recognition, and optical character recognition (OCR). It is particularly useful when the image has a bimodal histogram, meaning that the pixel intensities are divided into two distinct groups, such as in the case of foreground and background regions.

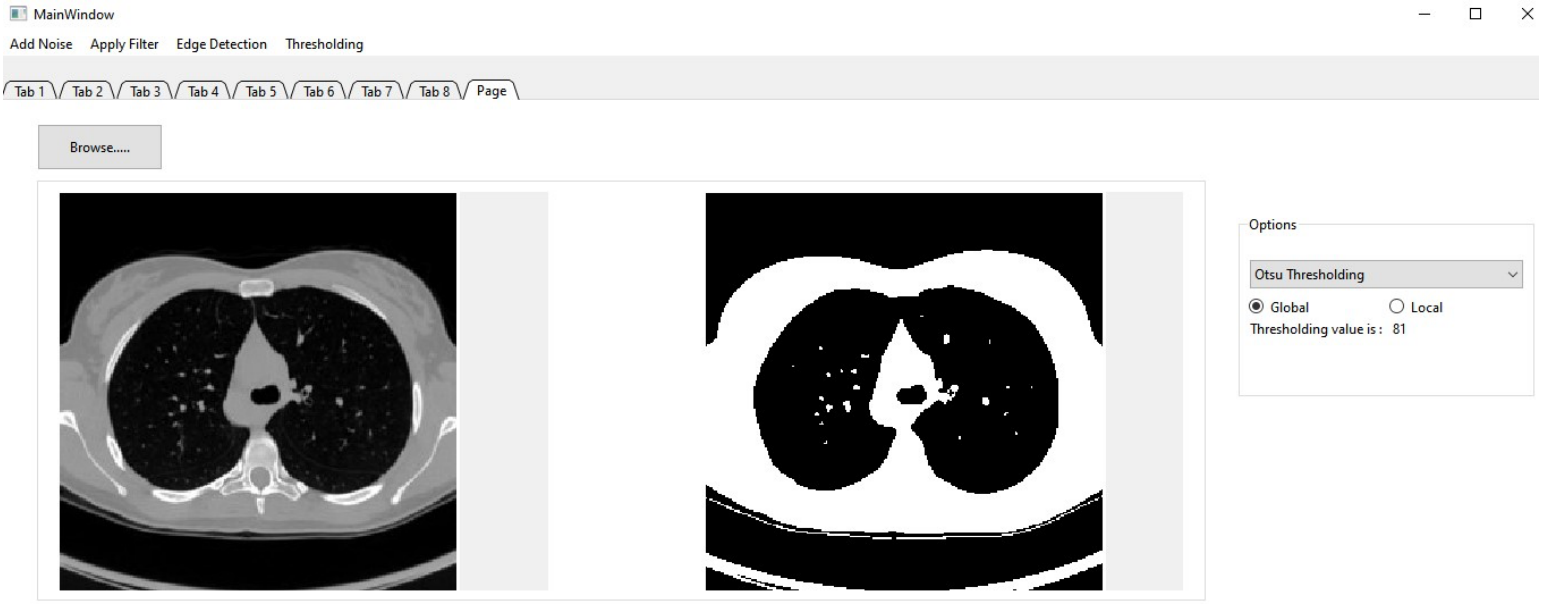
Our Algorithm:

1. Compute the histogram of pixel intensities in the input image. The histogram is represented as an array of size 256, with each element storing the frequency of occurrence of the corresponding intensity value (ranging from 0 to 255).
2. Compute the total number of pixels in the image.
3. Compute the sum of all pixel intensities in the image.
4. Initialize the variables for the within-class variance, threshold value, and weights of the background and foreground regions.
5. Iterate through all possible threshold values from 0 to 255, and for each threshold value:
    - Update the weights and within-class variance based on the current threshold value.
    - Compute the between-class variance as the product of the weights and the square of the difference between the means of the background and foreground regions.
    - Update the maximum between-class variance and corresponding threshold value if the current between-class variance is larger.
6. Return the threshold value that maximizes the between-class variance.

Output Examples:

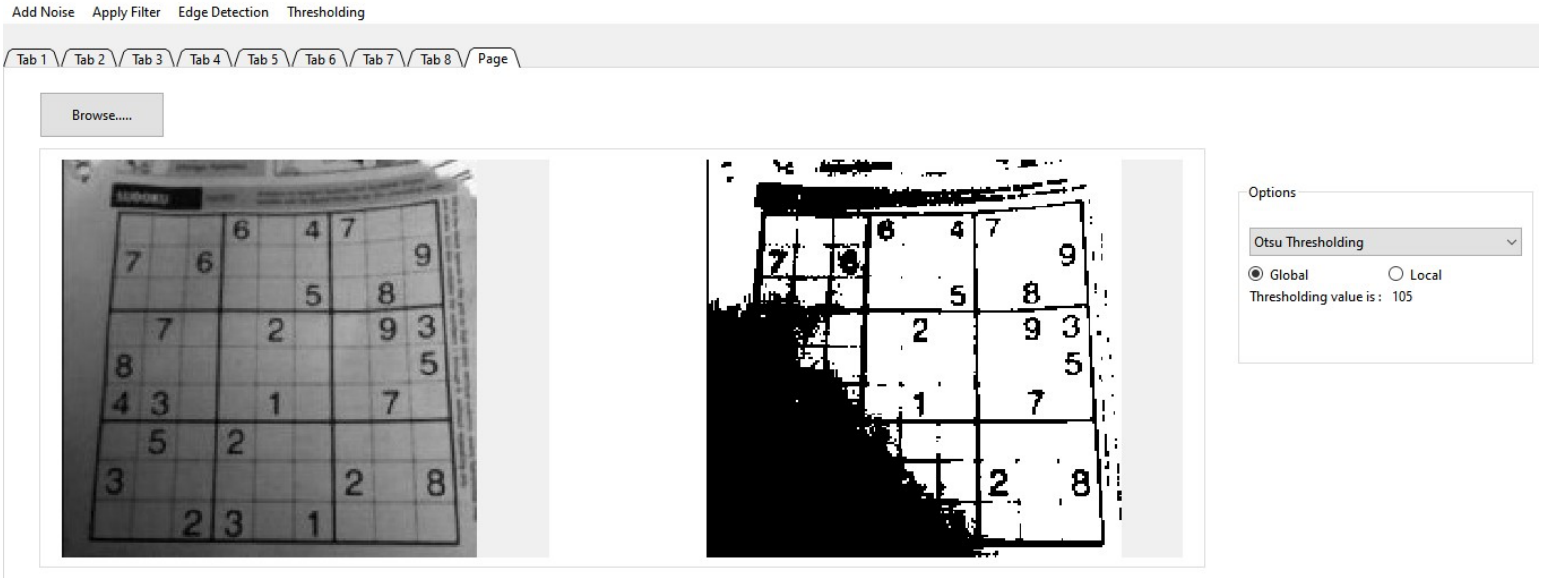
1- Otsu global thresholding:

1.1 Applying Otsu global thresholding on even illumination image:



1.2 Applying Otsu global thresholding on uneven illumination image:

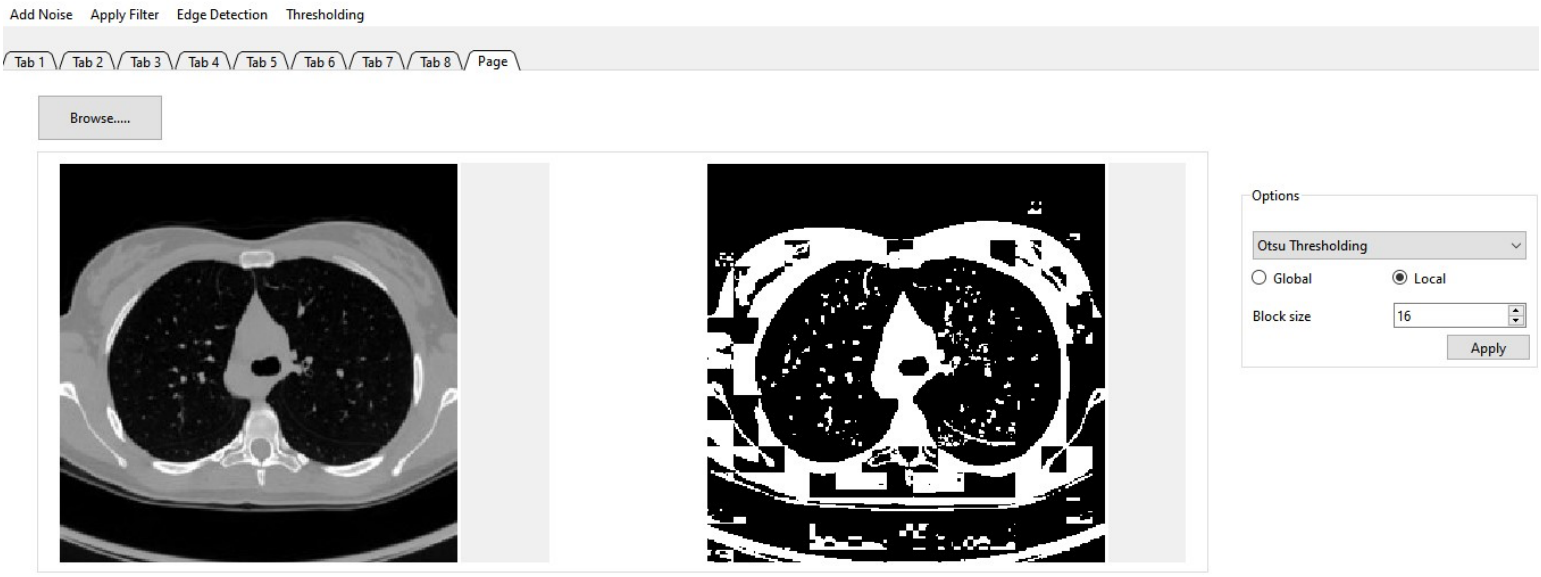
Global thresholding is not suitable in this case



2-Otsu local thresholding:

2.1 Applying Otsu local thresholding on even illumination image:

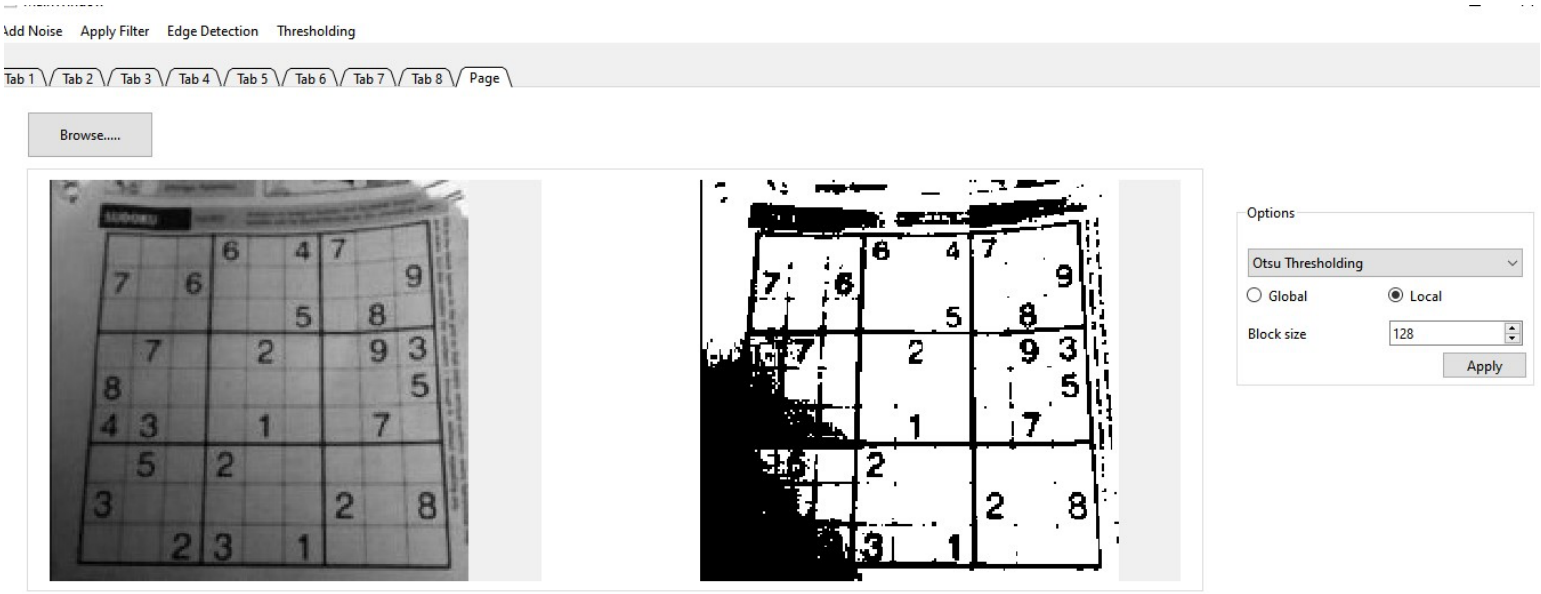
- **With small block size:**



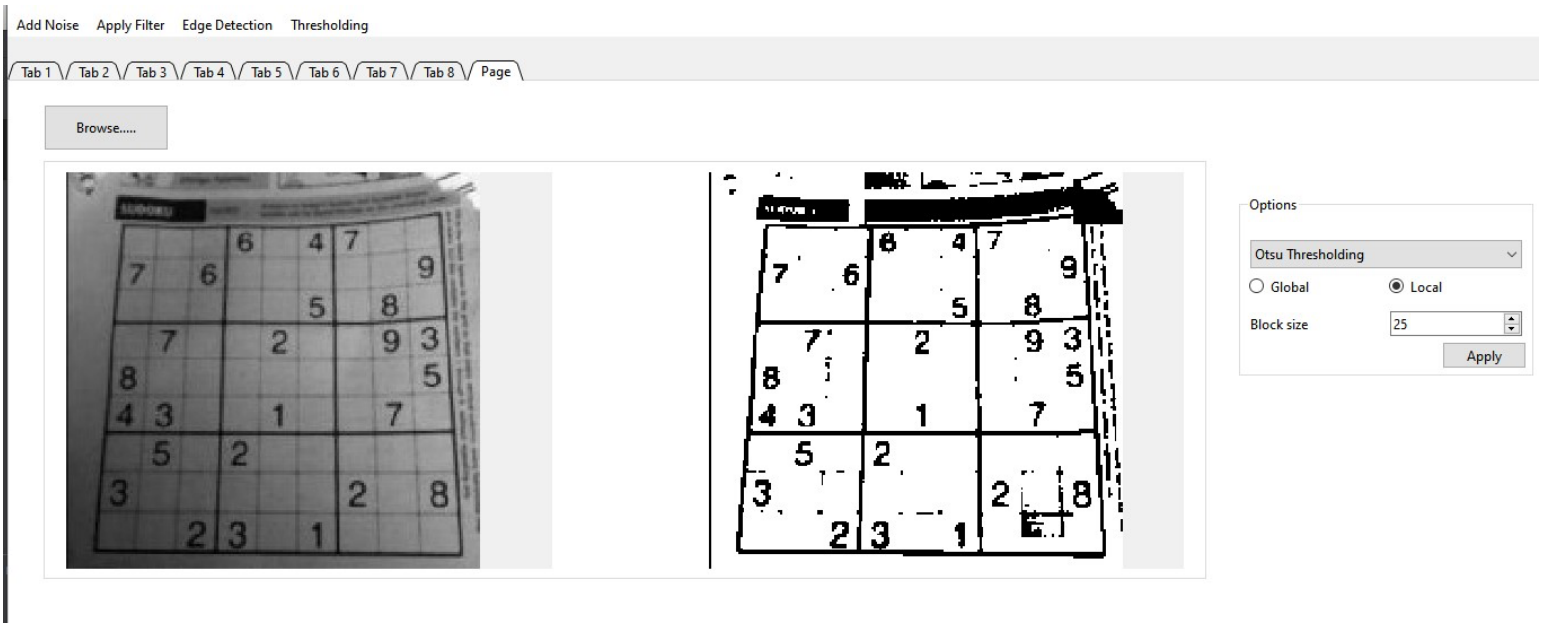
- **With big block size:**

2.1 Applying Otsu local thresholding on uneven illumination image:

- **With big block size:**



With small block size:



- ## Spectral Thresholding

  Spectral thresholding is dividing an image into multiple regions, where each region representing a distinct object or part of the image.

  In our algorithm, divide the image into 3 regions using 2 different threshold values.

  The regions are white , black , and gray.

  ### Our Algorithm:

  1. Calculate histogram of the input image.
  2. Calculate the cumulative sum of histogram values.
  3. Calculate the total number of pixels in the input image.
  4. Initialize the variables for the within-class variance, weights of the background and foreground regions, and the two threshold values.
  5. Calculate the within-class variance and between-class variance.
  6. Update the first threshold value if the between-class variance is the maximum variance calculated.
  7. Calculate the cumulative sum of the histogram values and updates tau_2 value.
  8. Calculate the value of second threshold value which is equal to the average value of the first threshold and tau2 values.
  9. Compensate the thresholds' values because spectral method tends to produce threshold values that are too high.
  10. Returns both values of the threshold.

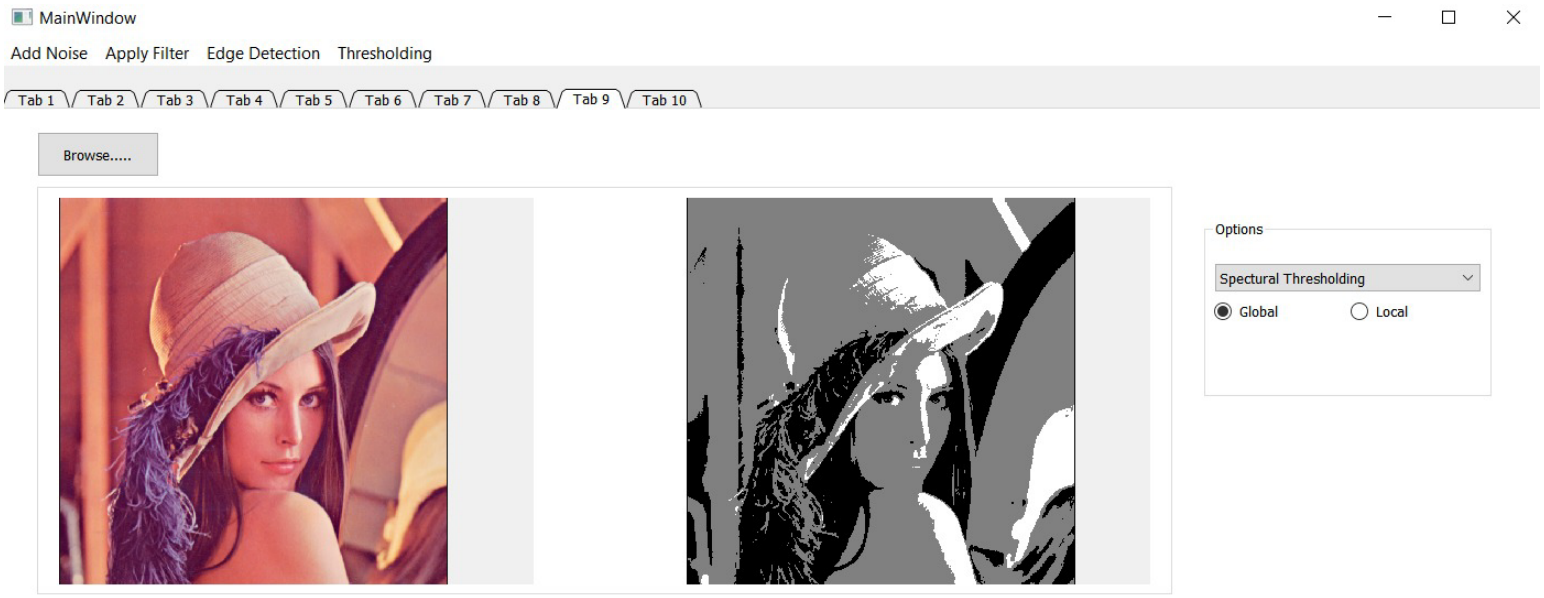
Double thresholding:

The calculated thresholds are then used in a function called double thresholding.

This function sets the conditions for each of the 3 regions by using these threshold values.

Output Examples:

1. Global Thresholding



2. Local Thresholding

- With block size 150.

- With block size 200.

Clustering:

- K-Means Clustering (Refer to lines 9-242 in clustering.cpp)

  Our Algorithm:

1- Get the value of k from the user.
2- Initialize k centroids by taking k random points from the image and for each centroid create an empty vector to store the points for each cluster. (Refer to lines 12-36 in clustering.cpp)
3- For each initial centroid get the associated pixels to get the initial clusters by computing the Euclidean Distance between each centroid and each pixel present in the image using the RGB intensity values. (Refer to lines 64-102 in clustering.cpp)
4- Adjust the k available centroids, by taking the mean of each cluster as the new centroids and then looping through each pixel again to get the minimum Euclidian Distance of the RGB values of that pixel with respect to that new centroid. (Refer to lines 111-157 in clustering.cpp)

5- Paint the found clusters by choosing a random color for each cluster and then painting the points(pixels) of each cluster with that random color to get the clustered image. (Refer to lines 165-190 in clustering.cpp)

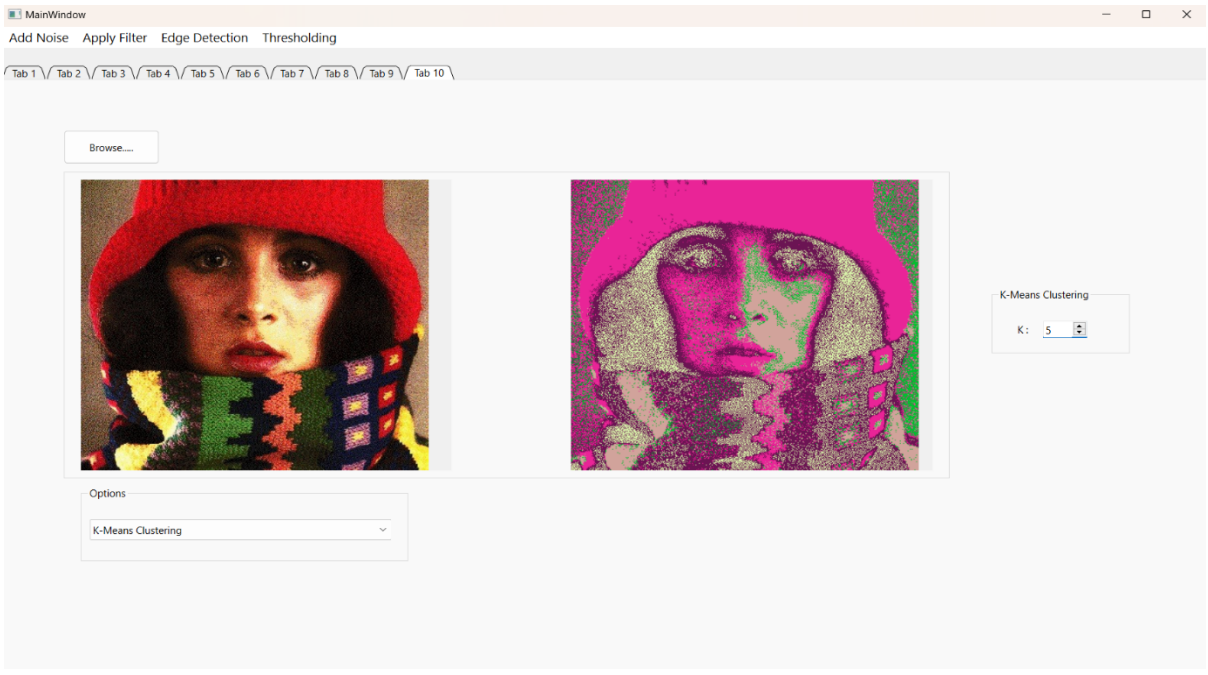
6- Perform the K-means clustering by taking the input image and the K value as inputs and then looping to get the centroids, and for each iteration compare the difference between the previous and adjusted centroid if this value is below certain threshold (set to 0.1 in our implementation) then the centroids didn't move much from the previous iteration, so stop iterating. (Refer to lines 196-242 in clustering.cpp)

Credits go to this paper:
(PDF) Image Segmentation using K-means Clustering (researchgate.net)

Output Examples:

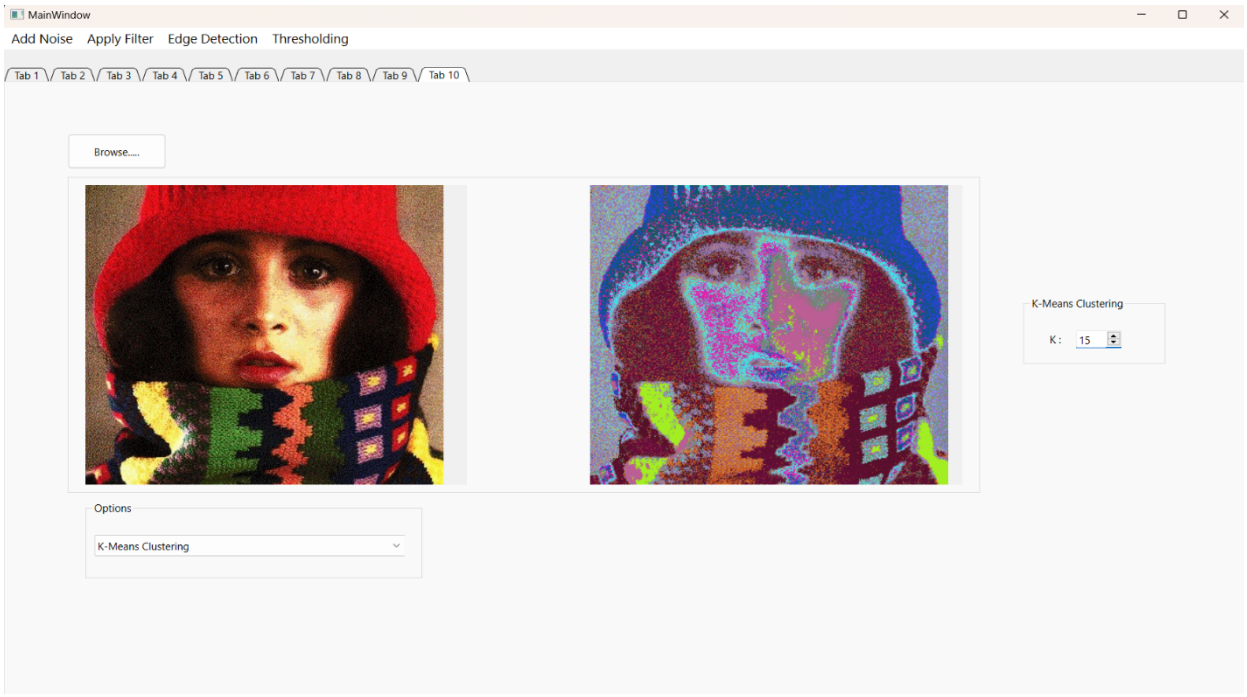
- **Case 1**

  - **K = 5**



- **Case 2**

  ➢ **K = 15**



- Agglomerative Clustering (Refer to lines 253-498 in clustering.cpp)

Our Algorithm:

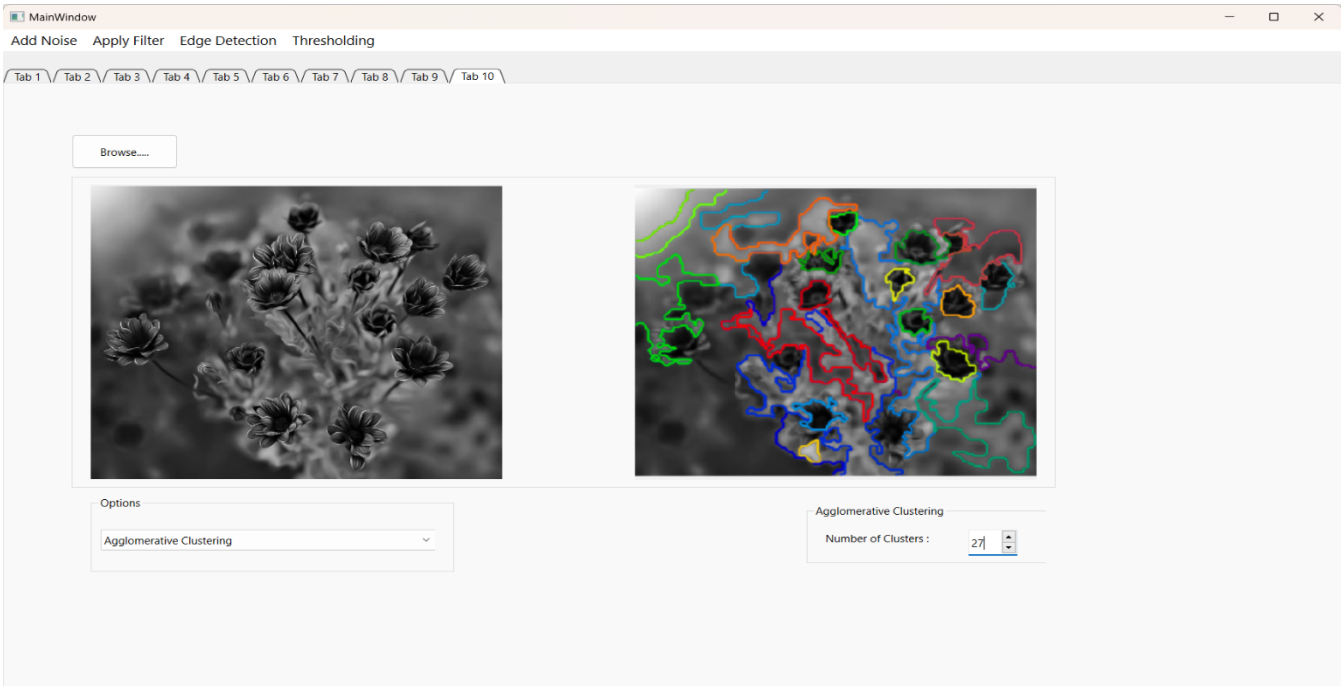
1- Get the input image & the number of desired clusters as inputs.
2- For each input image flatten the image into single 1D vector of pixels.
3- Get a vector that will hold all the found clusters & initially it has each pixel present as its own cluster.
4- For each pixel (cluster) present get the Euclidean Distance of the RGB values between it & all the previous clusters store these Euclidean distances for each cluster in 2D vector similar to the clusters vector but instead of storing points you are going to store double values.

5- Loop by starting with initial number of clusters equal to the total number of pixels and after each iteration decrease the number of clusters by 1 till you reach your desired number of clusters & through each iteration:

- Get the minimum nonzero Euclidean Distances among all clusters.
- Get the 2 clusters corresponding to the minimum Euclidean Distance & join them to form a new cluster.
- Update the Euclidean Distance vector of the newly combined clusters by getting the mean of both of the 2 combined clusters' Euclidean distances.
- Update the available clusters vector by shifting them towards the newly combined cluster and then pop the last cluster (became redundant).
- Paint the found clusters on the image by choosing a random color for each one of them.
- Clear all vectors.

Output Example:

- **K = 27**



Note:

- The above output was obtained using python having 27 as the number of clusters, as our implementation was computationally exhaustive and could barely run with very small images of 30,000 pixels.
- Also note, that as you increase the number of clusters towards the total number of pixels of the image, you will end up with less computational time, as we initially assumed that each pixel is a cluster.

1-Region Growing:

Region growing is a simple and effective image segmentation algorithm
It segments the images by grouping pixels together with similar properties into regions or objects.
Region growing works by starting with a seed pixel or a set of seed pixels and then iteratively adding neighbouring pixels to the same region based on some similarity criterion. The process continues until all pixels in the image have been assigned to a region.

Our Algorithm:

- **gray_diff** function calculates the absolute difference between the grayscale intensity values of two pixels in an image. it is used to determine the similarity between neighboring pixels and decide whether to group them together.

- The **neighboring_pixels** function returns a list of neighboring pixels that should be considered for grouping. The p parameter determines the type of connectivity (4 or 8) used for the algorithm ‘mostly used as 8 connections.

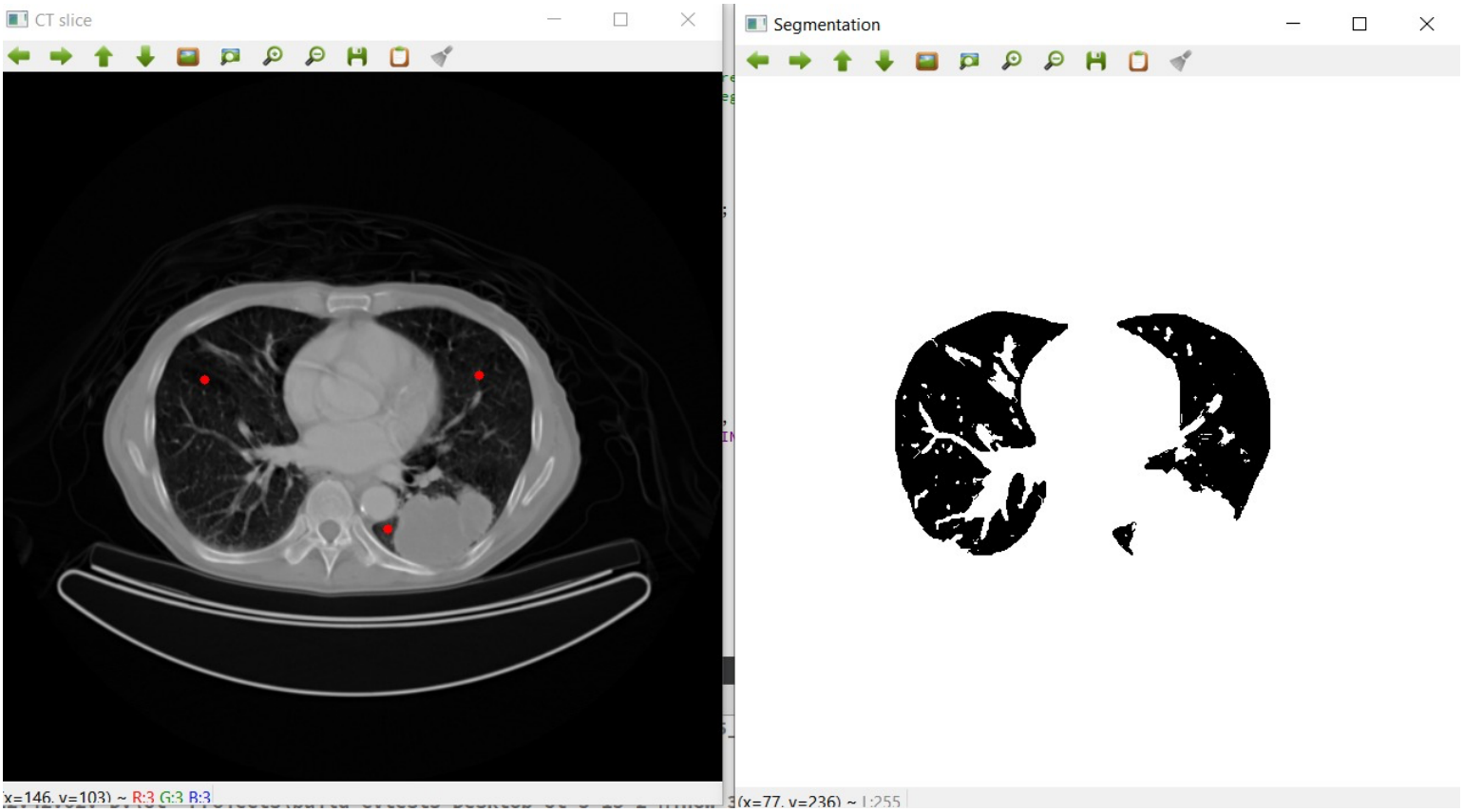
-The **Region_Growing** is the main implementation of the region growing algorithm. It takes in an input image, a set of seed points, a threshold value, and the type of connectivity to use. It then iteratively groups neighboring pixels together based on their similarity, until all pixels in the image have been assigned to a region. The output of the function is a labeled image, where each pixel is assigning a unique label corresponding to the region it belongs to.

We tried to 2 approaches

First Approach:

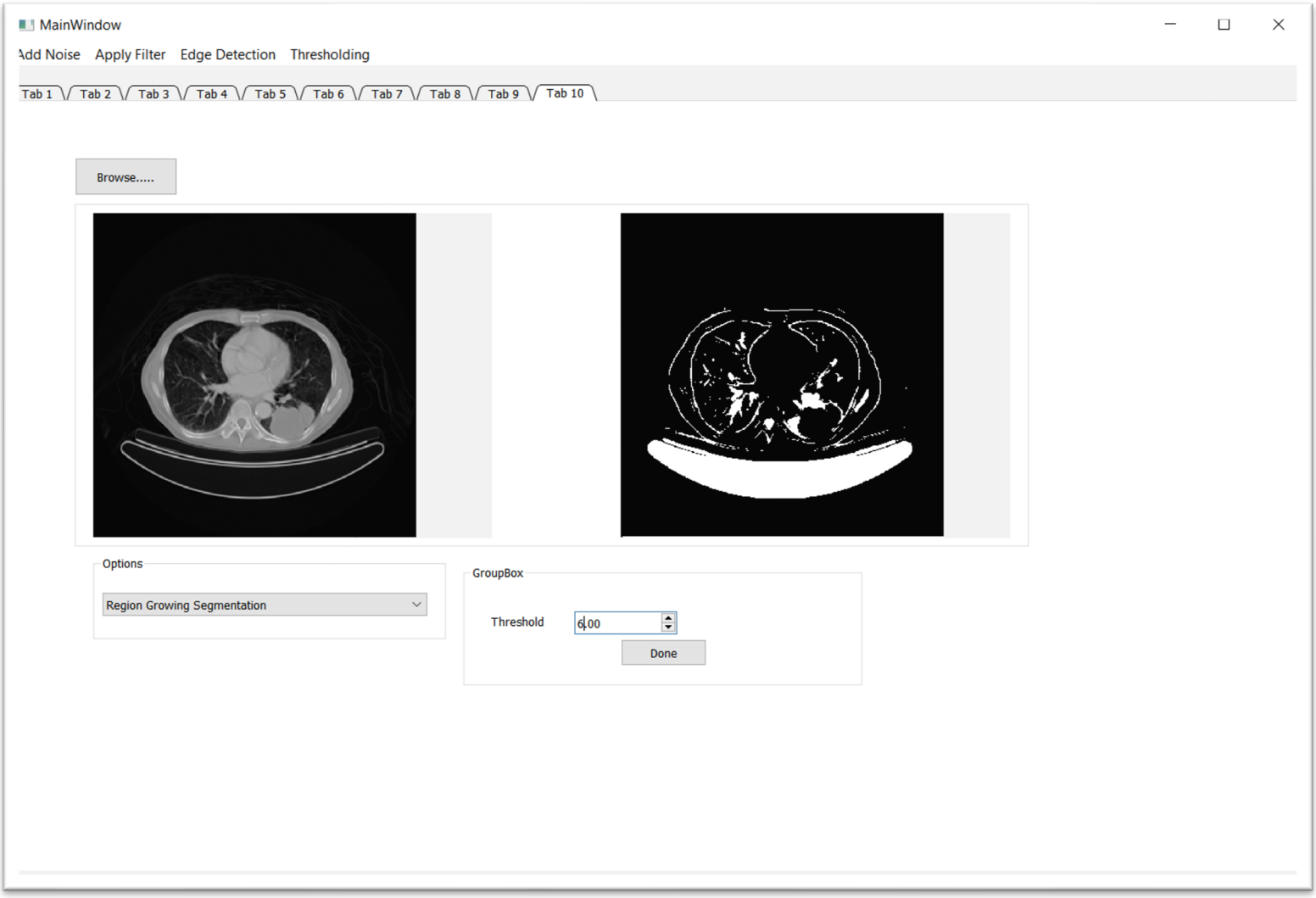
To make the user specify the seed points from which we can start our algorithm to detect certain object in the image
Note: unfortunately, we couldn't connect it to Qt but there will be a file with the project with the code

Output Example:



Second Approach:

we set the seed point so that the whole image will be segmented by our algorithm



1-Mean Shift:

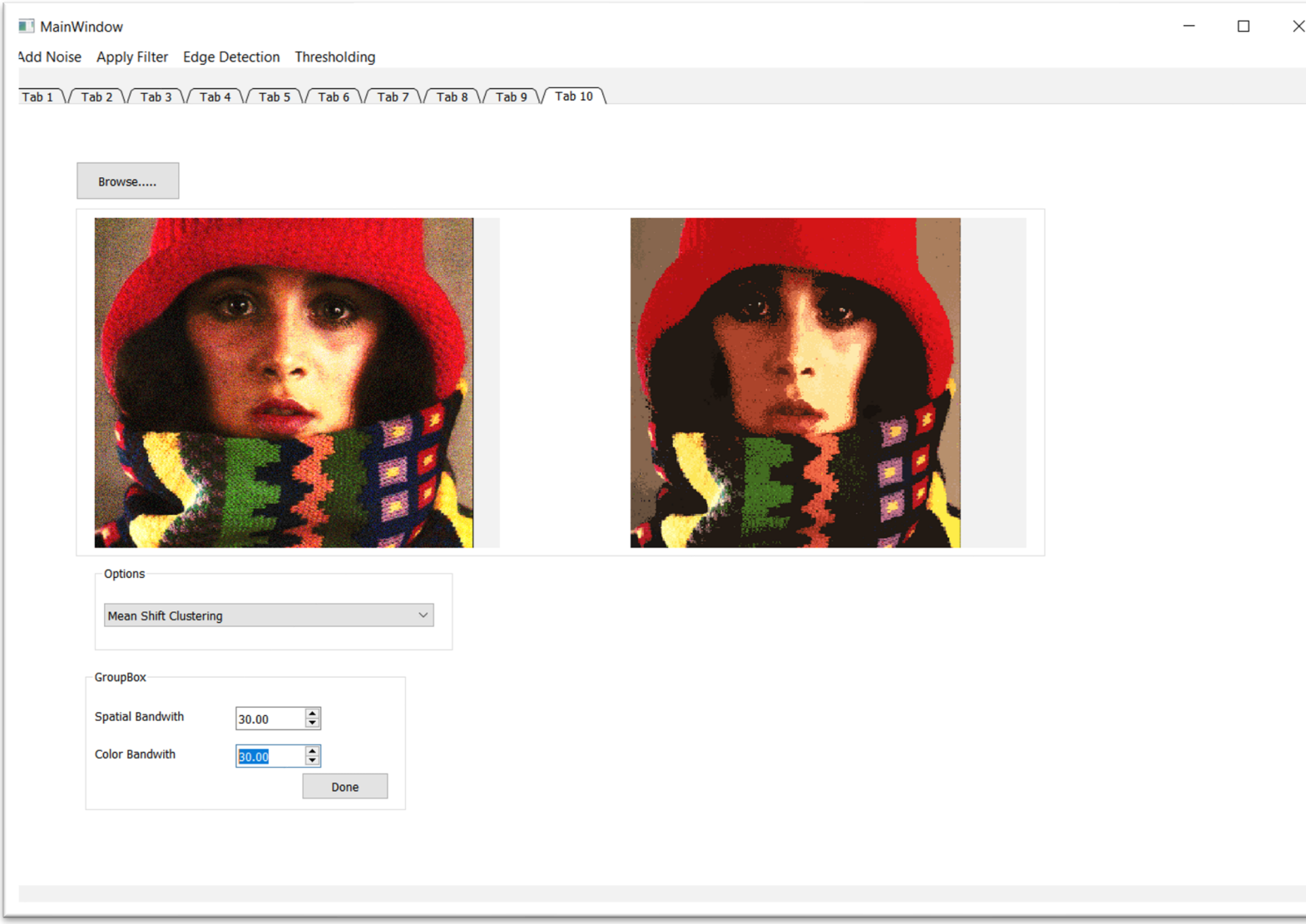
Mean shift segmentation is an image segmentation technique that uses a non-parametric clustering approach to group together regions in an image with similar properties, such as color or texture. The algorithm works by iteratively shifting the data points in the feature space towards the local modes of the probability density function, as it does not require the number of clusters to be predetermined, mean shift segmentation is an effective technique that does not require a priori knowledge of the number of clusters, making it useful in a wide range of applications.

The algorithm starts with an initial estimate of the density modes and iteratively refines these estimates by shifting the data points towards the nearest modes. The kernel function is used to weight the contribution of each neighbouring point to the mean shift vector, with the bandwidth parameter controlling the size of the region around each data point. The algorithm terminates when the data points no longer move or have converged to a local mode.

Our Algorithm:

1. The MeanShift algorithm starts with filtering the input image using the mean shift filtering technique, which is implemented in the function MSFiltering. This filtering step is used to smooth the image and reduce noise, while also preserving edges and details.
2. The MeanShift segmentation algorithm then proceeds to iterate over each pixel in the input image. For each pixel, a spatial window is defined around it with a radius determined by the spatial bandwidth parameter.
3. Within this spatial window, a color window is defined around the pixel with a radius determined by the color bandwidth parameter.
4. The MeanShift algorithm then iteratively computes the mean shift vector for the current pixel, which is the weighted average of the color values of the neighbouring pixels within the color window, with weights determined by a kernel function. The kernel function determines the contribution of each neighbouring pixel to the mean shift vector based on its distance from the current pixel.
5. The current pixel is then shifted to the position indicated by the mean shift vector.
6. Steps 4 and 5 are repeated until the pixel converges to a local mode, which is a region of high density in the feature space.
7. The algorithm assigns a unique label to each region of connected pixels and calculates the average color value for each region.
8. Finally, the algorithm uses the labels and average color values to generate the final segmented image, where each pixel is assigned the color value of its corresponding region.

Output Example:



Reference:

https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf