1- For grey images:

Optimal Thresholding

Optimal thresholding is an approach for image segmentation, aiming to find the most suitable threshold value that effectively separates objects from the background, optimal thresholding iteratively adjusts the threshold based on the mean intensities of object and background pixels until convergence is achieved.

- ### Global Optimal Thresholding:

Global optimal thresholding calculates a single threshold value for the entire image. Here's an overview of the process:

1. The mean intensity of the four corners of the image is computed, along with the mean intensity of the object area excluding the corners.
2. An initial threshold is calculated as the average of these two means.
3. Iteratively, the image is segmented based on the current threshold. The mean intensities of object and background pixels are computed.
4. A new threshold is calculated as the average of these mean intensities.
5. The process continues until the difference between the new and current thresholds falls below a predefined threshold.
6. Finally, the image is thresholded based on the converged threshold value.

- ### Comparison with the built-in function

Figure 1. Implemented Function

Figure 2. Built-in Function

- ### Local Optimal Thresholding

Local optimal thresholding extends the global approach by computing threshold values for smaller local regions within the image. Here's how it works:

1. The image is divided into non-overlapping blocks or overlapping blocks of a specified size.
2. For each block, a local region of interest (ROI) is defined.

3. The same optimal thresholding algorithm used in the global approach is applied independently to each local ROI, resulting in a local threshold value for each region.
4. These local threshold values are then assigned to their corresponding regions in the output image.
5. As a result, the thresholded image consists of multiple local regions, each with its own threshold, effectively capturing variations in illumination and object/background characteristics across the image.

- Comparison with the built-in function

Figure 1. Implemented Function

Figure 2. Built-in Function

Otsu Thresholding

Otsu thresholding is a widely used technique for image segmentation, which aims to separate objects of interest from the background in an image. It automatically determines an optimal threshold value based on the image's histogram.

- Global Otsu Thresholding:

It calculates a single threshold value for the entire image. Here are the steps for it:

1. The histogram of the grayscale image is computed, representing the frequency distribution of pixel intensities. And the cumulative sum of the histogram is calculated.
2. For each possible threshold value, the between-class variance is computed. The between-class variance measures the separability between foreground and background pixels. And the threshold that maximizes the between-class variance is selected as the optimal threshold value.

3. The selected threshold is applied to the image, resulting in a binary image where pixels above the threshold are assigned a value of 255 (white) and pixels below the threshold are assigned a value of 0 (black).

- ## Local Otsu Thresholding

It extends the global approach to calculate threshold values for smaller local neighborhoods within the image. Here are the steps for it:

1. The image is divided into overlapping or non-overlapping blocks of a specified size. For each block, a local region of interest (ROI) is defined, representing a smaller portion of the image.
2. The global Otsu thresholding algorithm is applied to each local ROI, resulting in a local threshold value specific to that region. The local threshold values are assigned to their corresponding regions in the output image, resulting in a thresholded image where each local region has its own threshold.

- Comparison with the built-in function

Figure 1. Built-in Function

Figure 2. Implemented Function

Spectral Thresholding

Spectral thresholding is a technique used for image segmentation, which aims to separate objects of interest from the background based on the spectral characteristics of the image.

- Global Spectral Thresholding

1. The histogram of the input image is computed to analyze the distribution of pixel intensities. The mean value of the entire image is calculated as the weighted average of the pixel intensities and their corresponding frequencies.

2. The algorithm iterates over all possible threshold values, comparing the variance between modes for each combination of high and low thresholds. The thresholds with maximum between-class variance are selected as the optimal values.
3. The optimal threshold values are used to classify pixels in the image. Pixels below the lower threshold are assigned a value of 0 (black), pixels between the lower and higher thresholds are assigned a value of 128 (gray), and pixels above the higher threshold are assigned a value of 255 (white).

- ## Local Spectral Thresholding

1. The image is divided into overlapping or non-overlapping blocks of a specified size. For each block, a local region of interest (ROI) is defined, representing a smaller portion of the image. The global spectral thresholding algorithm is applied to each local ROI, resulting in local optimal threshold values.
2. The local optimal threshold values are used to classify pixels in their respective regions of the output image.

- ## Comparison with the built-in function

Figure 3. Built-in Function

Figure 4. Implemented Function

2- For coloured images:

A) Map RGB to LUV

The RGB to Luv* conversion function takes RGB values as input and computes the corresponding Luv* values.

It begins by scaling the RGB values to the range [0, 1].

Next, it converts the RGB values to XYZ color space using a conversion matrix.Then, it normalizes the XYZ values with respect to a white reference.

After normalization, it calculates the Luv* values using the formulas defined for the transformation.

Finally, it scales the Luv* values to the range [0, 255] to ensure compatibility with the image data type.

Original Image

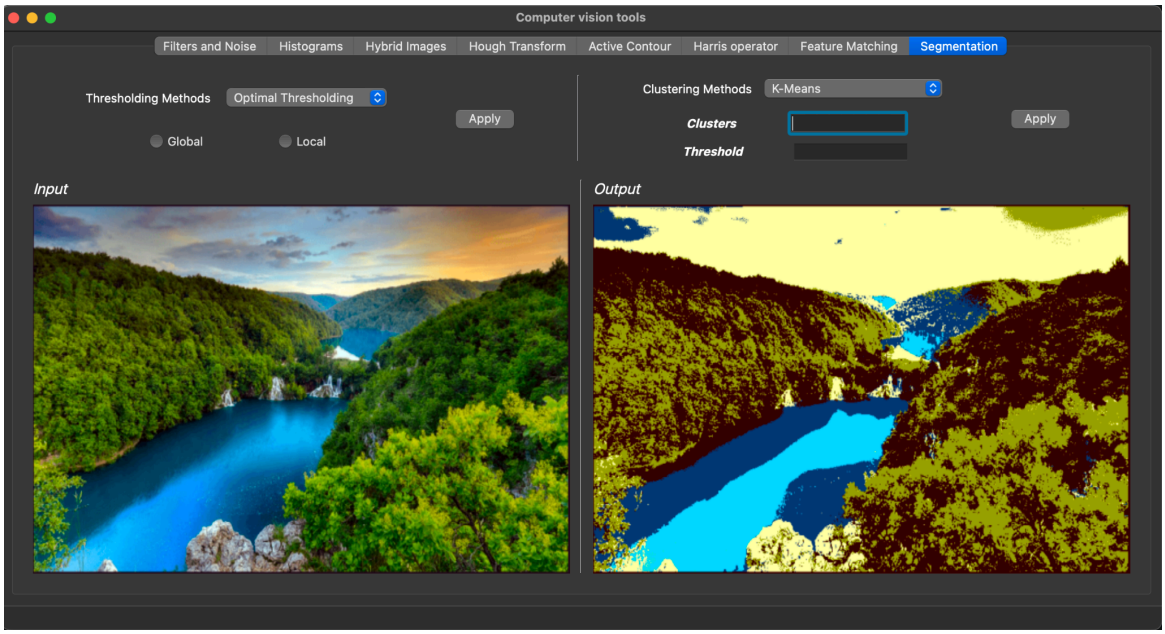
Mapped to L*u*v*

B) Segmentation by Clustering

k-means:

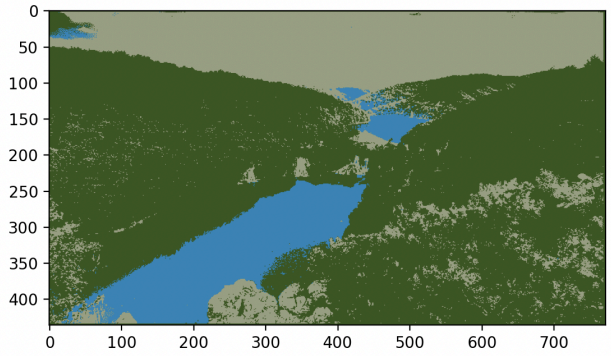
1. Choose the number of clusters you want to find which is k.
2. Randomly assign the data points to any of the k clusters.
3. Then calculate the center of the clusters.
4. Calculate the distance of the data points from the centers of each of the clusters.
5. Depending on the distance of each data point from the cluster, reassign the data points to the nearest clusters.
6. Again calculate the new cluster center.
7. Repeat steps 4,5 and 6 till data points don't change the clusters, or till we reach the assigned number of iterations.

results with clusters = 3



output using openCV :

note : we notice
some change in colors in our output because of our function that display the colored
image on a label



Region-Growing:

Region growing is a pixel-based image segmentation method. Here are the steps involved in the region growing algorithm:

1. **Initialization**: Select a seed point in the image. This can be done **manually** (e.g., by user input) or **automatically** (e.g., by selecting the pixel with the highest or lowest intensity value or by calculating seed points chosen as a function of the image dimensions).

2. **Region Growing:** Compare the seed point's value (e.g., intensity, color) to its neighboring pixels. If a neighboring pixel's value is similar to the seed point (within a predefined homogeneity threshold), add it to the region and update the region's average value. This pixel then becomes a new seed, and its neighbors are considered for inclusion in the region. Each region can be marked by an integer, letter, etc. to differentiate between them.

Using Euclidean Distance to measure pixel homogeneity (RGB image):

$$\Delta C_E = \sqrt{\Delta R^2 + \Delta G^2 + \Delta B^2},$$

Using Absolute Difference of image intensities to measure pixel homogeneity (Grayscale image):

$$D = \left| I[x_0, y_0] - I[x_1, y_1] \right|$$

3. **Iteration:** Repeat the region growing step for all the new seeds (i.e., the pixels that were added to the region in the previous step).

4. **Termination:** The algorithm stops when no more pixels can be added to the region, i.e., there are no more neighbors that meet the similarity criterion. The termination criteria can either be that (a) the number of iterations has reached a maximum or that (b) all pixels in the image belong to a region (have been visited).

There are a few things to consider when using region growing:

- **Seed Selection**: The choice of seed points can significantly affect the results. Poor seed selection can lead to over-segmentation or under-segmentation.
- **Similarity/Homogeneity Criterion:** The definition of similarity can vary. It could be based on pixel intensity, color, texture, or other pixel properties. The choice of similarity criterion depends on the specific application.
- **Connectivity**: Either 4-connected or 8-connected neighborhoods can be considered for region growing. 8-connected might result in more coherent regions, but it could also lead

to more noise. 4-connected can lead to less noisy, sharper regions, especially if the seed point is in a region that is more homogeneous in nature.
- **Threshold**: The threshold for determining whether a pixel is similar to the seed point can greatly affect the results. A high threshold might result in larger regions that encompass different objects, while a low threshold might result in over-segmentation.

The pros of region growing include:

- **Simple and Intuitive:** The algorithm is easy to understand and implement.
- **Flexible:** The algorithm can be easily adapted for different applications by changing the similarity criterion, connectivity, etc.
- **Preserves Region Homogeneity:** The algorithm ensures that the pixels within each region are similar to each other, which is not always the case with other segmentation methods.

There are also some cons to consider when using region growing:

- **Unrelated Pixels:** Unless the image has had a threshold function applied, a continuous path of points related to color may exist, which connects any two points in the image.
- **Slowness:** Random memory access slows down the algorithm

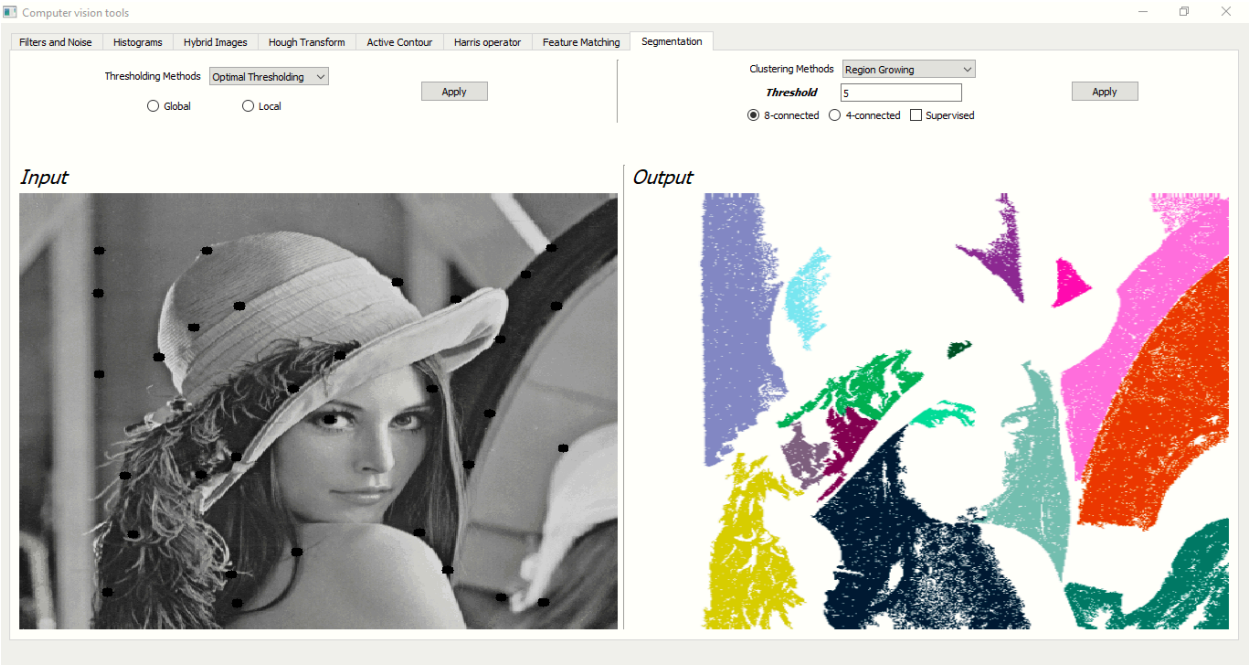
Algorithm Outputs:

Since Region Growing does not have a builtin implementation in OpenCV and in python's standard library , two examples will be shown below (one in RGB and one in grayscale) with different connectivities used to see the segmented image that results from the algorithm.

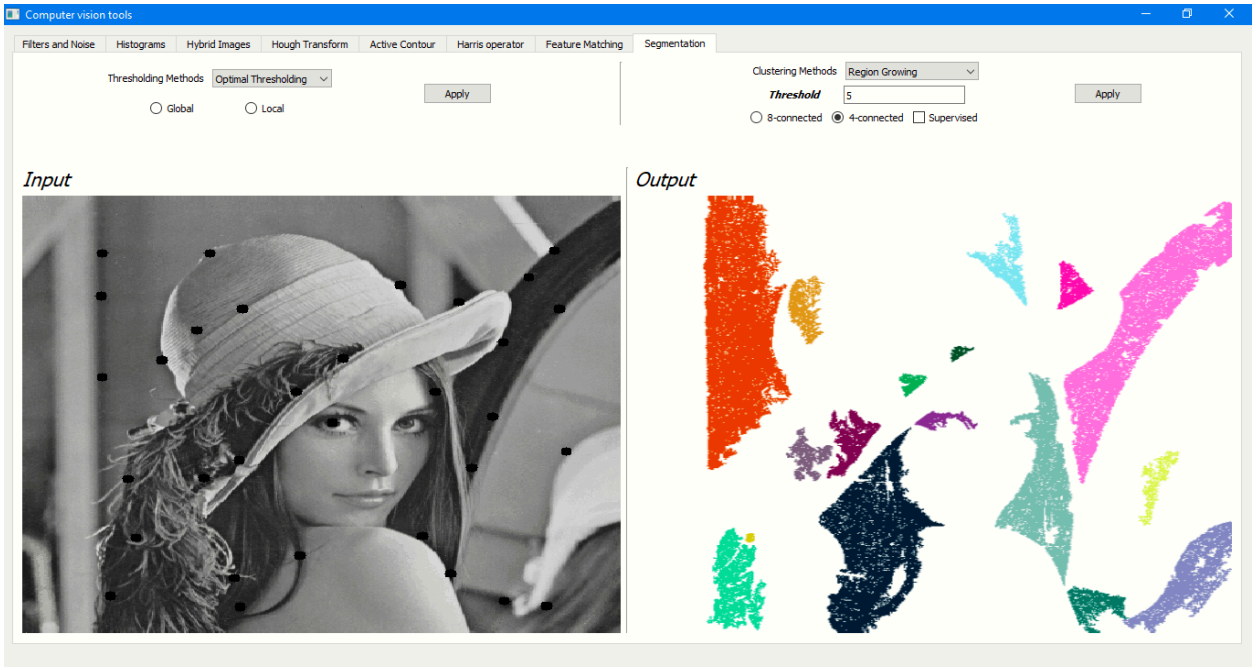
Note: The threshold used and seeds generated from the image's local minima can be found in the images. Also, each region is colored differently to separate them.

Grayscale:

→ *8-connected Neighborhood:* *(32 seed points, threshold = 5)*

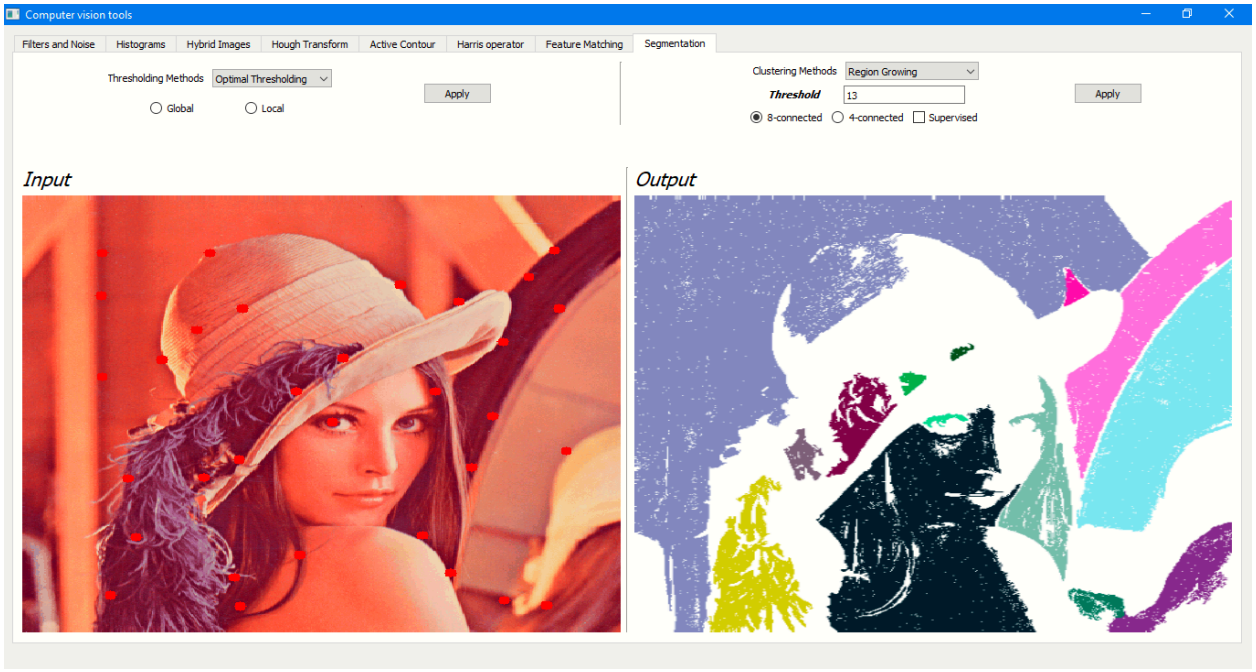


→ *4-connected Neighborhood:* *(32 seed points, threshold = 5)*

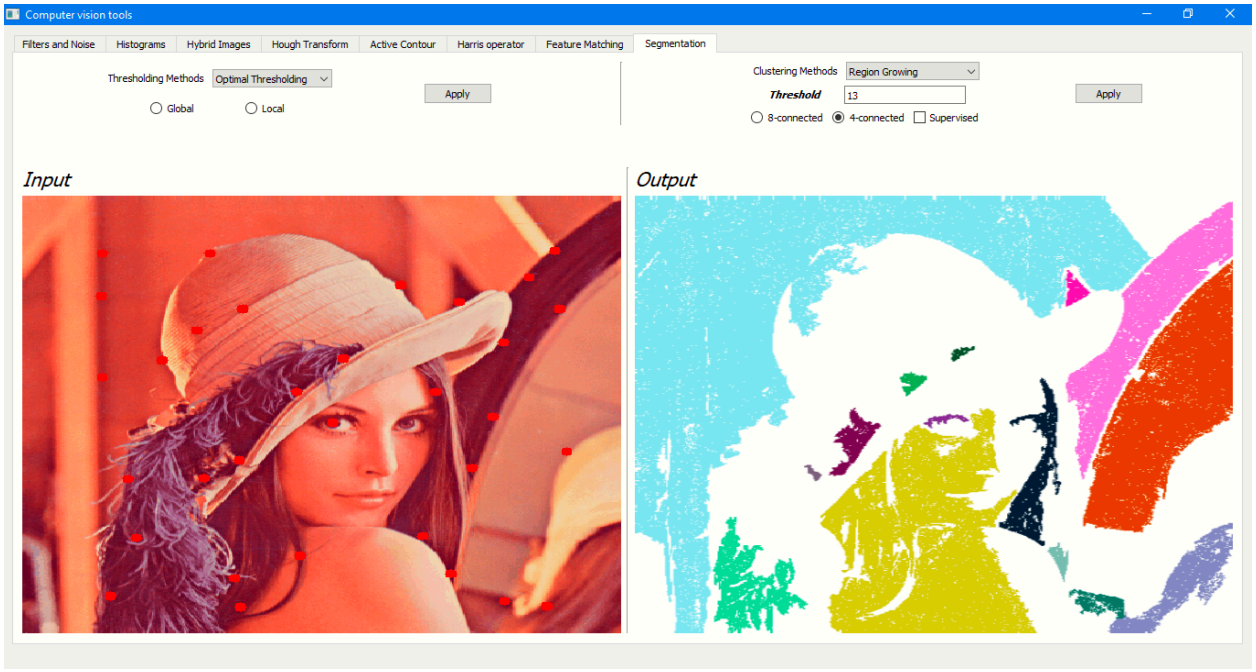


RGB:

→ ***8-connected Neighborhood:*** *(32 seed points, threshold = 13)*



→ ***4-connected Neighborhood:*** *(32 seed points, threshold = 13)*



Agglomerative clustering

Agglomerative clustering is a hierarchical clustering algorithm that iteratively merges clusters of data points based on their pairwise distances. It starts with each data point as a single cluster and then merges the closest clusters until a stopping criterion is met.

Steps:

1. Initialization:
    - Begin with each data point as a singleton cluster.

2. Compute Pairwise Distances:
    - Compute the distance (similarity) between each pair of clusters. Various distance metrics can be used, such as Euclidean distance or cosine similarity.

3. Merge Closest Clusters:
    - Identify the two closest clusters based on the computed distances.
    - Merge these two clusters into a single cluster.

4. Update Distance Matrix
    - Update the distance matrix to reflect the newly formed cluster and its distances to all other clusters.
    - Depending on the linkage criterion chosen (e.g., single-linkage, complete-linkage, average-linkage), the distance between clusters can be computed differently.
5. Repeat:
    - Repeat steps 2-4 until a single cluster containing all data points is obtained or until a specified number of clusters is reached.

Pros of Agglomerative Clustering in Image Segmentation:

1. Adaptability to Image Characteristics:
    - Agglomerative clustering can handle a wide range of image characteristics, including varying textures, colors, and shapes, making it versatile for different types of images.
2. Natural Groupings Preservation:
    - The hierarchical nature of agglomerative clustering allows for the preservation of natural groupings and structures in the image, leading to meaningful segmentation results.
3. Segmentation Hierarchy:
    - The hierarchical segmentation produced by agglomerative clustering provides insights into the image structure at multiple levels of detail, facilitating further analysis and interpretation.

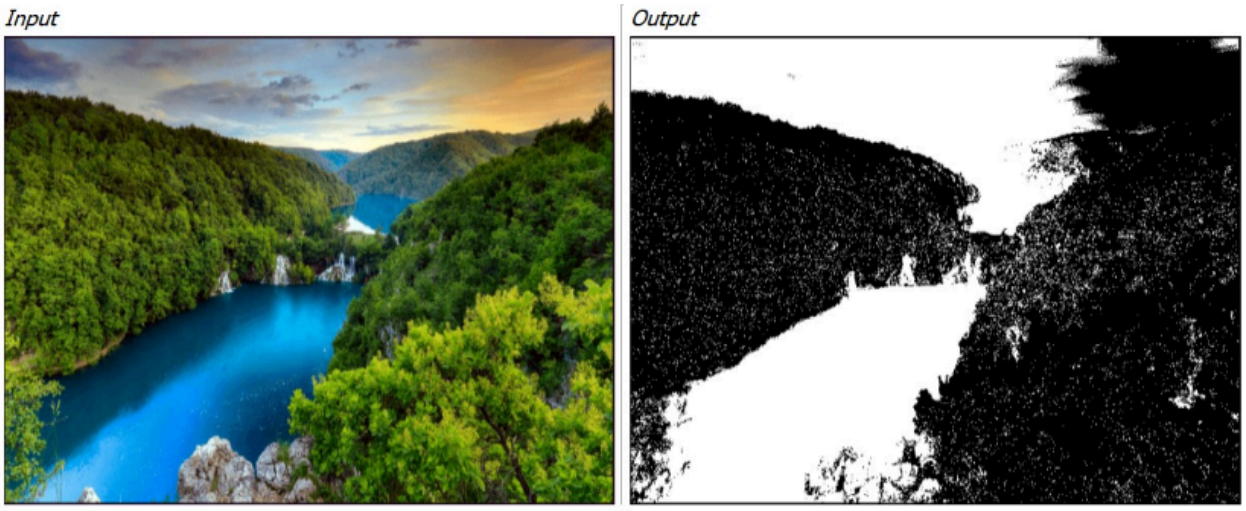
Cons of Agglomerative Clustering in Image Segmentation:

1. Computational Complexity:
    - Agglomerative clustering can be computationally intensive for large images due to the need to compute pairwise distances between all pixels.

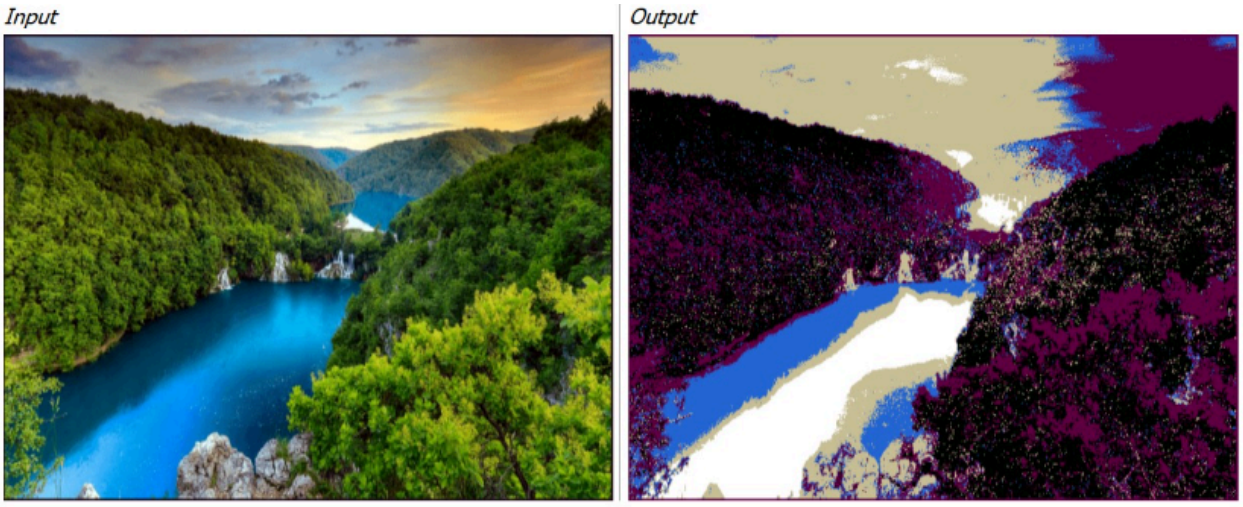
2. Memory Usage:
    - Storing the distance matrix for large images requires significant memory resources, which may limit its use with high-resolution images.

3. Sensitivity to Noise:
    - Agglomerative clustering is sensitive to noise in the image, which can result in the formation of unwanted clusters or segmentation artifacts.
4. Parameter Selection Difficulty:
    - Choosing appropriate distance metrics and linkage criteria can be challenging and may require experimentation to achieve optimal segmentation results.

Results:



Fig(1) Result with final clusters = 2



Fig(2) Result with final clusters = 5

Mean shift algorithm:

The Mean Shift algorithm is a non-parametric clustering algorithm commonly used for image segmentation and object tracking. It works by iteratively shifting data points towards the mode (peak) of the data distribution in the feature space. This results in the formation of clusters around the modes, leading to segmentation of the image into regions with similar characteristics.

`

Algorithm Steps:

1. Initialization:
    - Randomly select a data point from the feature space as the initial mean.

2. Shift Mean:
    - For each data point in the feature space:
        - Calculate the Euclidean distance between the current data point and the current mean.
        - Determine the neighboring data points within a specified window size around the current mean.
        - Compute the mean of the colors and positions of the neighboring data points.
        - Calculate the Euclidean distance between the computed mean and the current mean.
        - If the distance is below a specified convergence threshold:
            - Update the current mean to the computed mean.
            - Color all the neighboring data points with the color of the computed mean.
            - Remove the colored data points from the feature space.
        - Repeat the process until convergence or until no further updates are possible.

3. Segmentation:
    - After convergence, the image is segmented into regions based on the colored data points.

Pros:

- No Assumptions: Mean Shift does not make any assumptions about the shape or size of the clusters, making it suitable for clustering data with complex distributions.

- Automatic Cluster Number Detection: It automatically determines the number of clusters based on the data distribution, eliminating the need to specify the number of clusters beforehand.
- Robust to Outliers: Mean Shift is robust to outliers since it uses a density-based approach to cluster data points.

Cons:

- Computational Complexity: The algorithm's time complexity increases with the size of the dataset and the number of iterations required for convergence, making it computationally expensive for large datasets.
- Sensitivity to Parameters: Mean Shift performance is sensitive to parameters such as the window size and convergence threshold. Selecting optimal parameter values can be challenging and may require experimentation.
- Memory Consumption: Mean Shift requires storing the entire dataset in memory, which can be memory-intensive for large datasets, leading to scalability issues.

Results:

Input image Output image

Fig(3) Mean shift clustering