Cairo University

Faculty of Engineering

Systems and Biomedical Department

Spring 2024

Computer Vision(SBE 3230)

Assignment 4

"Thresholding and Segmentation"

Abdulrahman Emad

Mariam Ahmed Saied

Mourad Magdy

Youssef Ashraf

Ziad Ahmed Meligy

Under the supervision of

Prof/Ahmed Badawy

Eng/Layla Abbas

Eng/Omar Hesham

Table of Contents

Thresholding

Thresholding is one of the simple methods of Image segmentation, Image segmentation groups pixels based on a certain threshold or more than one threshold, there are multiple techniques for thresholding, in our application we implemented the optimum thresholding, Otsu thresholding, and Spectral thresholding, each has its local and global implementation the following.

Optimum Thresholding

In the class **"Threshold"**, the implementation is done in "**optimumThresholding" function** with the following **algorithm:**

Algorithm

1. Input parameters
    a. Mat Image: which is the image to be processed whether it's a grayscale or colored img
    b. Bool isGlobal: A boolean flag that determines if the thresholding is global or local, and the default value is true which is the global.
2. Initialization
    a. Make a copy of the input image img and store it in grey_img.
    b. If isGlobal is true, convert the image to grayscale, which is a handling of the difference between the global and local thresholding, because the local thresholding doesn't need to be converted to grayscale
3. Compute Initial Threshold
    Calculate an initial threshold value currentThreshold using the pixel intensities at four corners of the image, which is my initialization of the background (and the rest is the object) that will be altered iteratively.
4. Compute Probability Density Function (PDF)
    Generate a histogram of pixel intensities using the `thresholdingHistogram()` function, that was implemented in a previous task.
5. Iterative Threshold Optimization
6. following steps are applied in a loop until the difference between the current and previous threshold values is less than a constant epsilon threshold (0.002), which is the convergence
    a. Calculate the sum1, and sum2 which are the probability density function multiplied by the pixel intensity, if the value is more than currentThreshold then we calculate sum1 which is the sum of the background, but if it's loss then calculate that of the object which is sum2.
    b. Calculate the cdf1, and cdf2 which are the running sum of the probability density function of the the background or the object depending on the pixel value whether it's less than or more than the threshold
    c. Calculate the means for the object and the background which is sum1/cdf1 and sum2/cfd2.
    d. Update current threshold for be the mean of the previously calculated means
    e. Update till convergence
7. Apply Thresholding
    Apply the final optimized threshold to the input image using the applyThresholding() function, which takes the final threshold and sets anything less than it to zero anything more than it to white
8. Return the thresholded image grey_img.

Local Vs Global Thresholding

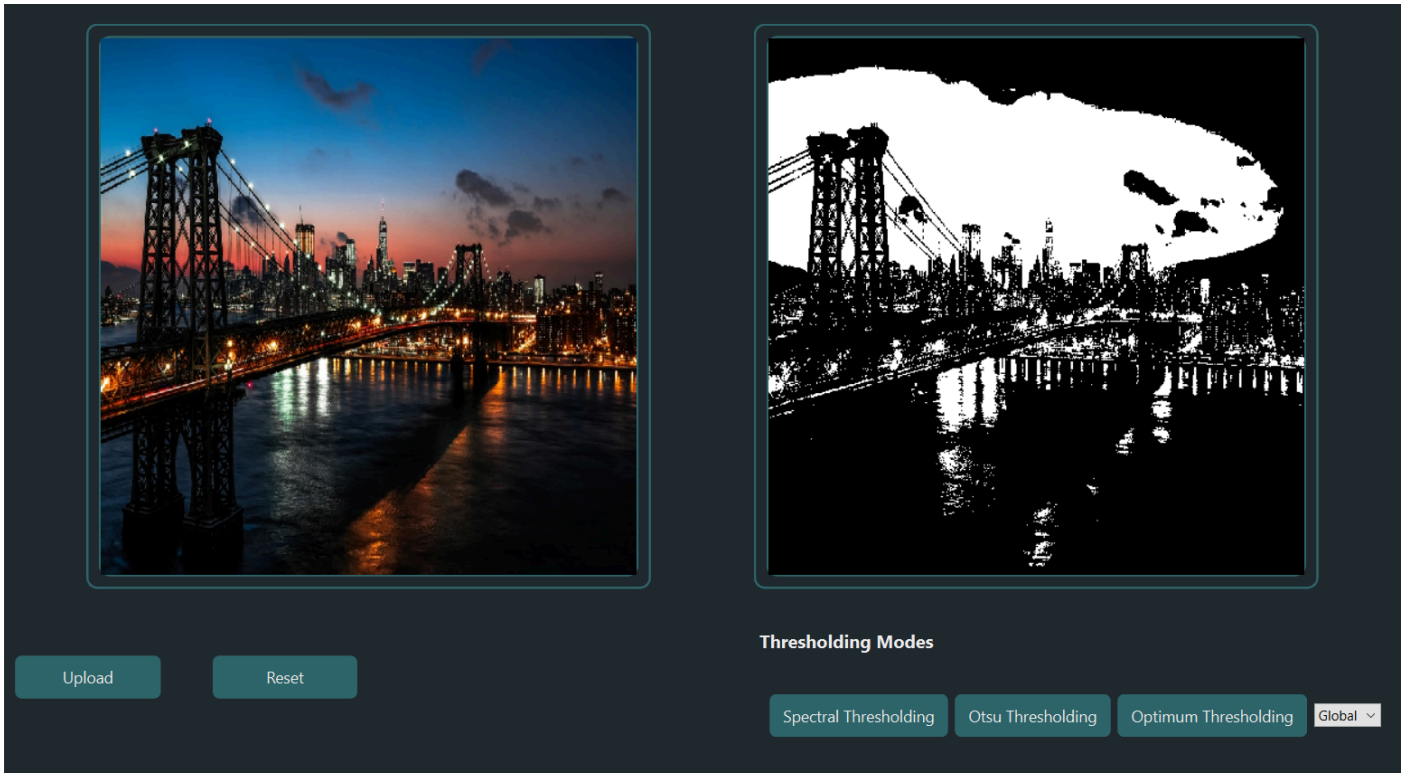
The local thresholding is done the exact same way the global is done except it's just divided into regions and the algorithm is performed on each region individually then the output is combined

Optimizations

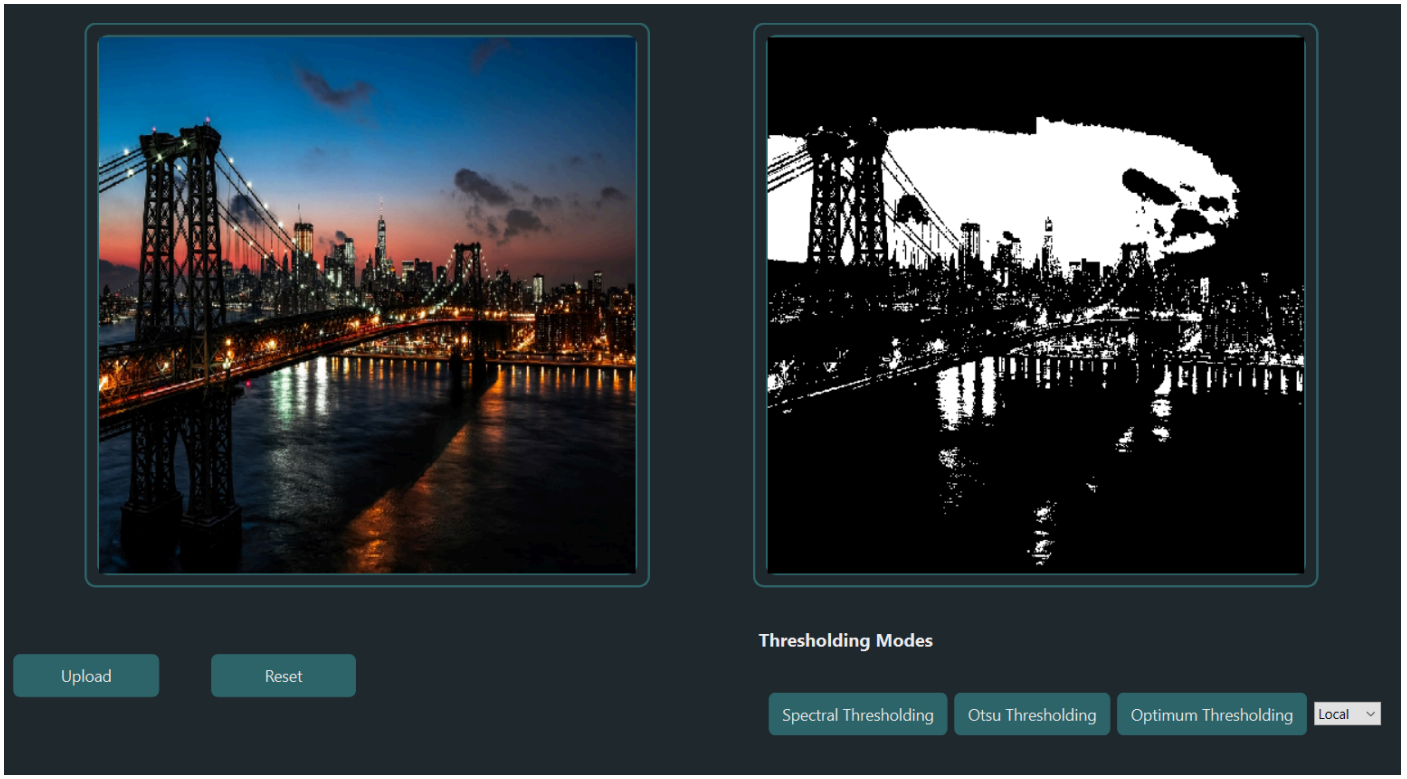
One optimization we did, is to change the original implementation which used loops to calculate the sum and mean to a prefix sum algorithm, using the histogram and its cfd and pdf to get the same result of that of the loops.

Results

The following is the result of the global optimum thresholding



The following is the result of local optimum thresholding



Otsu Thresholding

Otsu's method is a well-known technique used in computer vision and image processing for automatic image thresholding. The main purpose of Otsu's method is to determine an optimal threshold value that effectively separates pixels in an image into two distinct classes: foreground and background.

The algorithm works by analyzing the histogram of the image, which represents the distribution of pixel intensities. Otsu's method aims to find the threshold value that minimizes the variance within each class (intra-class variance) or, equivalently, maximizes the variance between classes (inter-class variance). By achieving this, it identifies the threshold that best discriminates the foreground and background regions, making it useful for image segmentation tasks, in our implementation we will use the maximization of inter-class variance.

Global

Steps of Algorithm:

1- Convert the image to Grayscale and calculate the histogram

2- Calculate Mean for Class 0 and Class 1, which are the background and object

3- Calculate Max inter-class variance and its corresponding threshold "k"

4- Apply Thresholding using the "k" which maximizes the inter-class variance

Local

Steps of Algorithm:

1- Convert the image to Grayscale

2- Divide the grayscale image into four regions of equal size using "Rect"

3- Apply Otsu's thresholding algorithm separately to each region using "otsuThresholding" function

4- Copy thresholded regions back to build up the resulting image using "copyTo" function

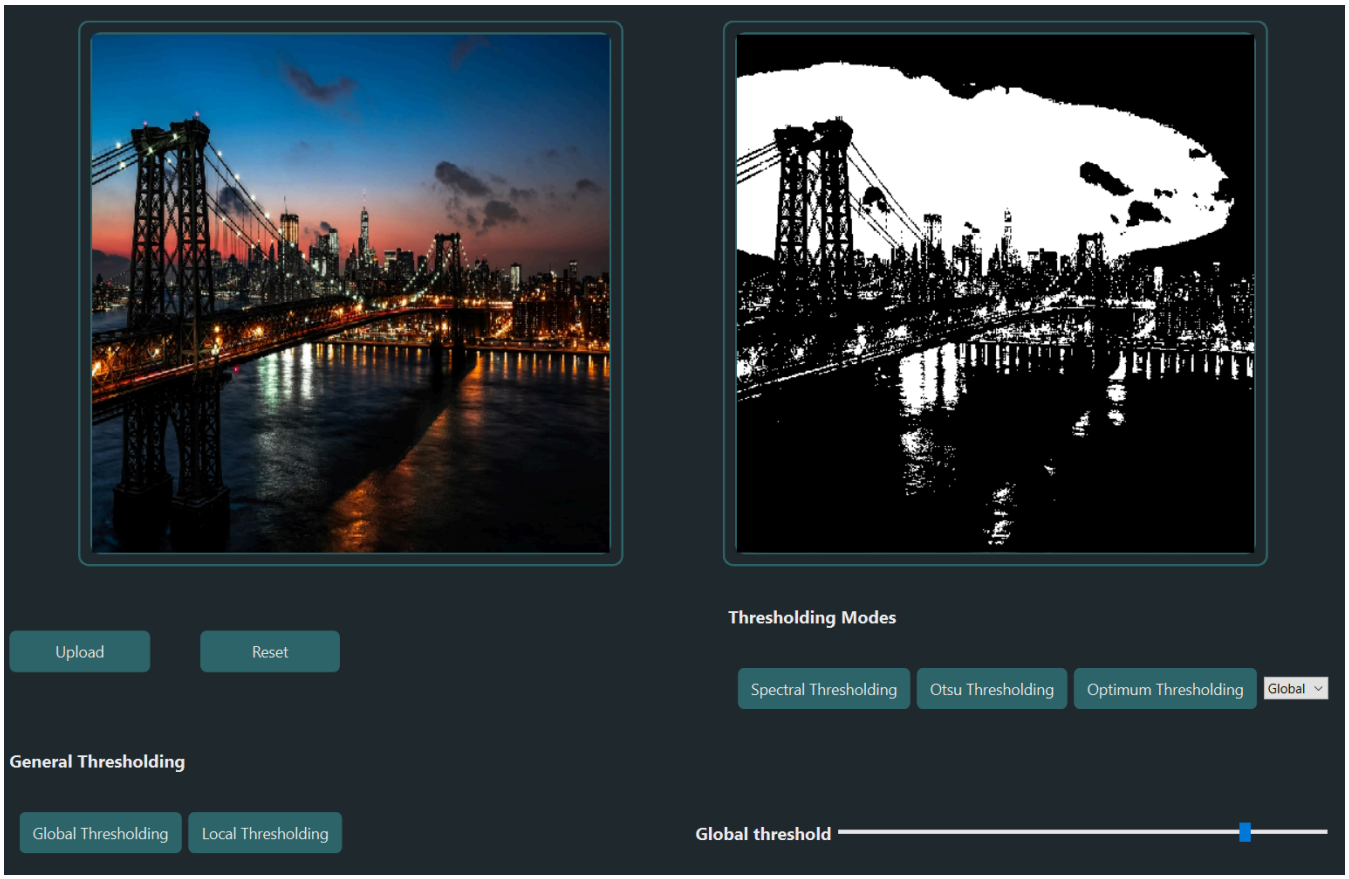
$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \qquad \mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \qquad \mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}$$

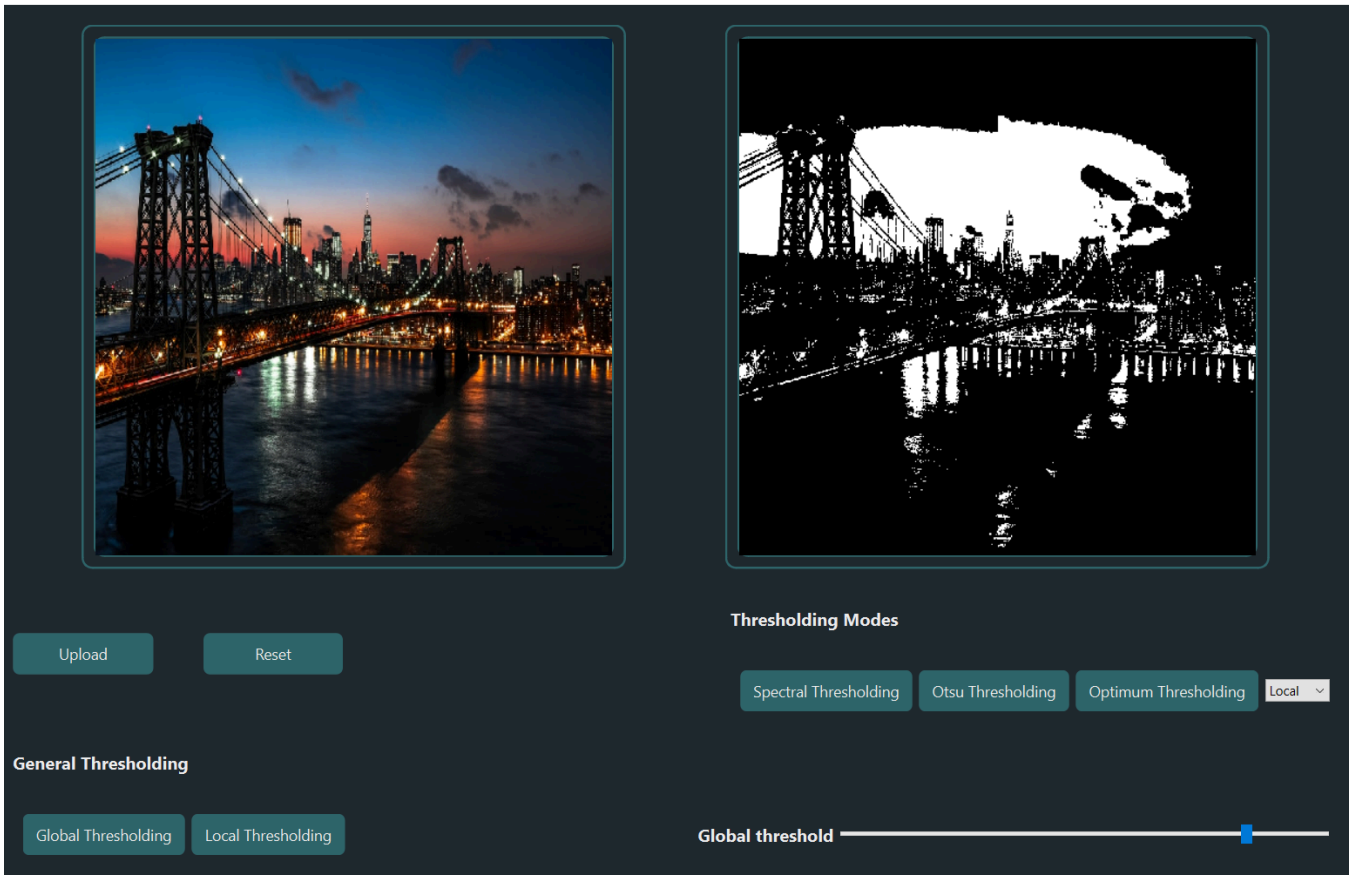
$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_0(t)(\mu_0 - \mu_T)^2 + \omega_1(t)(\mu_1 - \mu_T)^2$$
$$= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

Results

Otsu Global thresholding



Otsu Local Thresholding



Spectral Thresholding

Multilevel thresholding, specifically the Spectral Thresholding method, is a technique used in image processing to segment an image into multiple regions based on intensity levels. The primary goal is to find optimal thresholds that separate different objects or features within the image, enhancing their visibility or enabling further analysis.

The goal is to maximize Variance

***Variance**: σ^2 = w*(μo − μ)^2 + w1*(μ1 − μ)^2 + w2*(μ2 − μ)^2*

Global

Steps of algorithm:

1. **Compute Histogram:** Calculate the histogram of the input image, representing the distribution of pixel intensities.
2. **Calculate Mean Intensity:** Find the mean intensity value of the image pixels.
3. **Optimal Threshold Selection:**
    a. Iterate through possible threshold combinations:
    b. For each combination of low and high thresholds:
        i. Calculate the weight and mean intensity of pixels in three regions: below low threshold, between low and high thresholds, and above high threshold.
        ii. Compute the intra-class variance, which indicates the separation between intensity classes based on the thresholds.
        iii. Update the optimal thresholds if the variance is maximized.
4. **Binary Image Generation:** Create a binary image by assigning pixel values based on the optimal thresholds:
    a. Pixels with intensity below the low threshold are set to 0.
    b. Pixels with intensity between the low and high thresholds are set to a middle value (e.g., 128).
    c. Pixels with intensity above the high threshold are set to the maximum value (e.g., 255).
5. **Output:** Return the binary image representing the segmented regions based on the Spectral Thresholding method.

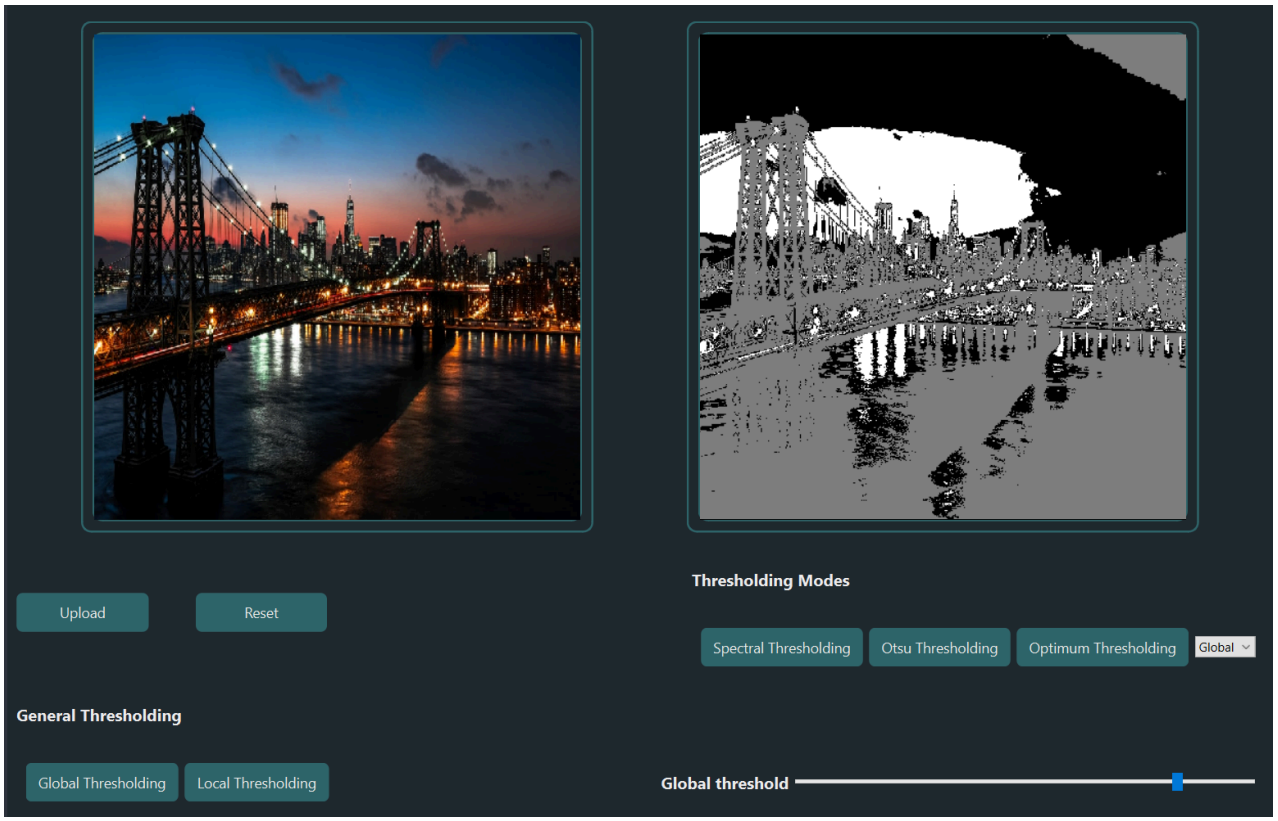
Local

Steps of Algorithm:

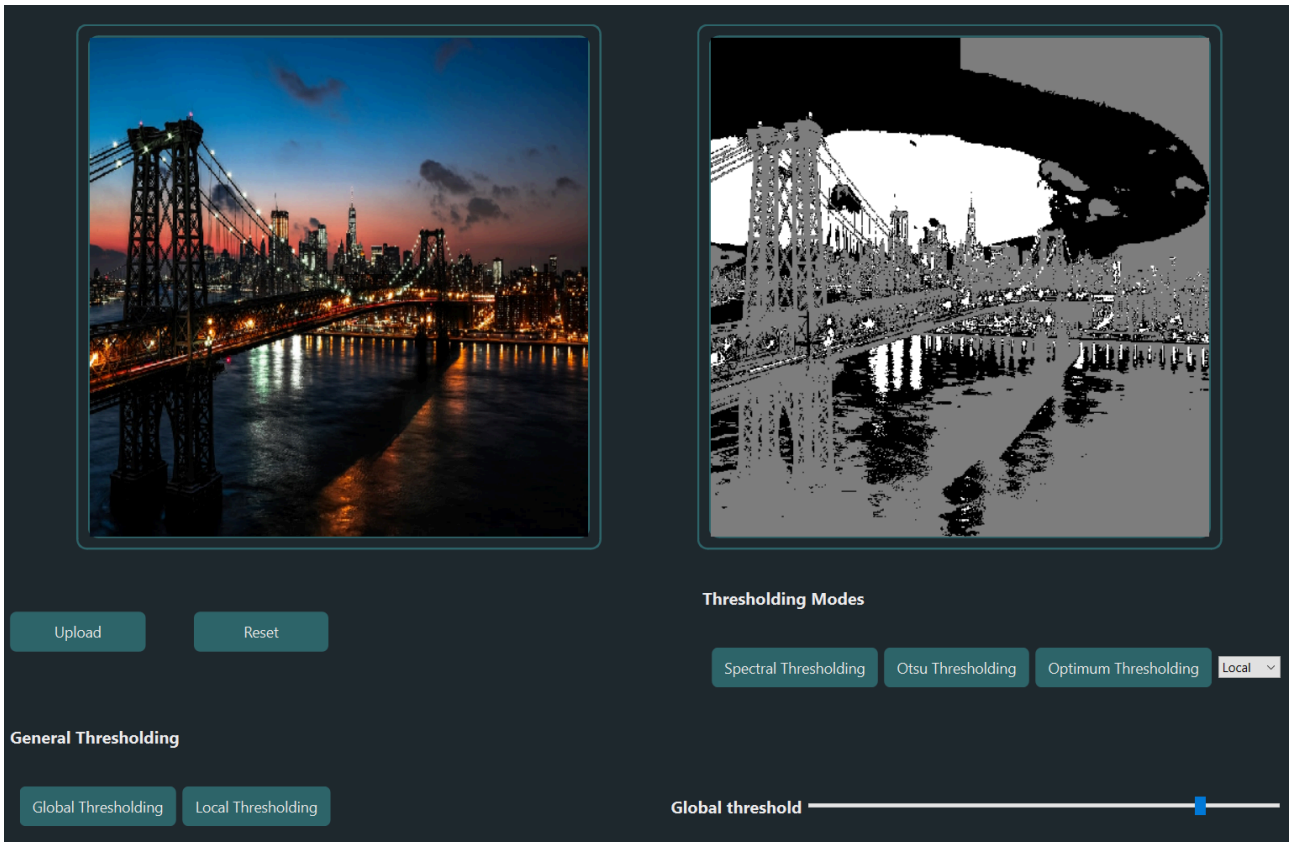
1- Convert the image to Grayscale
2- Divide the grayscale image into four regions of equal size using “Rect”
3- Apply Spectral’s thresholding algorithm separately to each region using “spectralThresholding” function
4- Copy thresholded regions back to build up the resulting image using “copyTo” function

Results

Spectral Global Thresholding



Spectral Local Thresholding



Segmentation

1. Kmeans Segmentation

K-means clustering is a popular unsupervised learning algorithm used to partition a dataset into K clusters. It aims to group similar data points together and discover underlying patterns or groupings in the data. Each cluster is represented by its centroid (the mean of data points assigned to the cluster), and the algorithm iteratively assigns data points to clusters based on their proximity to the cluster centroids.

Importance in Computer Vision:

K-means clustering is widely used in computer vision for tasks such as image segmentation, which involves partitioning an image into meaningful regions or segments. It helps in identifying objects or regions of interest by clustering similar pixels together. K-means can also be applied in feature space for tasks like image compression, where it can reduce the number of colors or features used to represent an image.

Application of K-means in Computer Vision:

One key application of K-means clustering in computer vision is image segmentation, where the algorithm can group similar pixels together to identify distinct objects or regions within an image.

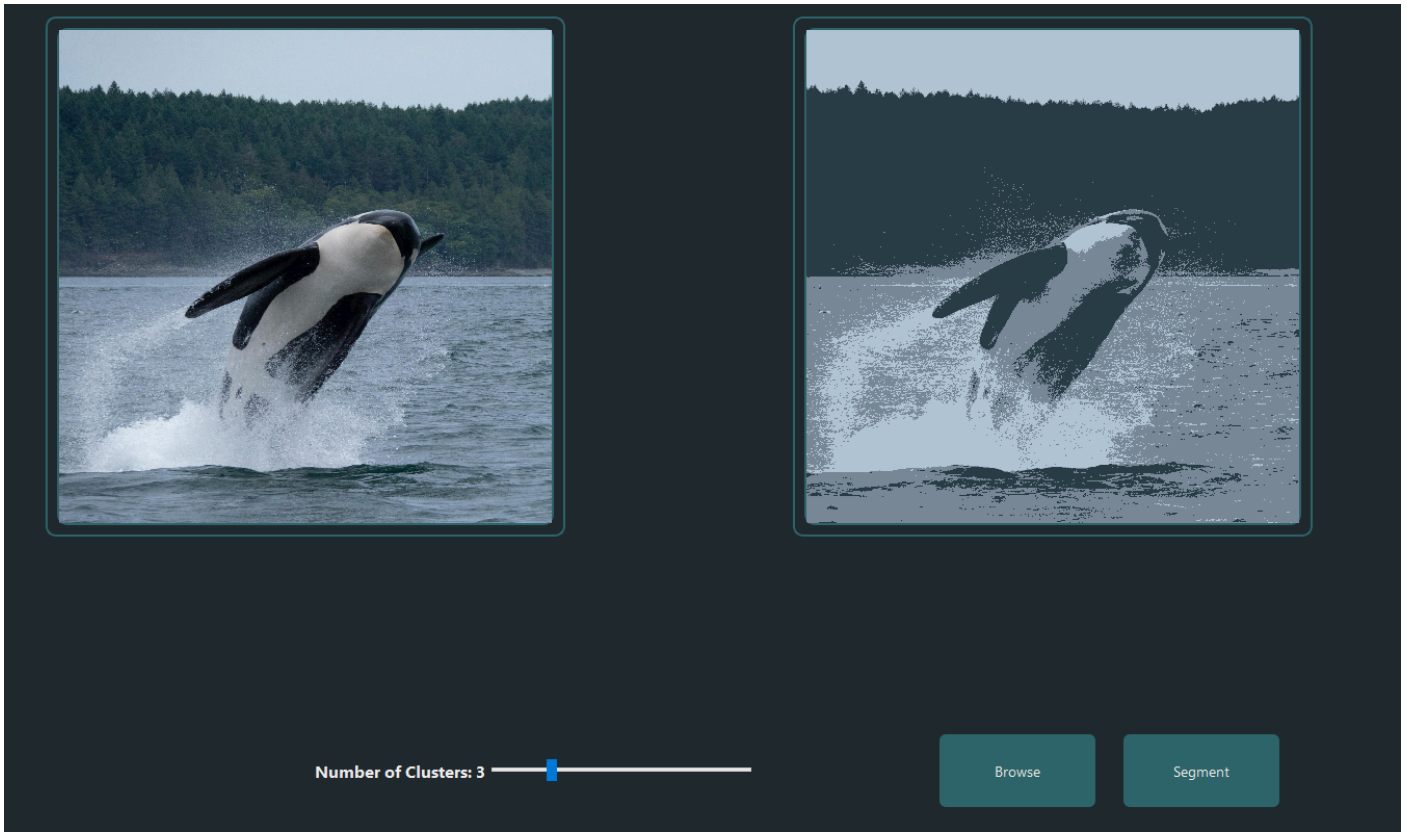
Steps to Perform K-means Clustering:

Note: In our case, we are working with a 3D space problem because we cluster based on the intensity of Red, Green and Blue in Images.

1. **Initialization**:
   - Choose the number of clusters *K* that you want to identify.
   - Initialize *K* cluster centroids randomly by assigning to each cluster a random Red, Green and Blue intensity.
2. **Assign Data Points to Clusters**:
   - For each pixel in the image, calculate the distance between Red, Green, and Blue intensities between the pixel and the centroid average colour intensities using Euclidean Distance
   - Assign each pixel to the cluster whose centroid is closest (i.e., has the minimum distance)
   - In this part, we made a mask that is the same size as the original image and each pixel will contain the cluster number it belongs to. this will be used to compare and colour the image using this data.

3. **Update Cluster Centroids**:
   - After assigning all data points to clusters, recalculate the centroids of the *K* clusters.
   - The new centroid of a cluster is the mean of all data points assigned to that cluster.
4. **Repeat Iteratively**:
   - Repeat the assignment and update steps until convergence criteria are met (e.g., centroids no longer change significantly (using a specific threshold) between iterations or a maximum number of iterations is reached).

Results:



2. Agglomerative Segmentation (Hierarchical Clustering)

Agglomerative Hierarchical Clustering is a bottom-up approach to clustering where each data point (or pixel, in the case of image segmentation) starts as its cluster and pairs of clusters are merged based on a specified distance metric until all data points belong to a single cluster. This technique can be adapted for image segmentation by treating each pixel as a data point and merging clusters of similar pixels based on their RGB colour values.

Importance of Agglomerative Clustering in Computer Vision:

Agglomerative Hierarchical Clustering is crucial in computer vision for image segmentation as it naturally groups pixels based on colour similarity without needing a predefined number of clusters. Its adaptability allows segmentation at varying granularities, accommodating different needs.

The hierarchical structure and interpretability through dendrograms aid in understanding relationships within segmented regions, making it effective for tasks requiring interpretable results. Its robustness to noise, outliers, and scalability further highlights its importance in segmenting images for various computer vision applications.

Algorithm Steps for Agglomerative Hierarchical Clustering:

1. Convert Image to Feature Vectors:
- Load the RGB image and represent each pixel as a feature vector in a high-dimensional space, typically using the RGB colour values (e.g., [*R*, *G*, *B*]) (We use Mat data type in OpenCV).

2. Initialize Clusters:

- Treat each pixel as an initial cluster (singleton cluster), resulting in **n** clusters where **n** is the number of pixels in the image.

3. Compute Pairwise Distances:

- Compute the pairwise distances (e.g., Euclidean distance) between all clusters (pixels) based on their feature vectors (RGB values). This results in a distance matrix representing the dissimilarity between clusters.

4. Merge Closest Clusters:

- Iteratively merge the two closest clusters based on the computed distances until only a single cluster (containing all pixels) remains.
- **Distance Metric**: Use a linkage criterion (e.g., single-linkage, complete-linkage, average-linkage) to determine the distance between clusters.
- **Merge Criteria**: Choose the merging criterion based on the distance metric (e.g., merge clusters that are closest in distance).

5. **Hierarchical Tree Construction**:

- Construct a dendrogram (hierarchical tree) that illustrates the sequence of cluster mergers, where each node represents a cluster and each leaf node represents a data point (pixel).

6. **Cut Dendrogram to Obtain Segmentation**:

- Decide where to cut the dendrogram to obtain the desired number of clusters (segments) for image segmentation.
- **Distance Threshold**: Use a distance threshold to determine when to stop merging clusters, resulting in a predefined number of segments.
- **Cluster Number**: Alternatively, specify the desired number of clusters and cut the dendrogram accordingly.

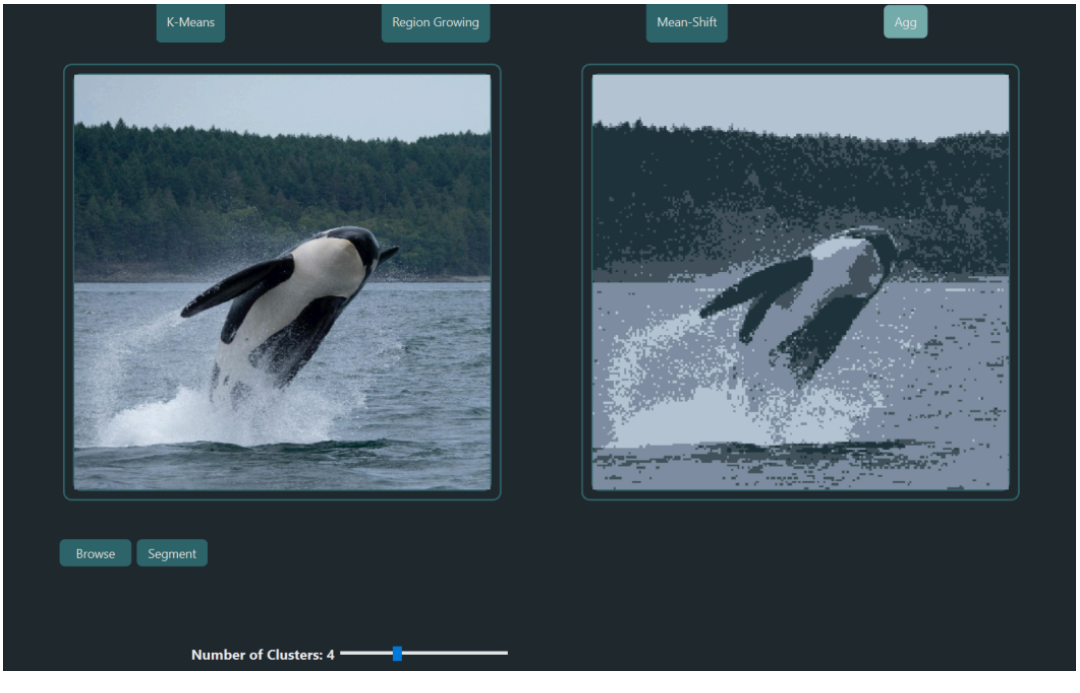
7. **Generate Segmented Image**:

- Assign each pixel to a cluster (segment) based on the results of the hierarchical clustering:
- Replace each pixel with the representative colour (e.g., centroid) of its assigned cluster to form the segmented image.

Key Points:

- Agglomerative Hierarchical Clustering is a versatile clustering technique that can be adapted for image segmentation by treating pixels as data points and merging clusters based on their similarity in feature space.
- The choice of distance metric and linkage criterion (e.g., single-linkage, complete-linkage) impacts the clustering results and segmentation quality.
- The hierarchical tree (dendrogram) provides a visual representation of the clustering process and can be used to interpret the hierarchical relationships between clusters.
- Cutting the dendrogram at an appropriate level allows for controlling the granularity of segmentation and the number of resulting segments in the image.

Results:



3. Region-Growing Segmentation

Region Growing is a technique used in image processing and computer vision for segmenting an image into regions or objects based on pixel similarity and connectivity. It starts with seed points (or seeds) and grows these points into larger regions by iteratively merging neighbouring pixels that satisfy certain similarity criteria. Region Growing is particularly useful for segmenting images where distinct regions or objects have homogeneous properties within them.

Importance in Computer Vision:

Region Growing segmentation is important in computer vision for tasks such as object detection, image segmentation, and feature extraction. It allows for the automatic delineation of objects or regions of interest within an image based on local pixel properties, facilitating subsequent analysis and understanding of visual data.

Application of Region Growing Segmentation:

One key application of Region Growing in computer vision is medical image analysis, where it is used for segmenting anatomical structures (e.g., tumors, organs) based on intensity or texture similarity. Another application is in remote sensing, where it can be applied to segment satellite images into land cover types (e.g., vegetation, and water bodies).

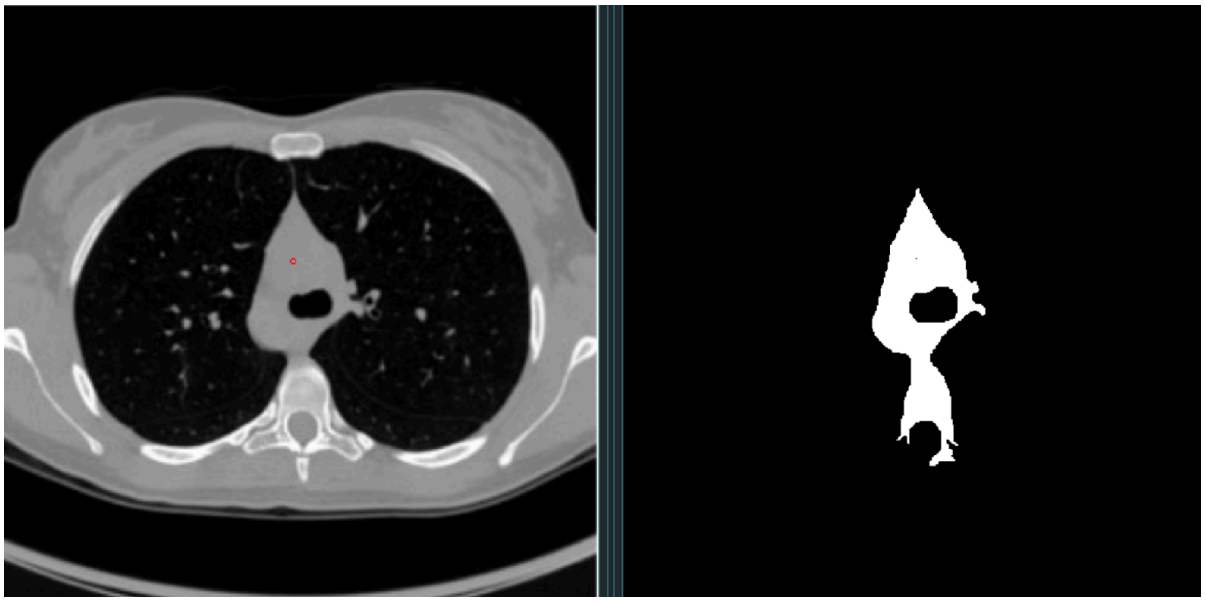
Steps to Perform Region-Growing Segmentation:

1. **Seed Selection**:
   - Choose one or more seed points within the image to initiate the region-growing process. Seeds can be manually selected based on prior knowledge of the image content or selected automatically (e.g., based on intensity peaks).
2. **Initialization**:
   - Initialize an empty region (or segmentation mask) and add the seed point(s) to this region.
3. **Region Growing Iteration:**
   - While there are unprocessed pixels adjacent to the current region:
     - **Pixel Selection**:
       - Select an unprocessed pixel neighbouring the current region.
     - **Similarity Check:**
       - Compare the selected pixel's properties (color intensity) with the properties of the pixels in the current region.
     - **Add to Region:**
       - If the selected pixel meets the similarity criteria (within an intensity threshold), add it to the current region.
       - Mark the pixel as processed to avoid redundant checks.

4. Stopping Criteria:
   - Define stopping criteria to determine when to stop growing the region, such as reaching a predefined region size or processing all eligible pixels.

5. Final Segmentation:
   - Once the region growing process is complete, all pixels added to the region(s) form the segmented regions of the image.

Results:



4. Mean Shift Segmentation

Mean Shift clustering is another popular technique used for image segmentation in computer vision. It is a non-parametric clustering algorithm that does not require specifying the number of clusters *K* beforehand/ Mean Shift works by iteratively shifting data points towards the mode (peak) of the density function in the feature space.

Mean Shift clustering aims to discover clusters by iteratively shifting data points towards the modes (local maxima) of the underlying probability density function of the data distribution. It is particularly effective for tasks like image segmentation where it can identify dense regions of similar pixels.

Importance in Computer Vision:

Mean Shift clustering is well-suited for computer vision tasks such as image segmentation, where it can effectively group pixels into coherent segments based on their similarity in feature space.

Application of Mean Shift in Computer Vision (Image Segmentation):

Mean Shift clustering can be applied to image segmentation, where it segments an image into regions of similar colour or texture. Each pixel in the image is treated as a point in a high-dimensional space (e.g., colour space), and Mean Shift is used to group similar pixels together based on their feature similarity.

Mean Shift Clustering Algorithm for Image Segmentation:

1. **Color-Space Conversion:**

Mean Shift begins by converting the input image from RGB colors to Lab colors. Lab colors separate brightness (L) from color (a, b), which helps in perceiving color differences more accurately.

2. Pixel Processing:

For each pixel in the image, Mean Shift looks at a nearby region defined by a "bandwidth." This region includes pixels that are close in both color and position to the current pixel.

3. Finding Dominant Colors (Modes):

Mean Shift tries to find the most common (dominant) color within the defined region around each pixel. It does this by repeatedly shifting each pixel towards where most similar colors are found, within the specified color range.

4. Iterative Process:

Mean Shift performs an iterative process on each pixel to find its mode (most dominant color in its neighborhood). The goal is to shift the pixel's location towards the mode until convergence.

During each iteration:

- A spatial window is defined around the current pixel.
- Neighbor pixels within this window are examined based on their Lab color distances from the current pixel.
- Pixels with similar colors (within a specified colorBandwidth) are accumulated to compute the new location (mode) of the pixel.

5. Region Segmentation:

- After mean shifting each pixel, the image is segmented into regions based on the converged modes. A region-growing approach is used to assign labels to connected pixels that share similar color characteristics within specified bandwidths.
- Each labeled region is associated with a mode (average color) computed from the pixels within that region.

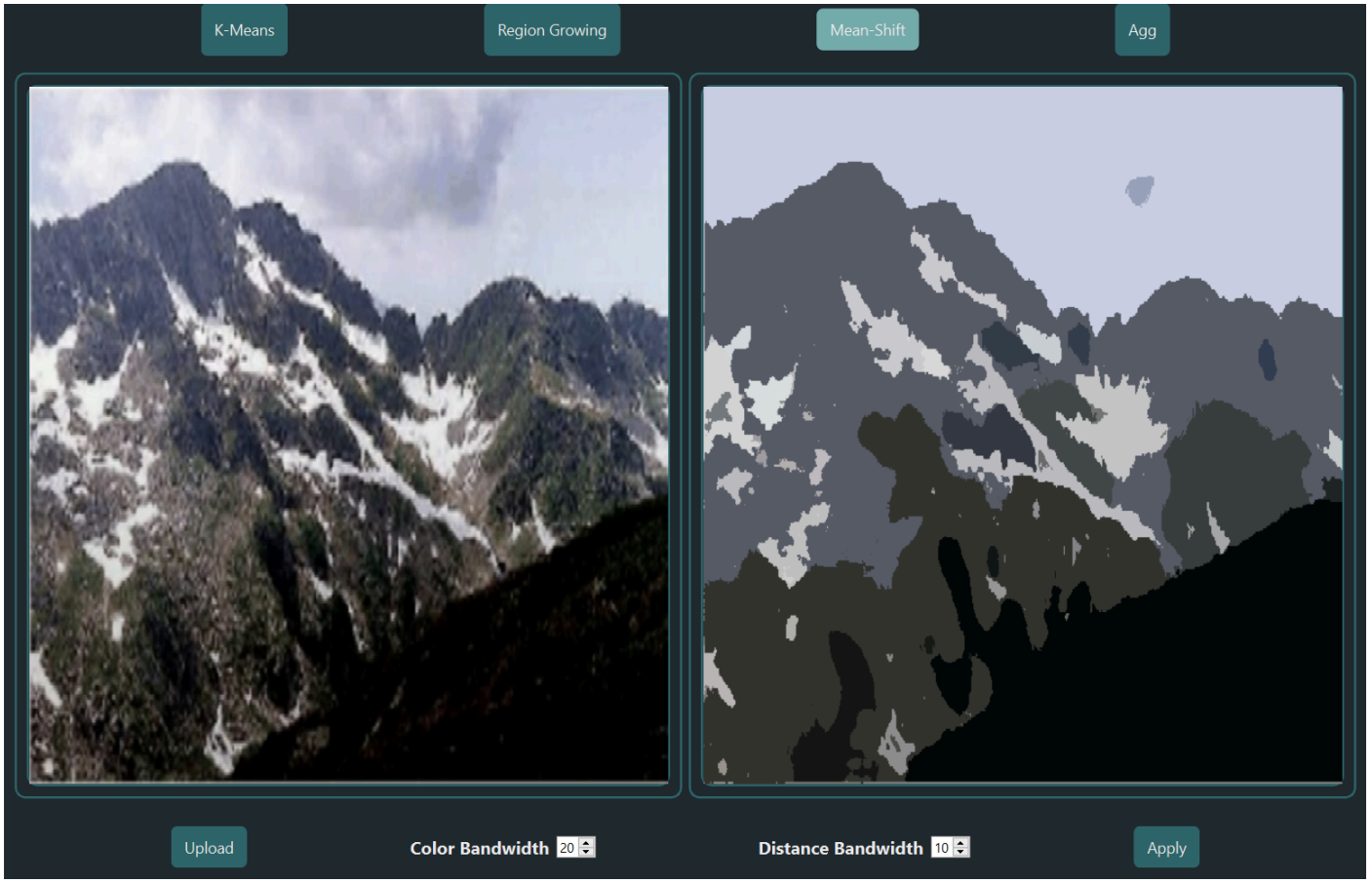
6. Final Image Result:

Finally, a new image is created using the dominant colours of these regions. Each pixel's colour is replaced with the dominant colour of the region it belongs to.

Key Points:

- Mean Shift clustering iteratively shifts data points towards regions of high density in the feature space, effectively grouping similar data points together.
- The bandwidth parameter h influences the size of the region over which the mean shift is computed and impacts the granularity of segmentation.
- Mean Shift is a non-parametric clustering algorithm and does not require specifying the number of clusters beforehand.
- The resulting segmented image highlights regions of the original image that exhibit similar characteristics (e.g., colour) based on the clustering of pixels in RGB space.

Results:


K-Means
Region Growing
Mean-Shift
Agg
Upload
Color Bandwidth 20
Distance Bandwidth 10
Apply