Thresholding and Segmentation

Team 07

Faculty of Engineering - Cairo University

Dr. Ahmed Badawy

Eng. Laila Abbas

Collaborators

Name	Section	B.N
Ahmed Emad	1	3
Amir Hesham Khalil	1	10
Omar Mohamed Ahmed	1	46
Hazem Zakaria	1	16
Omar Tarek	1	42

Introduction

In the field of image processing, thresholding is a technique that is used to partition an image into foreground and background. It is a fundamental technique that is used in many applications, including image segmentation, feature extraction, and object recognition.
There are two main types of thresholding techniques: global thresholding and local thresholding. Global thresholding applies a single threshold value to the entire image. Local thresholding, on the other hand, applies different threshold values to different parts of the image.
Optimal thresholding, Otsu's thresholding, and spectral thresholding are all examples of global thresholding techniques. Optimal thresholding selects the threshold value that minimizes the variance within the classes. Otsu's thresholding is a popular method for optimal thresholding. Spectral thresholding uses information from the image's spectrum to select the threshold value.

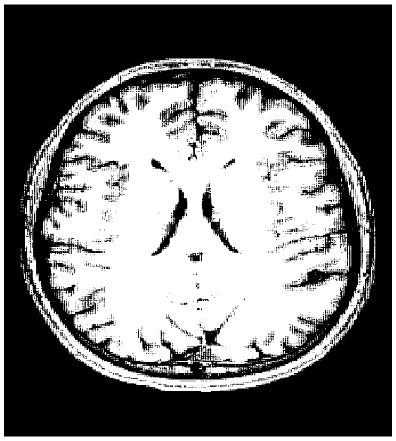
1. Thresholding

Thresholding is a fundamental technique used to separate objects or regions of interest from the background. It involves converting a grayscale image into a binary image, where pixels are classified as either foreground or background based on their intensity values. There are various algorithms for applying thresholding, but the below ones included in our program are the most popular.

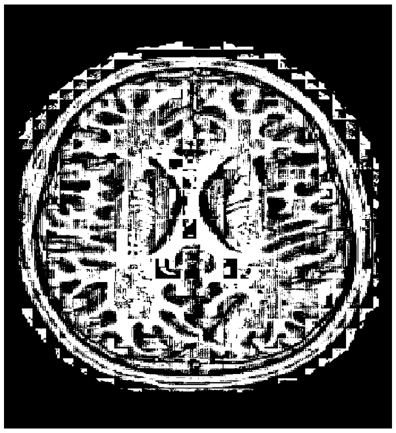
- **Optimal Thresholding:** Optimal thresholding methods aim to find the threshold that maximizes a certain criterion, such as between-class variance or entropy. These methods are used in image processing tasks like segmentation, where determining an appropriate threshold can effectively separate objects or features of interest from the background.

- **Otsu Thresholding:** Otsu's method calculates the optimal threshold by maximizing the between-class variance of the pixel intensities. It assumes that the image contains two classes of pixels (foreground and background) and finds the threshold that minimizes the intra-class variance. This technique is widely used for automatic image thresholding due to its simplicity and effectiveness, particularly in applications where manual threshold selection is impractical or time-consuming.

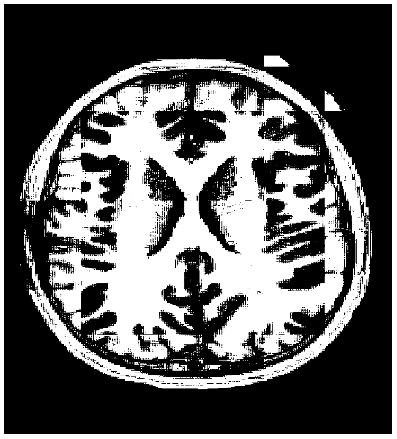
- **Spectral Thresholding:** While binary thresholding separates pixels into foreground and background classes, multi-class thresholding extends this concept to segment images into more than two classes. Spectral thresholding involves partitioning the image into multiple regions based on different intensity levels, which can be particularly useful in scenarios where objects or features in the image exhibit varying intensity characteristics. It enables more nuanced segmentation, allowing for the identification of multiple classes or categories within an image beyond simple foreground-background differentiation. Spectral thresholding techniques are employed in various fields, including medical imaging, remote sensing, and computer vision, where precise segmentation of complex scenes is essential for subsequent analysis or interpretation.



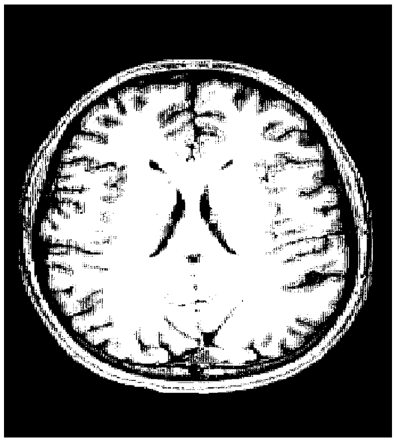
Optimal Thresholding (Global)



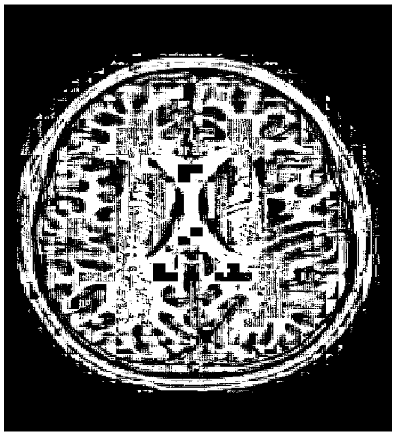
Optimal Thresholding (Local)
Box size: 20%



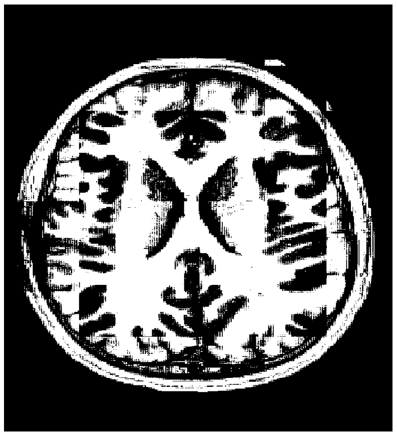
Optimal Thresholding (Local)
Box size: 90%



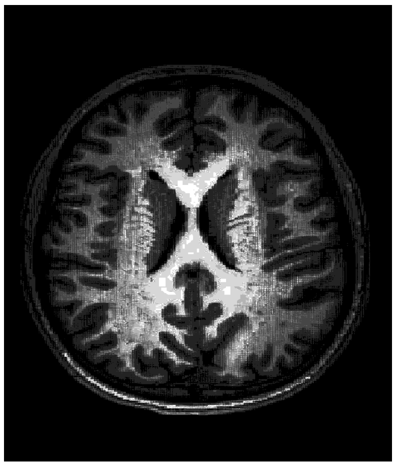
Otsu Thresholding (Global)



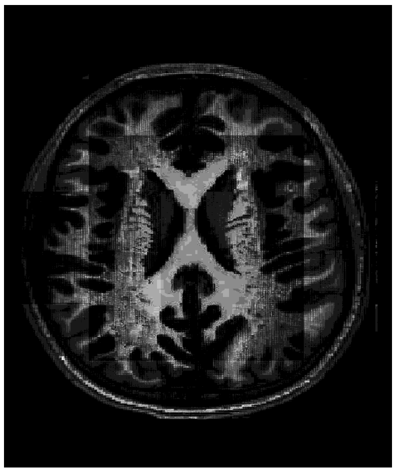
Otsu Thresholding (Local)
Box size: 20%



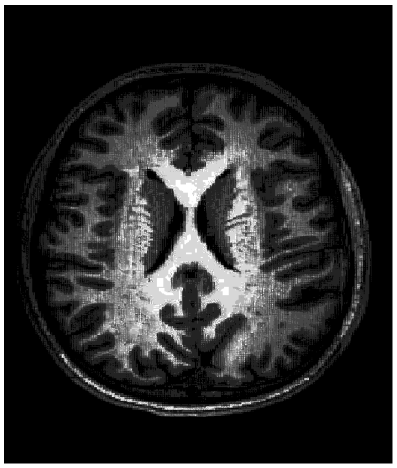
Otsu Thresholding (Local)
Box size: 90%



Spectral Thresholding (Global)
Num. Classes: 80



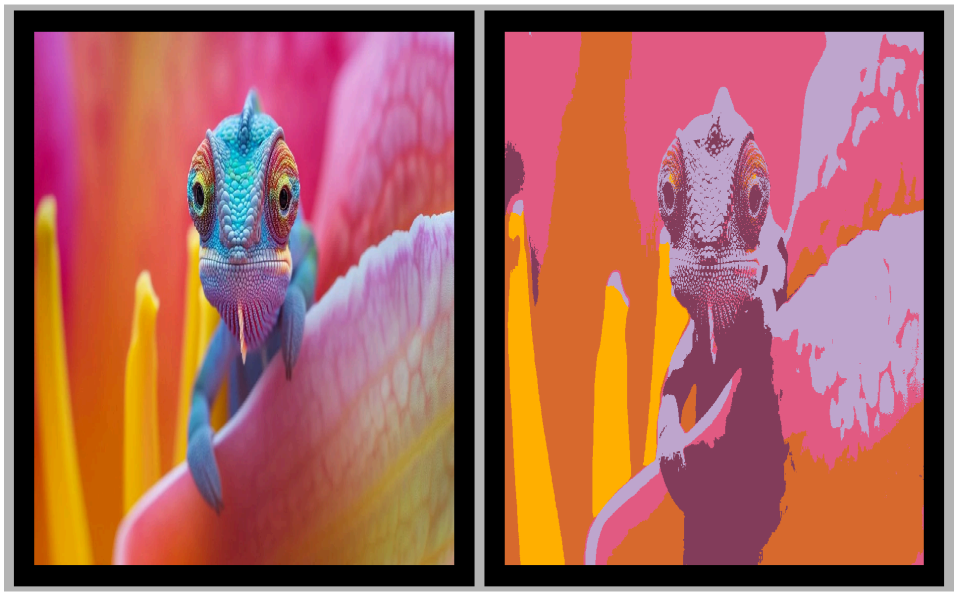
Spectral Thresholding (Local)
Num. Classes: 80
Box size: 20%



Spectral Thresholding (Local)
Num. Classes: 80
Box size: 90%

2. <u>Segmentation</u>

- **K-Means:** It is a clustering fundamental technique in computer vision for image segmentation, partitioning a given image into distinct regions based on pixel similarities. The algorithm iteratively assigns each pixel to the cluster with the nearest centroid, determined by minimizing the Euclidean distance between pixel values and cluster centroids. Initially, it randomly selects cluster centroids from the image's data points. Through successive iterations, centroids are recalculated as the mean of all pixels assigned to each cluster. This process continues until convergence, typically defined by minimal changes in centroid positions or a fixed number of iterations. By grouping pixels with similar characteristics, K-means effectively divides the image into coherent segments, enabling tasks such as object detection and image analysis. However, its performance may vary based on factors like the initial centroid selection and sensitivity to outliers.



Applying K-Means segmentation with No. of clusters = 5 and No. of iterations = 100

- **Mean Shift:** The mean shift algorithm, widely used in image segmentation tasks, iteratively shifts each pixel's position towards the mode of its local neighborhood in feature space. Our algorithm implements mean shift by iteratively updating the mean vector of pixels within a spatial window specified by the bandwidth until convergence. Initially, it assigns each pixel to a label denoting its cluster affiliation. The algorithm then calculates the mean vector for each cluster's pixels, iterating until convergence is achieved. Convergence occurs when either the mean vector stabilizes or the number of pixels assigned to the cluster remains unchanged. This process effectively segments the image into regions of similar color or texture. Subsequently, the algorithm assigns the mean color of each cluster to all pixels within that cluster, resulting in a segmented image with coherent regions. Despite its effectiveness, mean shift's performance heavily relies on the choice of **bandwidth** parameter, influencing the size of the spatial window and thereby affecting the granularity of segmentation. Additionally, it tends to be computationally expensive due to its exhaustive search within the feature space.

  Decreasing the bandwidth parameter to a very small value (~10), in the Mean Shift algorithm restricts the range of neighboring pixels considered for clustering to an extremely narrow window, wherein only pixels that are nearly identical in both color and spatial proximity are grouped together. In terms of code, this narrow window results in a highly constrained spatial and color window for computing the mean shift vector, with tight loop bounds and a condition that is satisfied only for neighboring pixels almost identical in color to the central pixel. Consequently, the segmented image tends to merge nearby pixels into the same cluster only if they are nearly identical in color, resulting in a segmented image with very limited variation and potentially appearing as a single color or a few distinct colors. Overall, decreasing the bandwidth produces a segmented image lacking detail and variation, as neighboring pixels with almost identical colors are grouped together, potentially resulting in oversimplification and limited discrimination among different regions or objects in the image.

Increasing the bandwidth parameter in the Mean Shift algorithm (~60) expands the range of neighboring pixels considered for clustering, allowing pixels with greater color and spatial differences to be grouped together. In the code, this leads to a larger spatial and color window for computing the mean shift vector, resulting in expanded loop bounds and a wider range of neighboring pixels satisfying the condition for clustering. The segmented image reflects these changes by forming larger and more homogeneous clusters with greater variation in color and intensity within each cluster. While increasing the bandwidth smooths out smaller variations and reduces noise, excessive bandwidth (~200) may lead to under-segmentation, where distinct regions merge into the same cluster. Thus, adjusting the bandwidth parameter allows for balancing detail capture and cluster coherence in the segmented image.

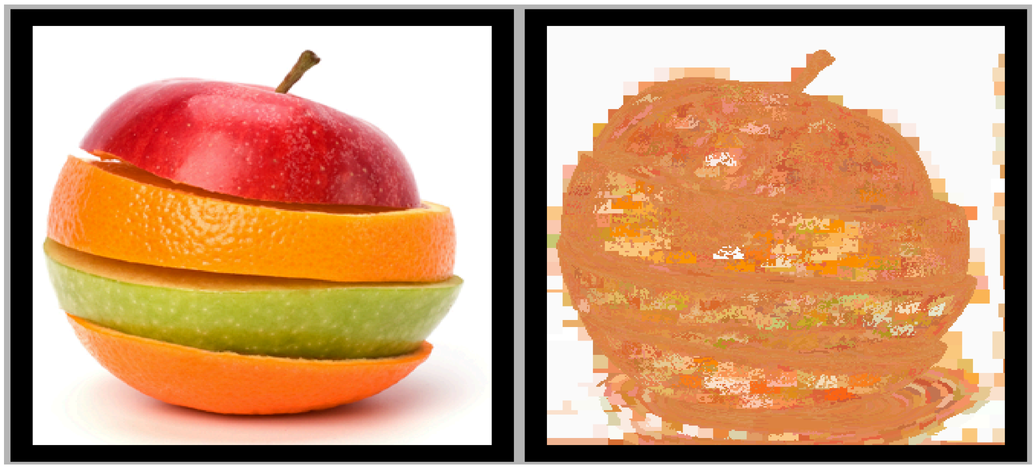


Image after applying Mean Shift algorithm with BW = 10

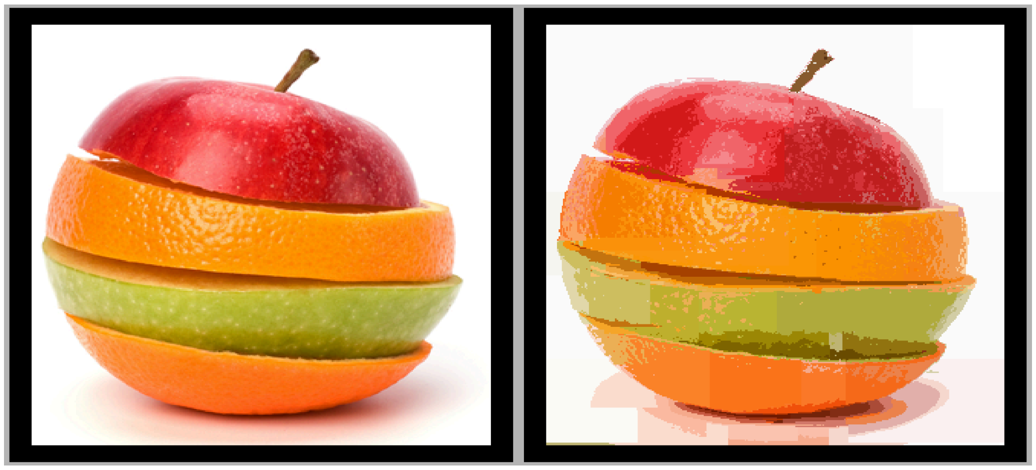


Image after applying Mean Shift algorithm with BW = 60

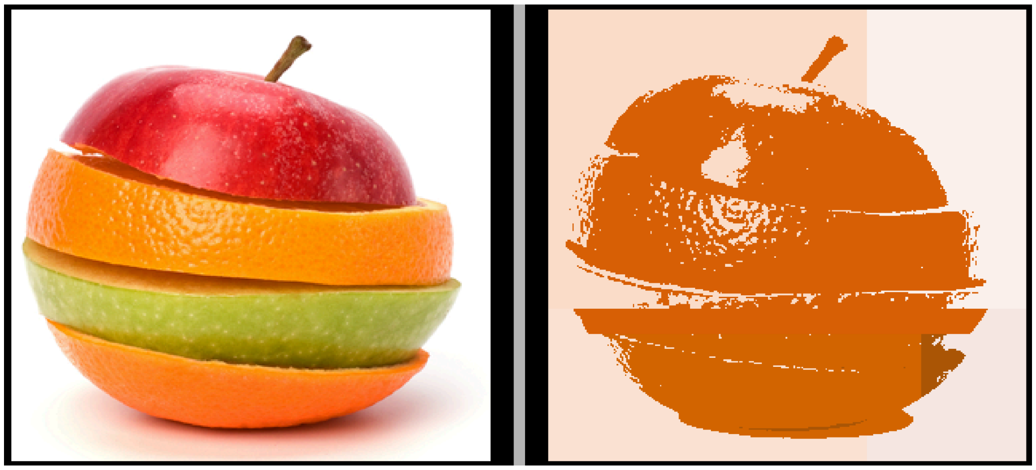


Image after applying Mean Shift algorithm with BW = 200

- **Agglomerative Clustering:** Agglomerative clustering is a powerful unsupervised learning technique used to group data points into clusters based on their similarity. Unlike supervised learning where data points have predefined labels, agglomerative clustering allows you to discover inherent structures within your data. This approach follows a **bottom-up** strategy. Initially, each data point is considered its own individual cluster. The algorithm then iteratively merges the most similar clusters based on a predefined distance metric, such as Euclidean distance. This process continues until all data points are merged into a single cluster, forming a hierarchy of clusters.

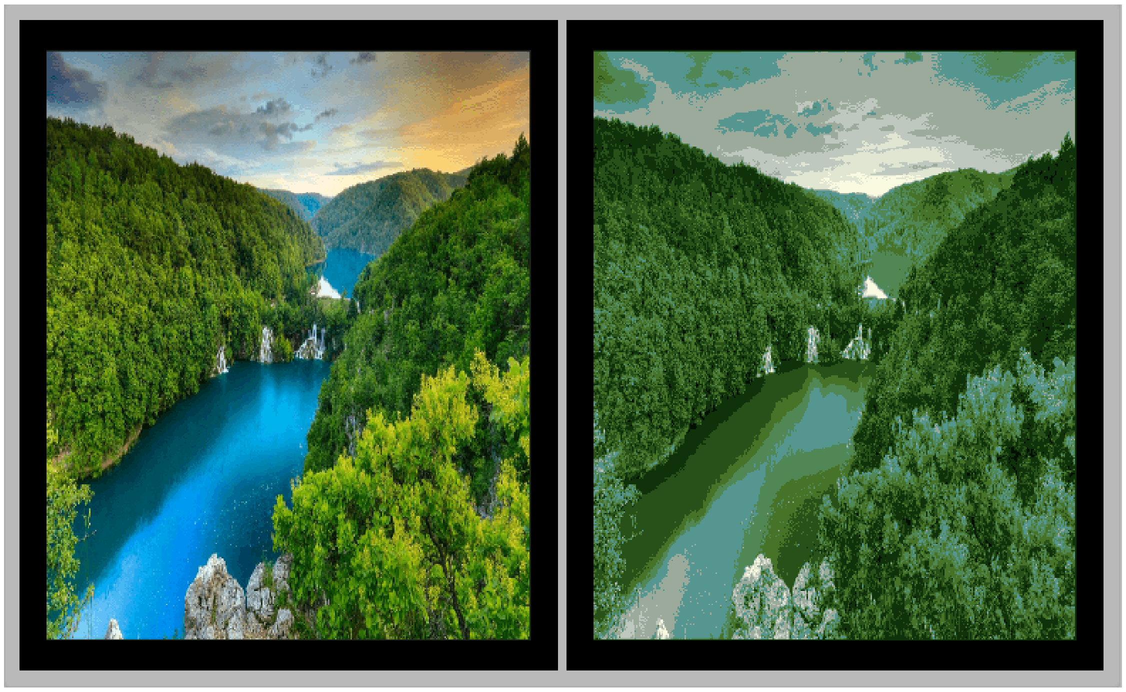


Image after applying Agglomerative Clustering

- **Region Growing:** It tackles the task of partitioning an image into distinct regions based on pixel similarities. Imagine segmenting an image containing objects – region growing helps us identify these objects by grouping pixels that share similar properties.

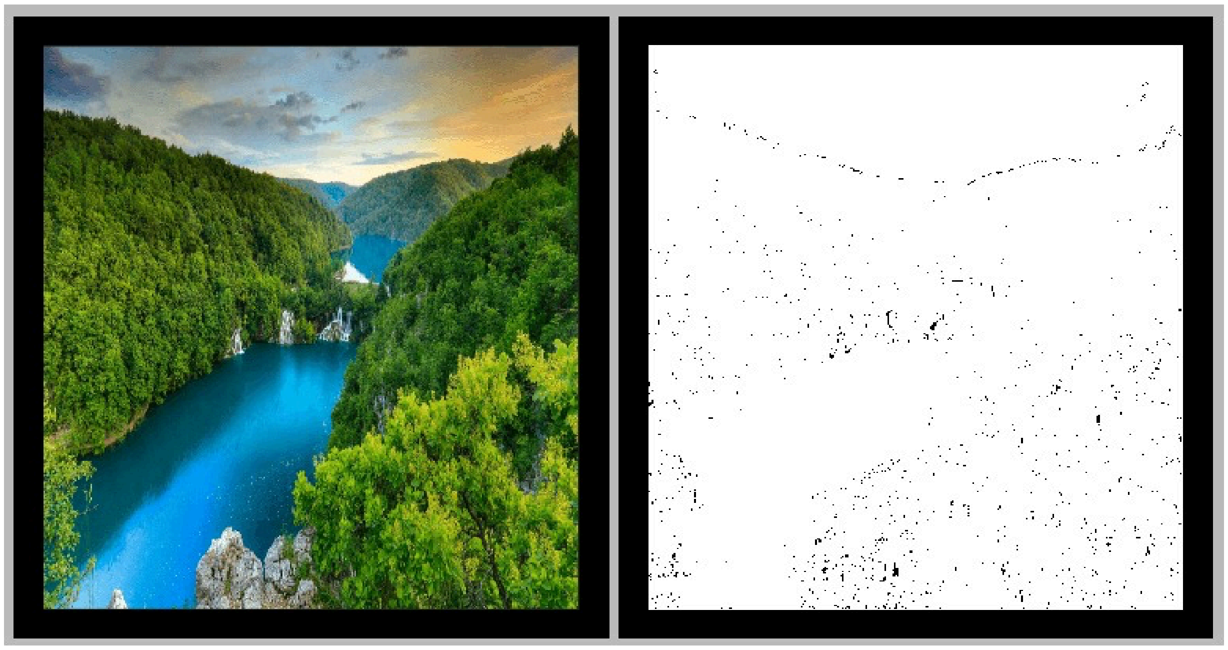


Image after applying Region Growing with threshold = 22

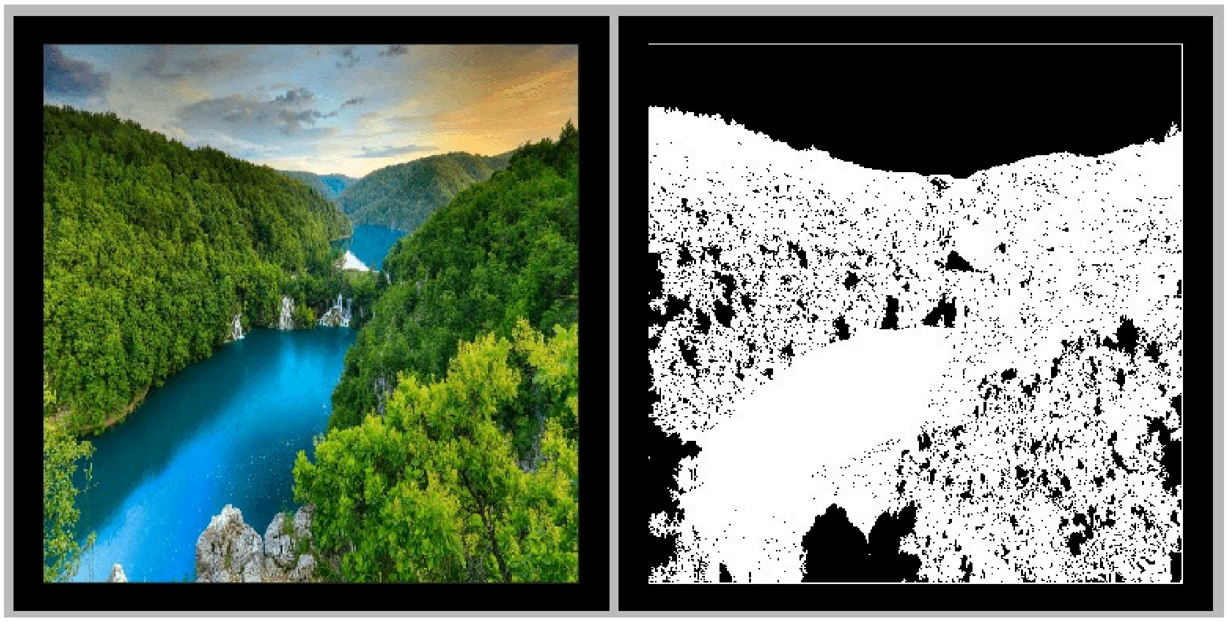


Image after applying Region Growing with threshold = 18

Software Design and Architecture

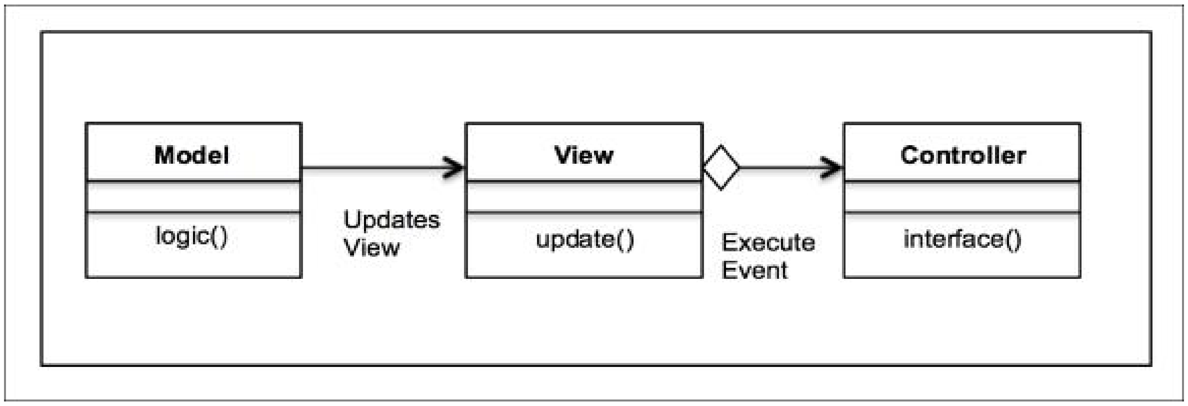
The application is built upon a robust software design architecture that leverages the Python programming language, PyQt framework, Qt Designer for UI design, and follows the Model-View-Controller (MVC) pattern for efficient organization and separation of concerns.

1. Model-View-Controller (MVC) Pattern

● Model: The Model component represents the data and business logic of the application. In our context, it encapsulates image processing algorithms, data structures for image representation, and manipulation functions.
The model in our case is represented in Image processing algorithms, data structures, and functions reside within the Model component, ensuring encapsulation and modularity.

● View: The View component is responsible for presenting the user interface to the user. Qt Designer is utilized to design the graphical user interface (GUI) elements, including buttons, sliders, image displays, and histograms.
The view in our case encompasses the graphical user interface designed using Qt Designer. It includes various widgets for user interaction and image display.

● Controller: The Controller acts as an intermediary between the Model and the View, handling user input and triggering appropriate actions. It binds GUI events to corresponding image processing functions, ensuring seamless interaction between the user and the application.
The controller in our case bridges the gap between the Model and the View, handling user events and orchestrating image processing tasks based on user input.



2. Used Technologies and Tools

● Python: Python serves as the primary programming language for our application, offering simplicity, readability, and extensive libraries for image processing tasks.

● PyQt: PyQt is a set of Python bindings for the Qt application framework. It enables the integration of Qt functionalities within Python applications, providing access to a vast array of GUI components and features for building cross-platform desktop applications.

● Qt Designer: Qt Designer is utilized for designing the graphical user interface of our application. It offers a drag-and-drop interface for creating and arranging UI elements, streamlining the GUI design process. Designed UI files (.ui) created in Qt Designer are seamlessly integrated into the Python codebase using PyQt, allowing for rapid development and modification of the user interface.

3. Goals

● Modularity: The MVC pattern facilitates modular design, enabling independent development and testing of Model, View, and Controller components.

● Separation of Concerns: MVC separates the presentation logic (View) from the application logic (Model), enhancing maintainability and scalability.

● Ease of Development: Python, PyQt, and Qt Designer collectively provide a conducive environment for rapid application development, allowing us, as developers, to focus on functionality rather than low-level details of GUI implementation.