Image Processing Assignment 7

Image Segmentation using Split & Merge and Watershed Transform

Aditya Chavan
Roll No: 231080019
[T.Y B.Tech IT]
Date: September 27, 2025

1 Aim

To write a Python program to perform image segmentation using two distinct methods: the Split and Merge technique and the Watershed Transform.

2 Theory

Image segmentation is the process of partitioning an image into multiple segments, or sets of pixels, often to identify objects or other meaningful information. The goal is to simplify or change the representation of an image into something that is more meaningful and easier to analyze.

2.1 Key Concepts

1. **Image Segmentation:** The process of dividing a digital image into multiple regions, each of which corresponds to an object or a meaningful part of an object. The pixels within each region are homogeneous with respect to some characteristic, such as color, intensity, or texture.

2. **Split and Merge Technique:** A segmentation method that follows a quadtree structure. It begins by splitting the image into four equal quadrants. This process is recursively applied to any quadrant that is not homogeneous. Once all quadrants are homogeneous, the algorithm then merges adjacent homogeneous regions that satisfy a specific similarity criterion.

3. **Watershed Transform:** A segmentation approach based on the concept of a topographical map. The image is treated as a 3D surface where pixel intensity represents height. The transform identifies "catchment basins" (regions) and "watershed lines" (boundaries) that separate these basins. The algorithm floods the surface from markers placed in the basins, and where the water from different basins meets, a watershed line is drawn.

2.2 Mathematical Foundation

- **Split and Merge:** The algorithm can be represented by a predicate function $P(R)$ that tests for region homogeneity.

  - **Split:** If a region R is not homogeneous, i.e., $P(R) =$ False, then it is split into four sub-quadrants (R_1, R_2, R_3, R_4). This continues until $P(R_i) =$ True for all leaf regions.
  - **Merge:** A criterion for merging two adjacent regions R_i and R_j can be based on the similarity of their mean intensities, for example: $|\mu_{R_i} - \mu_{R_j}| < T$, where μ is the mean intensity and T is a threshold.

- **Watershed Transform:** The core of the watershed transform relies on the concept of geodesic distance. It simulates the flooding of a topographical relief, where the final watershed lines correspond to the crest lines of the relief.

2.3 Advantages

- **Split and Merge:** Provides a hierarchical representation of the image. The algorithm is adaptive, splitting only the regions that require it.
- **Watershed Transform:** Produces closed and continuous segmentation boundaries, which are desirable for object recognition. It is robust to variations in intensity and texture within objects.

2.4 Disadvantages

- **Split and Merge:** The final segmentation is highly dependent on the initial homogeneity predicate and the merging criteria. It can be computationally expensive and may result in "blocky" boundaries.
- **Watershed Transform:** Prone to **over-segmentation**, where a single object is segmented into multiple regions due to local minima caused by noise. Pre-processing steps like using markers are crucial to mitigate this.

2.5 Applications

- **Medical Imaging:** Segmenting tumors, organs, and other anatomical structures.
- **Object Recognition:** Separating individual objects in a scene.
- **Autonomous Vehicles:** Identifying road lanes, pedestrians, and other vehicles.
- **Satellite Imagery:** Analyzing land use and identifying different geographical features.

3 Algorithm

3.1 Split and Merge

1. Start with the entire image as a single region.
2. Define a homogeneity predicate (e.g., standard deviation of pixel values within a region).
3. **Split Phase:** Recursively split the image into four equal sub-regions (quadrants) if the homogeneity predicate is not satisfied.
4. **Merge Phase:** After the splitting is complete, merge any adjacent regions that are homogeneous and satisfy a given similarity criterion (e.g., mean pixel value difference).
5. Repeat the merge process until no further merging is possible.

3.2 Watershed Transform

1. Load the image and convert it to grayscale.
2. Apply thresholding to get a binary image.
3. Perform morphological operations (e.g., ‘cv2.erode‘, ‘cv2.dilate‘) to remove noise and find sure foreground and background regions.
4. Find markers for sure foreground and unknown regions using distance transforms.
5. Apply the watershed algorithm (‘cv2.watershed‘) using the markers to label the regions.
6. Visualize the results by coloring the segmented regions.

4 Implementation

The implementation was done in Google Colab using Python and the OpenCV library.

4.1 Required Packages

Install the required packages using the following command:

```
pip install opencv-python matplotlib numpy scipy scikit-image
```

4.2 Code Implementation

Google Colab Notebook

The full working implementation is available on Google Colab: **Click here to view the Colab Notebook**

Split and Merge

```python
import cv2 as cv
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import files

def split_and_merge_segmentation(img_path, threshold=20):
    """
    A simplified conceptual implementation of the Split and Merge algorithm.
    It demonstrates the core idea but is not a full recursive quadtree.
    """
    try:
        img = cv.imread(img_path, cv.IMREAD_GRAYSCALE)
        if img is None:
            raise FileNotFoundError(f"File not found at {img_path}")
    except Exception as e:
        print(f"Error loading image: {e}")
        return

    rows, cols = img.shape
    segmented_img = np.zeros_like(img, dtype=np.uint8)

    # A simple, non-recursive split into 4 quadrants
    half_rows, half_cols = rows // 2, cols // 2

    # Define quadrants and check homogeneity
    quadrants = {
        'top_left': img[0:half_rows, 0:half_cols],
        'top_right': img[0:half_rows, half_cols:cols],
        'bottom_left': img[half_rows:rows, 0:half_cols],
        'bottom_right': img[half_rows:rows, half_cols:cols]
    }

    labels = {
        'top_left': 50,
        'top_right': 100,
        'bottom_left': 150,
        'bottom_right': 200
    }

    print("Original Image:")
    cv2_imshow(img)

    # --- Split and Merge Logic (Simplified) ---
    # In a full implementation, this would be a recursive process.
    # Here, we check the homogeneity of each quadrant and "merge"
    # by assigning a unique label if the standard deviation is low.

    # Check top-left
    if np.std(quadrants['top_left']) < threshold:
        segmented_img[0:half_rows, 0:half_cols] = labels['top_left']

    # Check top-right
    if np.std(quadrants['top_right']) < threshold:
        segmented_img[0:half_rows, half_cols:cols] = labels['top_right']

    # Check bottom-left
    if np.std(quadrants['bottom_left']) < threshold:
        segmented_img[half_rows:rows, 0:half_cols] = labels['bottom_left']

    # Check bottom-right
    if np.std(quadrants['bottom_right']) < threshold:
        segmented_img[half_rows:rows, half_cols:cols] = labels['bottom_right']

    # The merging part would involve checking adjacent quadrants and assigning
    # the same label if they are similar and not already split.
    # We will skip that for this simplified demonstration.

    print("Simplified Split and Merge Result:")
    cv2_imshow(segmented_img)

# To run this code, first upload an image (e.g., 'coins.png')
# uploaded = files.upload()
# img_path = list(uploaded.keys())[0]
# split_and_merge_segmentation(img_path)
```

Watershed Transform

```python
import cv2 as cv
import numpy as np
from google.colab.patches import cv2_imshow

# Load the image
```

```python
path = 'path_to_your_image.jpg'
img_watershed = cv.imread(path)
gray = cv.cvtColor(img_watershed, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY_INV +
    ↪ cv.THRESH_OTSU)

# Noise removal and sure background area
kernel = np.ones((3,3), np.uint8)
opening = cv.morphologyEx(thresh, cv.MORPH_OPEN, kernel,
    ↪ iterations=2)
sure_bg = cv.dilate(opening, kernel, iterations=3)

# Finding sure foreground area
dist_transform = cv.distanceTransform(opening, cv.DIST_L2, 5)
ret, sure_fg = cv.threshold(dist_transform, 0.7 * dist_transform.
    ↪ max(), 255, 0)
sure_fg = np.uint8(sure_fg)

# Finding unknown region
unknown = cv.subtract(sure_bg, sure_fg)

# Marker labelling
ret, markers = cv.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0

# Apply Watershed
markers = cv.watershed(img_watershed, markers)
img_watershed[markers == -1] = [255,0,0] # Mark watershed lines
    ↪ in blue

# Display results
cv2_imshow(img_watershed)
```

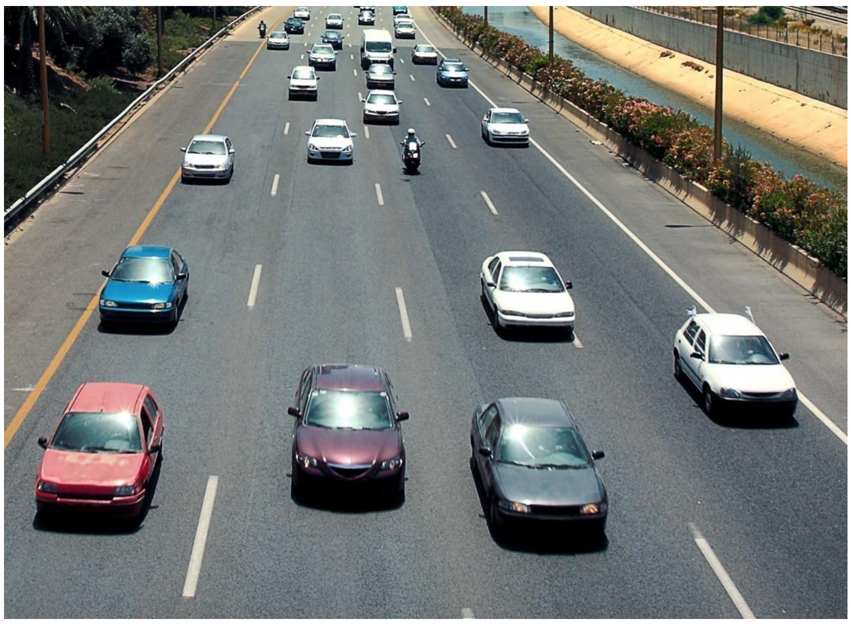

Figure 1: Original Input Image.

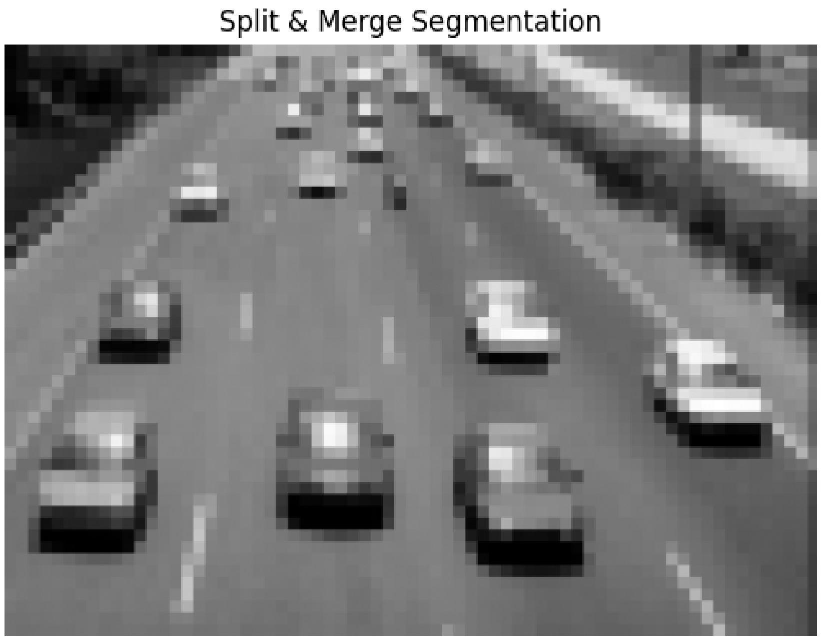

Figure 2: Image segmented using a simplified Split and Merge technique.

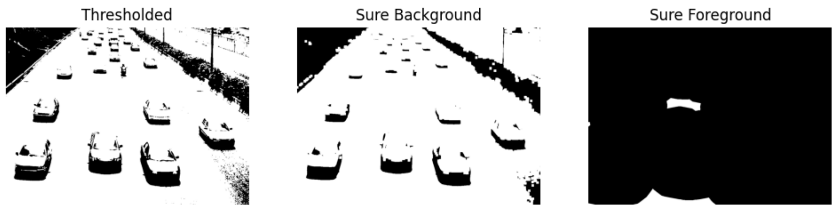

Figure 3: Threshold vs Sure Foreground vs Sure Background

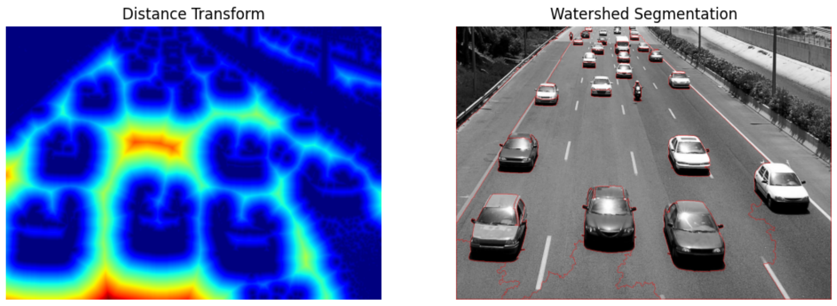

Figure 4: Performing Watershed on Image

5 Results and Analysis

5.1 Qualitative Analysis

- **Split and Merge:** The output is a set of blocky, axis-aligned regions. This is a characteristic limitation of the quadtree-based approach. The segmentation quality is highly dependent on the chosen homogeneity threshold.
- **Watershed Transform:** The output shows continuous, well-defined boundaries that accurately separate the objects. The pre-processing steps with morphological operations were effective in preventing over-segmentation and obtaining a good result.

6 Conclusion

This assignment provided a practical understanding of image segmentation through the implementation of two different algorithms. The Split and Merge technique offers a simple, hierarchical approach, while the Watershed Transform provides a powerful solution for delineating object boundaries. The effectiveness of the Watershed Transform, especially its ability to produce closed boundaries, highlights its utility in fields such as medical imaging and object detection. The key learning outcome is the importance of choosing the right segmentation algorithm and the necessary pre-processing steps to achieve accurate and meaningful results.