# **COMPUTER VISION**

# **Project report 2**

### Under the guidance of:

|  |
| --- |
| Prof.[Milan Gnjatović](http://www.gnjatovic.info/) |

|  |
| --- |
|  |

Presented by:

Rudradeep Palit

11038108

### 1.Introduction

Image segmentation is a crucial task in the field of computer vision, where the goal is to partition an image into meaningful segments, typically to simplify the representation or to make the image more understandable. In this project, we develop a graph-based segmentation algorithm to divide an image into different segments based on pixel intensity differences.

# 2. Methodology

#### 2.1 Data Preparation

* **Loading the Image**: We load the image from a file, converting it to grayscale for simplicity.
* **Resizing the Image**: To reduce computational complexity, we resize the image.

#### 2.2 Graph Construction and Segmentation

* **Graph Representation**: Each pixel in the image is treated as a node in a graph. Edges are created between neighboring pixels, with weights based on the intensity difference between pixels.
* **Union-Find Data Structure**: We use the Union-Find data structure to efficiently manage and merge segments.
* **Segmentation Algorithm**: The segmentation algorithm iteratively merges nodes based on edge weights and a predefined threshold, ensuring that similar pixels are grouped together.

# 3. Implementation

#### 3.1 Loading and Resizing the Image

We load the image and resize it to reduce computational costs.

python

import cv2import numpy as npfrom scipy.ndimage import zoomfrom matplotlib import pyplot as plt

def resize\_image(image, scale\_factor=0.5):

return zoom(image, scale\_factor)

image\_path = '/content/drive/MyDrive/objects.jpg'

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

image\_resized = resize\_image(image, scale\_factor=0.5)

#### 3.2 Building the Graph

We construct a graph where each pixel is a node, and edges are created between neighboring pixels with weights based on intensity differences.

python

import networkx as nx

def build\_graph(image):

rows, cols = image.shape

G = nx.Graph()

for x in range(rows):

for y in range(cols):

node = (x, y)

G.add\_node(node, intensity=image[x, y])

if x > 0: # Connect to the upper neighbor

G.add\_edge((x, y), (x-1, y), weight=abs(int(image[x, y]) - int(image[x-1, y])))

if y > 0: # Connect to the left neighbor

G.add\_edge((x, y), (x, y-1), weight=abs(int(image[x, y]) - int(image[x, y-1])))

return G

#### 3.3 Union-Find Data Structure

We implement the Union-Find data structure to efficiently manage segment merging.

python

class UnionFind:

def \_\_init\_\_(self, n):

self.parent = list(range(n))

self.rank = [0] \* n

def find(self, u):

if self.parent[u] != u:

self.parent[u] = self.find(self.parent[u])

return self.parent[u]

def union(self, u, v):

root\_u = self.find(u)

root\_v = self.find(v)

if root\_u != root\_v:

if self.rank[root\_u] > self.rank[root\_v]:

self.parent[root\_v] = root\_u

elif self.rank[root\_u] < self.rank[root\_v]:

self.parent[root\_u] = root\_v

else:

self.parent[root\_v] = root\_u

self.rank[root\_u] += 1

#### 3.4 Segmentation Algorithm

We segment the image by iteratively merging nodes based on edge weights.

python

def segment\_image(image, threshold):

G = build\_graph(image)

num\_nodes = len(G.nodes)

uf = UnionFind(num\_nodes)

node\_index = {node: idx for idx, node in enumerate(G.nodes)}

index\_node = {idx: node for node, idx in node\_index.items()}

edges = sorted(G.edges(data=True), key=lambda x: x[2]['weight'])

for (u, v, d) in edges:

if d['weight'] < threshold:

uf.union(node\_index[u], node\_index[v])

segments = {}

for idx in range(num\_nodes):

root = uf.find(idx)

if root not in segments:

segments[root] = []

segments[root].append(index\_node[idx])

segmented\_image = np.zeros((\*image.shape, 3), dtype=np.uint8)

for color, (root, nodes) in zip(random.sample(range(1, 256\*256\*256), len(segments)), segments.items()):

r = (color >> 16) & 255

g = (color >> 8) & 255

b = color & 255

for (x, y) in nodes:

segmented\_image[x, y] = [r, g, b]

return segmented\_image

#### 3.5 Displaying the Results

We visualize the original and segmented images.

threshold = 7 # Adjust the threshold value

segmented\_image = segment\_image(image\_resized, threshold)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(image\_resized, cmap='gray')

plt.subplot(1, 2, 2)

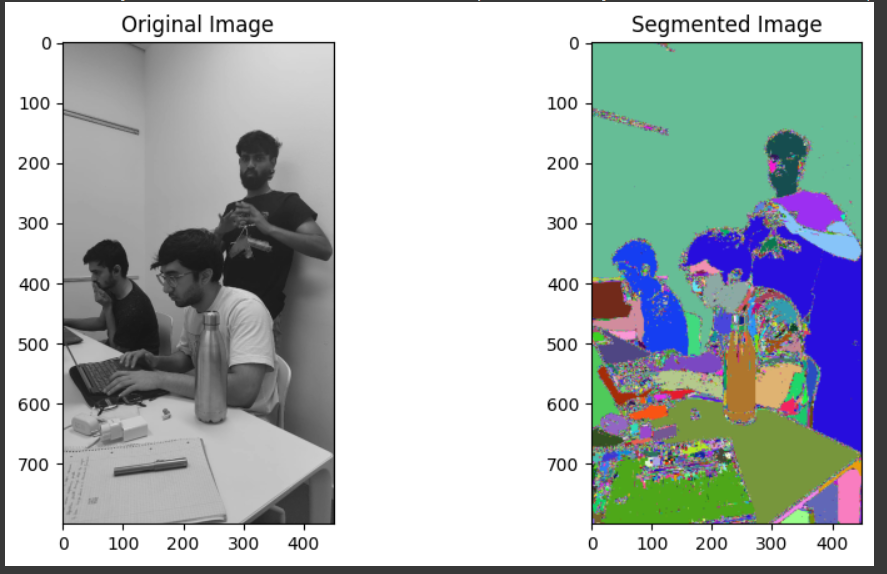
plt.title("Segmented Image")

plt.imshow(segmented\_image)

plt.show()

### 4. Results

The graph-based segmentation algorithm successfully segmented the image into distinct regions, each represented by a unique color. The effectiveness of the segmentation can be adjusted by modifying the threshold value, which controls the maximum allowed intensity difference for merging pixels.



result

# 5. Conclusion

We successfully implemented a graph-based segmentation algorithm to divide an image into meaningful segments. The use of a graph representation, combined with the Union-Find data structure, allows for efficient and effective image segmentation.

# 6. References

https://www.baeldung.com/cs/graph-based-segmentation

<https://github.com/salaee/pegbis>

https://link.springer.com/chapter/10.1007/978-3-540-89639-5\_27