Second report- Scene Segmentation and Interpretation

Kimberly Grace Sevillano C.

***Subject*— Labelling regions in image**

I. Objective

The objective of labeling regions in a binary image is to identify and distinguish separate objects or segments in the image. Labeling regions involves assigning a unique label or identifier to each distinct region in the binary image, usually with the goal of separating the image into meaningful parts for further analysis or manipulation. This can help in tasks such as object recognition, image segmentation, and image analysis.

A. Problem Description

Labelling regions in an image is a task in image processing where the aim is to assign a unique identifier (label) to each distinct region in an image. The main problem in this task is to accurately identify and separate the different regions in the image. This can be challenging in cases where the regions are touching or overlapping, or when the image has different types of textures, intensity variations, or noise. These difficulties can result in errors in the labelling process, such as missed regions, merged regions, or incorrect boundaries. To overcome these problems, various image processing techniques such as thresholding, morphological operations, and connected component analysis can be used to preprocess the image and improve the accuracy of the labelling process.

II. Methods

There are several methods for labeling regions in an image, including:

A. Connected Component Labelling (CCL)

This is a simple and popular method that assigns unique labels to connected components in a binary image. It is based on searching for connected pixels and grouping them into regions.

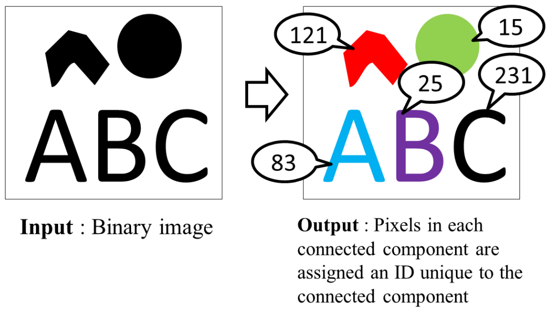

ABC
121
15
25
231
83
ABC
Input : Binary image
Output : Pixels in each connected component are assigned an ID unique to the connected component

Fig. 1. CCL example.

B. Watershed Transformation

This is a more advanced method for labeling regions in an image. It uses gradient information from the image to separate objects from their background.

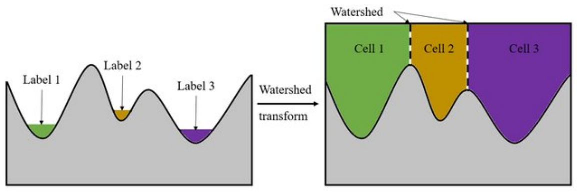


Fig. 2. The segmentation process of the watershed algorithm.

C. Region Growing

This is another popular method for labeling regions in an image. It starts with a set of seed pixels and iteratively grows the region by adding neighboring pixels that belong to the same object.

D. Level Set Methods

These are advanced methods for labeling regions in an image that use partial differential equations to track the evolution of regions over time.

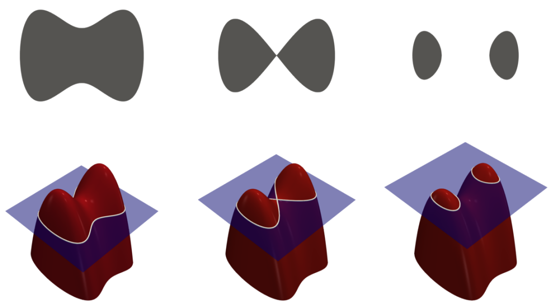


Fig. 3. An illustration of the level-set method.

E. Deep Learning Approaches

With the development of deep learning, neural networks have become popular methods for image segmentation and labeling. Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) have shown promising results in segmenting and labeling regions in images.

III. Methodology

This code is an implementation of connected components labeling algorithm. The function "connected_components" takes an input binary image represented as a 2D numpy array and returns another 2D numpy array with labels assigned to each connected component and the number of connected components. The connected components are defined as contiguous regions of non-zero elements in the input image. The algorithm works as follows:

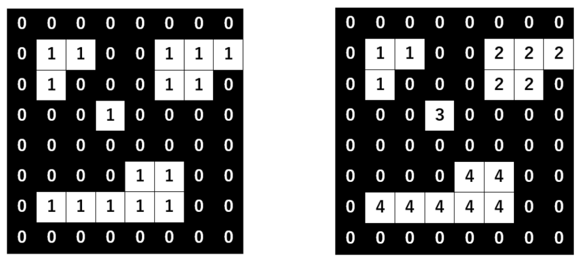
- Find the coordinates of non-zero elements in the input image.
- For each coordinate, if the corresponding position in the label image is zero, assign a new label and use breadth-first search (BFS) to label all its connected neighbors.
- Repeat this process until all non-zero elements in the input image have been assigned a label.

```python
def connected_components(image):
    m, n = image.shape
    coor_y, coor_x = np.where(image != 0)
    listita = []
    image_copy = np.zeros((m, n))
    label = 0
    for elem in range(len(coor_x)):
        i = coor_y[elem]
        j = coor_x[elem]
        if image_copy[i, j] == 0:
            image_copy[i, j] = label+1
            listita.append((i, j))
            while len(listita) != 0:
                current = listita.pop(0)
                i, j = current
                neighbors = [(i-1, j-1), (i-1, j),
                            (i-1, j+1), (i, j-1),
                            (i+1, j-1), (i+1, j)]
                for neighbor in neighbors:
                    x, y = neighbor
                    if x >= 0 and x < m and
                       y >= 0 and y < n and
                       image[x, y] != 0 and
                       image_copy[x, y] == 0:
                        image_copy[x, y] = label+1
                        listita.append((x, y))
            label + = 1
    return image_copy, label
```

Listing 1. Function for CCL calculation in python

IV. Results

The objective is to identify the non-zero elements in the matrix in 4 a) and assign them a label as shown in 4 b).



0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	1
0	1	0	0	0	1	1	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0
0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

a) Matrix original

0	0	0	0	0	0	0	0
0	1	1	0	0	2	2	2
0	1	0	0	0	2	2	0
0	0	0	3	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	4	4	0	0
0	4	4	4	4	4	0	0
0	0	0	0	0	0	0	0

b) Matrix with labels

Fig. 4. Original matrix a) and matrix when we have identified all the labels.

We plot the original image (5) and after passing through the function "connected_components" we have as a result each of the areas with their labels identified successfully.(6)

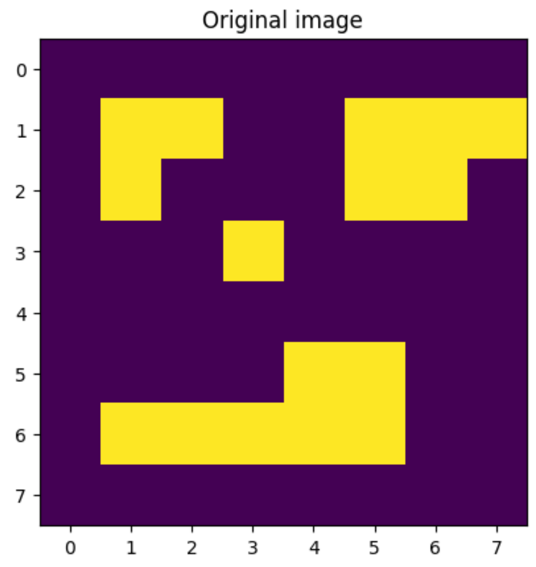


Fig. 5. Matrix plotting as image

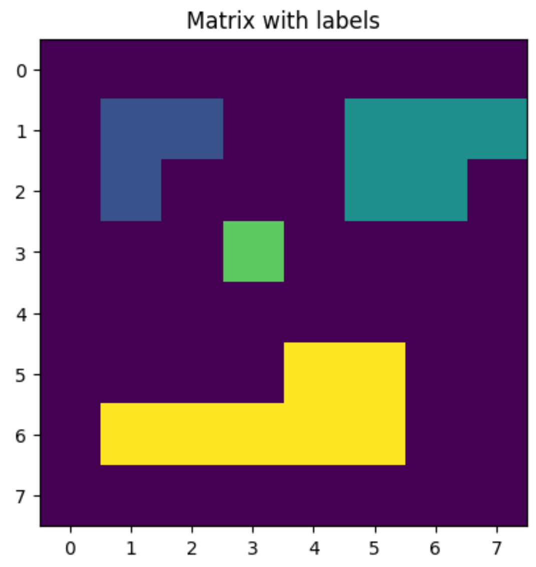


Fig. 6. Image result with labels.

V. Conclusions

In conclusion, the implementation of a connected component labeling algorithm from scratch was a valuable exercise that helped us understand the inner workings of this technique. We were able to identify the main steps involved in the algorithm, such as finding the coordinates of non-zero elements in the input image, assigning a new label to each connected component, and using breadth-first search to label all its connected neighbors.

The implementation was successful, as the algorithm was able to accurately label all the connected components in a binary image, giving us a clear visualization of the separate regions in the image. Additionally, the algorithm was able to handle a variety of different shapes and sizes of connected components.
However, it has some disadvantages, such as high computational costs, high memory requirements, and sensitivity to image noise and variability. Small changes in the image can cause mislabeling of connected components, making the algorithm unreliable for real-world applications.
An alternative approach is the maximum connected component (MCC) analysis. The MCC algorithm is a more reliable and efficient alternative to the connected component labeling algorithm, as it is more robust to image noise and variability, reduces memory requirements and computation time, and is suitable for large and complex images.

References

[1] https://towardsdatascience.com/implementing-a-connected-component-labeling-algorithm-from-scratch-94e1636554f.

[2] https://numpy.org/doc/stable/reference/generated/numpy.where.html.

[3] https://medium.com/swlh/image-processing-with-python-connected-components-and-region-labeling-3eef1864b951.