Department of Computer Science and Engineering (Data Science)
Image Processing and Computer Vision I (DJ19DSL603)

Name:Shreenissh Salian **Sap ID:60009210190** **CSDS-D2**

AIM:

THEORY:

1. Region Growing

The objective of segmentation is to partition an image into regions. One way to do is to by finding the regions directly. When we are segmenting based on regions, we are essentially finding similarities. In edge detection, we find differences. The basic approach is to start with a set of "seed" points, and from these grow regions by appending to each seed those neighboring pixels that have predefined properties similar to the seed. Properties can be: ranges of intensity, color, texture, shape, model etc. Region growth should stop when no more pixels satisfy the criteria for inclusion in that region. Criteria such as intensity values, texture, and color are local in nature and do not take into account the "history" of region growth. Additional criteria that can increase the power of a region-growing algorithm utilize the concept of size,likeness between a candidate pixel and the pixels grown so far (such as a comparison of the intensity of a candidate and the average intensity of the grown region), and the shape of the region being grown.

Let:

f(x,y): input image
S(x,y): seed array containing 1's at the locations of seed points and 0's elsewhere
Q: a predicate to be applied at each location (x, y)
Arrays f and S are assumed to be of the same size

1. Find all connected components in $S(x, y)$ and reduce each connected component to one pixel; label all such pixels found as 1. All other pixels in S are labeled 0.
2. Form an image f_Q such that, at each point (x, y), $f_Q(x,y) = 1$ if the input image satisfies a given predicate, Q, at those coordinates, $f_Q(x,y) = 0$ and otherwise. The predicate can use a threshold T for the same:

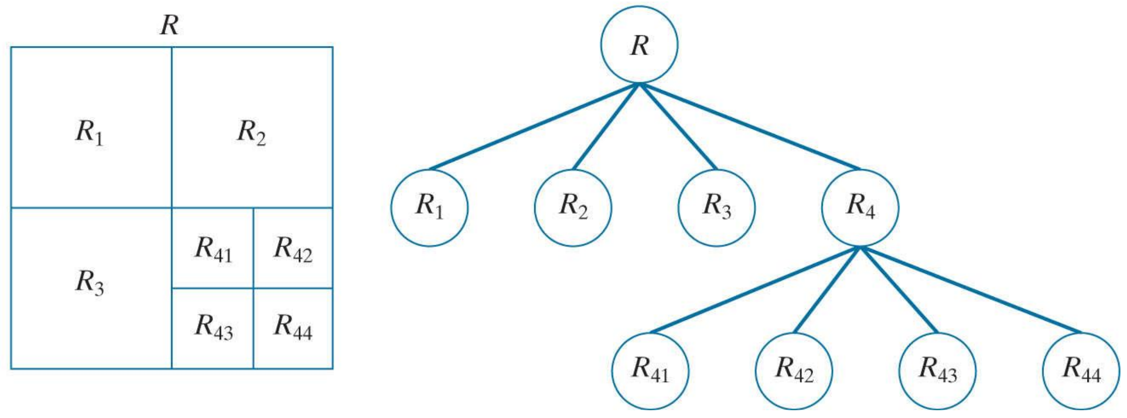
$$Q = \begin{cases} \text{TRUE} & \text{if the absolute difference of intensities} \\ & \text{between the seed and the pixel at } (x, y) \text{ is } \le T \\ \text{FALSE} & \text{otherwise} \end{cases}$$

3. Let g be an image formed by appending to each seed point in S all the 1-valued points in f_Q that are L-connected to that seed point. (L could be 4, 8 or m)
4. Label each connected component in g with a different region label (e.g.,integers or letters). This is the segmented image obtained by region growing

2. Region Splitting and Merging

An alternative is to subdivide an image initially into a set of disjoint regions and then merge and/or split the regions in an attempt to satisfy the conditions of segmentation stated in region growing
Let *R* represent the entire image region and select a predicate *Q*. One approach for segmenting *R* is to subdivide it successively into smaller and smaller quadrant regions so that, for any region. We start with the entire region, *R*. If we divide the image into quadrants. If *Q* is FALSE for any quadrant, we subdivide that quadrant into sub-quadrants, and so on. This splitting technique has a convenient representation in the form of so-called *quadtrees*; that is, trees in which each node has exactly four descendants

See a partitioned image and its corresponding quadtree:



1. Split into four disjoint quadrants any region R_i for which $Q(R_i)$ = FALSE
2. When no further splitting is possible, merge any adjacent regions R_j and R_k and for which $Q(R_j \cup R_k) = \text{TRUE}$
3. Stop when no further merging is possible.

Lab Assignments to complete in this session

Department of Computer Science and Engineering (Data Science)
Image Processing and Computer Vision I (DJ19DSL603)

Problem Statement: Develop a Python program utilizing the OpenCV library to manipulate images from the Fashion MNIST digits dataset. The program should address the following tasks:

1. Importing libraries
2. Read random image(s) from the MNIST fashion dataset.
3. **Dataset Link:** Digit MNIST Dataset
4. Extract a region of the input image depending on a start position and a stop condition.
5. The input should be a single channel 8 bits image and the seed a pixel position (x, y).
6. The threshold corresponds to the difference between outside pixel intensity and mean intensity of region.
7. In case no new pixel is found, the growing stops.
8. Output a single channel 8 bits binary (0 or 255) image. Extracted region is highlighted in white – REGION GROWING
9. For REGION MERGING AND SPLITTING, divide the whole image into regions if the threshold predicate does not match
10. When no further division is possible, merge regions if their unions satisfy the predicate.
11. Provide outputs of both #8 and #10

The solution to the operations performed must be produced by scratch coding without the use of built in OpenCV methods.

In [42]:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color
import cv2
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
```

In [43]:

```
(train_images, train_labels), (_, _) = mnist.load_data()
```

In [44]:

```
def get_random_image_by_digit(digit):
    indices = np.where(train_labels == digit)[0]
    random_index = np.random.choice(indices)
    return train_images[random_index]
```

In [45]:

```
image = get_random_image_by_digit(5)
image
```

Out[45]:

```
ndarray (28, 28)
```

show data

In [46]:

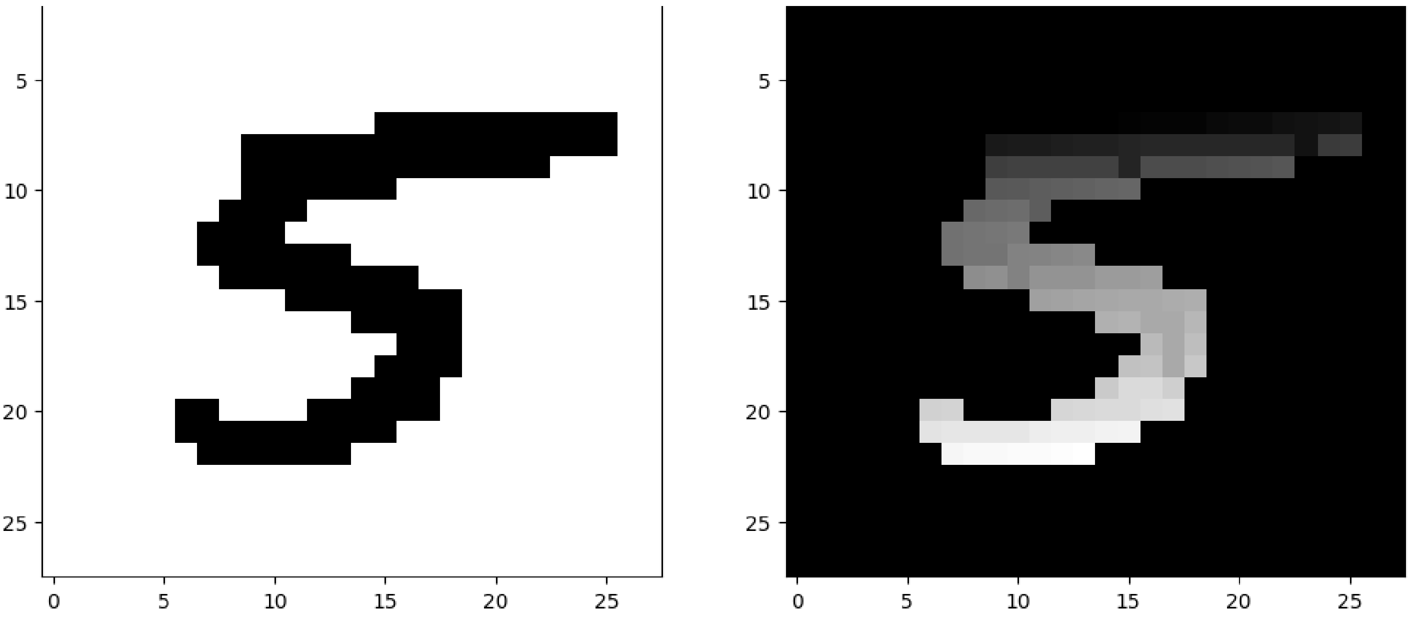
```
def region_growing(img, seed, threshold):
    x, y = seed
    height, width = img.shape
    region = np.zeros((height, width), dtype=np.uint8)
    region_list = [(x, y)]
    mean_intensity = float(img[x, y])

    while region_list:
        x, y = region_list.pop(0)
        for dx in [-1, 0, 1]:
            for dy in [-1, 0, 1]:
                nx, ny = x + dx, y + dy
                if 0 <= nx < height and 0 <= ny < width:
                    if region[nx, ny] == 0:
                        if abs(int(img[nx, ny]) - mean_intensity) < threshold:
                            region[nx, ny] = 255
                            region_list.append((nx, ny))
                            mean_intensity = (mean_intensity * np.sum(region == 255) + i
mg[nx, ny]) / (np.sum(region == 255) + 1)
    return region
```

In [47]:

```
def find_region(img, labels, x, y, region_id):
    stack = [(x, y)]
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    while stack:
        cx, cy = stack.pop()
        for dx, dy in directions:
            nx, ny = cx + dx, cy + dy
            if 0 <= nx < img.shape[0] and 0 <= ny < img.shape[1]:
                if labels[nx, ny] == -1 and img[cx, cy] == img[nx, ny]:
                    labels[nx, ny] = region_id
                    stack.append((nx, ny))

def split_image(img, threshold):
    labels = -1 * np.ones(img.shape, dtype=int)
    region_id = 0
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            if labels[x, y] == -1:
                labels[x, y] = region_id
                find_region(img, labels, x, y, region_id)
                region_id += 1
    return labels

def merge_regions(img, labels, threshold):
    height, width = img.shape
    region_means = {}
    for region in np.unique(labels):
        region_means[region] = np.mean(img[labels == region])

    neighbors = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    merged = False

    for x in range(height):
        for y in range(width):
            current_region = labels[x, y]
            for dx, dy in neighbors:
                nx, ny = x + dx, y + dy
                if 0 <= nx < height and 0 <= ny < width:
                    neighbor_region = labels[nx, ny]
                    if neighbor_region != current_region:
                        if abs(region_means[current_region] - region_means[neighbor_regi
on]) < threshold:
                            labels[labels == neighbor_region] = current_region
                            region_means[current_region] = (region_means[current_region]
+ region_means[neighbor_region]) / 2
                            del region_means[neighbor_region]
                            merged = True

    return labels, merged


def region_merging(img, initial_threshold, merge_threshold):
    labels = split_image(img, initial_threshold)
    while True:
        labels, merged = merge_regions(img, labels, merge_threshold)
        if not merged:
            break
    return labels
```

In [48]:

```
if __name__ == "__main__":
    img = image
    seed = (10, 20)
    threshold = 15

    grown_region = region_growing(img, seed, threshold)
    merged_image = region_merging(img, threshold, threshold)

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.title('Region Growing')
    plt.imshow(grown_region, cmap='gray')
    plt.subplot(1, 2, 2)
    plt.title('Region Merging')
    plt.imshow(merged_image, cmap='gray')
    plt.show()
```


5
10
15
20
25
0
5
10
15
20
25