DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

Assignment#1

DIGITAL IMAGE PROCESSING

Student Name: _____Salva Nadeem_____

Degree/Syndicate: ___CE 43 B_____

Skin Image Segmentation using Connected Component Labeling

➢ Abstract:

In this assignment, a method for segmenting tissue types in medical images using color-based segmentation and connected component analysis (CCA) is implemented. The goal is to accurately segment different tissue types in medical images such as dermis (DRM), epidermis (EPI), Dermal-epidermal junction (DEJ) and keratin (KER). The performance of the segmentation method is evaluated using the Dice coefficient, a common metric for evaluating segmentation accuracy.

➢ CODE:

- **V_Set:**

In order to get the V_set , I calculated the intensity ranges (V_SET) for different tissue types by analyzing pixel intensities in the image based on the labeled mask. It does this by extracting pixel intensities for each tissue type from the image and then calculating the mean and standard deviation of these intensities. These intensity ranges are essential for defining the color thresholds used in subsequent tissue segmentation tasks.

```
v_sets = {
    'BKG': {'B': (240, 255), 'G': (240, 255), 'R': (240, 255.0)},
    'DEJ': {'B': (100.86, 250.92), 'G': (100.13, 221.19), 'R': (221.52,
255.0)},
    'EPI': {'B': (100, 255), 'G': (0, 200), 'R': (150, 255)},
    'DRM': {'B': (178.21, 255), 'G': (138.12, 250.45), 'R': (220.55, 255.0)},
    'KER': {'B': (30, 200), 'G': (40, 200), 'R': (80, 180.5)}
}
```

- **Libraries Used:**

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import os
```

NumPy (import numpy as np): Used for numerical operations and array manipulation.

OpenCV (import cv2 as cv): Utilized for image processing tasks such as reading, processing, and saving images.

Matplotlib.pyplot (import matplotlib.pyplot as plt): Employed for visualizing images and results.

OS (import os): Used for file and folder operations.

Function Definitions:

- **CCA 8:**

This function performs eight-connected component analysis (CCA) based on the specified color ranges. It assigns a label to connected components and generates a mask containing labeled connected components.

```
def in_v_set(pixel, v_set):
    b, g, r = pixel  # Extracting the B, G, R channels
    for k, v in v_set.items():
        if k == 'B':
            b_range = v
        elif k == 'G':
            g_range = v
        elif k == 'R':
            r_range = v
    if b_range[0] <= b <= b_range[1] and g_range[0] <= g <= g_range[1] and
r_range[0] <= r <= r_range[1]:
        return True
    return False
def CCA_eight(v_set, image, color):
    img = image
    mask = np.zeros((img.shape[0], img.shape[1]), dtype=np.uint8)
    r, c, _ = image.shape
    label = color
```

```
    for i in range(r):
        for j in range(c):
            if i == 0 and j == 0: #top left corner
                if in_v_set(img[i, j], v_set):
                    mask[i, j] = label
            elif i == 0 and j > 0:#top row
                if in_v_set(img[i, j], v_set):
                    if in_v_set(img[i, j - 1],v_set):
                        mask[i, j] = label
            elif i > 0 and j == 0:#left column
                if in_v_set(img[i, j], v_set):
                    up = img[i - 1, j]
                    up_right = img[i - 1, j + 1]
                    if in_v_set(up, v_set) or in_v_set(up_right, v_set):
                        mask[i, j] = label
            elif i>0 and j==c-1:#right column
                if in_v_set(img[i,j],v_set):
                   left=img[i,j-1]
                   up=img[i-1,j]
                   up_left=img[i-1,j-1]
                   if in_v_set(up,v_set) or in_v_set(left,v_set) or
in_v_set(up_left,v_set):
                            mask[i,j]=label
            elif i > 0 and j > 0:#rest of the image
                if in_v_set(img[i, j], v_set):
                    up = mask[i - 1, j]
                    left = mask[i, j - 1]
                    up_left = mask[i - 1, j - 1]
                    up_right = mask[i - 1, j + 1]
                    neighbors = [up, left, up_left, up_right]
                    neighbors = [neighbor for neighbor in neighbors if
np.any(neighbor != 0)]
                    if len(neighbors) > 0:
                        min_neighbor = min([np.min(neighbor) for neighbor in
neighbors])
                        mask[i, j] = min_neighbor
                    else:
                        mask[i, j] = label
    return mask, label
```

- **Color_largest_component:**

This function identifies the largest connected component in a binary mask and assigns a specified color to it.

```
def color_largest_component(mask, label):
    num_labels, labeled_mask, stats, centroids =
cv.connectedComponentsWithStats(mask)
    largest_label = np.argmax(stats[0:, cv.CC_STAT_AREA]) + 1
    largest_component_mask = np.where(labeled_mask == largest_label, label,
0)
    largest_component_mask = np.uint8(largest_component_mask)
    return largest_component_mask
```

- **Color_mask:**

This function creates a color mask by applying a specified color to pixels that meet certain criteria in the input mask.

```
def color_mask(mask, color):
    colored_mask = np.ones((mask.shape[0], mask.shape[1], 3), dtype=np.uint8)
    for i in range(mask.shape[0]):
        for j in range(mask.shape[1]):
            if mask[i, j] == 255:
                colored_mask[i, j] = color
    return colored_mask
```

- **Creating_masks:**

This function generates masks for different tissue types by applying CCA and color assignment.

```
def creating_masks(img,color,v_set):
    proccessed_mask,l=CCA_eight(v_set,img,255)
    colored_mask1 = color_largest_component(proccessed_mask, 255)
    colored_mask=color_mask(proccessed_mask,color)
    return colored_mask
```

- **Masks:**

This function iterates over predefined tissue types, generates masks for each type, and displays them.

```
def masks(img):
    #Defining V_SET
    v_sets = {
        'BKG': {'B': (240, 255), 'G': (240, 255), 'R': (240, 255.0)},
        'DEJ': {'B': (166, 250.92), 'G': (116, 240), 'R': (221.52, 255.0)},
        'EPI': {'B': (100, 255), 'G': (0, 200), 'R': (150, 255)},
        'DRM': {'B': (178.21, 255), 'G': (138.12, 250.45), 'R': (220.55,
255.0)},
        'KER': {'B': (150, 200), 'G': (40, 200), 'R': (80, 180.5)}
    }
    tissue_colors = {
    'DEJ': [255, 172, 255],
    'DRM': [0, 255,190],
    'EPI': [160, 48, 112],
    'KER': [224, 224, 224],
    'BKG': [255, 255, 255]
    }
    masks=[]
    for tissue_type in v_sets:
        mask = creating_masks(img, tissue_colors[tissue_type],
```

```
v_sets[tissue_type])
        masks.append(mask)
    for mask in masks:
        plt.imshow(cv.cvtColor(mask, cv.COLOR_BGR2RGB))
        plt.show()
    return masks
```

- **Merge_masks:**

This function merges multiple masks into a single mask by bitwise OR operation.

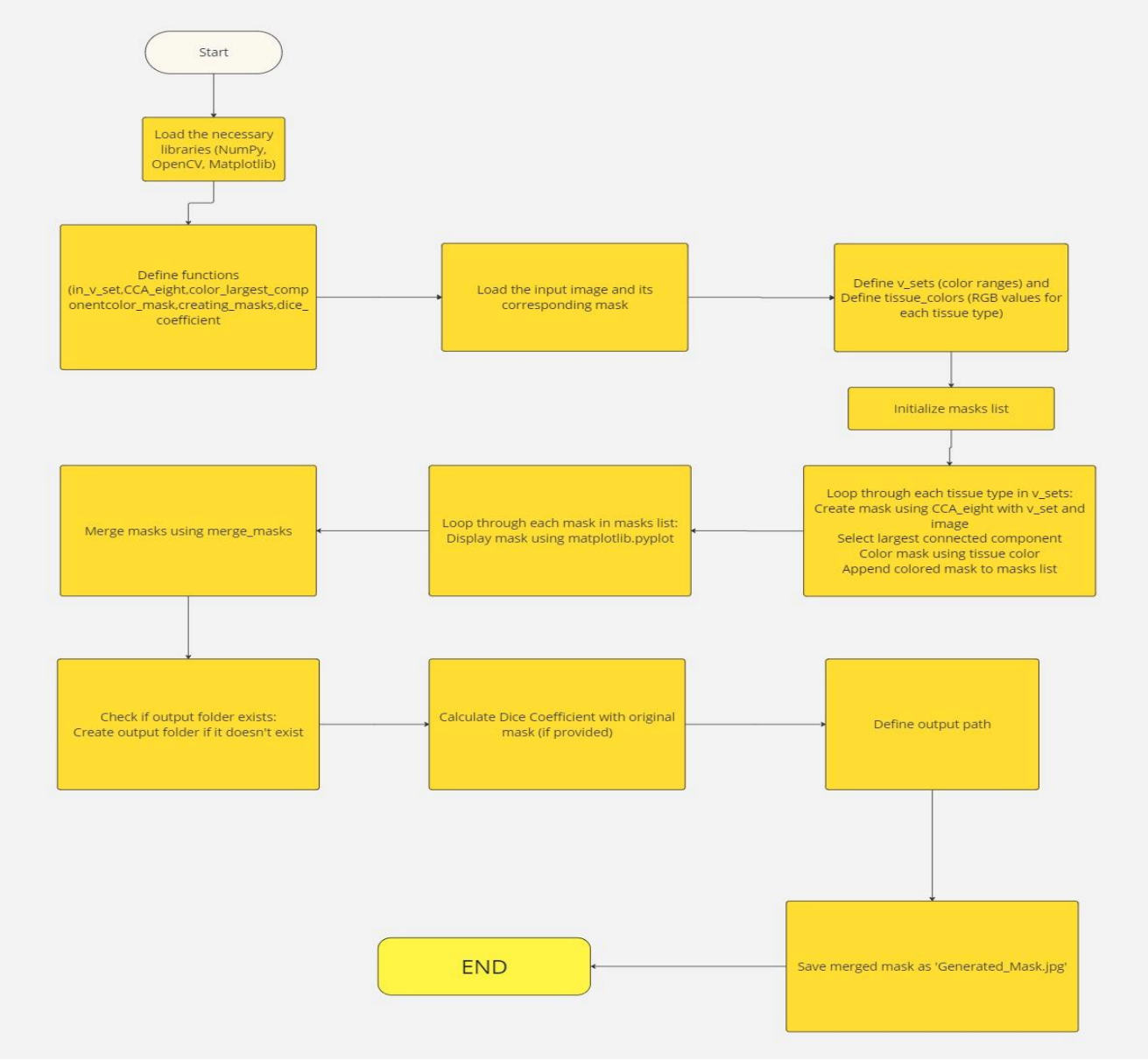
```
def merge_masks(masks):
    merged_mask = np.zeros_like(masks[0])
    for i,mask in enumerate(masks):
        if i == 0:
        #mask = np.bitwise_not(mask)
            continue
        N_black_pixels = np.any(mask != [0, 0, 0], axis=-1)
        merged_mask[N_black_pixels] |= mask[N_black_pixels]
    return merged_mask
```

- **Main:**

1. Loads the input image and its corresponding original mask.
2. Calls the masks function to generate masks for different tissue types.
3. Merges the generated masks into a single mask.
4. Displays the merged mask and saves it to a specified output folder.
5. Calculates and displays the Dice coefficient between the generated mask and the original mask.

```
•   output_folder = 'Assignment #1/Test/Output/'
    img = cv.imread(r"Test/tissue/RA23-01882-A1-1-PAS.[10240x1536].jpg")
    or_mask=cv.imread("Test/Mask/RA23-01882-A1-1.[10240x1536].png")
    plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
    plt.show()
    #Calling the function to create masks
    masking=masks(img)
    merged_masks=merge_masks(masking)
    plt.imshow(cv.cvtColor(merged_masks, cv.COLOR_BGR2RGB))
    plt.title('Merged Mask with Black Background')
    plt.show()
    if not os.path.exists(output_folder):
       os.makedirs(output_folder)
    output_path = os.path.join(output_folder, "Generated_Mask.jpg")
    cv.imwrite(output_path, merged_masks)
    dice_coefficient(merged_masks,or_mask)
```

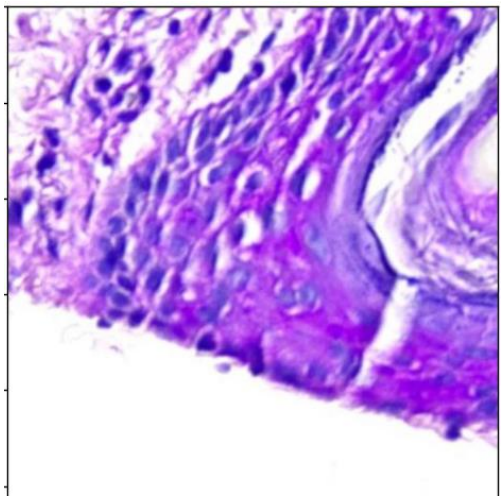
➢ Flow Chart:

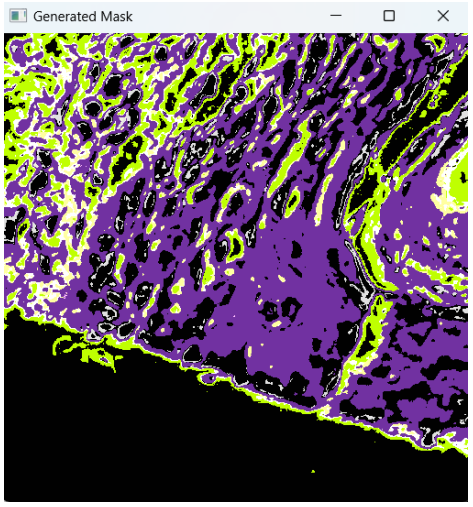

Start
Load the necessary libraries (NumPy, OpenCV, Matplotlib)
Define functions (in_v_set,CCA_eight,color_largest_component color_mask,creating_masks,dice_coefficient
Load the input image and its corresponding mask
Define v_sets (color ranges) and Define tissue_colors (RGB values for each tissue type)
Initialize masks list
Loop through each tissue type in v_sets:
Create mask using CCA_eight with v_set and image
Select largest connected component
Color mask using tissue color
Append colored mask to masks list
Loop through each mask in masks list:
Display mask using matplotlib.pyplot
Merge masks using merge_masks
Check if output folder exists:
Create output folder if it doesn't exist
Calculate Dice Coefficient with original mask (if provided)
Define output path
Save merged mask as 'Generated_Mask.jpg'
END

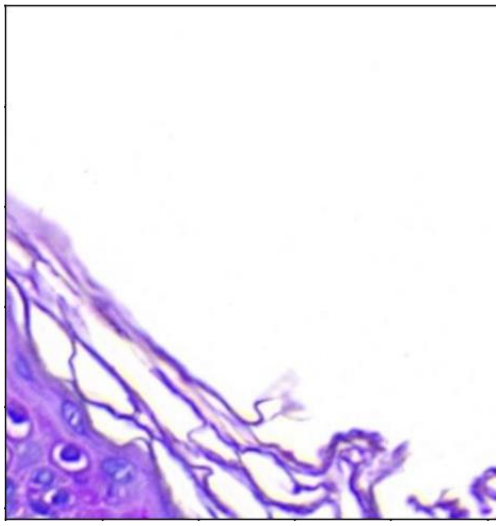
➢ Output:

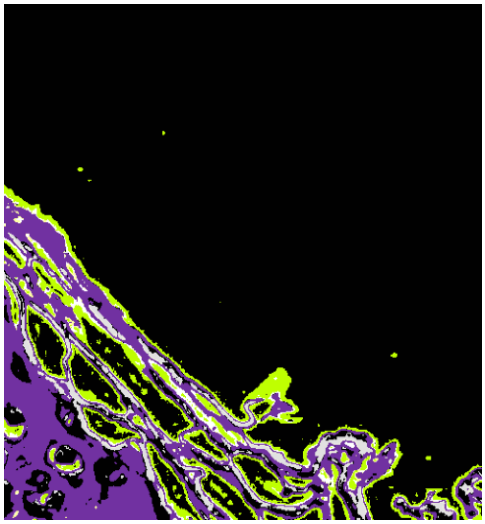
The images below show original image, ground truth and the segmented masks

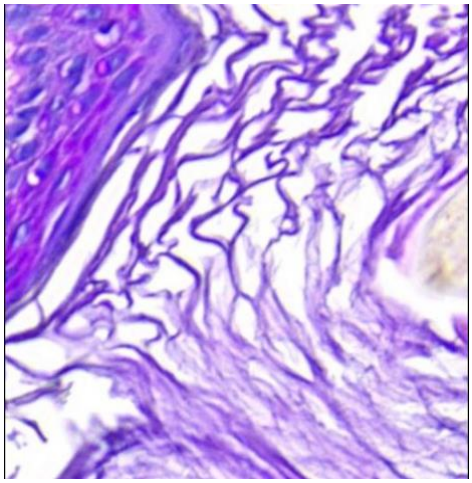
Original | Ground Truth | Generated Mask

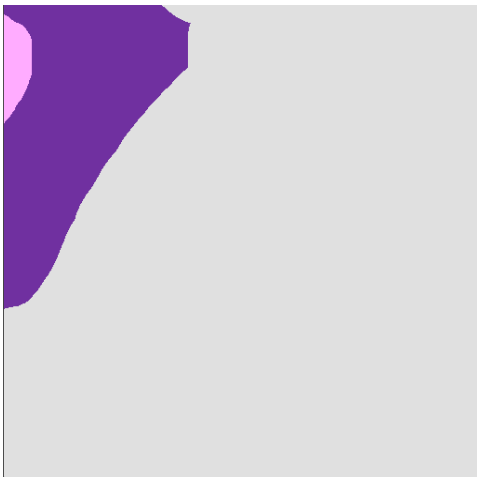


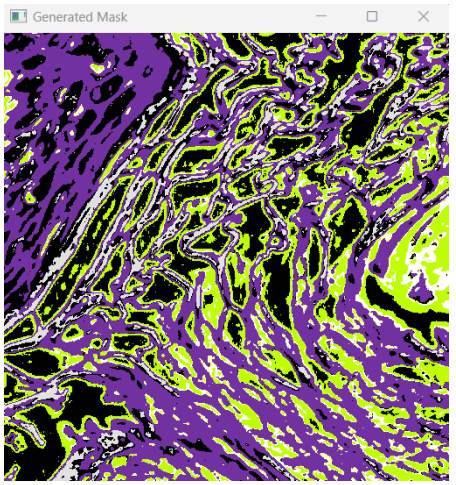


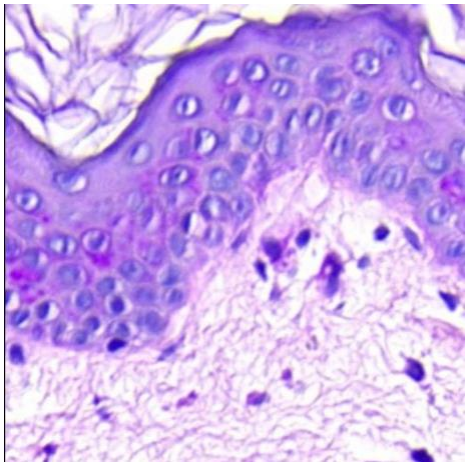


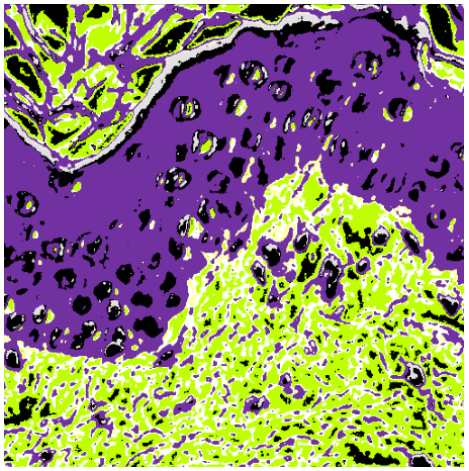












➢ Conclusion:

The implemented method for skin image segmentation using connected component Analysis (CCA) and color-based segmentation provides an effective way to segment different tissue types in medical images. By defining specific color ranges for each tissue type and employing CCA for connected component analysis, the algorithm identifies and separates tissue layers such as dermis, epidermis, dermal-epidermal junction, and keratin.