DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

Digital Image Processing

Assignment #01

Student Name: **Muhammad Hamza Tariq – 371070**

Degree/ Syndicate: 43 CE B

Skin Image Segmentation using Connected Component Labeling

Introduction

Image segmentation and masking are fundamental techniques in computer vision used for identifying objects or regions of interest within an image. This article aims to explain the working of these techniques, focusing on the steps involved and the methodologies employed.

Initial Image Processing

Reading Images from Folders

The process begins with reading original images and their corresponding masks from designated folders. This step ensures that the necessary data is available for further analysis.

Saving and Reading V Set

The V set, representing the color intensities of different layers, is saved and later read from a file. This set serves as a reference for subsequent processing stages.

Background Removal

Removing Background from Images

Background removal is crucial to isolate the foreground objects from the rest of the image. By applying thresholding and connected component analysis, the background pixels are identified and eliminated.

Refining Layer Intensities

To enhance the accuracy of segmentation, the intensities of different layers are refined based on their frequency distribution. This refinement step optimizes the segmentation process.

Color Intensity Analysis

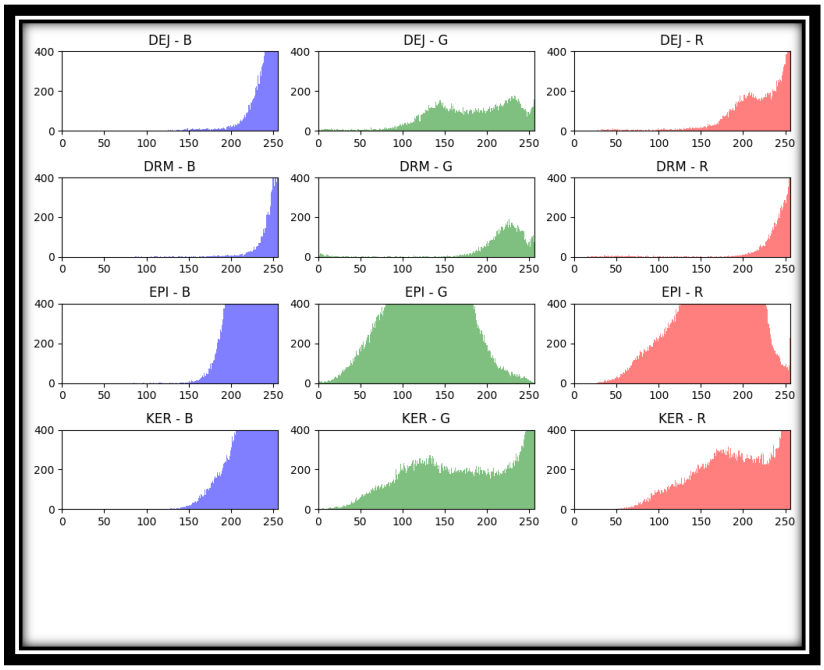
Smoothing Images

Smoothing techniques are applied to reduce noise and irregularities in the images, resulting in cleaner segmentation results.

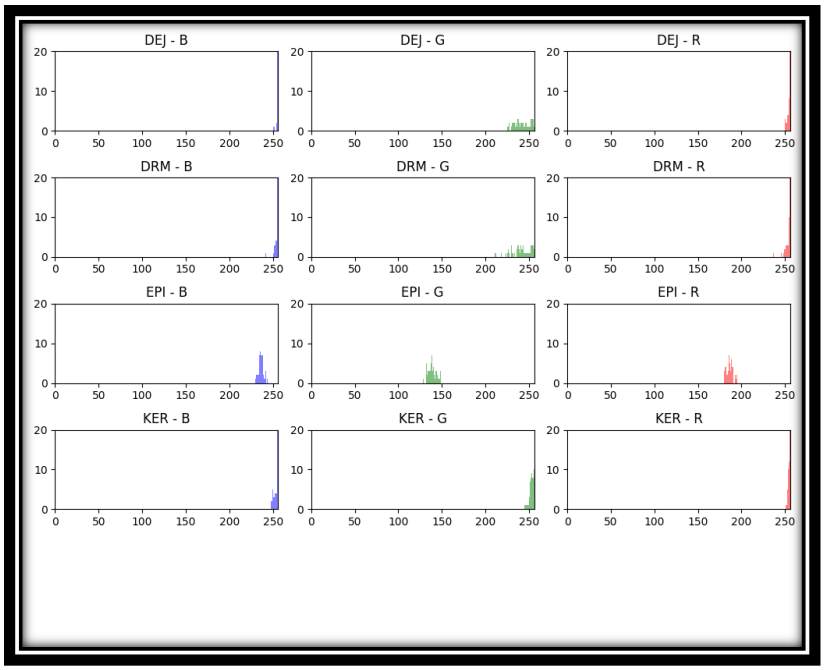
Displaying Histograms

Histograms of color intensities for each layer are displayed to visualize their distribution and identify potential segmentation boundaries.

Before refining:



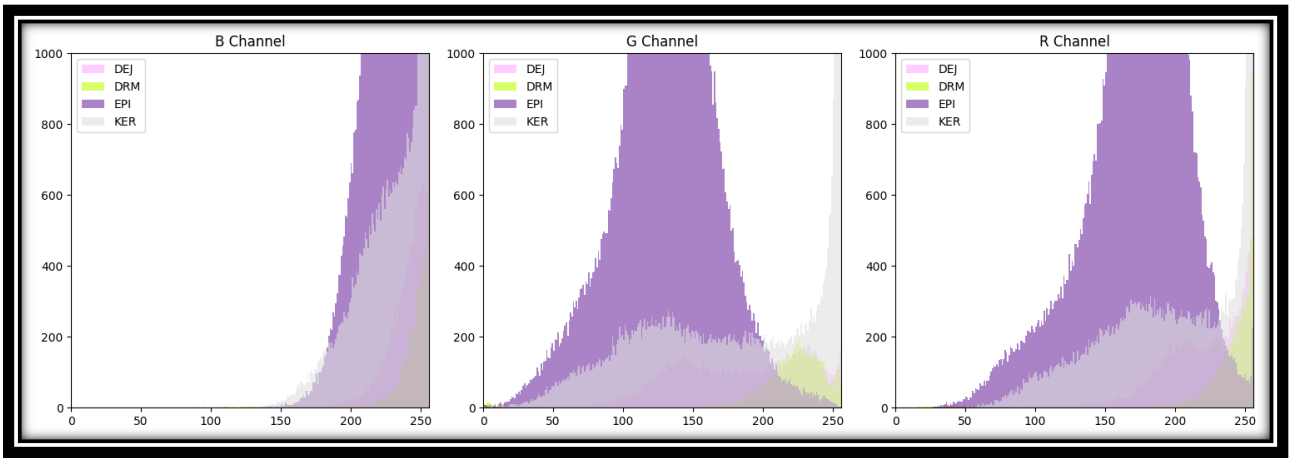
After refining:



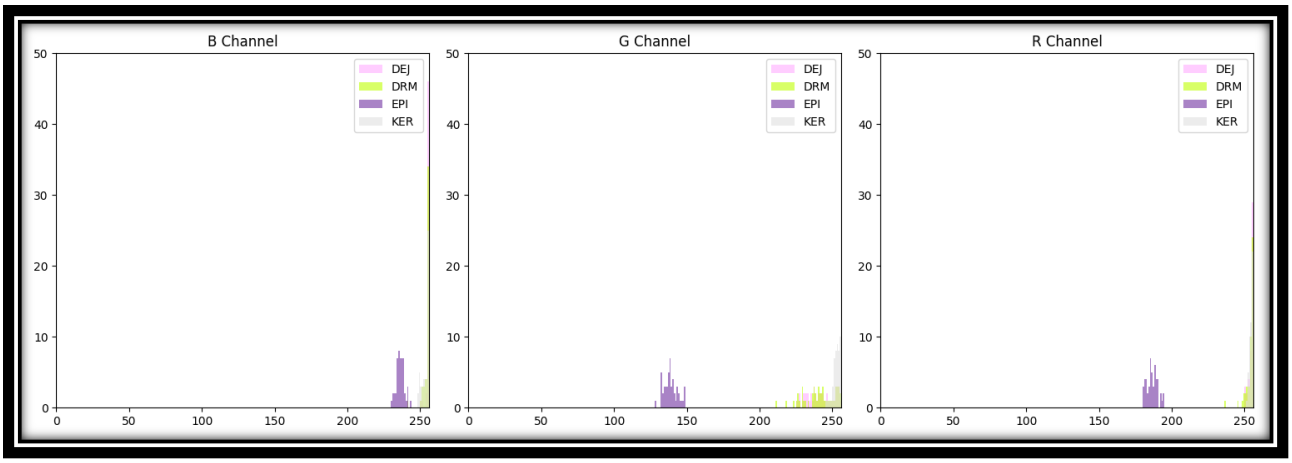
Resultant Histograms

The resultant histograms provide insights into the distribution of color intensities after segmentation, aiding in the evaluation of segmentation quality.

Before refining:



After refining:



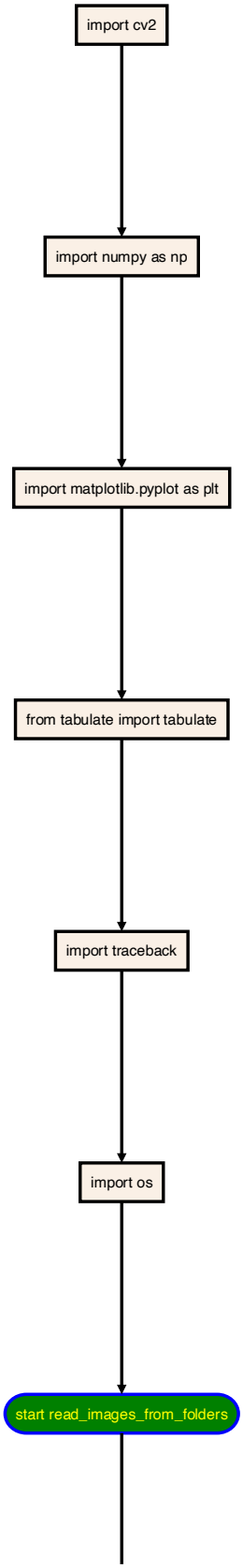
Component Labeling and Analysis

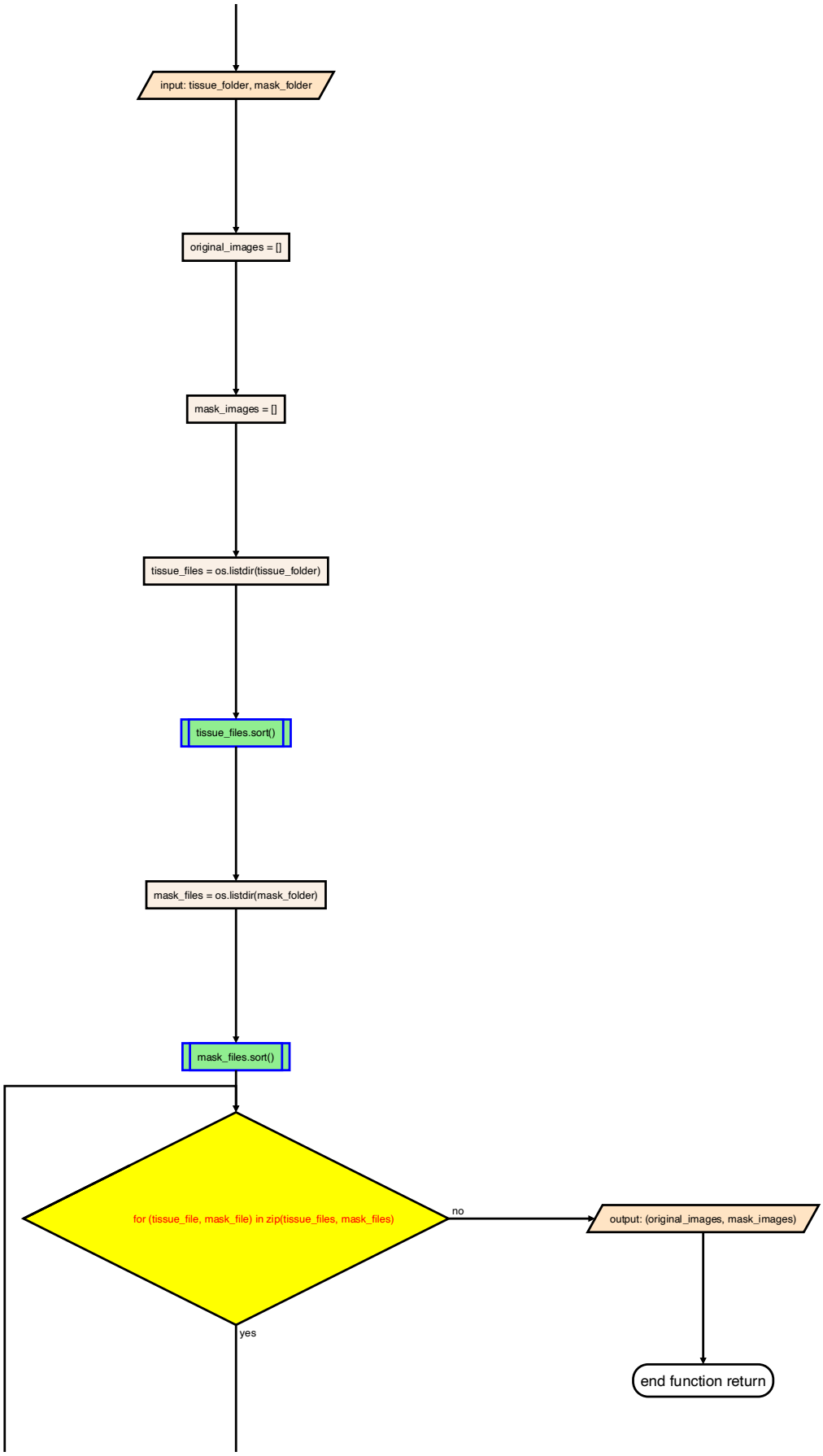
Connected Component Analysis (CCA)

CCA is employed to label and analyze connected regions within the segmented images, facilitating further processing and analysis.

Building Masks

Masks representing segmented regions are constructed based on the identified color intensities, enabling the isolation and extraction of specific objects or regions.


import cv2
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate
import traceback
import os
start read_images_from_folders


input: tissue_folder, mask_folder
original_images = []
mask_images = []
tissue_files = os.listdir(tissue_folder)
tissue_files.sort()
mask_files = os.listdir(mask_folder)
mask_files.sort()
for (tissue_file, mask_file) in zip(tissue_files, mask_files)
no
output: (original_images, mask_images)
yes
end function return

Evaluation

Computing Dice Coefficients

Dice coefficients are computed to quantify the similarity between the segmented masks and the ground truth masks, providing a measure of segmentation accuracy.

Writing Dice Coefficients to File

The computed Dice coefficients are written to a file for record-keeping and further analysis.

Working Process Explanation:

1. **Reading Images and Masks:**
    - Original images and their corresponding masks are read from designated folders.
    - Each image represents the tissue sample, and its mask indicates the regions of interest within the image.
2. **Initial Processing and V Set Creation:**
    - The images and masks are processed to extract relevant features and information.
    - A V set is created, representing the color intensities of different layers within the images.
    - This V set serves as a reference for subsequent processing stages.
3. **Background Removal:**
    - Background removal techniques are applied to isolate the foreground objects from the rest of the image.
    - Thresholding and connected component analysis are used to identify and eliminate background pixels.
    - The V set is refined based on the foreground intensities, optimizing the segmentation process.
4. **Color Intensity Analysis:**
    - Smoothing techniques are applied to reduce noise and irregularities in the images.
    - Histograms of color intensities for each layer are displayed to visualize their distribution and identify segmentation boundaries.
    - The resultant histograms provide insights into the distribution of color intensities after segmentation.
5. **Component Labeling and Analysis:**
    - Connected component analysis (CCA) is employed to label and analyze connected regions within the segmented images.
    - Masks representing segmented regions are constructed based on the identified color intensities, enabling the isolation and extraction of specific objects or regions.
6. **Evaluation:**
    - Dice coefficients are computed to quantify the similarity between the segmented masks and the ground truth masks.

- The computed coefficients are written to a file for record-keeping and further analysis.

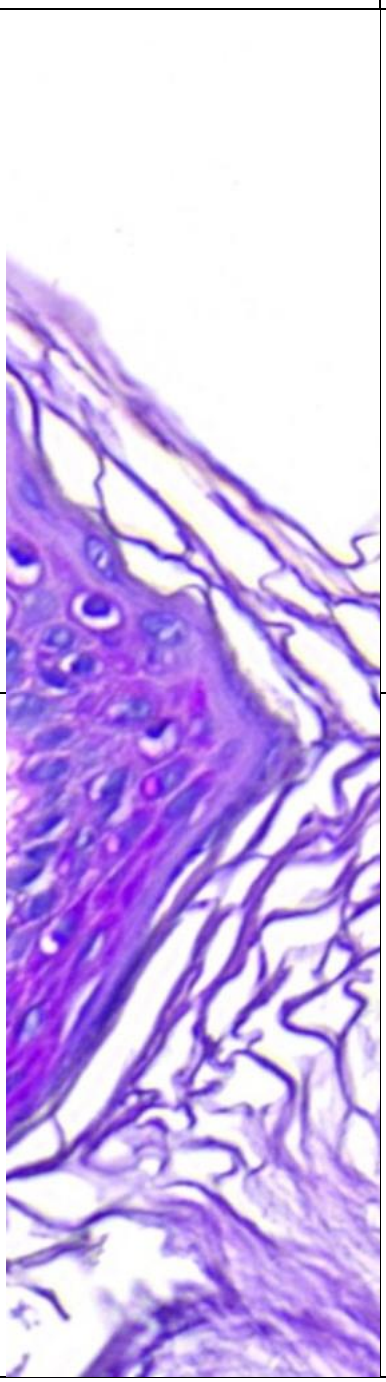
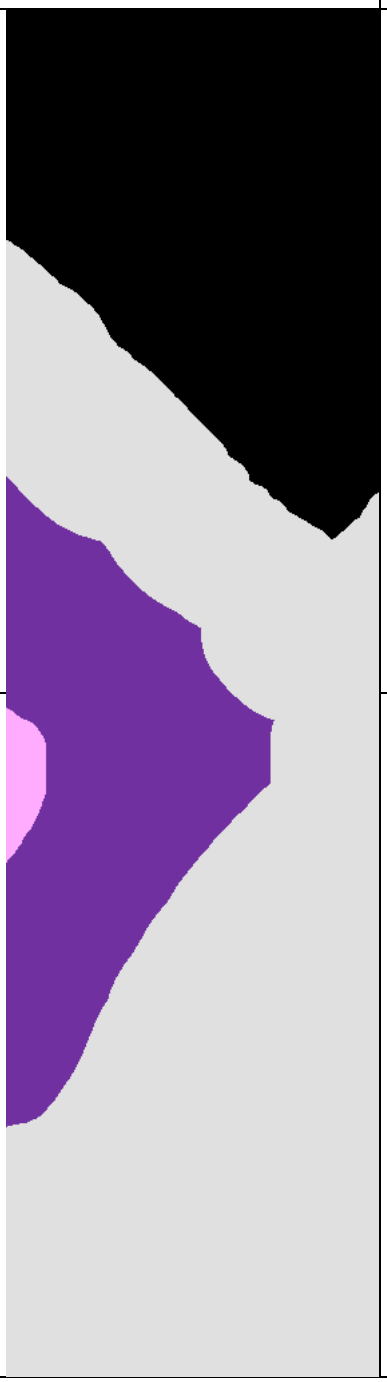
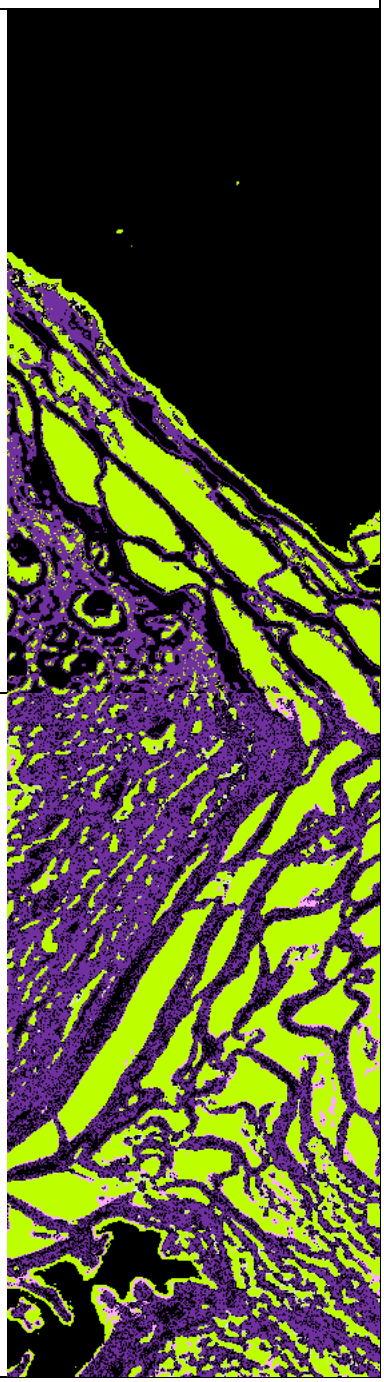
Dice Coefficient	Image	Original Mask	Created Mask
Image 1: 74.38			
Image 2: 33.15			

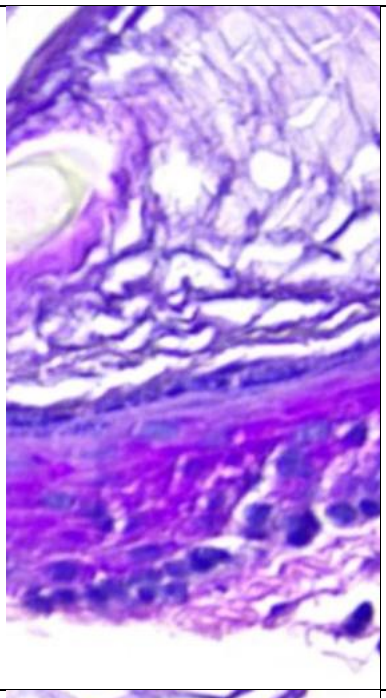
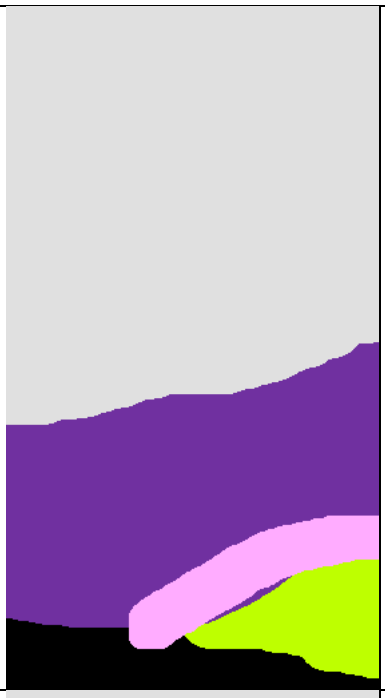
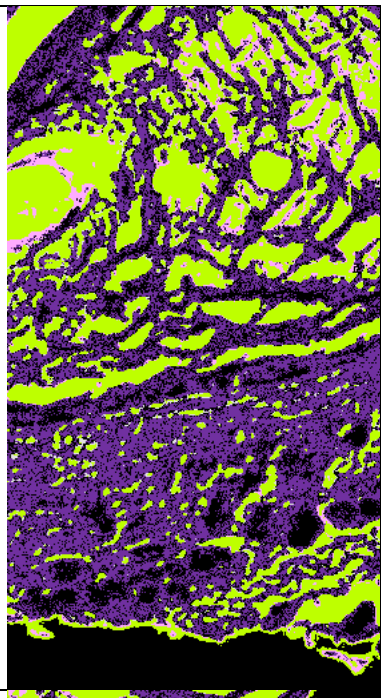
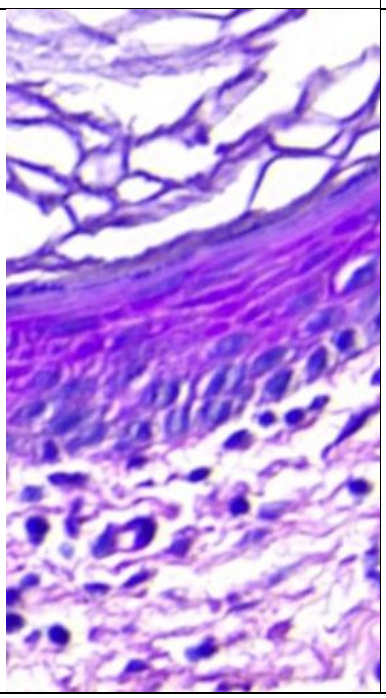
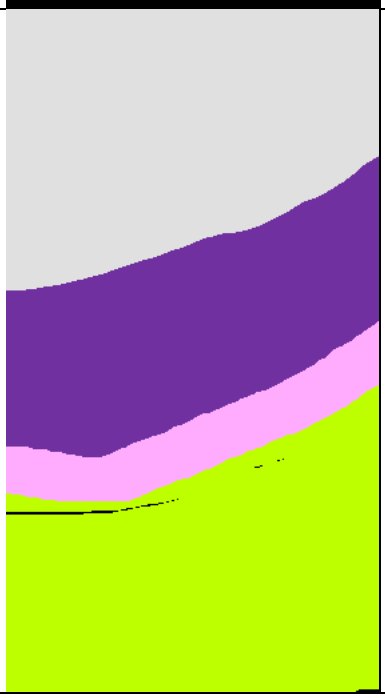
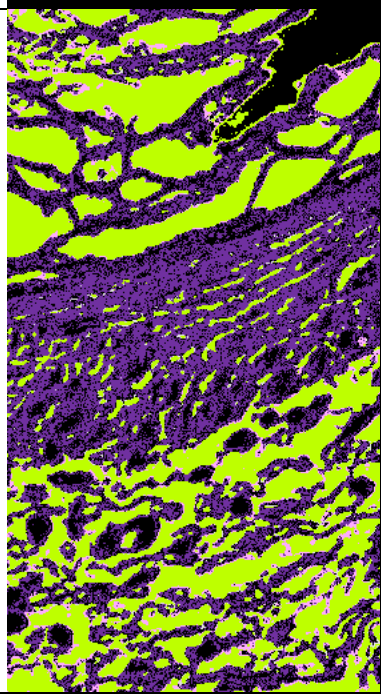
Image 3: 45.23			
Image 4: 53.42			

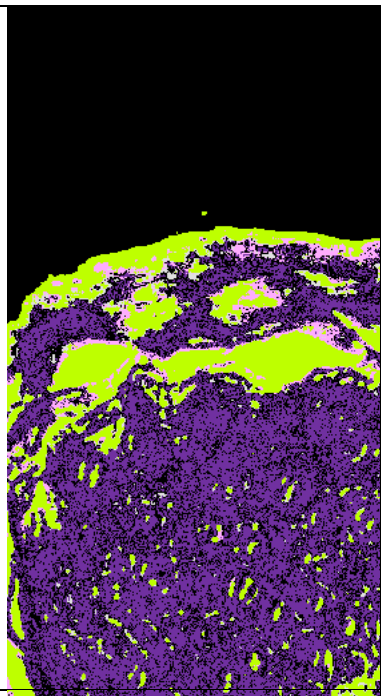
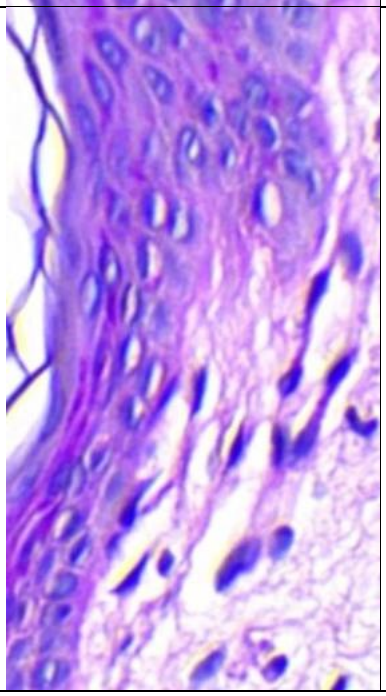
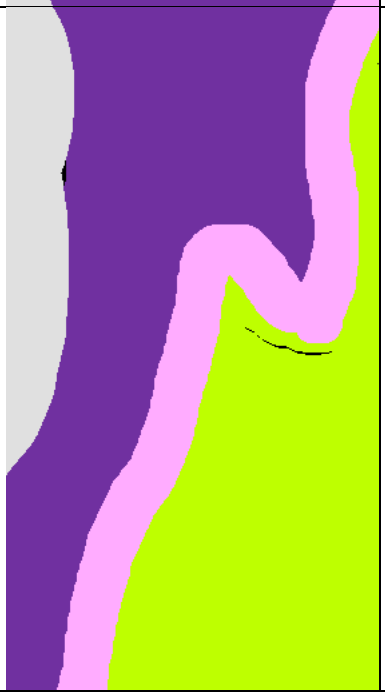
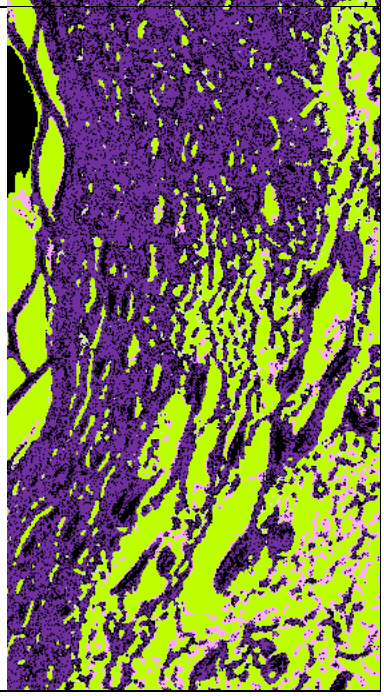
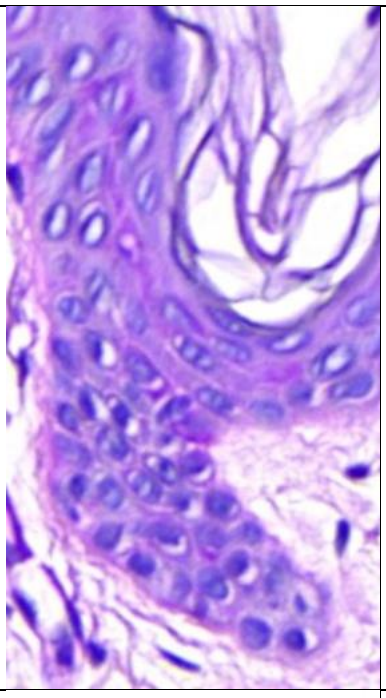
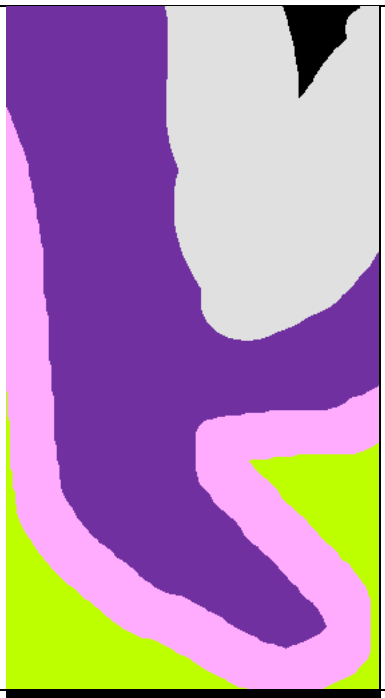
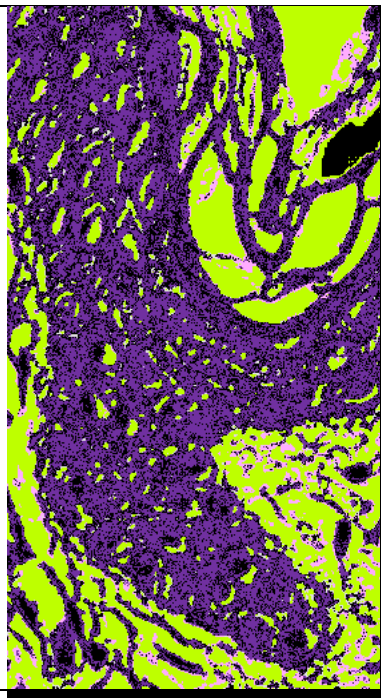
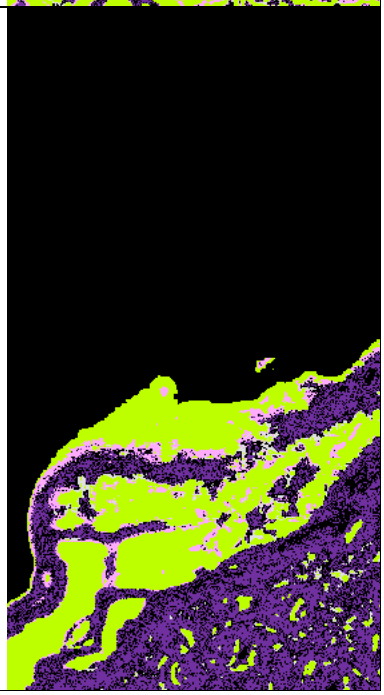
Image 5: 62.56			
Image 6: 50.43			

Image 7: 77.41			
Image 8: 54.50			

and so on….

Average Dice Coefficient: 65.91%

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate
import traceback
import os

def read_images_from_folders(tissue_folder, mask_folder):
    original_images = []
    mask_images = []
    tissue_files = os.listdir(tissue_folder)
    tissue_files.sort()
    mask_files = os.listdir(mask_folder)
    mask_files.sort()
    for tissue_file, mask_file in zip(tissue_files, mask_files):
        tissue_path = os.path.join(tissue_folder, tissue_file)
        mask_path = os.path.join(mask_folder, mask_file)
        tissue_img = cv2.imread(tissue_path, cv2.IMREAD_COLOR)
        mask_img = cv2.imread(mask_path, cv2.IMREAD_COLOR)
        original_images.append(tissue_img)
        mask_images.append(mask_img)
    return original_images, mask_images

def save_v_set(v_set, filename):
    try:
        with open(filename, 'w') as file:
            for layer, values in v_set.items():
                file.write(f"{layer}:\n")
                for value in values:
                    file.write(','.join(map(str, value)) + '\n')
                file.write('\n')
        print(f"v_set saved to {filename} successfully.")
    except Exception as e:
        print(f"Error saving v_set: {e}")

def read_v_set(filename):
    v_set = {}
    try:
        with open(filename, 'r') as file:
            lines = file.readlines()
            i = 0
            while i < len(lines):
                layer = lines[i].strip(': \n')[0:3]
                values = []
                i += 1
```

```
                while i < len(lines) and lines[i] != '\n':
                    value = list(map(int, lines[i].strip().split(',')))
                    values.append(value)
                    i += 1
                v_set[layer] = np.array(values)
                i += 1
        print(f"v_set loaded from {filename} successfully.")
        return v_set
    except Exception as e:
        print(f"Error reading v_set: {e}")
        return None

def save_image(image, path):
    try:
        directory = os.path.dirname(path)
        filename = os.path.basename(path)
        os.makedirs(directory, exist_ok=True)
        filepath = os.path.join(directory, filename)
        cv2.imwrite(filepath, image)
        print(f"Image saved successfully as '{filepath}'.")
    except Exception as e:
        print(f"Error saving image: {e}")

def write_dice_coefficients_to_file(dice_coefficients, filename):
    try:
        with open(filename, 'w') as file:
            for i, coefficient in enumerate(dice_coefficients):
                file.write(f"Image {i+1}: {coefficient}\n")
        print(f"Dice coefficients written to '{filename}' successfully.")
    except Exception as e:
        print(f"Error writing Dice coefficients to file: {e}")

def read_images_from_folder(folder_path):
    images = []
    file_names = os.listdir(folder_path)
    file_names.sort()
    for file_name in file_names:
        file_path = os.path.join(folder_path, file_name)
        image = cv2.imread(file_path, cv2.IMREAD_COLOR)
        images.append(image)
    return images

def remove_background(image):
    try:
        if image is None:
            raise ValueError("Image could not be loaded.")
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(gray_image, 250, 255, cv2.THRESH_BINARY)
```

```
        num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(thresh, connectivity=8)
        max_area_label = 1
        max_area = -1
        for label in range(1, num_labels):
            area = stats[label, cv2.CC_STAT_AREA]
            if area > max_area:
                max_area = area
                max_area_label = label
        mask = np.uint8(labels == max_area_label) * 255
        mask = cv2.bitwise_not(mask)
        result_image = cv2.bitwise_and(image, image, mask=mask)
        # cv2.imshow('Result Image', result_image)
        # cv2.waitKey(0)
        # cv2.destroyAllWindows()
        return result_image
    except Exception as e:
        print(f"Error: {e}")
        return None

    #  for code, layer_values in v_set.items():
    #         min = [0, 0, 0]
    #         max = [256, 256, 256]
    #         if code == 'DEJ':
    #             min = [239, 100, 157]
    #             max = [256, 175, 211]
    #         elif code == 'DRM':
    #             min = [197, 148, 211]
    #             max = [239, 256, 256]
    #         elif code == 'EPI':
    #             min = [220, 25, 143]
    #             max = [242, 90, 177]
    #         elif code == 'KER':
    #             min = [238, 90, 167]
    #             max = [256, 248, 233]
    #         v_set[code] = v_set[code][(v_set[code] >= min).all(axis=1) &
(v_set[code] <= max).all(axis=1)]

def refine_layer(v_set):
    try:
        for code, layer_values in v_set.items():
            most_common_values = []
            unique_values, counts = np.unique(layer_values, axis=0,
return_counts=True)
            sorted_indices = np.argsort(counts)[::-1]
            for idx in sorted_indices:
                intensity = unique_values[idx]
                if counts[idx] > 1:
```

```
                    most_common_values.append(intensity)
                if len(most_common_values) >= 50:
                    break
            v_set[code] = np.array(most_common_values)
    except Exception as e:
        print(f"Error: {e}")
    return v_set

def smooth_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, binary_mask = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY)
    kernel = np.ones((10, 10), np.uint8)
    closed_mask = cv2.morphologyEx(binary_mask, cv2.MORPH_CLOSE, kernel)
    smoothed_image = cv2.bitwise_and(image, image, mask=closed_mask)

    return smoothed_image

def display_histograms(v_set,height,name):
    plt.figure(figsize=(10, 8))
    i = 1
    for layer_code, layer_values in v_set.items():
        for j in range(3):
            plt.subplot(5, 3, i)
            plt.hist(layer_values[:, j], bins=256, range=(0, 256), color=['b', 'g',
'r'][j], alpha=0.5)
            plt.title(f"{layer_code} - {'BGR'[j]}")
            plt.ylim(0, height)
            plt.xlim(0, 256)
            i += 1

    plt.tight_layout()
    plt.savefig(name)
    plt.show()

def display_resultant_histogram(v_set, height,name):
    color_codes = {
        (255, 172, 255): 'DEJ',
        (0, 255, 190): 'DRM',
        (160, 48, 112): 'EPI',
        (224, 224, 224): 'KER',
        (0, 0, 0): 'BKG'
    }
    layer_colors = [(color[2]/255, color[1]/255, color[0]/255) for color in
color_codes.keys()]
    plt.figure(figsize=(15, 5))
    num_channels = 3
    for j in range(num_channels):
        plt.subplot(1, num_channels, j+1)
```

```
        for idx, (layer_code, layer_values) in enumerate(v_set.items()):
            color_idx = idx % len(layer_colors)
            plt.hist(layer_values[:, j], bins=256, range=(0, 256),
color=layer_colors[color_idx], alpha=0.6, label=layer_code)
        plt.title(f"{'BGR'[j]} Channel")
        plt.ylim(0, height)
        plt.xlim(0, 256)
        plt.legend()
    plt.tight_layout()
    plt.savefig(name)
    plt.show()
    return v_set

def print_table(v_set):
    table = []
    for layer, values in v_set.items():
        table.append([layer, len(values)])
    print(tabulate(table, headers=["Layer", "Unique Intensities"]))

def cca(images, masks):
    try:
        i = 0
        for image, mask in zip(images, masks):
            i += 1
            if image is None or mask is None:
                raise ValueError("One or both of the images could not be loaded.")

            color_codes = {
                (255, 172, 255): 'DEJ',
                (0, 255, 190): 'DRM',
                (160, 48, 112): 'EPI',
                (224, 224, 224): 'KER',
                (0, 0, 0): 'BKG'
            }

            v_set = {
                'DEJ': [], 'DRM': [], 'EPI': [], 'KER': []
            }

            for x in range(mask.shape[0]):
                for y in range(mask.shape[1]):
                    pixel = tuple(mask[x, y])
                    intensity = image[x, y]
                    if pixel == (0, 0, 0):
                        continue
                    elif pixel in color_codes.keys():
                        layer = color_codes[pixel]
                        v_set[layer].append(intensity)
```

```
            for layer, values in v_set.items():
                v_set[layer] = np.unique(values, axis=0)
                v_set[layer] = np.array(values)


            print('Segmented and Masked Images: ', i)

        for layer, values in v_set.items():
            v_set[layer] = np.unique(values, axis=0)
            v_set[layer] = np.array(values)

        return v_set

    except Exception as e:
        traceback.print_exc()
        return None, None, None

def build_mask(v_set, images):
    masked_images = []
    i = 0
    try:
        for image in images:
            i += 1
            image = remove_background(image)
            masked_image = np.zeros_like(image, dtype=np.uint8)
            for x in range(image.shape[0]):
                for y in range(image.shape[1]):
                    intensity = image[x, y]
                    for layer, values in v_set.items():
                        if tuple(intensity) in v_set['DRM']:
                            masked_image[x, y] = [0, 255, 190]
                        elif tuple(intensity) in v_set['DEJ']:
                            masked_image[x, y] = [255, 172, 255]
                        elif tuple(intensity) in v_set['EPI']:
                            masked_image[x, y] = [160, 48, 112]
                        elif tuple(intensity) in v_set['KER']:
                            masked_image[x, y] = [224, 224, 224]
            masked_image = smooth_image(masked_image)
            masked_images.append(masked_image)
            save_image(masked_image, f"Assignment-1/Results/Masked Image {i}.png")
            print(f"Masked Image {i} saved successfully.")
            cv2.imshow('Segmented and Masked Images',masked_image)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
        return masked_images
    except Exception as e:
```

```
        print(f"Error in build_mask: {e}")
        return None

def compute_dice_coefficient(rebuild_masks, original_masks):
    dice_coefficients = []
    for rebuild_mask, original_mask in zip(rebuild_masks, original_masks):
        true = np.sum(np.logical_and(rebuild_mask, original_mask))
        false = np.sum(np.logical_xor(rebuild_mask, original_mask))
        dice_coefficient = (2.0 * true) / (2.0 * true + false)
        dice_coefficients.append(dice_coefficient * 100)

    avg = sum(dice_coefficients) / len(dice_coefficients)
    print(f"Average Dice Coefficient: {avg:.2f}%")
    return (dice_coefficients)


train_images, train_masks = read_images_from_folders('Assignment-1/Train/Tissue/',
'Assignment-1/Train/Mask/')
test_images, test_masks = read_images_from_folders('Assignment-1/Test/Tissue/',
'Assignment-1/Test/Mask/')

train_images = [train_images[0]]
train_masks = [train_masks[0]]
test_images = [test_images[0]]
test_masks = [test_images[0]]

v_set = cca(train_images, train_masks)
save_v_set(v_set, 'v_set.txt')
v_set = read_v_set('v_set.txt')
display_histograms(v_set,400, 'individual_histogram_before_refinement.png')
display_resultant_histogram(v_set, 1000, 'histogram_before_refinement.png')
v_set = refine_layer(v_set)
display_histograms(v_set,20, 'individual_histogram_after_refinement.png')
display_resultant_histogram(v_set, 50, 'histogram_after_refinement.png')
rebuild_masks = build_mask(v_set, test_images)

print_table(v_set)


original_mask_folder = 'Assignment-1/Test/Mask/'
masked_images_folder = 'Assignment-1/Results/'

original_masks = read_images_from_folder(original_mask_folder)
masked_images = read_images_from_folder(masked_images_folder)
original_masks_binary = [(mask > 0).astype(np.uint8) for mask in original_masks]
masked_images_binary = [(mask > 0).astype(np.uint8) for mask in masked_images]
dice_coefficients = compute_dice_coefficient(masked_images_binary,
original_masks_binary)
```

```
write_dice_coefficients_to_file(dice_coefficients, 'dice_coefficients.txt')
```

Masks: