DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

Subject Name
Digital Image Processing

Assignment
1

SUBMITTED TO:
Dr. Usman Akram

SUBMITTED BY:
Student Name

1. Wahaaj Nasir

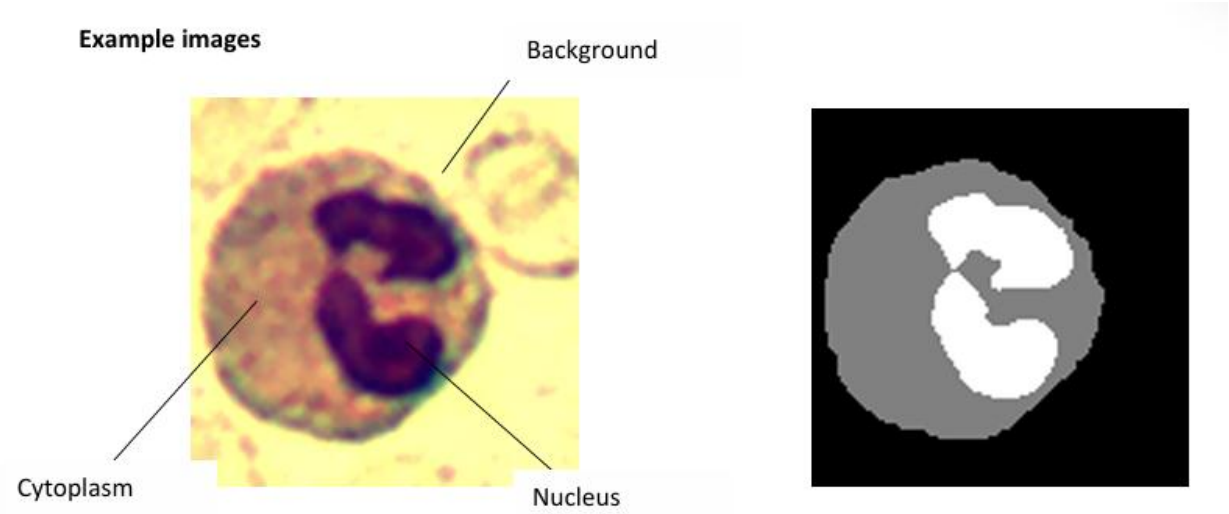
Reg#413238
DE- 44 Dept C&SE

Objectives:

Blood smear films are thin layers of blood spread on a microscope slide and stained to allow microscopic examination of blood cells. They are crucial in diagnosing various hematological disorders by analyzing the morphology and count of blood cells. White blood cell (WBC) disorders include leukemias, lymphomas, and conditions such as neutropenia and leukocytosis, which indicate infections, immune system abnormalities, or malignancies. Accurate segmentation of blood smear images aids in the diagnosis and classification of these disorders.

Automated segmentation of hematological images is essential for blood cell analysis and disease diagnosis. In this assignment, we will implement Connected Component Labeling (CCL) for segmenting different components of blood cell images. The dataset consists of microscopic images of blood smears
with corresponding masks that classify each pixel into different categories:

- Nucleus (White Mask Region)
- Cytoplasm (Gray Mask Region)
- Background (Black Mask Region)



Using this dataset, we will develop a segmentation pipeline that applies pre-processing techniques, connected component labeling, and post-processing refinement. Dataset Details The dataset consists of paired microscopic images and manually annotated ground truth masks. Each image has a corresponding labeled mask where:

- White region represents the nucleus of the WBC.
- Gray region represents the cytoplasm of the WBC.
- Black region represents the background. Each image is labeled pixel-wise, enabling precise segmentation.

Dataset Link:
https://drive.google.com/drive/folders/1DUDnYXZQF6zSZDl0RJIsdo8lqiZOJQoV?usp=sharing

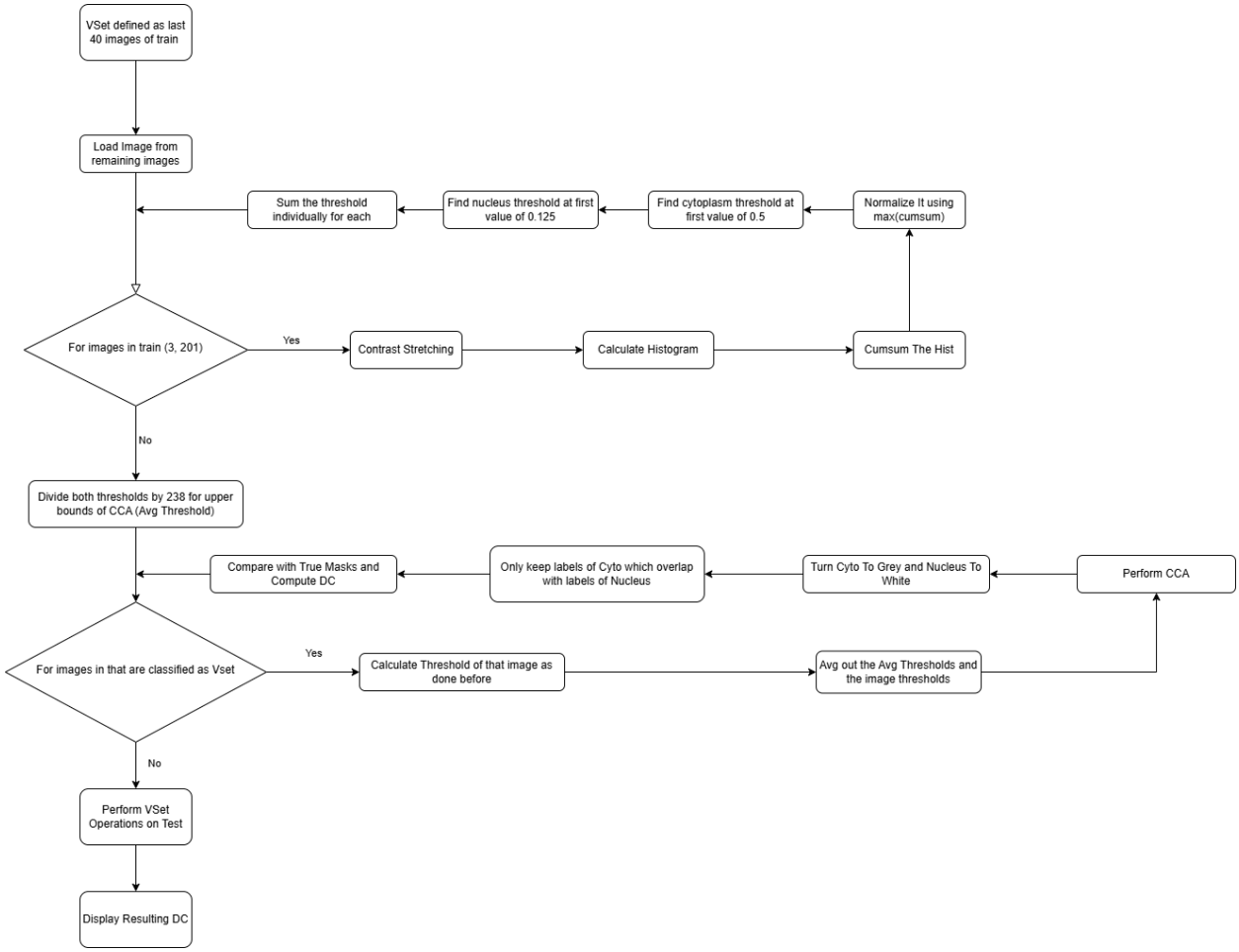
VSet:

The VSet for both Task 1 and Task 2 has been kept the same. That is:

Last 40 images of Training Set

Flow Diagram:

The following is the flow diagram that tells about my approach:



Part 1 – Calculating Avg Thresholds:

Part 1 of my assignment consists of loading all the training data images, stretch their contrast, calculate their histograms and determine the thresholds for the range of CCA.

Main Code Section:

```
total_threshold_cyto = 0
total_threshold_nucleus = 0

for i in range (3, 201):
    if i < 10:
        temp = "00" + str(i)
    elif i < 100:
        temp = "0" + str(i)
    else:
```

```
        temp = str(i)

    image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/images/" + temp + ".bmp",0)  # Grayscale image
    test_img = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/masks/" + temp + ".png", 0)
    image = contrast_stretch(image)
    histogram = histogram_creating(image)
    cumsum = hist_cumsum(histogram)

    cdf = cumsum/max(cumsum)
    thresh_cyto = (np.where(cdf >= 0.4)[0][0])
    thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

    print(f"For img {str(i)} the Thresh Cytoplasm: {thresh_cyto}")
    print(f"For img {str(i)} the Thresh Nucleus: {thresh_nucleus}")

    # ----------------------------------------------------
    # Plotting Histogram and CDF for visualization
    # plt.figure(figsize=(10, 5))
    #
    # # Histogram (PDF)
    # plt.subplot(1, 2, 1)
    # plt.bar(range(256), histogram, color='gray')
    # plt.axvline(x=thresh_cyto, color='blue', linestyle='--', label=f'Cyto
Thresh = {thresh_cyto}')
    # plt.axvline(x=thresh_nucleus, color='red', linestyle='--',
label=f'Nucleus Thresh = {thresh_nucleus}')
    # plt.title(f"Histogram for Image {temp}")
    # plt.xlabel("Pixel Intensity")
    # plt.ylabel("Frequency")
    # plt.legend()
    #
    # # CDF Plot
    # plt.subplot(1, 2, 2)
    # plt.plot(range(256), cdf, color='black')
    # plt.axhline(y=0.4, color='blue', linestyle='--', label=f'0.4 (Cyto
Thresh)')
    # plt.axhline(y=0.125, color='red', linestyle='--', label=f'0.125
(Nucleus Thresh)')
    # plt.title(f"CDF for Image {temp}")
    # plt.xlabel("Pixel Intensity")
    # plt.ylabel("CDF")
    # plt.legend()
    #
    # plt.show()
    # ---------------------------------------------

    total_threshold_cyto = total_threshold_cyto + thresh_cyto
    total_threshold_nucleus = total_threshold_nucleus + thresh_nucleus

avg_threshold_cyto = total_threshold_cyto // 238
avg_threshold_nucleus = total_threshold_nucleus // 238

print(f"Avg Thresh Cyto: {avg_threshold_cyto}")
print(f"Avg Thresh Nucleus: {avg_threshold_nucleus}")
```

Contrast Stretching:

```
def contrast_stretch(image):
    im_min_5 = np.percentile(image, 5)
    im_max_95 = np.percentile(image, 95)
    rows,cols = image.shape
    new_img = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if(image[i][j] < im_min_5):
                new_img[i][j] = 0
            elif(image[i][j] > im_max_95):
                new_img[i][j] = 255
            else:
                new_img[i][j] = 255 * ((image[i][j] - im_min_5) / (im_max_95
- im_min_5))

    return new_img
```

The contrast stretching helps to separate the cytoplasm and the nucleus from the original image.

Histogram Creation, Cumsum and CDF:

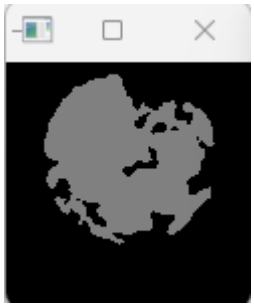
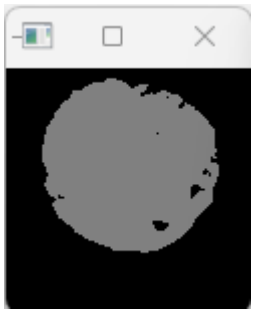
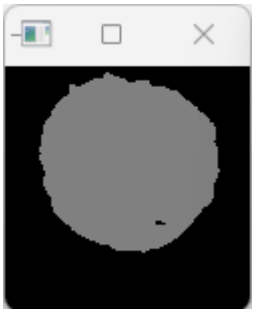
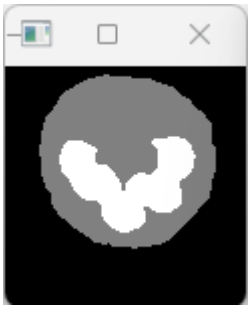
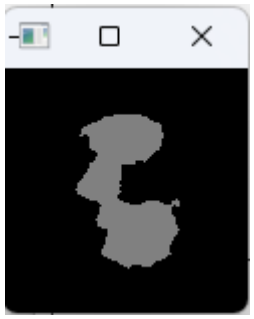
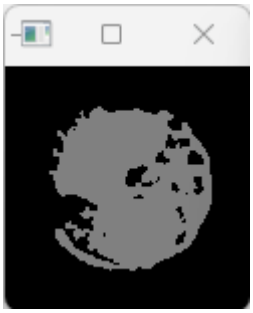
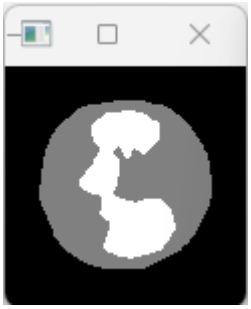
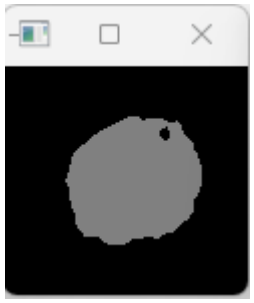
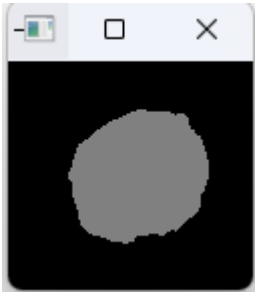
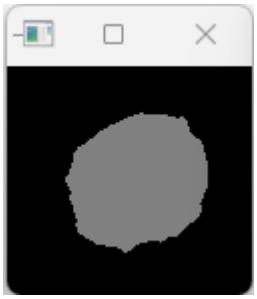
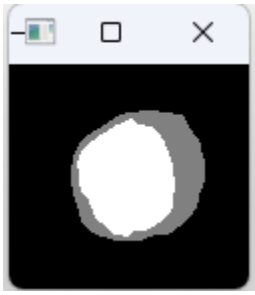
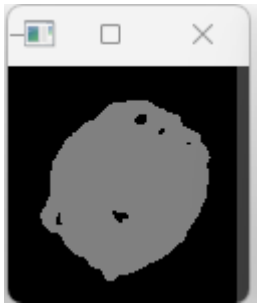
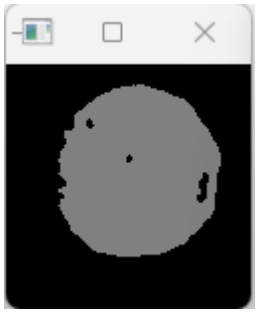
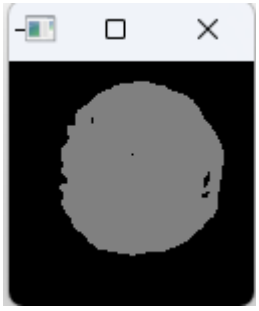
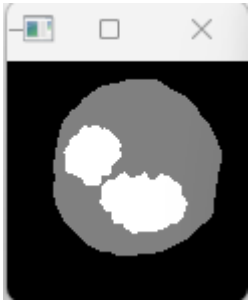
```
def histogram_creating(image):
    rows, cols = image.shape
    histogram = np.zeros(256, dtype = int)

    for i in range(rows):
        for j in range(cols):
            val = image[i][j]
            histogram[val] += 1

    return histogram

def hist_cumsum(histogram):
    cumsum = np.zeros(len(histogram), dtype = int)
    cumsum[0] = histogram[0]
    for i in range(1, len(histogram)):
        cumsum[i] = cumsum[i-1] + histogram[i]

    return cumsum

# Main Portion For Normalizing Cumsum
cdf = cumsum/max(cumsum)
```

The above mentioned code is used in the program to first compute the histogram, calculate the cumsum of the histogram and then normalize it. This normalization of the cumsum is known as the ***Cumulative Distribution Function.***

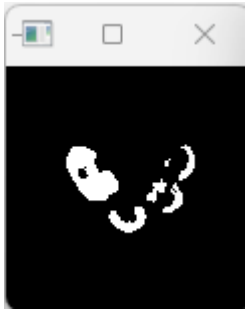
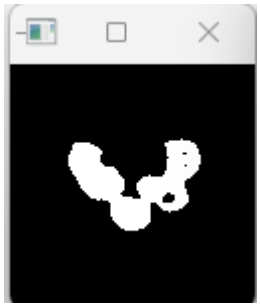
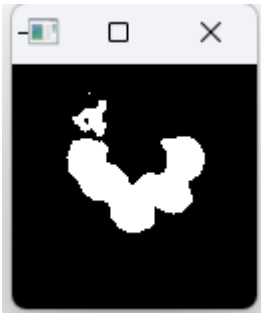
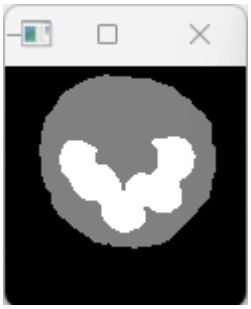
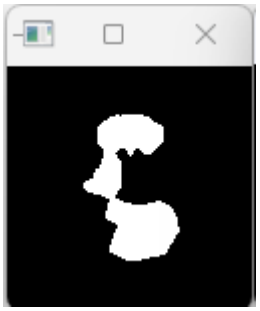
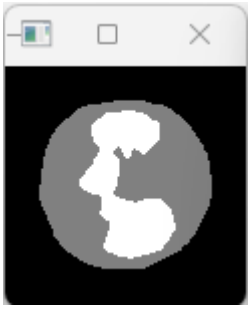
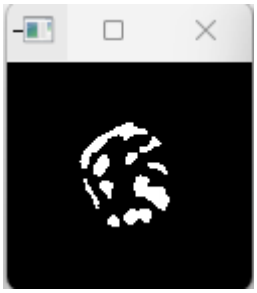
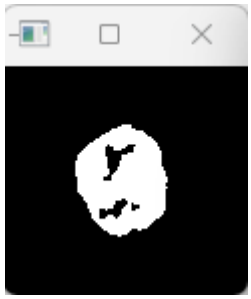
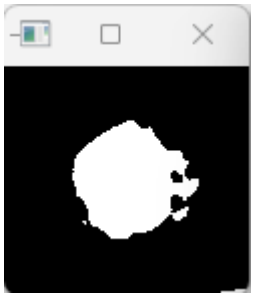
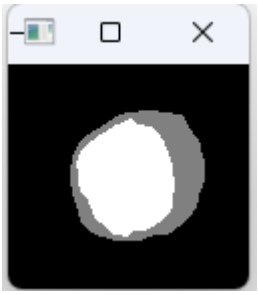
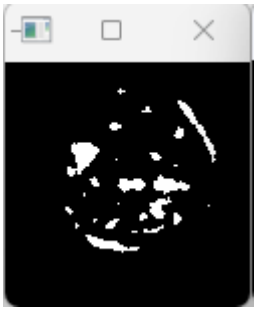
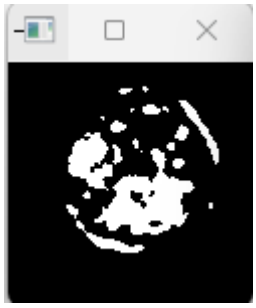
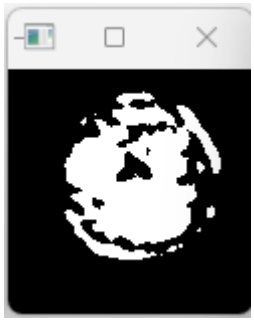
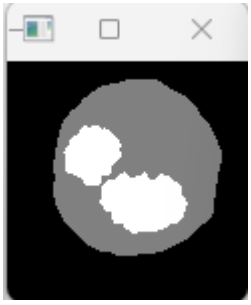
The CDF was utilized in my code as it allows me to find the first pixel in the image with a threshold of ***0.4 for the cytoplasm*** and ***0.1 for the nucleus.***

The values of 0.4 and 0.1 were determined through trial and analysis on the images. I will now show you the effect these values have on the images after inputting the bounds in CCA:

Cytoplasm:

Image	0.3	0.4	0.5	True Mask
003.bmp				
50.bmp				
114.bmp				
198.bmp				

Nucleus:

Image	0.05	0.1	0.15	True Mask
003.bmp				
50.bmp				
114.bmp				
198.bmp				

By comparing the above seen values, as well as after calculating Dice Coefficients in the end, it was determined that 0.4 and 0.1 are the values that produces the best results.

Avg Threshold Determination:

```
    total_threshold_cyto = total_threshold_cyto + thresh_cyto
    total_threshold_nucleus = total_threshold_nucleus + thresh_nucleus

avg_threshold_cyto = total_threshold_cyto // 238
avg_threshold_nucleus = total_threshold_nucleus // 238

print(f"Avg Thresh Cyto: {avg_threshold_cyto}")
print(f"Avg Thresh Nucleus: {avg_threshold_nucleus}")
```

This final part of the code helps to determine the avg threshold. This threshold is the pixel value that is given as the upper bound for the CCA function.

Part 2 – Applying With CCA On VSet:

Part 2 involves using the VSet and applying CCA on it for separating the components. This is then compared with true masks to calculate the dice coefficient.

Main Code Section:

```
print("\n---------------------------------CHECKING DICE COEFFICIENT ON VSET--
----------------------------------------")
total_dc_black = 0
total_dc_cyto = 0
total_dc_nucleus = 0
for i in range (201, 241):
    if i < 10:
        temp = "00" + str(i)
    elif i < 100:
        temp = "0" + str(i)
    else:
        temp = str(i)

    image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/images/" + temp + ".bmp",0)  # Grayscale image
    test_img = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/masks/" + temp + ".png", 0)
    image = contrast_stretch(image)
    histogram = histogram_creating(image)
    cumsum = hist_cumsum(histogram)

    cdf = cumsum / max(cumsum)
    thresh_cyto = (np.where(cdf >= 0.4)[0][0])
    thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

    #Taking avg of current and avg thresholds
    avg_threshold_cyto_new = (avg_threshold_cyto + thresh_cyto)//2
    avg_threshold_nucleus_new = (avg_threshold_nucleus + thresh_nucleus) // 2

    image_padded = padding(1, image)
    img_cc_cyto,img_cc_cyto_dict = cc(image_padded, 0, avg_threshold_cyto)
    img_cc_nucleus, img_cc_nucleus_dict = cc(image_padded, 0,
avg_threshold_nucleus)
```

```
    image_cc_cyto = remove_padding(img_cc_cyto, 1)
    image_cc_nucleus = remove_padding(img_cc_nucleus, 1)

    image_cc_cyto, image_cc_nucleus, img_cc_cyto_dict, img_cc_nucleus_dict =
overlapping_labels(image_cc_cyto, image_cc_nucleus)
    image_cc_cyto = cyto_to_gray(img_cc_cyto, img_cc_cyto_dict)
    image_cc_nucleus = nuclei_to_white(img_cc_nucleus, img_cc_nucleus_dict)

    own_mask = merge_for_mask(image_cc_cyto, image_cc_nucleus)

    # cv.imshow("Original", image)
    # cv.imshow("Mask", test_img)
    # cv.imshow("Cyto", image_cc_cyto)
    # cv.imshow("Nucleus", image_cc_nucleus)
    # cv.imshow("Own Mask", own_mask)
    # cv.waitKey()

    dc_black = calculate_dice_coefficient(test_img, own_mask, 0)
    dc_cyto = calculate_dice_coefficient(test_img, own_mask, 128)
    dc_nucleus = calculate_dice_coefficient(test_img, own_mask, 255)

    print(f"\nDCs for img {str(i)}: ")
    print(f"DC for Black: {dc_black}")
    print(f"DC for Cytoplasm: {dc_cyto}")
    print(f"DC for Nucleus: {dc_nucleus}")

    total_dc_black = total_dc_black + dc_black
    total_dc_cyto = total_dc_cyto + dc_cyto
    total_dc_nucleus = total_dc_nucleus + dc_nucleus

avg_dc_black = total_dc_black / 40
avg_dc_cyto = total_dc_cyto / 40
avg_dc_nucleus = total_dc_nucleus / 40

print("\n------AVG DCs FOR VSET------")
print(f"Avg DC Black: {avg_dc_black}")
print(f"Avg DC Cyto: {avg_dc_cyto}")
print(f"Avg DC Nucleus: {avg_dc_nucleus}")
```

Averaging with Prev Calculated Threshold:

In the previous section, we calculated the average pixel values for both Nucleus and Cytoplasm. In this section, I similarly calculate those values for the currently loaded image. I then proceed to average out those values with the previously determined pixel values, as show here:

```
histogram = histogram_creating(image)
cumsum = hist_cumsum(histogram)

cdf = cumsum / max(cumsum)
thresh_cyto = (np.where(cdf >= 0.5)[0][0])
thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

#Taking avg of current and avg thresholds
avg_threshold_cyto_new = (avg_threshold_cyto + thresh_cyto)//2
avg_threshold_nucleus_new = (avg_threshold_nucleus + thresh_nucleus) // 2
```

Applying CCA – with 8 Connectivity:

The next step is performing CCA on our image. For CCA, we need to send a padded image, which is done by the function "***padding***". After that, we send the padded image to the CCA function, which returns our labeled image as well as our dictionary. The CCA function is as follows:

```
def cc(orig, lower_bound, upper_bound):
    rows, cols = orig.shape
    new_img = np.zeros((rows, cols), dtype=np.uint8)
    my_dict = {}
    count = 1

    for i in range(1, rows):
        for j in range(1, cols):
            if ((orig[i][j] >= lower_bound) & (orig[i][j] <= upper_bound)) :
                neighbors = []  # Store nonzero neighboring labels

                # Check all 8-connected neighbors
                if ((orig[i - 1][j] >= lower_bound) & (orig[i - 1][j] <=
upper_bound)):
                    neighbors.append(new_img[i - 1][j])
                if ((orig[i][j-1] >= lower_bound) & (orig[i][j-1] <=
upper_bound)):
                    neighbors.append(new_img[i][j - 1])
                if ((orig[i-1][j-1] >= lower_bound) & (orig[i-1][j-1] <=
upper_bound)):
                    neighbors.append(new_img[i - 1][j - 1])
                if ((j + 1 < cols) and (lower_bound <= orig[i - 1][j + 1] <=
upper_bound)):
                    neighbors.append(new_img[i - 1][j + 1])

                if not neighbors:  # No connected neighbors, assign new label
                    new_img[i][j] = count
                    my_dict[count] = count
                    count += 1
                else:
                    min_label = min(neighbors)
                    new_img[i][j] = min_label

                    # Merge equivalence classes
                    for label in neighbors:
                        root1 = find_root(my_dict, min_label)
                        root2 = find_root(my_dict, label)
                        if root1 != root2:
                            my_dict[max(root1, root2)] = min(root1, root2)

    for i in range(1, rows):
        for j in range(1, cols):
            if new_img[i][j] > 0:
                new_img[i][j] = find_root(my_dict, new_img[i][j])

    return new_img, my_dict


# Path compression to find root label
```

```
def find_root(my_dict, x):
    #Added to avoid that the background coming in the dictionaries
    if x == 0:
        return 0
    if x not in my_dict:
        my_dict[x] = x
        return x
    while my_dict[x] != x:
        my_dict[x] = my_dict[my_dict[x]]  # Path compression
        x = my_dict[x]
    return x
```

The above mentioned CCA function works in the following manner:

- Step 1: The loop checks if the current pixel lies between the bounds of gray levels (determined previously by CDF)
- Step 2: If it does, it checks all the neighbors. These neighbors include the left, top-left, top, and top-right of the current pixel.
- Step 3: If the neighbors array is empty, it means this is a new label. Thus, it gets assigned a new value
- Step 3.5: In case the neighbor includes an already defined label, we assign the lowest label to the current label. We update our equivalency list, using the find_root function. The function loops through every value in the dictionary and updates it accordingly, so that no extra labels remain. For example, if 1->1, 2->2 and 3->3, but in a further iteration we find that 2 and 1 are actually connected and 2 and 3 are connected as well, we need to go through the equivalency list so that 1->1, 2->1 and 3->1. The while loop in the find_root function makes sure that this happens.
- Step 4: We loop through the entire image one last time, making sure that all labels have been correctly updated

Removing Padding, Converting Cytoplasm to Gray, Nucleus to White and Further Optimization:

Now we focus on this part of the main:

```
image_cc_cyto = remove_padding(img_cc_cyto, 1)
image_cc_nucleus = remove_padding(img_cc_nucleus, 1)

image_cc_cyto, image_cc_nucleus, img_cc_cyto_dict, img_cc_nucleus_dict =
overlapping_labels(image_cc_cyto, image_cc_nucleus)
image_cc_cyto = cyto_to_gray(img_cc_cyto, img_cc_cyto_dict)
image_cc_nucleus = nuclei_to_white(img_cc_nucleus, img_cc_nucleus_dict)

own_mask = merge_for_mask(image_cc_cyto, image_cc_nucleus)
```

Firstly, we remove the extra padding that we had added for CCA. We do this, as leaving this in will cause a mismatch in image size, leading to incorrect DC calculations.

Now, we are aware that a nucleus can only be located inside a cytoplasm. Thus, a function has been written that compares the labels for cytoplasm and nucleus from their images. If a nucleus label is found that is not inside a cytoplasm label, we know that this is a false positive, and thus, remove that label

```
def overlapping_labels(cyto_img, nucleus_img):
    rows, cols = cyto_img.shape
    new_cyto = np.zeros((rows, cols), dtype = np.uint8)
    new_nucleus = np.zeros((rows, cols), dtype = np.uint8)

    cyto_labels_keep = set()
    nucleus_labels_keep = set()

    for i in range(rows):
        for j in range(cols):

            #Check for overlapping areas
            if((cyto_img[i][j] > 0) & (nucleus_img[i][j] > 0)):
                cyto_labels_keep.add(cyto_img[i][j])
                nucleus_labels_keep.add(nucleus_img[i][j])

    for i in range(rows):
        for j in range(cols):

            if (cyto_img[i][j] in cyto_labels_keep):
                new_cyto[i][j] = cyto_img[i][j]

            if(nucleus_img[i][j] in nucleus_labels_keep):
                new_nucleus[i][j] = nucleus_img[i][j]

    return new_cyto, new_nucleus, cyto_labels_keep, nucleus_labels_keep
```

After this, we simply convert the cytoplasm labels to all be gray (128) and nucleus to be white (255).

```
def cyto_to_gray(cyto_img, cyto_dict):
    rows, cols = cyto_img.shape
    new_img = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if cyto_img[i][j] in cyto_dict:
                new_img[i][j] = 128

    return new_img

def nuclei_to_white(nucleus_img, nucleus_dict):
    rows, cols = nucleus_img.shape
    new_img = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if nucleus_img[i][j] in nucleus_dict:
                new_img[i][j] = 255

    return new_img
```

For comparison, we need to merge the two images, cytoplasm and nuclei, into a single image. This is done using the following function:

```
def merge_for_mask(cyto_img, nuclei_img):
    rows, cols = cyto_img.shape
    mask = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if((cyto_img[i][j] == 128) & (nuclei_img[i][j] != 255)):
                mask[i][j] = 128
            elif((cyto_img[i][j] == 128) & (nuclei_img[i][j] == 255)):
                mask[i][j] = 255
            elif((cyto_img[i][j] != 128) & (nuclei_img[i][j] == 255)):
                mask[i][j] = 0 #Reduce false positives of nucleus as nuclei
should only be inside cyto

    return mask
```

Another check has been applied here for checking nucleus and cytoplasm values, that ensures that the no nuclei occur outside the cytoplasm. This is incase our labels miss something.

Dice Coefficient Calculation:

Now we calculate the Dice Coefficient for the image, which is determined by the following formula:

$$D.C = \frac{2 * (X \cap Y)}{X + Y}$$

Where X is the no. of our predicted pixels, Y is the actual no. of pixels of the corresponding image that exist and X ∩ Y is the no. of true positive pixels (correct predicitons).

We calculate the dice coefficient for all 3 labels (background, cytoplasm and nucleus), using the following function:

```
#D.C = 2 * (X ∩ Y) / X  + Y
#X is Predicted Pixels
#Y is Actual Pixels
#X ∩ Y is true Positives
def calculate_dice_coefficient(true_mask, own_mask, label):
    rows, cols = true_mask.shape
    X = 0
    Y = 0
    TP = 0

    for i in range(rows):
        for j in range(cols):
            if(own_mask[i][j] == label):
                X += 1

    for i in range(rows):
        for j in range(cols):
            if (true_mask[i][j] == label):
                Y += 1

    for i in range(rows):
        for j in range(cols):
```

```
            if ((own_mask[i][j] == label) & (true_mask[i][j] == label)):
                TP += 1

    DC = (2 * TP) / (X+Y)

    return DC
```

Part 3 – Testing on Actual Test Set:

The calculations and methods performed on the VSet are then all performed on the actual test set, which consists of 60 images. These 60 images give us the 3 more DCs about how well the algorithm works.

Main Code:

```
print("\n--------------------------------CHECKING DICE COEFFICIENT ON TEST--
----------------------------------------")
total_dc_black = 0
total_dc_cyto = 0
total_dc_nucleus = 0
for i in range (241, 301):
    if i < 10:
        temp = "00" + str(i)
    elif i < 100:
        temp = "0" + str(i)
    else:
        temp = str(i)

    image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/test/images/" + temp + ".bmp",0)  # Grayscale image
    test_img = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/test/masks/" + temp + ".png", 0)
    image = contrast_stretch(image)
    histogram = histogram_creating(image)
    cumsum = hist_cumsum(histogram)

    cdf = cumsum / max(cumsum)
    thresh_cyto = (np.where(cdf >= 0.4)[0][0])
    thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

    #Taking avg of current and avg thresholds
    avg_threshold_cyto_new = (avg_threshold_cyto + thresh_cyto)//2
    avg_threshold_nucleus_new = (avg_threshold_nucleus + thresh_nucleus) // 2

    image_padded = padding(1, image)
    img_cc_cyto,img_cc_cyto_dict = cc(image_padded, 0,
avg_threshold_cyto_new)
    img_cc_nucleus, img_cc_nucleus_dict = cc(image_padded, 0,
avg_threshold_nucleus_new)
    image_cc_cyto = remove_padding(img_cc_cyto, 1)
    image_cc_nucleus = remove_padding(img_cc_nucleus, 1)

    image_cc_cyto, image_cc_nucleus, img_cc_cyto_dict, img_cc_nucleus_dict =
overlapping_labels(image_cc_cyto, image_cc_nucleus)
    image_cc_cyto = cyto_to_gray(img_cc_cyto, img_cc_cyto_dict)
```

```
    image_cc_nucleus = nuclei_to_white(img_cc_nucleus, img_cc_nucleus_dict)

    own_mask = merge_for_mask(image_cc_cyto, image_cc_nucleus)

    # cv.imshow("Original", image)
    # cv.imshow("Mask", test_img)
    # cv.imshow("Cyto", image_cc_cyto)
    # cv.imshow("Nucleus", image_cc_nucleus)
    # cv.imshow("Own Mask", own_mask)
    # cv.waitKey()

    dc_black = calculate_dice_coefficient(test_img, own_mask, 0)
    dc_cyto = calculate_dice_coefficient(test_img, own_mask, 128)
    dc_nucleus = calculate_dice_coefficient(test_img, own_mask, 255)

    print(f"\nDCs for img {str(i)}: ")
    print(f"DC for Black: {dc_black}")
    print(f"DC for Cytoplasm: {dc_cyto}")
    print(f"DC for Nucleus: {dc_nucleus}")

    total_dc_black = total_dc_black + dc_black
    total_dc_cyto = total_dc_cyto + dc_cyto
    total_dc_nucleus = total_dc_nucleus + dc_nucleus

avg_dc_black = total_dc_black / 60
avg_dc_cyto = total_dc_cyto / 60
avg_dc_nucleus = total_dc_nucleus / 60

print("\n------AVG DCs FOR TEST------")
print(f"Avg DC Black: {avg_dc_black}")
print(f"Avg DC Cyto: {avg_dc_cyto}")
print(f"Avg DC Nucleus: {avg_dc_nucleus}")
```

Part 4 – Testing With Different Values:

After the entire pipeline is completed, we check different values of CDF to see which give us the best avg DC when applied to our dataset. The following are the results of our experimentation:

Cyto CDF Thresh	**Nucleus CDF Threshold**			
	0.05 **(Avg BG)** **(Avg Cyto)** **(Avg Nuclei)**	**0.1** **(Avg BG)** **(Avg Cyto)** **(Avg Nuclei)**	**0.15** **(Avg BG)** **(Avg Cyto)** **(Avg Nuclei)**	**0.2** **(Avg BG)** **(Avg Cyto)** **(Avg Nuclei)**
0.4	0.98 0.83 0.71	**0.98** **0.88** **0.91**	0.98 0.79 0.81	0.98 0.56 0.66
0.45	0.98 0.82	0.98 0.87	0.98 0.78	0.97 0.55

	0.71	0.91	0.81	0.66
0.5	0.97 0.81 0.71	0.97 0.86 0.91	0.97 0.76 0.81	0.97 0.54 0.66
0.55	0.96 0.80 0.71	0.96 0.84 0.91	0.96 0.75 0.81	0.96 0.53 0.66
0.6	0.95 0.77 0.71	0.95 0.82 0.91	0.95 0.72 0.81	0.95 0.51 0.66
0.65	0.94 0.77 0.71	0.94 0.81 0.91	0.94 0.71 0.81	0.94 0.51 0.66

(Further calculations omitted as clear downward trend was noticeable)

The best result on VSet was thus determined to be from ***0.4 and 0.1***

When running on the test images, we receive the following results:

------AVG DCs FOR TEST------

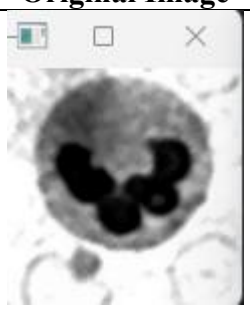
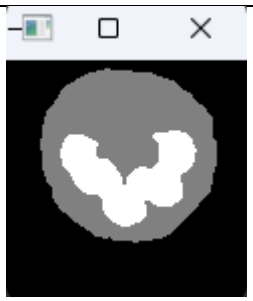
Avg DC Black: 0.9851816758716043

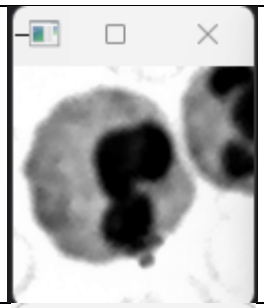
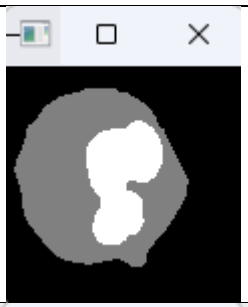
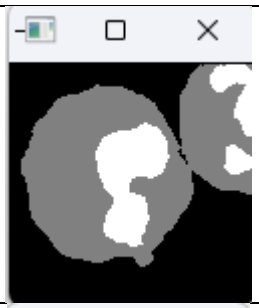
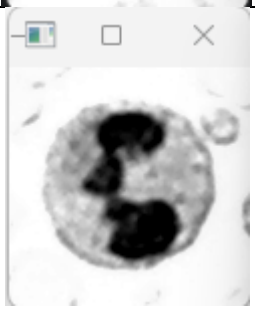
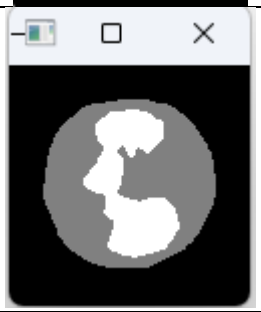
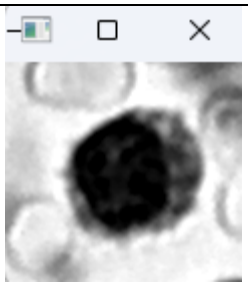
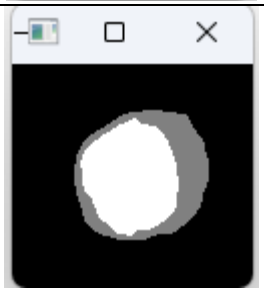
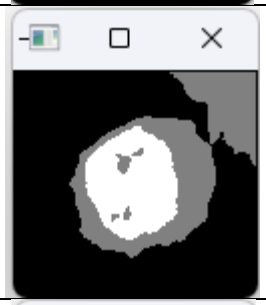
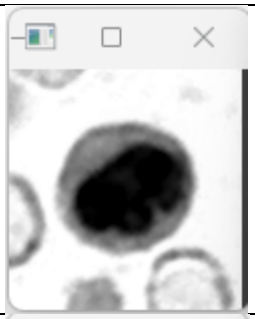
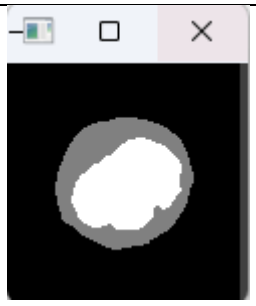
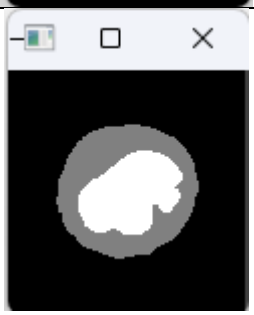
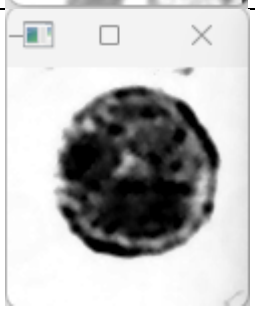
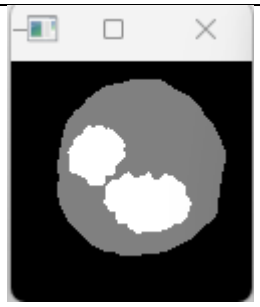
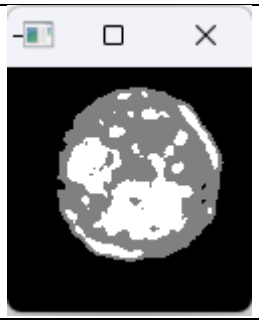
Avg DC Cyto: 0.8899387637305892

Avg DC Nucleus: 0.8947152018533242

Final Images:

The following are a set of images from the training set, their actual masks and our created ones:

Image	Original Image	Actual Mask	Our Mask
003.bmp			

005.bmp			
050.bmp			
114.bmp			
131.bmp			
198.bmp			

Complete Editable Code:

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

def padding(pad, orig):
    rows, cols = orig.shape
    padded_arr = np.ones((rows+ 2 * pad, cols+ 2 * pad), dtype =
np.uint8)*255

    for i in range(rows):
        for j in range(cols):
            padded_arr[i+pad][j+pad] = orig[i][j]

    return padded_arr

def remove_padding(padded_img, pad):
    rows, cols = padded_img.shape
    return padded_img[pad:rows-pad, pad:cols-pad]

# def lower_by_x(image, thresh):
#     rows, cols = image.shape
#     new_image = np.ones((rows, cols), dtype=np.uint8)
#
#     for i in range(rows):
#         for j in range(cols):
#             if (image[i, j] >= 0 and image[i, j] <= thresh):
#                 new_image[i, j] = 0
#             elif (image[i, j] >= thresh+1 and image[i, j] <= 255):
#                 new_image[i, j] = 255
#
#     return new_image
#
# def lower_by_2(image):
#     rows, cols = image.shape
#     new_image = np.ones((rows, cols), dtype=np.uint8)
#
#     for i in range(rows):
#         for j in range(cols):
#             if (image[i, j] >= 0 and image[i, j] <= 127):
#                 new_image[i, j] = 0
#             elif (image[i, j] >= 128 and image[i, j] <= 255):
#                 new_image[i, j] = 255
#
#     return new_image

def cc(orig, lower_bound, upper_bound):
    rows, cols = orig.shape
    new_img = np.zeros((rows, cols), dtype=np.uint8)
    my_dict = {}
    count = 1

    for i in range(1, rows):
        for j in range(1, cols):
            if ((orig[i][j] >= lower_bound) & (orig[i][j] <= upper_bound)) :
```

```
                neighbors = []  # Store nonzero neighboring labels

                # Check all 8-connected neighbors
                if ((orig[i - 1][j] >= lower_bound) & (orig[i - 1][j] <=
upper_bound)):
                    neighbors.append(new_img[i - 1][j])
                if ((orig[i][j-1] >= lower_bound) & (orig[i][j-1] <=
upper_bound)):
                    neighbors.append(new_img[i][j - 1])
                if ((orig[i-1][j-1] >= lower_bound) & (orig[i-1][j-1] <=
upper_bound)):
                    neighbors.append(new_img[i - 1][j - 1])
                if ((j + 1 < cols) and (lower_bound <= orig[i - 1][j + 1] <=
upper_bound)):
                    neighbors.append(new_img[i - 1][j + 1])

                if not neighbors:  # No connected neighbors, assign new label
                    new_img[i][j] = count
                    my_dict[count] = count
                    count += 1
                else:
                    min_label = min(neighbors)
                    new_img[i][j] = min_label

                    # Merge equivalence classes
                    for label in neighbors:
                        root1 = find_root(my_dict, min_label)
                        root2 = find_root(my_dict, label)
                        if root1 != root2:
                            my_dict[max(root1, root2)] = min(root1, root2)

    for i in range(1, rows):
        for j in range(1, cols):
            if new_img[i][j] > 0:
                new_img[i][j] = find_root(my_dict, new_img[i][j])

    return new_img, my_dict


# Path compression to find root label
def find_root(my_dict, x):
    #Added to avoid that the background coming in the dictionaries
    if x == 0:
        return 0
    if x not in my_dict:
        my_dict[x] = x
        return x
    while my_dict[x] != x:
        my_dict[x] = my_dict[my_dict[x]]  # Path compression
        x = my_dict[x]
    return x

def histogram_creating(image):
    rows, cols = image.shape
    histogram = np.zeros(256, dtype = int)

    for i in range(rows):
```

```
        for j in range(cols):
            val = image[i][j]
            histogram[val] += 1

    return histogram

def hist_cumsum(histogram):
    cumsum = np.zeros(len(histogram), dtype = int)
    cumsum[0] = histogram[0]
    for i in range(1, len(histogram)):
        cumsum[i] = cumsum[i-1] + histogram[i]

    return cumsum

def cyto_to_gray(cyto_img, cyto_dict):
    rows, cols = cyto_img.shape
    new_img = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if cyto_img[i][j] in cyto_dict:
                new_img[i][j] = 128

    return new_img

def nuclei_to_white(nucleus_img, nucleus_dict):
    rows, cols = nucleus_img.shape
    new_img = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if nucleus_img[i][j] in nucleus_dict:
                new_img[i][j] = 255

    return new_img

def overlapping_labels(cyto_img, nucleus_img):
    rows, cols = cyto_img.shape
    new_cyto = np.zeros((rows, cols), dtype = np.uint8)
    new_nucleus = np.zeros((rows, cols), dtype = np.uint8)

    cyto_labels_keep = set()
    nucleus_labels_keep = set()

    for i in range(rows):
        for j in range(cols):

            #Check for overlapping areas
            if((cyto_img[i][j] > 0) & (nucleus_img[i][j] > 0)):
                cyto_labels_keep.add(cyto_img[i][j])
                nucleus_labels_keep.add(nucleus_img[i][j])

    for i in range(rows):
        for j in range(cols):

            if (cyto_img[i][j] in cyto_labels_keep):
                new_cyto[i][j] = cyto_img[i][j]
```

```
            if(nucleus_img[i][j] in nucleus_labels_keep):
                new_nucleus[i][j] = nucleus_img[i][j]

    return new_cyto, new_nucleus, cyto_labels_keep, nucleus_labels_keep

def merge_for_mask(cyto_img, nuclei_img):
    rows, cols = cyto_img.shape
    mask = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if((cyto_img[i][j] == 128) & (nuclei_img[i][j] != 255)):
                mask[i][j] = 128
            elif((cyto_img[i][j] == 128) & (nuclei_img[i][j] == 255)):
                mask[i][j] = 255
            elif((cyto_img[i][j] != 128) & (nuclei_img[i][j] == 255)):
                mask[i][j] = 0 #Reduce false positives of nucleus as nuclei
should only be inside cyto

    return mask

#D.C = 2 * (X ∩ Y) / X  + Y
#X is Predicted Pixels
#Y is Actual Pixels
#X ∩ Y is true Positives
def calculate_dice_coefficient(true_mask, own_mask, label):
    rows, cols = true_mask.shape
    X = 0
    Y = 0
    TP = 0

    for i in range(rows):
        for j in range(cols):
            if(own_mask[i][j] == label):
                X += 1

    for i in range(rows):
        for j in range(cols):
            if (true_mask[i][j] == label):
                Y += 1

    for i in range(rows):
        for j in range(cols):
            if ((own_mask[i][j] == label) & (true_mask[i][j] == label)):
                TP += 1

    DC = (2 * TP) / (X+Y)

    return DC

#Purely for checking purposes
def neg_img(image):
    l = 256
    rows, cols = image.shape
    new_img = np.zeros((rows, cols), dtype = np.uint8)
    for i in range(rows):
```

```
        for j in range(cols):
            r = int(image[i][j])
            s = (256-1)-r
            new_img[i][j] = np.uint8(s)

    return new_img

def contrast_stretch(image):
    im_min_5 = np.percentile(image, 5)
    im_max_95 = np.percentile(image, 95)
    rows,cols = image.shape
    new_img = np.zeros((rows, cols), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):
            if(image[i][j] < im_min_5):
                new_img[i][j] = 0
            elif(image[i][j] > im_max_95):
                new_img[i][j] = 255
            else:
                new_img[i][j] = 255 * ((image[i][j] - im_min_5) / (im_max_95
- im_min_5))

    return new_img


#Main
total_threshold_cyto = 0
total_threshold_nucleus = 0

for i in range (3, 201):
    if i < 10:
        temp = "00" + str(i)
    elif i < 100:
        temp = "0" + str(i)
    else:
        temp = str(i)

    image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/images/" + temp + ".bmp",0)  # Grayscale image
    test_img = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/masks/" + temp + ".png", 0)
    image = contrast_stretch(image)
    histogram = histogram_creating(image)
    cumsum = hist_cumsum(histogram)

    cdf = cumsum/max(cumsum)
    thresh_cyto = (np.where(cdf >= 0.4)[0][0])
    thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

    print(f"For img {str(i)} the Thresh Cytoplasm: {thresh_cyto}")
    print(f"For img {str(i)} the Thresh Nucleus: {thresh_nucleus}")

    # ----------------------------------------------------
    # Plotting Histogram and CDF for visualization
    # plt.figure(figsize=(10, 5))
    #
```

```
    # # Histogram (PDF)
    # plt.subplot(1, 2, 1)
    # plt.bar(range(256), histogram, color='gray')
    # plt.axvline(x=thresh_cyto, color='blue', linestyle='--', label=f'Cyto
Thresh = {thresh_cyto}')
    # plt.axvline(x=thresh_nucleus, color='red', linestyle='--',
label=f'Nucleus Thresh = {thresh_nucleus}')
    # plt.title(f"Histogram for Image {temp}")
    # plt.xlabel("Pixel Intensity")
    # plt.ylabel("Frequency")
    # plt.legend()
    #
    # # CDF Plot
    # plt.subplot(1, 2, 2)
    # plt.plot(range(256), cdf, color='black')
    # plt.axhline(y=0.4, color='blue', linestyle='--', label=f'0.4 (Cyto
Thresh)')
    # plt.axhline(y=0.125, color='red', linestyle='--', label=f'0.125
(Nucleus Thresh)')
    # plt.title(f"CDF for Image {temp}")
    # plt.xlabel("Pixel Intensity")
    # plt.ylabel("CDF")
    # plt.legend()
    #
    # plt.show()
    # ---------------------------------------------

    total_threshold_cyto = total_threshold_cyto + thresh_cyto
    total_threshold_nucleus = total_threshold_nucleus + thresh_nucleus

avg_threshold_cyto = total_threshold_cyto // 198
avg_threshold_nucleus = total_threshold_nucleus // 198

print(f"Avg Thresh Cyto: {avg_threshold_cyto}")
print(f"Avg Thresh Nucleus: {avg_threshold_nucleus}")

image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/images/198.bmp",0)  # Grayscale image
test_img = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/masks/198.png", 0)
image = contrast_stretch(image)
histogram = histogram_creating(image)
cumsum = hist_cumsum(histogram)

cdf = cumsum / max(cumsum)
thresh_cyto = (np.where(cdf >= 0.4)[0][0])
thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

#Taking avg of current and avg thresholds
avg_threshold_cyto_new = (avg_threshold_cyto + thresh_cyto)//2
avg_threshold_nucleus_new = (avg_threshold_nucleus + thresh_nucleus) // 2

image_padded = padding(1, image)
img_cc_cyto,img_cc_cyto_dict = cc(image_padded, 0, avg_threshold_cyto)
img_cc_nucleus, img_cc_nucleus_dict = cc(image_padded, 0,
avg_threshold_nucleus)
image_cc_cyto = remove_padding(img_cc_cyto, 1)
```

```
image_cc_nucleus = remove_padding(img_cc_nucleus, 1)

image_cc_cyto, image_cc_nucleus, img_cc_cyto_dict, img_cc_nucleus_dict =
overlapping_labels(image_cc_cyto, image_cc_nucleus)
image_cc_cyto = cyto_to_gray(img_cc_cyto, img_cc_cyto_dict)
image_cc_nucleus = nuclei_to_white(img_cc_nucleus, img_cc_nucleus_dict)

own_mask = merge_for_mask(image_cc_cyto, image_cc_nucleus)

cv.imshow("Original", image)
cv.imshow("Mask", test_img)
cv.imshow("Cyto", image_cc_cyto)
cv.imshow("Nucleus", image_cc_nucleus)
cv.imshow("Own Mask", own_mask)
cv.waitKey()

print("\n--------------------------------CHECKING DICE COEFFICIENT ON VSET--
----------------------------------------")
total_dc_black = 0
total_dc_cyto = 0
total_dc_nucleus = 0
for i in range (201, 241):
    if i < 10:
        temp = "00" + str(i)
    elif i < 100:
        temp = "0" + str(i)
    else:
        temp = str(i)

    image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/images/" + temp + ".bmp",0)  # Grayscale image
    test_img = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/train/masks/" + temp + ".png", 0)
    image = contrast_stretch(image)
    histogram = histogram_creating(image)
    cumsum = hist_cumsum(histogram)

    cdf = cumsum / max(cumsum)
    thresh_cyto = (np.where(cdf >= 0.4)[0][0])
    thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

    #Taking avg of current and avg thresholds
    avg_threshold_cyto_new = (avg_threshold_cyto + thresh_cyto)//2
    avg_threshold_nucleus_new = (avg_threshold_nucleus + thresh_nucleus) // 2

    image_padded = padding(1, image)
    img_cc_cyto,img_cc_cyto_dict = cc(image_padded, 0, avg_threshold_cyto)
    img_cc_nucleus, img_cc_nucleus_dict = cc(image_padded, 0,
avg_threshold_nucleus)
    image_cc_cyto = remove_padding(img_cc_cyto, 1)
    image_cc_nucleus = remove_padding(img_cc_nucleus, 1)

    image_cc_cyto, image_cc_nucleus, img_cc_cyto_dict, img_cc_nucleus_dict =
overlapping_labels(image_cc_cyto, image_cc_nucleus)
    image_cc_cyto = cyto_to_gray(img_cc_cyto, img_cc_cyto_dict)
    image_cc_nucleus = nuclei_to_white(img_cc_nucleus, img_cc_nucleus_dict)
```

```
    own_mask = merge_for_mask(image_cc_cyto, image_cc_nucleus)

    # cv.imshow("Original", image)
    # cv.imshow("Mask", test_img)
    # cv.imshow("Cyto", image_cc_cyto)
    # cv.imshow("Nucleus", image_cc_nucleus)
    # cv.imshow("Own Mask", own_mask)
    # cv.waitKey()

    dc_black = calculate_dice_coefficient(test_img, own_mask, 0)
    dc_cyto = calculate_dice_coefficient(test_img, own_mask, 128)
    dc_nucleus = calculate_dice_coefficient(test_img, own_mask, 255)

    print(f"\nDCs for img {str(i)}: ")
    print(f"DC for Black: {dc_black}")
    print(f"DC for Cytoplasm: {dc_cyto}")
    print(f"DC for Nucleus: {dc_nucleus}")

    total_dc_black = total_dc_black + dc_black
    total_dc_cyto = total_dc_cyto + dc_cyto
    total_dc_nucleus = total_dc_nucleus + dc_nucleus

avg_dc_black = total_dc_black / 40
avg_dc_cyto = total_dc_cyto / 40
avg_dc_nucleus = total_dc_nucleus / 40

print("\n------AVG DCs FOR VSET------")
print(f"Avg DC Black: {avg_dc_black}")
print(f"Avg DC Cyto: {avg_dc_cyto}")
print(f"Avg DC Nucleus: {avg_dc_nucleus}")

print("\n---------------------------------CHECKING DICE COEFFICIENT ON TEST--
----------------------------------------")
total_dc_black = 0
total_dc_cyto = 0
total_dc_nucleus = 0
for i in range (241, 301):
    if i < 10:
        temp = "00" + str(i)
    elif i < 100:
        temp = "0" + str(i)
    else:
        temp = str(i)

    image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/test/images/" + temp + ".bmp",0)  # Grayscale image
    test_img = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment
1/dataset_DIP_assignment/test/masks/" + temp + ".png", 0)
    image = contrast_stretch(image)
    histogram = histogram_creating(image)
    cumsum = hist_cumsum(histogram)

    cdf = cumsum / max(cumsum)
    thresh_cyto = (np.where(cdf >= 0.4)[0][0])
    thresh_nucleus = (np.where(cdf >= 0.1)[0][0])

    #Taking avg of current and avg thresholds
```

```
    avg_threshold_cyto_new = (avg_threshold_cyto + thresh_cyto)//2
    avg_threshold_nucleus_new = (avg_threshold_nucleus + thresh_nucleus) // 2

    image_padded = padding(1, image)
    img_cc_cyto,img_cc_cyto_dict = cc(image_padded, 0,
avg_threshold_cyto_new)
    img_cc_nucleus, img_cc_nucleus_dict = cc(image_padded, 0,
avg_threshold_nucleus_new)
    image_cc_cyto = remove_padding(img_cc_cyto, 1)
    image_cc_nucleus = remove_padding(img_cc_nucleus, 1)

    image_cc_cyto, image_cc_nucleus, img_cc_cyto_dict, img_cc_nucleus_dict =
overlapping_labels(image_cc_cyto, image_cc_nucleus)
    image_cc_cyto = cyto_to_gray(img_cc_cyto, img_cc_cyto_dict)
    image_cc_nucleus = nuclei_to_white(img_cc_nucleus, img_cc_nucleus_dict)

    own_mask = merge_for_mask(image_cc_cyto, image_cc_nucleus)

    # cv.imshow("Original", image)
    # cv.imshow("Mask", test_img)
    # cv.imshow("Cyto", image_cc_cyto)
    # cv.imshow("Nucleus", image_cc_nucleus)
    # cv.imshow("Own Mask", own_mask)
    # cv.waitKey()

    dc_black = calculate_dice_coefficient(test_img, own_mask, 0)
    dc_cyto = calculate_dice_coefficient(test_img, own_mask, 128)
    dc_nucleus = calculate_dice_coefficient(test_img, own_mask, 255)

    print(f"\nDCs for img {str(i)}: ")
    print(f"DC for Black: {dc_black}")
    print(f"DC for Cytoplasm: {dc_cyto}")
    print(f"DC for Nucleus: {dc_nucleus}")

    total_dc_black = total_dc_black + dc_black
    total_dc_cyto = total_dc_cyto + dc_cyto
    total_dc_nucleus = total_dc_nucleus + dc_nucleus

avg_dc_black = total_dc_black / 60
avg_dc_cyto = total_dc_cyto / 60
avg_dc_nucleus = total_dc_nucleus / 60

print("\n------AVG DCs FOR TEST------")
print(f"Avg DC Black: {avg_dc_black}")
print(f"Avg DC Cyto: {avg_dc_cyto}")
print(f"Avg DC Nucleus: {avg_dc_nucleus}")
```