COMP448 HOMEWORK1 REPORT – SPRING 2023

Name – Surname: Barış Kaplan **(KU ID Number:** 0069054, **KU Login:** bkaplan18**)**

Part-1

Plots of the Estimated Masks:

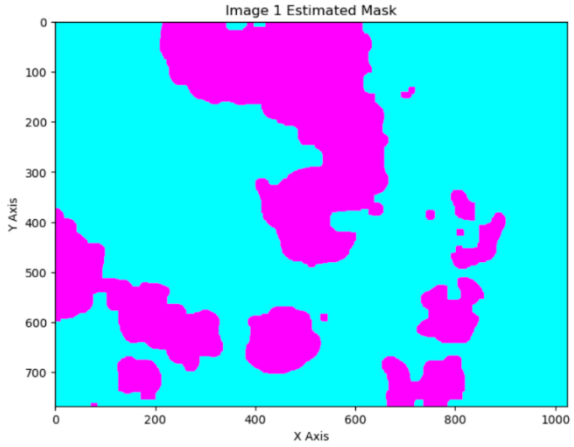


Figure 1: The plot of the estimated mask for Image1

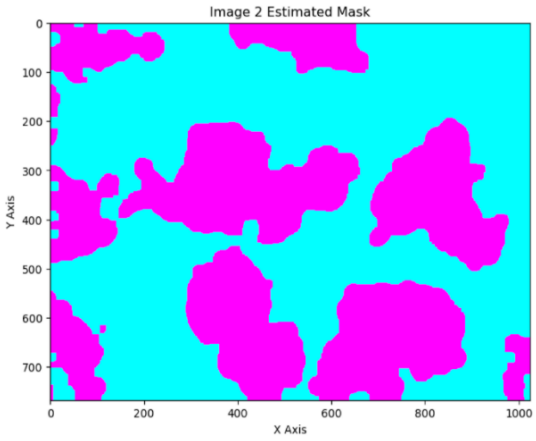


Figure 2: The plot of the estimated mask for Image2

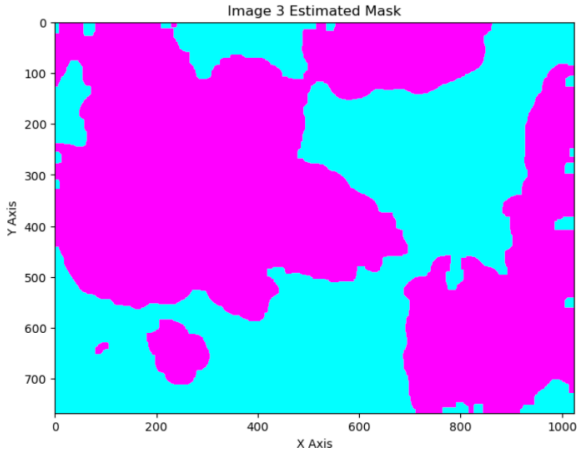


Figure 3: The plot of the estimated mask for Image3

Screenshots of the Precision, Recall, and F-Score values for each image:

```
Metrics for the image 1:

Precision: 0.9545192834523555
Recall: 0.8860016981221527
F-score: 0.9189851292058105
```

Figure 4: Pixel-level metric values for image1

```
Metrics for the image 2:

Precision: 0.9810016779955064
Recall: 0.9249381759675833
F-score: 0.9521453688893524
```

Figure 5: Pixel-level metric values for image2

```
Metrics for the image 3:

Precision: 0.9911611599555716
Recall: 0.936627837130928
F-score: 0.9631231788337817
```

Figure 6: Pixel-level metric values for image3

A table showing the Precision, Recall, and F-Score values for each image (pixel-level):

Images	Precision	Recall	F-score
Image1	0.9545192834523555	0.8860016981221527	0.9189851292058105
Image2	0.9810016779955064	0.9249381759675833	0.9521453688893524
Image3	0.9911611599555716	0.936627837130928	0.9631231788337817

Pseudocode:

```
for i in range(1, 4):
  read_ith_gold_mask_data()
  blurred_img = preprocess_image(image_ith_jpg_file)
  estimated_mask = apply_thresholding(blurred_img)
  estimated_mask = morphological_operations(estimated _mask)
  tp, fp, fn = calculate_tp_fp_fn(estimated_mask, gold_mask_data)
  prec = calculate_precision(tp,fp)
  recall = calculate_recall(tp,fn)
  f_score = calculate_fscore(prec,recall)
  display_metrics()
  plot_estimated_mask_for_ith_image()
```

Explanation (including the list of parameters):

Firstly, by using the genfromtxt function of the numpy library, I read the Txt file data of the gold mask of each image. Then, I applied the preprocessing to each image. In the preprocessing step; I read each image and converted it to grayscale. Next, I have utilized the "contrast enhancing/rising" technique to improve the distinguishability and visibility of a medical image's hard-to-differentiate parts, regions, and structures, such as the boundaries and edges. Since it gives better results, I used 1.70 as the enhancing coefficient. After contrast enhancement, I used a non-linear filter called "median filter" with a 5-by-5 structuring element to preserve boundaries & edges better than linear filters (e.g., gaussian filter) and to remove noise. I used the median filter since the white and black pixels are somehow randomly scattered in the provided medical images (noise like salt & pepper). After preprocessing, I converted the image to its grayscale version by using a corresponding code of gray color and then applied the thresholding method of Otsu. While applying thresholding, I used cv2.threshold() function from the OpenCV library of Python. In medical imaging, it is common practice to utilize 8-bit pixel representation for images where 0 (min value of an 8-bit binary number) represents the darkest pixel (black) and 255 (max value of an 8-bit binary number) represents the brightest pixel (white). In medical imaging, the brighter regions are generally treated as foreground while the darker regions are generally treated as background. Therefore; as parameters in the cv2.threshold() function which uses 8-bit pixel representation, I have set the background pixel threshold to 0 and the foreground pixel threshold to 255. Subsequently, for postprocessing, I applied several morphological operations. Initially, while applying binary closing, I used a large (13-by-13) structuring element to more effectively fill and close the small gaps/holes, disrupt the narrow connections in the foreground, and remove small foreground subregions. After that, I applied dilation with a 3-by-3 structuring element to make the foreground structures/objects bigger, reduce the holes within them, and connect with the unconnected pixel subregions. Next, I filled the binary holes to capture small details within the pixel regions such as gaps, boundaries, and holes. Subsequently, I used binary erosion with a 5-by-5 kernel in order to effectively remove the noise from foreground objects, and thus make the objects in the foreground smaller. Finally, in postprocessing, I applied binary closing once with a 7-by-7 structuring element and then binary opening twice with 9-by-9 and 11-by-11 kernels respectively. By applying these binary closing and opening methods with their corresponding kernel sizes, I got better results than the other outcomes I obtained through my trial-error-based experimentation. Then, I calculated the true positive, false positive, and false negative counters by comparing the estimated foreground mask with the gold mask txt data pixel by pixel. After that, by using the TP, FP, and FN counter values I obtained and applying the respective formula of each metric, I calculated the pixel-level recall, f-score, and precision values. Ultimately, I displayed the metrics I found and plotted the estimated foreground mask for each medical image using the "Matplotlib" library.

Part-2

Plots of the Regional Maxima Maps

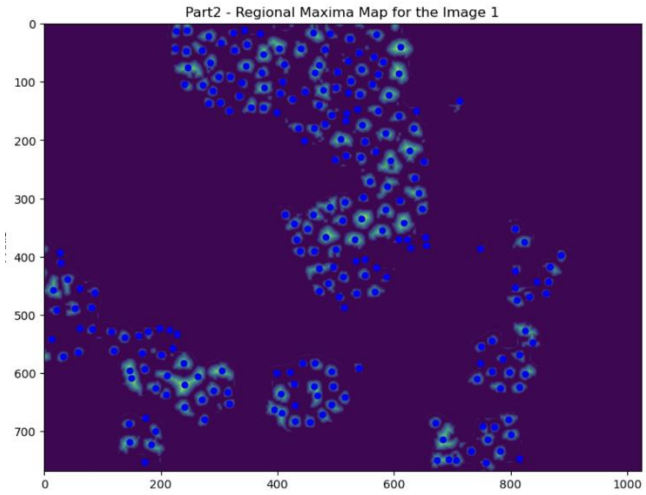


Figure7: *The regional maxima map of the Image1*

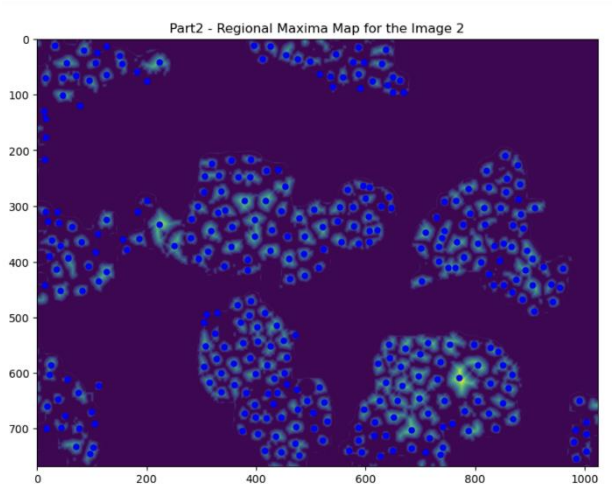


Figure8: *The regional maxima map of the Image2*

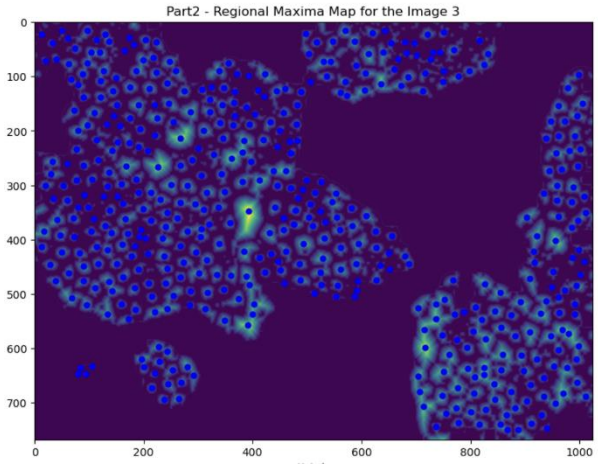
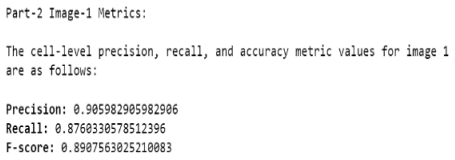


Figure9: *The regional maxima map of the Image3*

Screenshots of Precision, Recall, and F-score values for each image

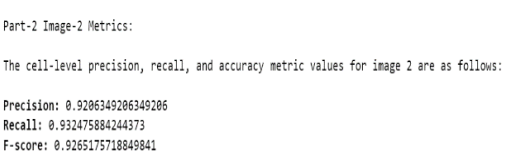


Part-2 Image-1 Metrics:

The cell-level precision, recall, and accuracy metric values for image 1 are as follows:

Precision: 0.905982905982906
Recall: 0.8760330578512396
F-score: 0.8907563025210083

Figure10: *Part2 Image1 Metrics*

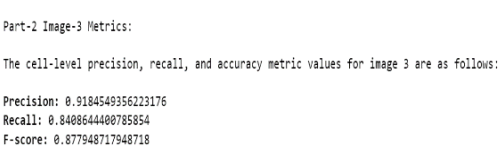


Part-2 Image-2 Metrics:

The cell-level precision, recall, and accuracy metric values for image 2 are as follows:

Precision: 0.9206349206349206
Recall: 0.932475884244373
F-score: 0.9265175718849841

Figure11: *Part2 Image2 Metrics*



Part-2 Image-3 Metrics:

The cell-level precision, recall, and accuracy metric values for image 3 are as follows:

Precision: 0.9184549356223176
Recall: 0.8408644400785854
F-score: 0.877948717948718

Figure12: *Part2 Image3 Metrics*

A table showing the precision, recall, and F-score values for each image (cell level)

Images	Precision	Recall	F-score
Image1	0.905982905982906	0.8760330578512396	0.8907563025210083
Image2	0.9206349206349206	0.932475884244373	0.9265175718849841
Image3	0.9184549356223176	0.8408644400785854	0.877948717948718

Pseudocode:

```
for i in range(1,4):
    read_ith_gold_cells_data()
    find_boundary()
    preprocess_image()
    remove_appearing_white_regions()
    calculate_and_normalize_distance_transform()
    calculate_regional_maxima()
    plot_regional_maxima()
    calculate_cell_level_metrics()
    display_cell_level_metrics()
```

Explanation (including the list of parameters):

By using the genfromtxt function of the numpy library, I have read the gold cells data. After that, I found the cell boundaries by considering pixel values less than or equal to 180 as black and more than 180 as white. Then, for preprocessing, I applied "non-local means denoising" to the blurred image by using the fastNlMeansDenoising function of the OpenCV library. In this function; I have passed 4 to h, 8 to the window size of the template, and 32 to the search window size. I have made the relationship between the template window size and search window size directly proportional to obtain more accurate denoising results and performance. While deciding specific values for h, template size, and search window size; I followed a trial and error approach and observed the changes in the values of f-score, precision, and accuracy metrics. Then, by utilizing a 5 by 5 kernel, I applied the erosion operation in order to reduce the sizes of (or remove) small foreground areas/objects and open/disrupt narrow connections between the connected components in the foreground. After that, by using a 3 by 3 structuring element, I applied the binary opening operation to remove/shrink small objects or noises in the foreground, connect the pixel regions near each other, and reduce/fill holes caused by erosion in the foreground. I have kept these kernels small to shrink/remove finer-grained objects/pixel regions, and to fill finer-grained holes in the foreground. Subsequently, I have shallow copied the estimated foreground mask. Next, I removed the white regions created after the binary opening and treated those regions as background in the copied mask. Subsequently, by using chebysev distance as a parameter (DIST_C parameter) and the mask size as 5, I have constructed the distance transform. I have also tried Euclidean distance but Chebyshev gives higher precision, accuracy, and f-score values. While deciding on mask size value in distance transform, I followed a trial and error approach and tried to obtain higher cell-level metric values. Then, by using the "normalize" function of the OpenCV library, I normalized the distance transform I constructed. In the normalize function, I used the default alpha and beta factor values suggested by the OpenCV documentation. To make the normalization by using alpha and beta values, which are lower and upper boundaries, I have used the NORM_MINMAX parameter in normalize function. Then, I found the regional maxima with peak_local_max(). In this function, I passed 12 to min distance to seek regional maxima points that are at least 12 pixels away from each other. Then by using the regional maxima points, I obtained the approximate cell locations. After that, I plotted distance transform and approximated cell locations by using the matplotlib library. Next, by iterating over approximate cell locations, I found the value of the true positive counter and false positive counters for part 2. While iterating, if the pixel is a background pixel (i.e., pixel == 0), I incremented the false positive counter. If only one centroid matched with the gold cell, I incremented the true positive counter. In this iteration, to detect matches only with 1 centroid, I have also subtracted the number of duplicate detections of the same cell location from the true positive counter and reset the number of duplicates when a new and correct cell location is found. Then, by using the counters I calculated and the total number of cell boundaries in the ground truth gold cells data, I calculated cell-level precision, recall, and f-score values. Finally, I displayed the metric values I found. For deciding the optimal value of each parameter in Part 2, I followed an experimentation approach based on trial and error.

Part-3

Segmentation Maps

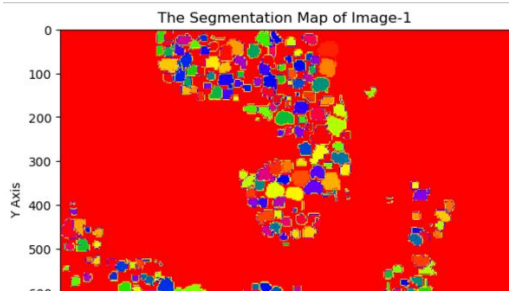


Figure13: *Segmentation Map of Image1*

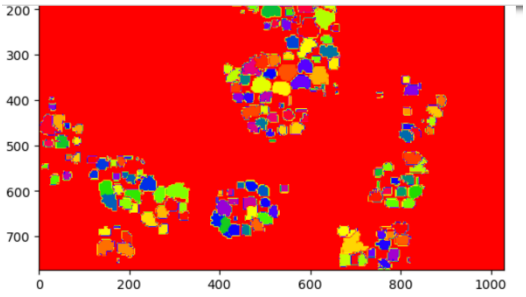


Figure13: *Segmentation Map of Image1 (remaining part)*

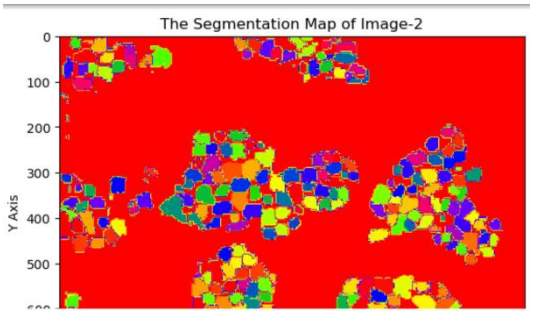


Figure14: *Segmentation Map of Image2*

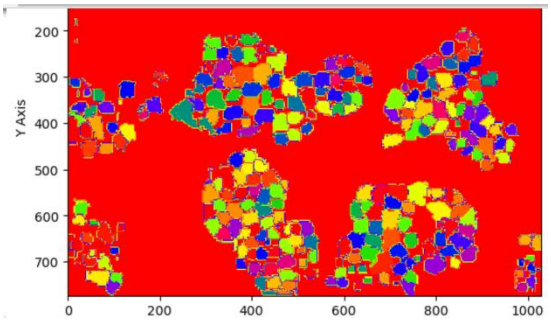


Figure14: *Segmentation Map of Image2 (remaining part)*

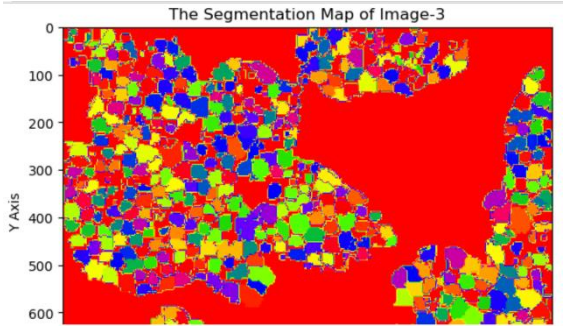


Figure15: *Segmentation Map of Image3*

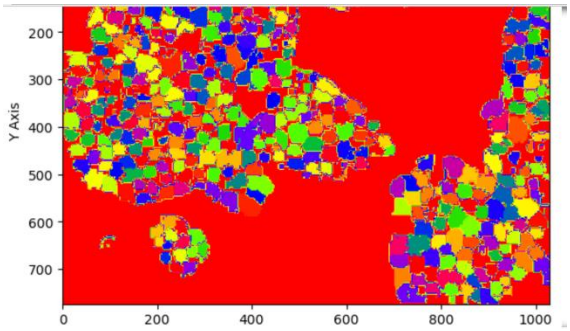


Figure15: *Segmentation Map of Image3 (remaining part)*

Quantitative Metrics (Cell-level Dice index and Intersection Over Union Metrics)

```
Cell-level dice index and intersection over union metrics of Part3:

For Threshold being 0.5:

Dice Index: 0.8034188034188035
Intersection Over Union: 0.6923076923076923

For Threshold being 0.75:

Dice Index: 0.5555555555555556
Intersection Over Union: 0.09401709401709402

For Threshold being 0.90:

Dice Index: 0.021367521367521368
Intersection Over Union: 0.0
```

Figure16: *Dice Index and IOU Metrics of Image1*

```
Cell-level dice index and intersection over union metrics of Part3:

For Threshold being 0.5:

Dice Index: 0.7570093457943925
Intersection Over Union: 0.6728971962616822

For Threshold being 0.75:

Dice Index: 0.4953271028037383
Intersection Over Union: 0.06542056074766354

For Threshold being 0.90:

Dice Index: 0.003115264797507788
Intersection Over Union: 0.0
```

Figure17: *Dice Index and IOU Metrics of Image2*

```
Cell-level dice index and intersection over union metrics of Part3:

For Threshold being 0.50:

Dice Index: 0.68
Intersection Over Union: 0.5810526315789474

For Threshold being 0.75:

Dice Index: 0.4631578947368421
Intersection Over Union: 0.10105263157894737

For Threshold being 0.90:

Dice Index: 0.010526315789473684
Intersection Over Union: 0.0
```

Figure18: *Dice Index and IOU Metrics of Image3*

A table showing the cell-level Dice Index and Intersection Over Union (IOU) metrics for each image

Images	Threshold = 0.50	Threshold = 0.75	Threshold = 0.90
Image1	**Dice Index =** 0.8034188034188 **IOU =** 0.6923076923076923	**Dice Index =** 0.55555555 **IOU =** 0.09401709401709402	**Dice Index =** 0.02136752136752 **IOU =** 0.00
Image2	**Dice Index =** 0.7570093457944 **IOU =** 0.6728971962616822	**Dice Index =** 0.495327102803 **IOU =** 0.06542056074766354	**Dice Index =** 0.00311526479751 **IOU =** 0.00
Image3	**Dice Index =** 0.68 **IOU =** 0.5810526315789474	**Dice Index =** 0.463157894737 **IOU =** 0.10105263157894737	**Dice Index =** 0.01052631578947 **IOU =** 0.00

Pseudocode:

```
for i in range(1,4):
        grow_regions_with _neighbors_of_regional_maximas()
        label_cell_pixels_in_segmentation_map()
        find_tp_fn_tp_counts_by_comparing_estimated_maxima_ and_gold_cell_pixels()
        find_dice_index_and_iou_scores_from_tp_fn_tn_counts()
        append_dice_index_and_iou_scores()
        find_overlap_amounts_of_dice_and_iou_scores_based_on_thresholds()
        calculate_and_display_cell_level_dice_index_and_iou_metrics()
        display_segmentation_map()
```

Explanation (including the list of parameters):

Firstly, I have excluded/omitted/erased the current foreground cell pixels. For excluding/erasing current foreground pixels, I have used a pixel value (-999) out of the valid range, which is between 0 and 255. I have applied this exclusion so that the cell boundaries in each contiguous subgroup of the segmentation map become more consistent and accurate. Then, I iterated through all regional maxima points that I estimated in Part 2. In this iteration, for each specific cell id value, I have found the successor of each regional maxima. Then, I appended the regions with the corresponding successor/neighbor I found for each regional maxima. For capturing enough adjacent pixels in the image boundaries while applying the region growing algorithm, I have applied padding by 3 rows to the bottom and top sides and 3 columns to the right and left sides of the segmentation map. I have decided on this padding amount by performing trial-and-error-based experimentation. In this experiment, I observed the changes in the values of cell-level dice index and intersection over union (IOU) metrics and tried to obtain higher values of those metrics. I have applied padding operation to the segmentation map so that I capture the additional pixels which will be added in the growing process. Then, for each reset pixel in the segmentation map, I applied the "region growing" algorithm on the segmentation map by assigning the same cell id to each contiguous subgroup of adjacent pixels in the map. While growing regions, I have kept each subgroup small to make the segmentation map of each image more accurate. As the initial seed values in the region growing process, I have utilized the predicted cell locations. By using the x location and y location values in each of those estimated cell locations, I have constructed the list of neighbor coordinates. In this "region growing" algorithm, I have also kept a list of already visited cell locations. For constructing this list, I have used the values of the x location, y location, and the cell id of each estimated cell location. Subsequently, I have iterated over the successors of a specific id and checked whether the cell id in those successors is already visited. If the cell id is new (not yet visited), then I have added this cell id to the list of all regions I keep. When the length of the list of regions becomes zero, I terminated the "region growing" algorithm. At the end of this algorithm, I have continuously popped the first element of the list of regions since it is already visited and there is no need to apply the region growing for that element. After popping, I have updated the length of the list of estimated regions. After the region growing process, I treated the reset pixels as background. Next, in order to find FP, TP, and FN counts and construct the lists of intersection and dice scores; I iterated through all estimated regional maxima points, applied pixel matching among gold cells and estimated cells, and used respective formulas of dice coefficient and IOU metrics in each iteration. Finally; I found overlap amounts of dice and IOU values/scores based on the threshold values, calculated the cell-level dice index and IOU metrics, and displayed the segmentation map & metric values.

Part-4

Segmentation Maps

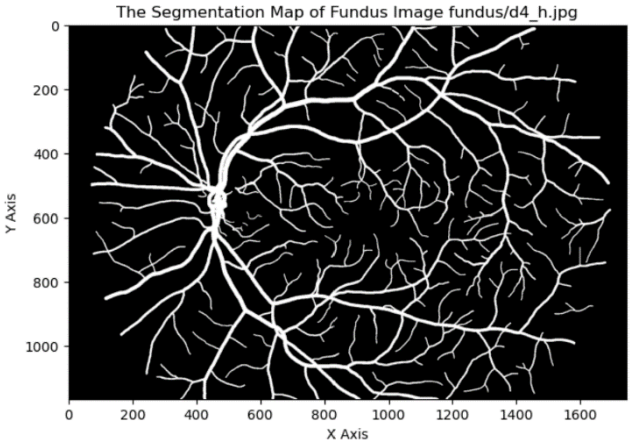


Figure 19: *The Segmentation Map of "d4_h.jpg"*

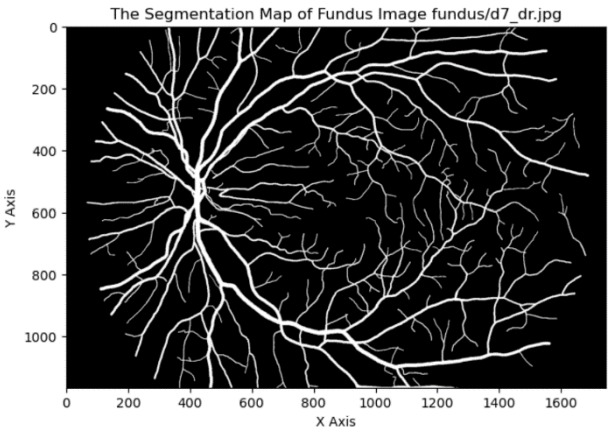


Figure 20: *The Segmentation Map of "d7_dr.jpg"*

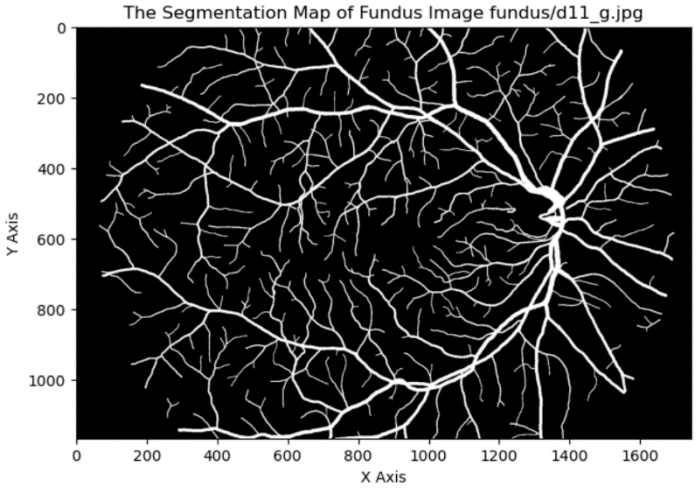


Figure 21: *The Segmentation Map of "d11_g.jpg"*

Screenshot of Pixel-level Precision, Recall, and F-score Metrics

```
The pixel-level precision, recall, and f-score metrics of the fundus image fundus/d7_dr.jpg are as follows:

Precision value: 0.12017720434279987
Recall value: 0.6817610639761522
F-score value: 0.2043352760061579

The pixel-level precision, recall, and f-score metrics of the fundus image fundus/d4_h.jpg are as follows:

Precision value: 0.1454234569274367
Recall value: 0.7501050437143457
F-score value: 0.24361674349274579

The pixel-level precision, recall, and f-score metrics of the fundus image fundus/d11_g.jpg are as follows:

Precision value: 0.12782447059361732
Recall value: 0.8646043794026101
F-score value: 0.2227214516596969
```

Figure22: The screenshot of pixel-level precision, recall, and f-score metrics for each image

A table showing the values of pixel-level precision, recall, and f-score metrics for each image

Images	Precision	Recall	Fscore
d7_dr.jpg	0.12017720434279987	0.6817610639761522	0.2043352760061579
d4_h.jpg	0.1454234569274367	0.7501050437143457	0.24361674349274579
d11_g.jpg	0.12782447059361732	0.8646043794026101	0.2227214516596969

Pseudocode:

```
for j in range(0,3):
    fundus = read_jth_fundus_image()
    gold = read_jth_gold_mask()
    estimated_mask = estimate_mask_of_vessels(fundus)
    precision = evaluate_precision(gold, estimated_mask)
    recall = evaluate_recall(gold, estimated_mask)
    f_score = evaluate_fscore(precision, recall)
    display_pixel_level_metrics_for_jth_fundus_image()
    plot_segmentation_map_for_jth_fundus_image()
```

Explanation (including the list of parameters):

By using a for loop, I have iterated over all fundus images and gold masks. By using the "cv2.imread" function from the OpenCV library of Python, I have read each fundus image and gold mask. As the flag argument, in order to read the image in the grayscale format, I have passed the "cv2.IMREAD_GRAYSCALE" to each "cv2.imread()" function I used. After reading the fundus image and the gold mask, I have estimated the mask of blood vessels in each fundus image. While estimating this mask, I have followed an approach similar to Part 1. Firstly, I have converted the grayscale image to the pil format by using the function called "Image.fromarray ()". After that, I applied the "contrast enhancing" technique to the PIL image. While enhancing the contrast, since it gives better metric results and segmentation maps, I have used "1.65" as the enhancing coefficient value. I have decided on this value by conducting several experiments with different enhance coefficient values and observing the pixel-level metric values (precision, recall, f-score) & segmentation maps accordingly. Then; by using an 11 by 11 structuring element, I applied the Gaussian blur to the enhanced PIL image as it is effective for removing/reducing the noise in images. While applying this blur, I used a large kernel in order to reduce/remove more noise. Subsequently, while estimating the mask of vessels, I used Otsu's thresholding method with the usage of the "cv2.threshold()" function. As the first argument, I have passed Gaussian blurred image to this function. As the second argument, I have passed 0 since the background pixels are stored as 0 in the gold mask images. As the third argument of the "cv2.threshold()" function, I have passed 1 since the foreground pixels are treated as 1 in the gold mask images. Subsequently, I have calculated the values of the pixel-level precision, recall, and f-score metrics. While calculating these metrics, I have pixel-by-pixel compared each mask of vessels I estimated with its corresponding gold mask image. If the gold mask pixel is 1 and the estimated mask pixel is 1, then I treated it as a true positive. If the gold mask pixel is 1 and the estimated mask pixel is 0, then I treated it as a false negative. If the gold mask pixel is 0 and the estimated mask pixel is 1, then I treated it as a false positive. After calculating these pixel-level metrics, I have displayed them. Finally, I have plotted the segmentation map of each fundus image by applying the "gold == 1" filter on the corresponding gold mask of each image. For plotting each segmentation map in the grayscale format, I have passed "gray" to the cmap parameter (cmap stands for colormap) of the "plt.imshow()" function. While plotting the segmentation maps, I used the "Matplotlib" library of Python.

Note-1: Observing the outputs of the third part takes some time. You need to wait nearly 20-25 minutes to see all outputs of Part 1, Part 2, and Part 3 (I have implemented them in a common for loop). You can see the final outputs of each part within the "ipynb" file I submitted.

Note-2: By using the "pip install library_name" command, you can install the libraries which do not exist in your computer but exist in my code (Here, library_name refers to the library which does not exist in your computer).