Medical Image Analysis Homework 1

Muhammed Esad Simitcioglu – msimitcioglu22

Arman Torikoglu – atorikoglu22

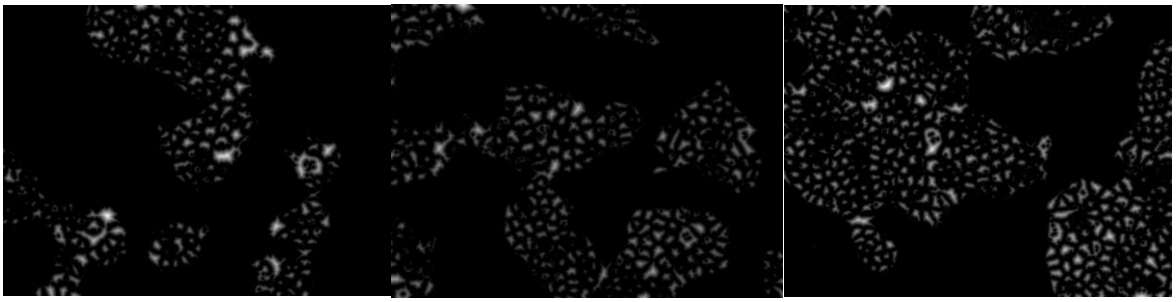
ObtainForegroundMask

The first step in the code is to calculate the best threshold to apply to the image. We calculate a histogram of the input test image using the OpenCV function cv2.calcHist(). The histogram is a graphical representation of the distribution of pixel intensities in an image. In this case, the histogram is calculated for the intensity values in the test image's first channel (0). The histogram is then normalized by dividing each bin value by the total sum of histogram values, resulting in a normalized histogram. Next, the normalized histogram is used to compute the cumulative distribution function (CDF) by taking the cumulative sum of the normalized histogram values using the cumsum() function from NumPy. The CDF represents the accumulated probability of pixel intensities up to a certain intensity level and is used to determine the optimal threshold for image segmentation. The code then iterates over all possible threshold values (0-255) and calculates the probabilities of the background and foreground classes using the CDF. The background class probability is obtained directly from the CDF, while the foreground class probability is calculated as 1 minus the background class probability. For each threshold value, the code calculates the means of the background and foreground classes using the normalized histogram and the corresponding probabilities. The intra-class variance, which represents the separability between the background and foreground classes, is then calculated using probabilities and means. The optimal threshold value is selected as the one that maximizes the intra-class variance. Using the optimal threshold value, the code then thresholds the input test image using the cv2.threshold() function from OpenCV, which converts the image into a binary image where pixels with intensities above the threshold are set to 255 (white) and pixels with intensities below the threshold are set to 0 (black). Morphological closing operation is then applied to the binary image using the cv2.morphologyEx() function, which helps to fill gaps in the binary image and smooth the edges of the generated mask. A structuring element in the shape of an ellipse with a size of (3, 3) is used for the morphological closing operation. The number of iterations for the closing operation is set to 20, which determines the extent of the smoothing and filling of gaps.

Pixel-Based	Precision	Recall	F-Score
Image 1	0.90	0.99	0.94
Image 2	0.94	0.98	0.96
Image 3	0.97	0.98	0.97

FindCellLocations

The algorithm uses the optimal threshold obtained from Otsu's method to threshold the RGB image and obtain a binary image. Morphological operations such as morphological opening and erosion are iterated once to the binary image using a pre-defined structuring element (a 3x3 ellipse) to remove noise and smooth the binary image. The resulting binary image is then inverted using bitwise not operation with a mask image to obtain the cells as white objects on a black background. Another thresholding operation is applied to the inverted binary image to refine the cell regions further. Distance transform is then applied to the refined binary image to obtain the distance of each pixel to the nearest cell boundary. The distance map is normalized and further processed with morphological operations (morphological closing and median filtering) to improve the accuracy of cell localization. Finally, contours are detected in the processed binary image using the OpenCV function findContours(), and the centroids of the contours are calculated as the locations of the detected cells.



Cell-Based	Precision	Recall	F-Score
Image 1	0.85	0.82	0.83
Image 2	0.8	0.84	0.82
Image 3	0.83	0.82	0.82

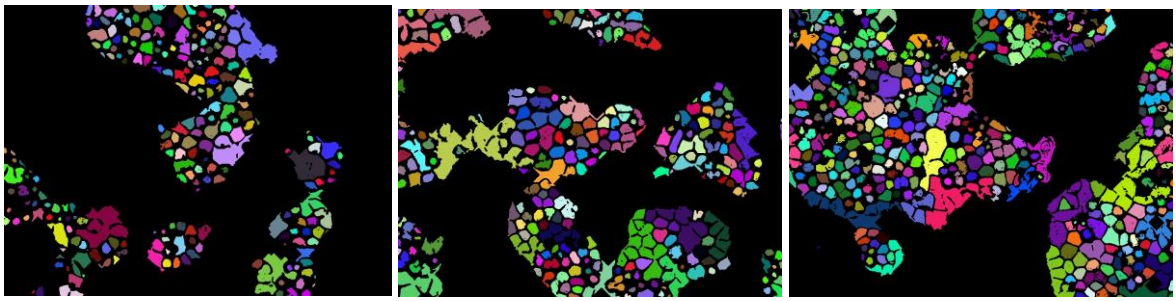
FindCellBoundaries

The code consists of the following components:

Stack Class: This class implements a stack data structure, which is used as a data structure for the region-growing algorithm.

The Region Growing Function is responsible for implementing a seed-region growing algorithm, a method used for segmenting regions in an image based on intensity similarity. The function takes several inputs, including a seed pixel, the image to be segmented, a mask, a threshold value, a segmentation map, and a seed counter. The threshold value used in this function is set to 40. The algorithm starts the segmentation process by searching for the centroid coordinate of the seed pixel. This search is performed using a 4-component-based cell approach. During the search, the function iterates over the grayscale version of the RGB image. At each iteration, it checks the difference between the grayscale value of the iterated pixel and the grayscale value of the seed pixel. If the result is greater than the threshold value, the growing process continues. In the growing process, we gave a unique ID for every seed's growing component.

Random Color Assignment: Once the algorithm has iterated over all the seeds and segmented the image into distinct regions, the next step is to colorize each unique seed. This is achieved by assigning a random color to each seed, resulting in a visually distinguishable and vibrant representation of the segmented regions. The random colorization adds an element of visual appeal to the segmented image, making it easier to visually identify and analyze the different regions of interest. This step enhances the overall visualization and interpretation of the image segmentation results, making them more accessible and informative for further analysis or visualization.



	Dice-Index
Image 1	0.62
Image 2	0.56
Image 3	0.58

Parameter = 0.5	Precision with IoU	Recall with IoU	F-Score with IoU
Image 1	0.42	0.4	0.41
Image 2	0.49	0.4	0.46
Image 3	0.56	0.44	0.5

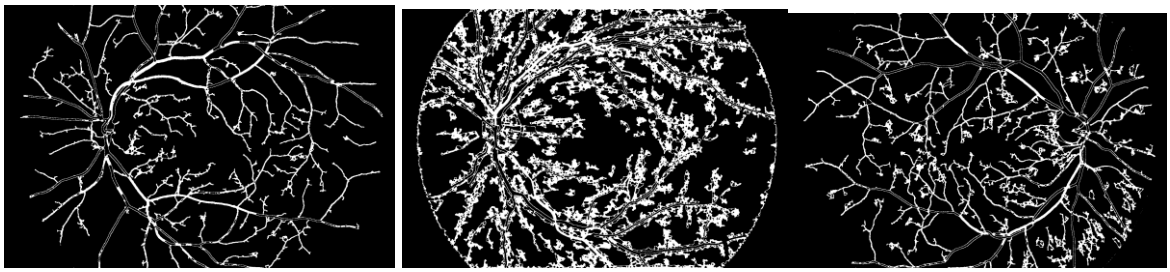
Parameter = 0.75	Precision with IoU	Recall with IoU	F-Score with IoU
Image 1	0.15	0.11	0.11
Image 2	0.23	0.18	0.2
Image 3	0.12	0.1	0.1

Parameter = 0.9	Precision with IoU	Recall with IoU	F-Score with IoU
Image 1	0.004	0.004	0.004
Image 2	0.01	0.01	0.01
Image 3	0.005	0.003	0.004

Segmentation of blood vessels in fundus photography images

The grayscale images are then preprocessed using a LoG filter to enhance blood vessels and suppress noise. The filtered images are then normalized to the range [0, 255] and converted to uint8 data type for further processing. Next, Gaussian blur is applied to the grayscale image to reduce noise. The LoG filter is applied again to obtain a Laplacian response image, which is then normalized and converted to binary using Otsu's thresholding. Contours are detected in the binary image and filtered based on their area to remove small contours that are likely noise or unrelated structures. A mask is created using filtered contours and applied to the binary image to retain only the blood vessels. The resulting blood vessel binary image is then converted to grayscale and thresholded to obtain a binary mask. Dilation is applied to the mask using an elliptical structuring element to enhance the blood vessels further. Distance transform is then applied to the dilated mask, and the resulting distance map is normalized and thresholded to obtain a binary blood vessel mask. Median filtering is applied to the

binary mask to remove remaining noise, and erosion is performed to refine the blood vessel mask. The final blood vessel mask is displayed using OpenCV's imshow function, and the result is saved as an output image. Pixel-level metrics, including precision, recall, and F-score, are calculated by comparing the detected blood vessel mask with the ground truth data using a custom pixel_level function.



Pixel-Based	Precision	Recall	F-Score
Image 1	0.75	0.55	0.62
Image 2	0.2	0.4	0.3
Image 3	0.55	0.6	0.6