Basic Cell Segmentation

Pouya Pourakbarian Niaz
ID: 0073978
Koç University
Spring 2023
pniaz20@ku.edu.tr

Abstract

This project aims to help us understand and apply some of the classical image processing techniques to perform segmentation without using machine/deep learning. To that end, three similar images containing bodies of cells packed densely together are given. In order to perform cell segmentation on the images, firstly, the binary foreground mask of the images will be extracted. Then, cell regions will be identified along with cell centers, and then, using the watershed algorithm, entire grown cells will be constructed, identified and segmented from one another. The OpenCV library in Python is used for the implementation of the image processing operations.

Introduction

In this project, three images (first row of Fig. 1) are given, including some densely packed transparent cells with white boundaries, spread across a beige-colored background. The objective is to separate the cells from one another so their properties can be studied later. To this end, since we do not intend to use an end-to-end machine/deep learning pipeline, we will use classical image processing techniques such as gray-scale conversion, thresholding/binarization, filtering, morphological operations, distance transforms, region-growth, etc.

This project comprises of three main sections. In the first section, the binary foreground mask of the image is extracted, which will later help us identify cell regions and segment them properly. In the second section, the cell region seeds will be identified, and cell centers will be calculated. In the third section, the watershed algorithm will be used for growing the seeds into fully segmented cells. As an additional extra, in the fourth section vessel segmentation will be performed on fundus photography.

1 Obtaining the Foreground Mask

In order to help us identify cell regions and perform cell segmentation, it is beneficial to know which parts (pixels) of the image are foreground, and contain the cells or packs of cells, and which parts are background, and contain nothing but empty space. The binary foreground mask will be an image which is white throughout the foreground, and black everywhere else. The `ObtainForegroundMask` function in the provided source code contains the implementation which we used for obtaining the binary foreground mask. Algorithm 1 briefly summarizes the procedure.

Algorithm 1 is a function that takes s_1 which is the structuring element size for morphological opening, and $t_{\min}$ which is the relative intensity for binarizing the distance transform so that a binary mask can be obtained. In this report, after some trial and error, $s_1 = 40$ and $t_{\min} = 0$ are selected as the possible optimal values. Inner images generated by Algorithm 1 as well as the original image and the final mask can be observed in Fig. 1.

Algorithm 1 Obtain the foreground mask of the image

function OBTAINFOREGROUNDMASK(image, s_1, $t_{\min}$)
  gray $\leftarrow$ ConvertToGrayscale(image)
  thresh $\leftarrow$ Binarize(gray, Inverse + Otsu)
  opening $\leftarrow$ MorphOpening(thresh, Ellipsis, s_1)
  dist $\leftarrow$ DistanceTransform(opening)
  mask $\leftarrow$ Binarize(dist, $t_{\min} \times \max$(dist))
  mask $\leftarrow 1 -$ mask
  return mask
end function

As it can be observed, Algorithm 1 first converts the image to gray scale so that it is monochromatic and therefore easier to process. Then, it inverse-binarizes the gray image using Otsu's method, such that the threshold is optimally defined and the binarization is inverted. Then, an optional morphological opening can be performed in order to remove noise and get rid of small regions. A morphological closing can also be tried and tested. The code implementation actually allows it as an optional input. By default, an elliptic structural element is chosen, but the code implementation allows any other structuring element, e.g., cross, rectangle, and diamond, as an optional input to the function. In our preliminary trial and error it was observed that an elliptic structuring element produced the best results. After the morphological operation, the morphed image is passed through a function that calculated the distance transform map of the image. The distance transform can give us information as to how close or far every pixel is from its boundaries, thereby producing a virtual 3D surface which can later be used for binarization, etc. This way, for instance, all pixels that are farther/closer to the foreground/background boundary than some threshold can be blacked out. Distance transforms are used for shrinking or enlarging foreground or background regions that are too small, too large, too far from the center, too close to some boundary, and so forth. After obtaining the distance transform, the map is binarized so that the inner regions of the foreground are blacked out, as seen on Fig. 1. By inverting this binarized distance map, we can obtain a binary foreground mask for the image, which is done in the last line of the algorithm, before returning the mask.

In order to evaluate the foreground mask, a ground truth is provided for each of the three images. Since generating the foreground mask is a pixel-level binary classification task, pixel-level statistics such as accuracy, F1 score, etc., can be generated. Table 1 demonstrates the metrics achieved by the algorithm.

Table 1: Foreground mask generation performance

Image	**Accuracy**	**Precision**	**Recall**	**F-Score**
1	0.9637	0.8916	0.9984	0.9420
2	0.9663	0.9357	0.9891	0.9617
3	0.9775	0.9738	0.9894	0.9815

2 Finding Cell Locations

In order to segment the cells and separate them from one another, we need to identify where the cells are located. In some applications where the cells are individually separable coin-shaped elements, this step is relatively straightforward. By contrast, when images contain regions of densely connected neighboring cells, simple distance transform will not help separate the neighboring cells within the foreground. Slightly more complex procedures may therefore be required. In our particular application, a glimpse at the original images reveals the distinctly white-colored boundaries between the cells, a contrasting feature that can be used for locating the cells. As such, the cell location method is briefly explained in Algorithm 2.

In Algorithm 2, $t_3 = 200$ is the initial binarization threshold, $s_{o,d} = 3$ is the structuring element size of the opening which is used for denoising and smoothing, and $t_{\text{rel}} = 0.2$ is the relative binarization threshold for extracting the binary local maxima map from the distance transform map. In this algorithm, after gray-scaling and thresholding the image, parts of it that correspond to the background in the foreground mask, are blacked out. This helps us avoid having unnecessary distance

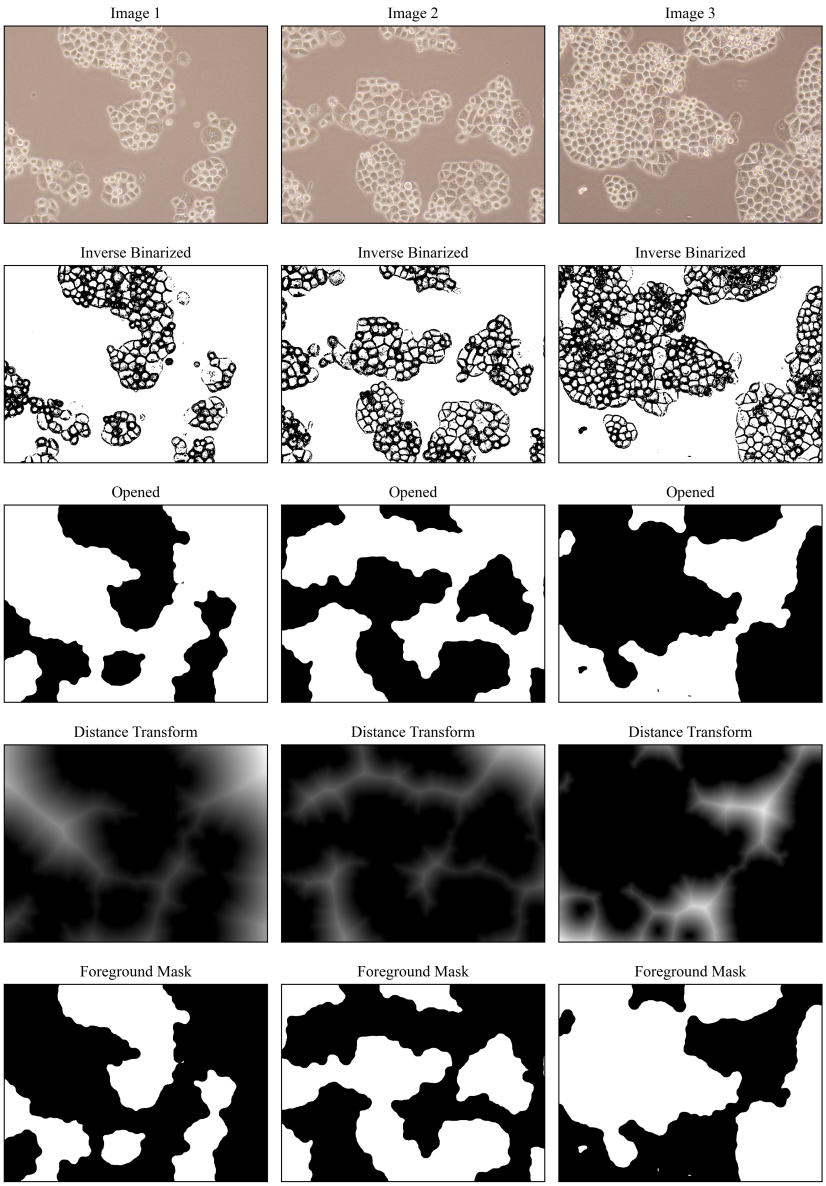

Figure 1: Obtaining the binary foreground mask, per Algorithm 1

transform values across the background. The procedure explained in Algorithm 2 can also be seen with more details in Fig. 2. In this algorithm, after blacking out the background, a morphological opening is applied to get rid of any noise, if necessary. In the resulting image, the cells are white, their boundaries are black, and the background is also black. In order to emphasize the boundaries between the cells, a distance transform is applied, which is maximum at cell centers and minimum at boundaries. Using this transform, as it can be seen in Fig. 2, the cells are farther and more clearly separated from each other. In order to fully separate the cell regions, the binary local maxima map is extracted. In a binary local maxima map, the local maxima of the distance transform are extracted, and binarized to white, while everything else is blacked out. After extracting the local maxima, a simple clustering algorithm would be necessary to check the connectivity (surroundings) of each cell and determine which cell region each pixel belongs to. Most image processing libraries have built-in functions for checking the connectivity and extracting connected components from local maxima maps via contouring, using either 4-connectivity or 8-connectivity. For the `ConnectedComponents`

Algorithm 2 Generating cell regions and calculating cell centroid locations

```
function FindCellLocations(image, foregroundMask, t_3, s_{o,d}, t_rel)
    gray ← ConvertToGrayscale(image)
    thresh ← Binarize(gray, t_3)
    for pixel in thresh do
        if foregroundMask[pixel] = 0 then
            thresh[pixel] ← 0
        end if
    end for
    opening ← MorphOpening(thresh, Ellipsis, s_{o,d})
    dist ← DistanceTransform(opening)
    localMaxima ← Binarize(dist, t_rel × max(dist))
    [cellLabels, centroids] ← ConnectedComponents(localMaxima)
    return [localMaxima, cellLabels, centroids]
end function
```

Table 2: Cell location performance

Image	Precision	Recall	F-Score
1	0.9014	0.7934	0.8440
2	0.8618	0.8424	0.8520
3	0.8912	0.7721	0.8274

function of Algorithm 2, in Python OpenCV, the `cv2.connectedComponentsWithStats` or the `cv2.findContours` functions can both be utilized.

For evaluating the performance of cell location with Algorithm 2, a ground truth is provided for each image, where the background is black, and all cells are labeled accordingly. For assessment, the calculated centroids are used. Every centroid that falls into the background of the ground truth is a false positive. True positives are the ground truth cells that contain (match with) only one of the calculated centroids, and the rest of the ground truth cells are false negatives. There are no true negatives in this context. Nevertheless, precision, recall and F-scores can be calculated using these definitions. Table 2 outlines the performance of Algorithm 2 in cell location.

3 Cell Segmentation

The final step after cell location is cell segmentation. There are many ways of achieving this, but the watershed algorithm, which is a seed-growth algorithm, can be utilized by using the local maxima map as a marker. By knowing beforehand which areas of the image are known (labeled), which areas are unknown and need to be segmented (labeled) by the algorithm, and which areas are the background and need to stay that way, the watershed algorithm checks for similarities between foreground pixels and their surroundings, while filling in the unlabeled foreground area, until no further growth/expansion is possible. Algorithm 3 briefly explains how this is achieved.

In this algorithm, function parameter image is the RGB image itself, foregroundMask is the binary foreground mask where foreground is white and background is black, and labelsMap is an image where the background including inter-cell boundaries are black (0), and cell regions that are marked by the local maxima map obtained in the previous section, are numbered from 1 to N, where N is the total number of cell regions (seeds) in the binary local maxima map. The Python OpenCV implementation of the Watershed algorithm is such that the input marker needs to be 0 at unknown foreground pixels, 1 at background pixels, and one more than the label throughout the rest of the image. The output segmentation is also such that background pixels are assigned -1, and unlabeled foreground pixels are assigned 0. The labeled foreground pixels are assigned one more than their labels. This means that generating the markers image and postprocessing the output of the Watershed method requires some procedures, which are marked in Algorithm 3. Fig. 3 shows the segmentation map produced by Algorithm 3, comparing the ground truth and the calculated segmentations.

For evaluating the cell segmentation performance, object-level metrics are used. To this end, the object-level dice index of the segmentation is calculated. Furthermore, object-level metrics such

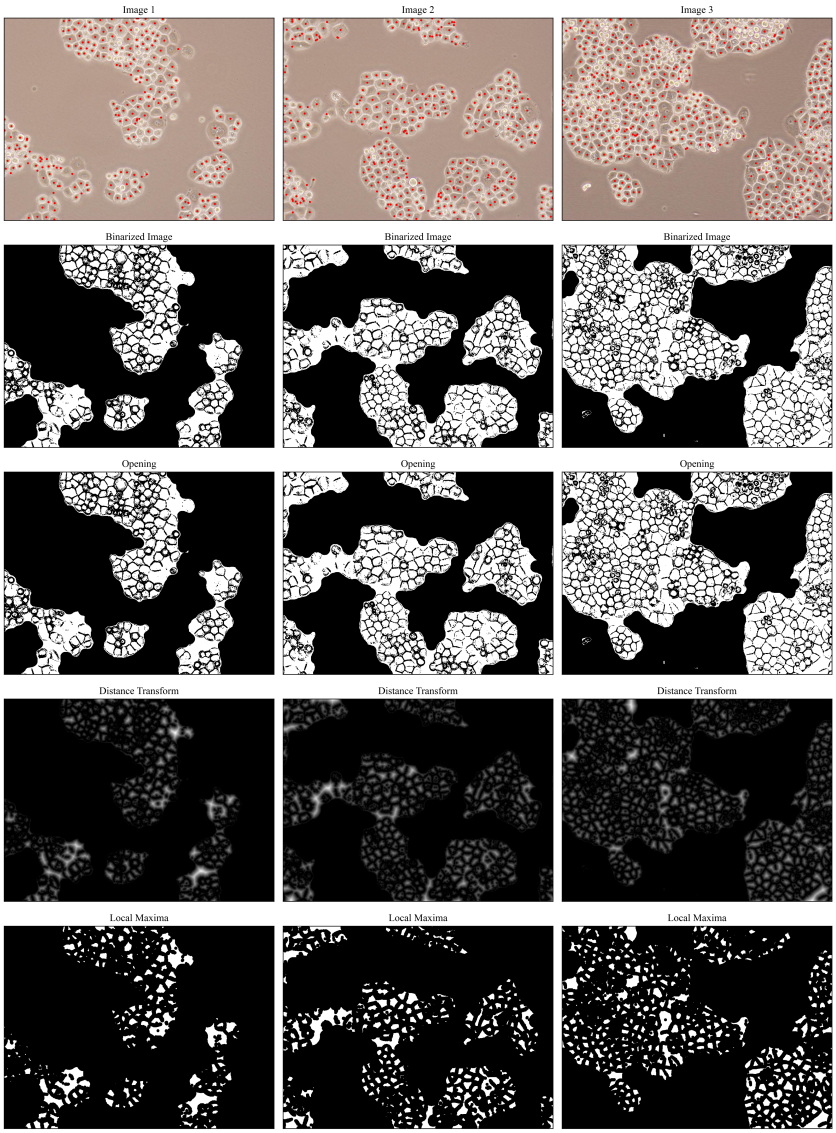

Figure 2: Extracting cell regions, centroids, and local maxima, per Algorithm 2. The red dots in the first row represent the cell centroids after running the algorithm.

as precision, recall and F-score are calculated using the assumption that a segmented cell is a true positive if and only if it has a 50% overlap with its matching ground truth cell, and the ground truth cell also has a minimum 50% overlap with the segmented cell. In other words, it is assumed that a segmented cell is a true positive if and only if it has an intersection-over-union (IOU) of at least 1/3 with its matching ground truth cell. In addition, IOU-based object-level metrics are also calculated. When using IOU for calculating metrics, IOU thresholds of 0.5, 0.75 and 0.9 are considered. In this case, a segmented cell is a true positive if it has an IOU with its matching ground truth cell of greater than or equal to the chosen threshold. As expected, the higher the threshold, the more strict true positive requirements will be. Therefore, the higher the threshold, the lower the metrics. Table 3 shows the cell segmentation metrics achieved using Algorithm 3. As it can be seen on this table, metrics are acceptable when the true positive requirements are slightly lenient. At higher IOU thresholds, however, the segmentation suffers from very low metrics because almost none of the segmented cells have an IOU of 0.9 or higher with their ground truth cells.

Algorithm 3 Cell segmentation

```
function FindCellBoundaries(image, foregroundMask, labelsMap)
    markers ← Copy(labelsMap)
    for pixel in image do
        if foregroundMask[pixel] = 0 then
            markers[pixel] ← 1
        else if labelsMap[pixel] = 0 then
            markers[pixel] ← 0
        else
            markers[pixel] ← labelsMap[pixel] + 1
        end if
    end for
    seg ← Watershed(image, markers)
    for pixel in seg do
        if seg[pixel] < 1 then
            seg[pixel] ← 1
        end if
        seg[pixel] ← seg[pixel] − 1
    end for
     return seg
end function
```

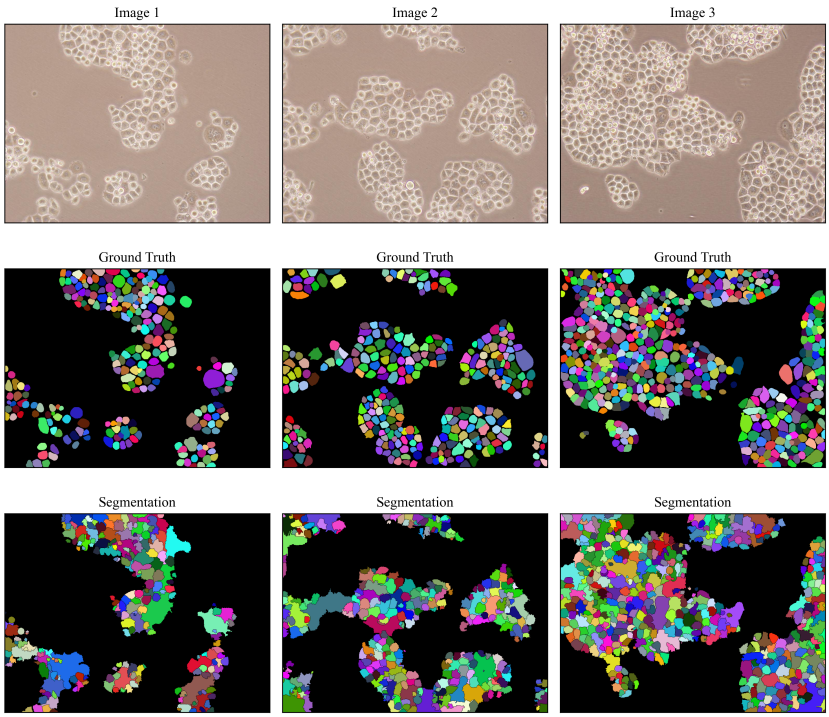

Figure 3: Cell segmentation, per Algorithm 3

4 Vessel Segmentation in Fundus Photography

In this extra section of the assignment, vessel segmentation is performed on retinal fundus photography. The goal is to segment blood vessels out of the rest of the fundus images, so they can be studied further. Algorithm 4 summarizes the procedures for this task. In this algorithm, $t_{\text{lens}} = 8$ is a small threshold used for extracting the lense shutter mask which will be used later, $s_{\text{blur}} = 31$ is the size of the Gaussian blurring mask, $s_{o,\text{init}} = 3$, $s_{c,\text{pre}} = 9$ and $s_{o,\text{pre}} = 3$ are the structuring element

Table 3: Cell segmentation object-level performance

Image	Dice Index	Object-Level Metrics			IOU Metrics per Threshold								
					Precision			Recall			F-Score		
		Precision	Recall	F-Score	*0.5*	*0.75*	*0.9*	*0.5*	*0.75*	*0.9*	*0.5*	*0.75*	*0.9*
1	0.6148	0.7354	0.6777	0.7054	0.4888	0.1839	0.0045	0.4504	0.1694	0.0041	0.4688	0.1763	0.0043
2	0.6298	0.6342	0.6913	0.6615	0.4838	0.1652	0.00	0.5273	0.1801	0.00	0.5046	0.1723	0.00
3	0.6676	0.7591	0.6994	0.7280	0.6162	0.2302	0.0064	0.5678	0.2122	0.0059	0.5910	0.2209	0.0061

sizes of the morphological opening, closing and opening operations performed respectively in order, $t_{\text{area}} = 150$ is the area of the smallest contour (connected region) that will be kept when refining (masking) the image, and finally, $s_{c,\text{fin}} = 11$ and $s_{o,\text{fin}} = 3$ are the element sizes of the closing and opening operations that will be respectively performed on the refined mask at the end. In this algorithm, the green channel is extracted foremost because of its high contrast. Then, the lense shutter mask is extracted from it. Then, a Gaussian blur is deployed to blur the image and smoothen it in the process. Afterwards, by dividing the green channel intensity by the blurred image, a neat map is revealed that expresses useful features. Then, parts of it that fall outside the lense shutter are discarded. After that, the image is binarized and morphed multiple times to get rid of small islands and small gaps. Then, small regions are discarded from the image, and only the vessels remain. The image is morphed twice more to result in the final segmentation image. Fig. 4 shows the segmentation results of the three fundus images provided. Additionally, Table 4 shows the pixel-level performance metrics of this segmentation task. It is not feasible to use simple image processing techniques and keep even the most narrow vessels active and segmented. Most of the operations done in Algorithm 4 will eliminate some of the smaller, narrower vessels. Nevertheless, the results show in Table 4 are acceptable for most applications.

Algorithm 4 Vessel segmentation from fundus photography

function SEGMENT_VESSELS(image, t_{lens}, s_{blur}, $s_{o,\text{init}}$, $s_{c,\text{pre}}$, $s_{o,\text{pre}}$, $s_{c,\text{fin}}$, $s_{o,\text{fin}}$, t_{area})
  green ← GetGreenChannel(image)
  lenseMask ← Binarize(green, t_{lens})
  blurred ← GaussianBlur(green, s_{blur})
  division ← Divide(green, blurred)
  maskedDiv ← Copy(division)
  for pixel **in** lenseMask **do**
    if lenseMask [pixel] = 0 **then**
      maskedDiv [pixel] ← 255
    end if
  end for
  thresh ← Binarize(maskedDiv, Otsu)
  thresh ← 255 - thresh
  morphed ← Copy(thresh)
  morphed ← MorphOpening(morphed, Ellipsis, $s_{o,\text{init}}$)
  morphed ← MorphClosing(morphed, Ellipsis, $s_{c,\text{pre}}$)
  morphed ← MorphOpening(morphed, Ellipsis, $s_{o,\text{pre}}$)
  mask ← ZerosLike(thresh)
  contours ← FindContours(thresh)
  for contour **in** contours **do**
    if GetContourArea(contour) $\geq t_{\text{area}}$ **then**
      for pixel **in** contour **do** mask [pixel] ← 255
      end for
    end if
  end for
  morphed ← MorphClosing(mask, Ellipsis, $s_{c,\text{fin}}$)
  output ← MorphOpening(morphed, Ellipsis, $s_{o,\text{fin}}$)
  return output
end function

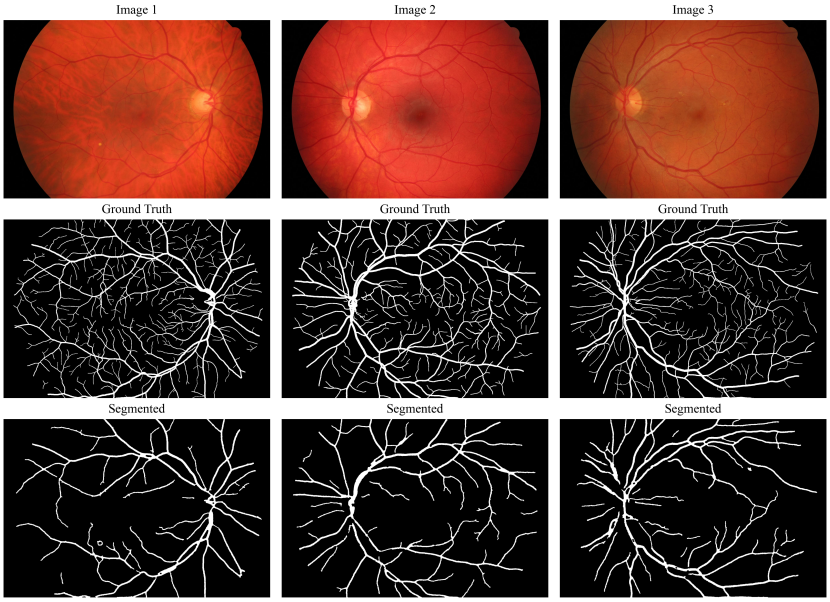

Figure 4: Vessel segmentation on fundus images, per Algorithm 4

Table 4: Vessel segmentation performance on three given fundus images

Image	**Accuracy**	**Precision**	**Recall**	**F-Score**
1	0.9416	0.8411	0.5650	0.6759
2	0.9460	0.9354	0.5972	0.7290
3	0.9428	0.7897	0.6346	0.7037