LAB 1 - OBJECT COUNTING ON IMAGES

Fromsa T. Negasa, Nadeer Hassan

Photonics for Security, Reliability, and Safety (PSRS)

Medical Biometrics

Yasmina CHENOUNE, Instructor

17th October 2024

```
clc; clear; close all
```

I. Objective

Apply the different concepts of morphological processing for object counting on a blood cells image

II. Required tools

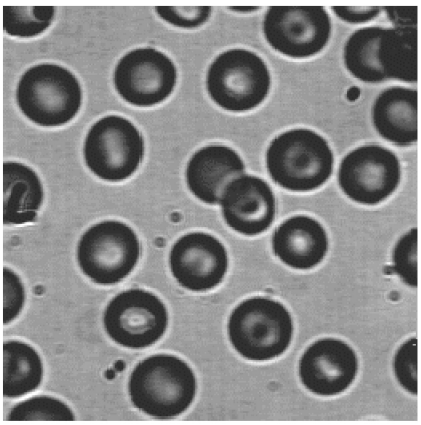
The Image Processing Toolbox of Matlab or Python Image Library.

III. Procedure

1. Load and show the image

Read the image bloodcells

```
img = imread("BLOODCELLS.bmp");
figure;
imshow(img)
```

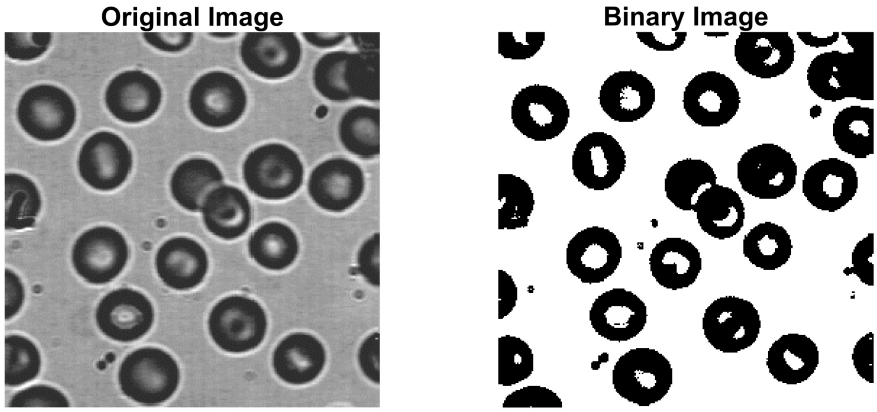


Transform this image into a binary image BW using a thresholding process or a dedicated function of Matlab or Python.

```
BW = imbinarize(img, graythresh(img));
```

Show the two images with titles within the same figure.

```
figure;
subplot(1,2,1), imshow(img), title('Original Image');
subplot(1,2,2), imshow(BW), title('Binary Image');
```



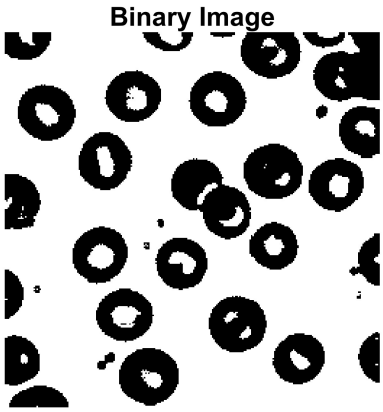
The morphological operator operates on the white pixels in the binary image. What do you observe on the image? What's the solution of this problem?

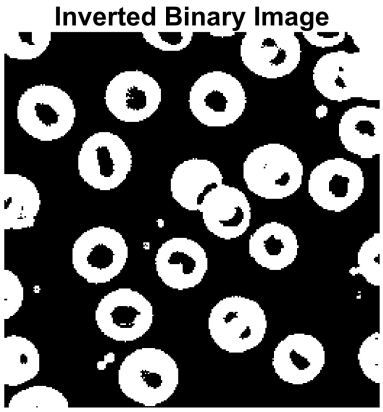
- Morphological operators typically work on white pixels (foreground), and this could be problematic if the objects are represented by dark pixels.
- **Solution:** Invert the binary image if the foreground (cells) is dark.

```
BW_inv = ~BW;
```

Show the inverted binary image

```
figure;
subplot(1,2,1), imshow(BW), title('Binary Image');
subplot(1,2,2), imshow(BW_inv), title('Inverted Binary Image');
```





Erosion

Open and study the help page of the erosion function.

```
help imerode
```

```
  imerode - Erode image
     This MATLAB function erodes the grayscale, binary, or packed binary
     image I using the structuring element SE.

     Syntax
       J = imerode(I,SE)
       J = imerode(I,nhood)
       J = imerode(___,packopt,m)
       J = imerode(___,shape)

     Input Arguments
       I - Input image
         grayscale image | binary image | packed binary image
       SE - Structuring element
         strel object | offsetstrel object | array of strel objects |
         array of offsetstrel objects
       nhood - Structuring element neighborhood
         matrix of 0s and 1s
       packopt - Indicator of packed binary image
         'notpacked' (default) | 'packed'
       m - Row dimension of original unpacked image
         positive integer
       shape - Size of output image
         'same' (default) | 'full'

     Output Arguments
       J - Eroded image
         grayscale image | binary image | packed binary image

     Examples
       Erode Binary Image with Line Structuring Element
```

```
  Erode Grayscale Image with Rolling Ball
  Erode MRI Stack Volume Using Cubic Structuring Element

See also bwpack, bwunpack, conv2, filter2, imclose, imdilate, imopen,
  strel, offsetstrel

Introduced in Image Processing Toolbox before R2006a
Documentation for imerode
Other uses of imerode
```

Choose your appropriate structuring element SE (shape and size) and justify your choice.

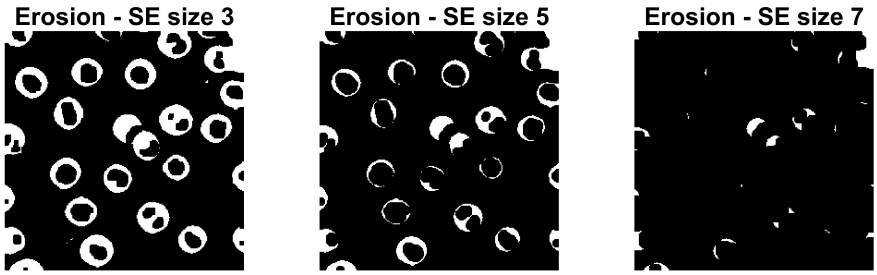
- Given that the image contains circular cells, the most appropriate structuring element (SE) to use is a **disk-shaped SE.** This shape matches the geometry of the objects (circular cells and the small debris) in the image and is optimal for preserving their roundness during morphological operations.

Apply the erosion operator on the binary image BW and Repeat the operation with 3 different sizes of SE.

```
SE1 = strel('disk', 3); % Small structuring element
SE2 = strel('disk', 5); % Medium structuring element
SE3 = strel('disk', 7); % Large structuring element

eroded1 = imerode(BW_inv, SE1);
eroded2 = imerode(BW_inv, SE2);
eroded3 = imerode(BW_inv, SE3);
```

Visualize the obtained results, the eroded images.

```
figure;
subplot(1,3,1), imshow(eroded1), title('Erosion - SE size 3');
subplot(1,3,2), imshow(eroded2), title('Erosion - SE size 5');
subplot(1,3,3), imshow(eroded3), title('Erosion - SE size 7');
```



Discuss the results and explain the observed differences.

- The size and shape of the structuring element (SE) impact the results of the erosion. Smaller SEs preserve object details, while larger SEs erode more aggressively. The roundedness is preserved.

Dilation

Open and study the help page of the dilation function.

```
help imdilate
```

```
  imdilate - Dilate image
     This MATLAB function dilates the grayscale, binary, or packed binary
     image I using the structuring element SE.

     Syntax
       J = imdilate(I,SE)
       J = imdilate(I,nhood)
       J = imdilate(___,packopt)
       J = imdilate(___,shape)

     Input Arguments
       I - Input image
         grayscale image | binary image | packed binary image
       SE - Structuring element
         strel object | offsetstrel object | array of strel objects |
         array of offsetstrel objects
       nhood - Structuring element neighborhood
         matrix of 0s and 1s
       packopt - Indicator of packed binary image
         'notpacked' (default) | 'packed'
       shape - Size of output image
         'same' (default) | 'full'
```

```
Output Arguments
  J - Dilated image
    grayscale image | binary image | packed binary image

Examples
  Dilate Image with Vertical Line Structuring Element
  Dilate Grayscale Image with Rolling Ball
  Determine Domain of Composition of Structuring Elements
  Dilate Points in 3D Space Using Spherical Structuring Elements

See also bwpack, bwunpack, conv2, filter2, imclose, imerode, imopen,
  strel, offsetstrel

Introduced in Image Processing Toolbox before R2006a
Documentation for imdilate
Other uses of imdilate
```

Choose your appropriate structuring element SE (shape and size).

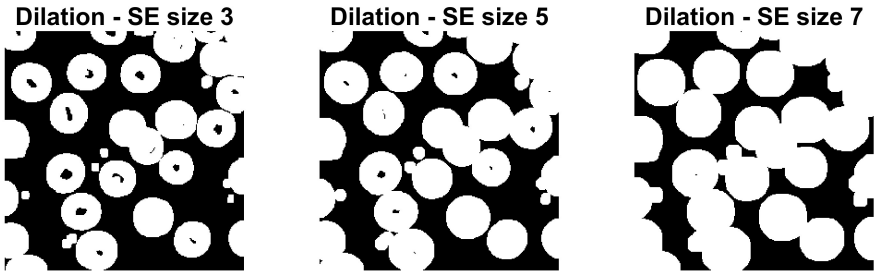
- For the task at hand, where the objects of interest are circular cells, the same is true that the most suitable structuring element (SE) is a **disk-shaped SE.** Dilation, here, helps to close the holes in the binary image.

Apply the dilation operator on the binary image BW, and Repeat the operation with 3 different sizes of the SE.

```
dilated1 = imdilate(BW_inv, SE1);
dilated2 = imdilate(BW_inv, SE2);
dilated3 = imdilate(BW_inv, SE3);
```

Visualize the obtained results.

```
figure;
subplot(1,3,1), imshow(dilated1), title('Dilation - SE size 3');
subplot(1,3,2), imshow(dilated2), title('Dilation - SE size 5');
subplot(1,3,3), imshow(dilated3), title('Dilation - SE size 7');
```



Discuss the results and explain the observed differences

- Dilation enlarges objects. Using larger structuring elements increases the object's size, filling gaps between objects.

Improvement

Are erosion and dilation operators sufficient to extract all the cells with a good precision?

- No,these fundamental morphological operations alone are not enough. A combination of the two such as by opening or closing, or advanced techniques like watershed segmentation might be required.

Which solution do you propose to improve the segmentation results?

- Opening, with a structuring element larger that the maximum debris but smaller than the smallest cell, would be ideal to remove the debris. Then, a closing operation or imfill can be used to close the hole in the cells.

Give a solution and test it on your image

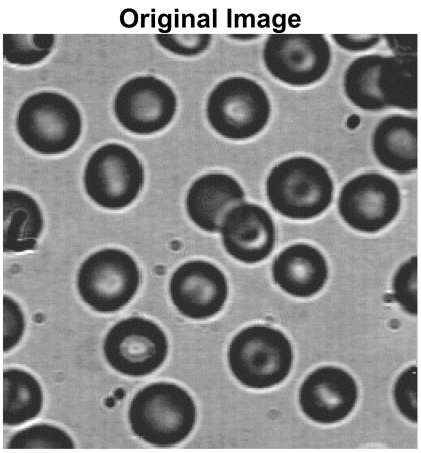
- We proposed the following preprocessing procedure. It helped us to try different operations in different sequence with different argument. After try and error method, we obtaind a better result with the following setting.

```
dbtype('preProcess.m')
```

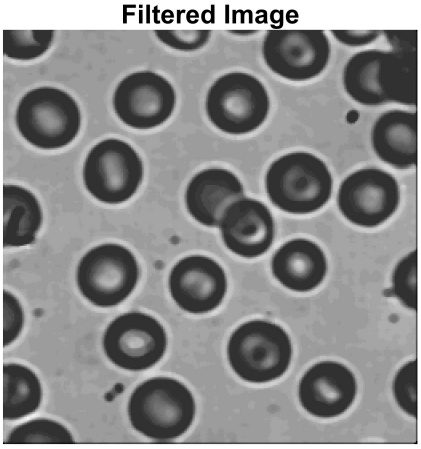
```
function processedImage = preProcess(imageFilename)

    % Step 1: Read the input image
    img = imread(imageFilename);
    figure; imshow(img); title('Original Image');

    % Step 2: Apply median filtering
    filterSize = 4;
    filtImg = medfilt2(img, [filterSize filterSize]);
    figure; imshow(filtImg); title('Filtered Image');

    % Step 3: Convert to binary image using Otsu's thresholding
    BW = imbinarize(filtImg, graythresh(img));

    % Step 4: Invert the binary image
    BW_inv = ~BW;
    figure; imshow(BW_inv); title('Inverted Binary Image');

    % Step 5: Fill holes in the binary image
    BW_filled = imfill(BW_inv, "holes");
    figure; imshow(BW_filled); title('Filled Binary Image');

    % Step 6: Apply morphological opening (remove small objects)
    BW_open = imopen(BW_filled, strel('disk', 2));
    figure; imshow(BW_open); title('After Opening Operation');

    % Step 7: Apply morphological closing (close small gaps)
    BW_close = imclose(BW_open, strel('disk', 2));
    figure; imshow(BW_close); title('After Closing Operation');

    % Step 8: Remove small connected components
    BW_area_open = bwareaopen(BW_close, 80, 8);
    figure; imshow(BW_area_open); title('Final Processed Image');

    % Return the final processed binary image
    processedImage = BW_area_open;
end
```

Call the function.

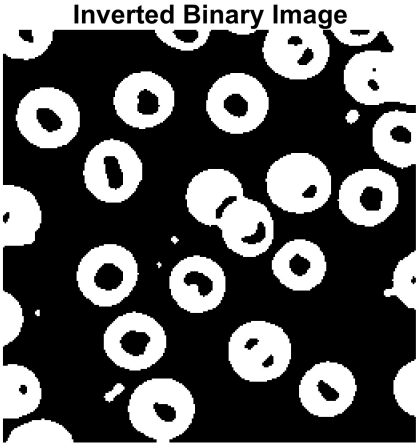
```
processedImage = preProcess('BLOODCELLS.bmp');
```



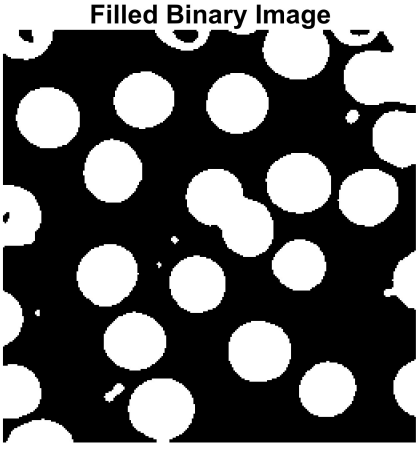
Filtered Image

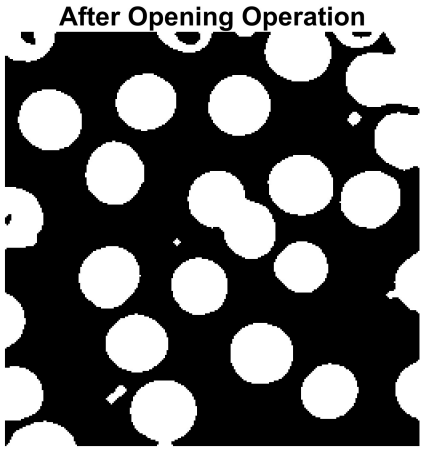


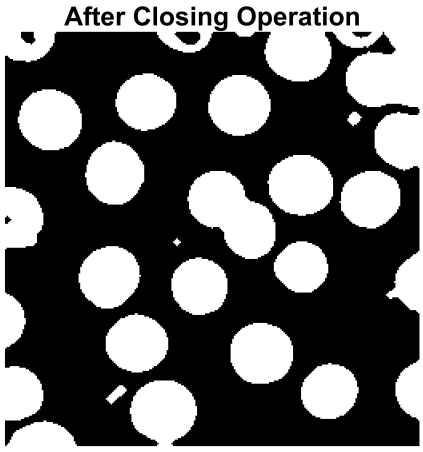
Inverted Binary Image

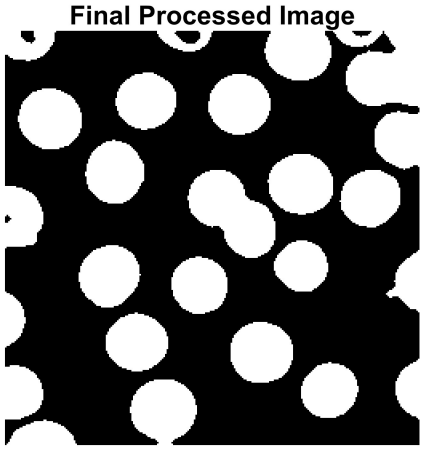


Filled Binary Image









Label connected components and counting

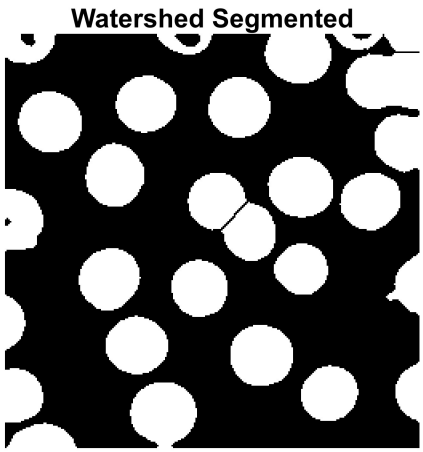
Find the appropriate function for labeling the image objects.

```
help bwlabel
```

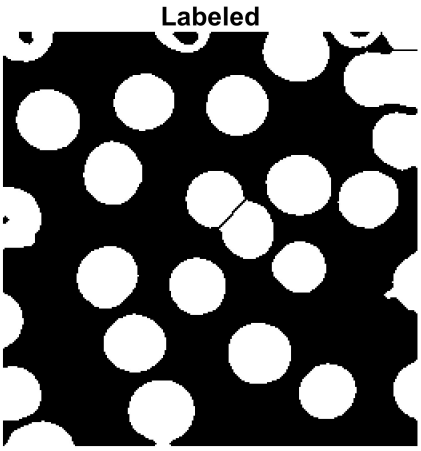
```
 bwlabel - Label connected components in 2-D binary image
    This MATLAB function returns the label matrix L that contains labels for
    the 8-connected objects found in BW.

    Syntax
      L = bwlabel(BW)
      L = bwlabel(BW,conn)
      [L,n] = bwlabel(___)

    Input Arguments
      BW - Binary image
        2-D numeric matrix | 2-D logical matrix
      conn - Pixel connectivity
        8 (default) | 4

    Output Arguments
      L - Label matrix
        matrix of nonnegative integers
      n - Number of connected objects
        nonnegative integer

    Examples
      Label Components Using 4-connected Objects

    See also bwconncomp, bwlabeln, bwselect, labelmatrix, label2rgb,
      regionprops

    Introduced in Image Processing Toolbox before R2006a
    Documentation for bwlabel
    Other uses of bwlabel
```

Use this function to label the segmented objects obtained after the enhancement step.

```
imSegmented = watershedSegmentation(processedImage);
figure; imshow(imSegmented); title('Watershed Segmented')
```




```
[connectedComponents, TotalCells] = bwlabel(imSegmented);
figure; imshow(connectedComponents); title('Labeled')
```



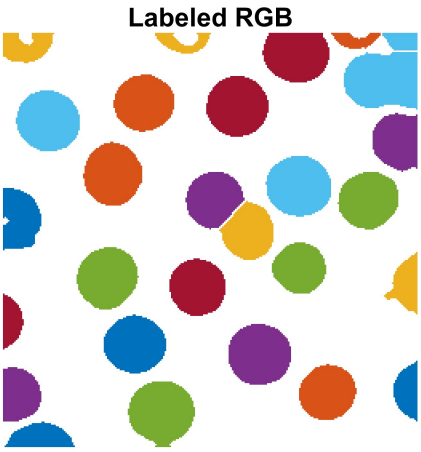
How many cells did you obtain?

```
disp("Total cell obtained: " + TotalCells)
```

```
Total cell obtained: 28
```

Show the labelled cells using different colors.

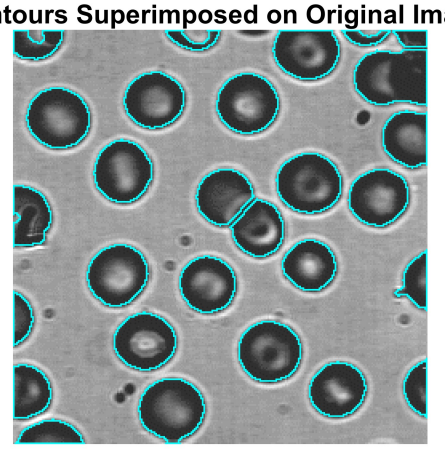
```
imLabeledRGB = label2rgb(connectedComponents, 'lines', 'white',
'shuffle');
figure; imshow(imLabeledRGB); title('Labeled RGB')
```



Extract the contours of all the segmented cells and superimpose them on the original image.

```
contours = bwperim(connectedComponents, 8);

superimposed = imoverlay(img, contours, "cyan"); % cyan contours
figure;
imshow(superimposed), title('Contours Superimposed on Original Image');
```

Contours Superimposed on Original Image



Open and study the help page of the regionprops function of Matlab or the equivalent function on Python.

```
help regionprops
```

```
  regionprops - Measure properties of image regions
     The regionprops function measures properties such as area, centroid, and
     bounding box, for each object (connected component) in an image.

     Syntax
       stats = regionprops(BW,properties)
       stats = regionprops(CC,properties)
       stats = regionprops(L,properties)
       stats = regionprops(regions,I,properties)
       stats = regionprops(outputFormat,___)

     Input Arguments
       BW - Binary image
         logical array
       CC - Connected components
         structure
       L - Label image
         numeric array | categorical array
       properties - Type of measurement
         comma-separated list of string scalars or character vectors |
         array of string scalars | cell array of character vectors | "all" |
         "basic" (default)
       I - Image to be measured
         grayscale image
       outputFormat - Output format
         "struct" (default) | "table"

     Output Arguments
       stats - Measurement values
         struct array | table

     Examples
       Calculate Centroids and Superimpose Locations on Image
       Estimate Center and Radii of Circular Objects and Plot Circles
       Create Binary Image with Subset of Filtered Objects
       Isolate Vertically Oriented Regions

     See also Image Region Analyzer, regionprops3, bwpropfilt, bwconncomp,
       bwferet, watershed, labelmatrix

     Introduced in Image Processing Toolbox before R2006a
```

```
Documentation for regionprops
Other uses of regionprops
```

Use this function to perform three measurements on one or more objects (areas, perimeter…)

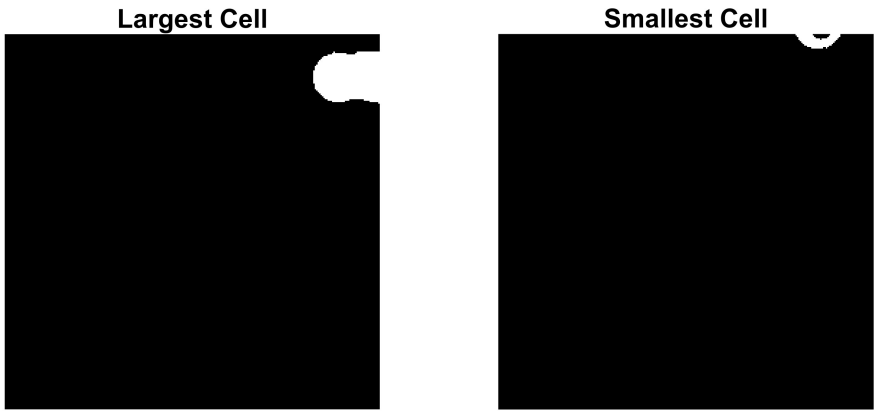
```
cellProps = regionprops("table", connectedComponents, "Area",
"Perimeter", "Centroid")
```

`cellProps` = 28×3 table

	Area	Centroid		Perimeter
		1	2	
1	447	15.0380	9.3960	97.6540
2	757	11.1400	117.1889	112.0440
3	320	5.7406	180.9406	76.8960
4	679	11.0147	226.7216	95.7680
5	492	23.8699	253.1301	97.2900
6	1196	28.8286	55.6037	119.8520
7	1147	65.7323	153.6992	118.1900
8	1126	69.1083	89.0142	116.7840
9	1098	82.9472	195.3761	115.0060
10	1052	88.7025	44.3717	112.3840
11	1289	99.6214	237.8813	129.7020
12	239	112.8326	6.6862	89.5830
13	991	122.1302	159.6226	108.9020
14	941	132.0733	104.3337	108.7460
15	1158	147.0130	46.5112	117.5280
16	873	154.2027	125.2486	105.3800
17	1201	160.8201	201.6278	119.7360
18	1090	183.8972	14.8055	118.3540
19	1222	185.4542	96.4092	121.2660
20	832	185.9075	147.6514	99.6400
21	963	203.3136	225.4330	107.7860
22	197	220.7665	5.1269	69.9250
23	1060	228.9425	105.1160	113.1200
24	1431	238.1104	30.0224	145.5710
25	849	246.7020	68.3522	108.6120
26	207	249.9614	5.6329	56.0630
27	452	252.4580	157.8606	97.8700
28	407	253.2383	224.5651	85.4060

Show in one figure the largest cell and on another figure the smallest one.

```
% Get the indices
allAreas = [cellProps.Area];
[~, largestIdx] = max(allAreas);
[~, smallestIdx] = min(allAreas);

% Display largest and the smallest
largestCellMask = ismember(connectedComponents, largestIdx);
smallestCellMask = ismember(connectedComponents, smallestIdx);

figure;
subplot(121); imshow(largestCellMask), title('Largest Cell');
subplot(122); imshow(smallestCellMask), title('Smallest Cell');
```

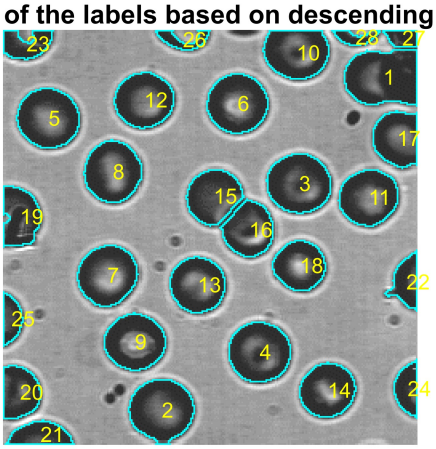


The above figure is a little bit misleading since that is not the actuall truth. The indicated labels are, for the largest cell, an overlapping of two cells that were segmented as one, and for the smallest cell, a cropped part of a cell on the border of the image. Hence, we used a visualization approach to show the order of the area of the labels to help the expert decide the largest and smallest cell from the overall perspective.

```
% Sort the areas in descending order and get the indices
[sortedAreas, sortIndexes] = sort(allAreas, 'descend');

% Step 4: Overlay the numbers on the image based on sorted area
figure; imshow(superimposed); title('Order of the labels based on
descending area') % hold on;
for i = 1:TotalCells
    % Get the index of the current object in the sorted order
    currentLabel = sortIndexes(i);

    % Get the centroid of the current object from the table
```

```
    centroid = cellProps(currentLabel, :).Centroid;

    % Overlay the number corresponding to the object's order in the
sorted list
    text(centroid(1), centroid(2), num2str(i), 'Color', 'Yellow',
'FontSize', 8);
end
```

Order of the labels based on descending area



Alternative way, to apprach this problem using: Circular Hough Transform.

```
[centers, radii] = imfindcircles(processedImage, [15 40], ...
                                 'ObjectPolarity', 'bright', ...
                                 'Sensitivity', 0.88, ...
                                 'EdgeThreshold', 0.25);

% Display original image and detected circles
figure;
imshow(img, 'Parent', gca);
hold on;
viscircles(centers, radii, 'Color', 'r');
hold off;
```