LAB 1 - OBJECT COUNTING ON IMAGES

Fromsa T. Negasa, Nadeer Hasan

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

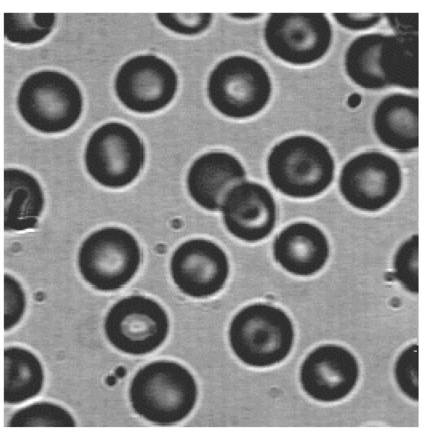
The Image Processing Toolbox of Matlab or Python Image Library.

III. Procedure

1. Load and show the image

Read the image bloodcells

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

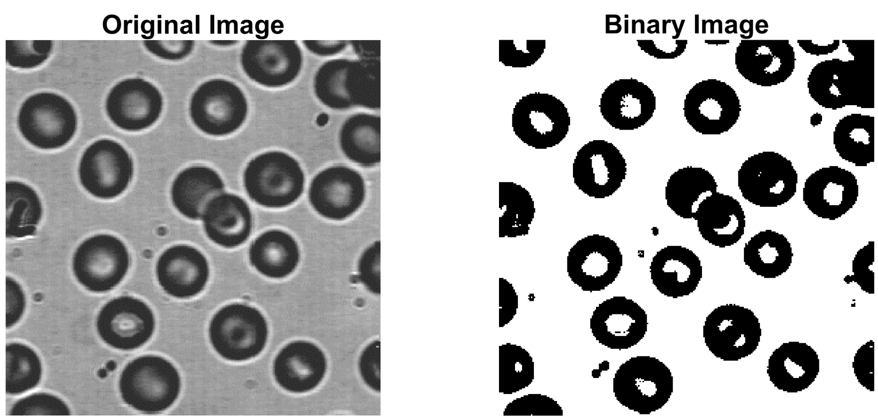


Transform this image into a binary image BW using a thresholding process or a dedicated function of Matlab or Python.

```
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');
```



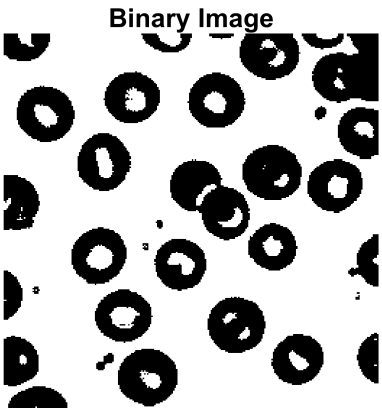
The morphological operator operates on the white pixels in the binary image. What do you observe on the image? What's the solution of this problem?

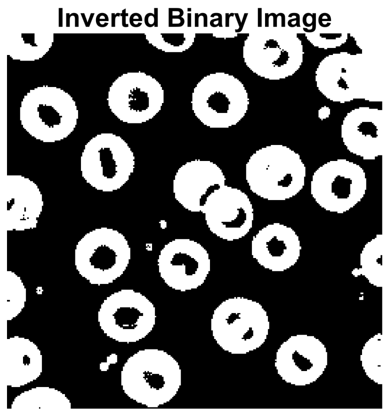
- Morphological operators typically work on white pixels (foreground), and this could be problematic if the objects are represented by dark pixels.
- **Solution**: Invert the binary image if the foreground (cells) is dark.

```
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.
    IM2 = imerode(IM,SE) erodes the grayscale, binary, or packed binary image
    IM, returning the eroded image, IM2.  SE is a structuring element
    object, or array of structuring element objects, returned by the
    STREL or OFFSETSTREL functions.

    If IM is logical and the structuring element is flat, imerode
    performs binary erosion; otherwise it performs grayscale erosion.  If
    SE is an array of structuring element objects, imerode performs
    multiple erosions of the input image, using each structuring element
    in succession.

    IM2 = imerode(IM,NHOOD) erodes the image IM, where NHOOD is an array
    of 0s and 1s that specifies the structuring element.  This is
    equivalent to the syntax imerode(IM,STREL(NHOOD)).  imerode uses this
    calculation to determine the center element, or origin, of the
    neighborhood:  FLOOR((SIZE(NHOOD) + 1)/2).

    IM2 = imerode(IM,SE,PACKOPT,M) or imerode(IM,NHOOD,PACKOPT,M) specifies
    whether IM is a packed binary image and, if it is, provides the row
    dimension, M, of the original unpacked image.  PACKOPT can have
    either of these values:

        'ispacked'    IM is treated as a packed binary image as produced
                      by BWPACK.  IM must be a 2-D uint32 array and SE
                      must be a flat 2-D structuring element.  If the
```

```
                  value of PACKOPT is 'ispacked', SHAPE must be
                  'same'.

    'notpacked'   IM is treated as a normal array.  This is the
                  default value.

If PACKOPT is 'ispacked', you must specify a value for M.

IM2 = imerode(...,SHAPE) determines the size of the output image.
SHAPE can have either of these values:

    'same'        Make the output image the same size as the input
                  image.  This is the default value.  If the value of
                  PACKOPT is 'ispacked', SHAPE must be 'same'.

    'full'        Compute the full erosion.

Class Support
-------------
IM can be numeric or logical and it can be of any dimension.  If IM is
logical and the structuring element is flat, then output will be
logical; otherwise the output will have the same class as the input. If
the input is packed binary, then the output is also packed binary.

Example 1
---------
% Erode the binary image in text.png with a vertical line
    originalBW = imread('text.png');
    se = strel('line',11,90);
    erodedBW = imerode(originalBW,se);
    imshowpair(originalBW,erodedBW,'montage');

Example 2
---------
% Erode the grayscale image in cameraman.tif with a rolling ball
    originalI = imread('cameraman.tif');
    se = offsetstrel('ball',5,5);
    erodedI = imerode(originalI,se);
    imshowpair(originalI,erodedI,'montage')

Example 3
---------
% Erode the mristack volume, using a cube of side 3
    % Create a binary volume
    load mristack
    BW = mristack < 100;

    % Erode the volume with a cubic structuring element
    se = strel('cube',3);
    erodedBW = imerode(BW, se);

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

Documentation for imerode
Other uses of imerode
```

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

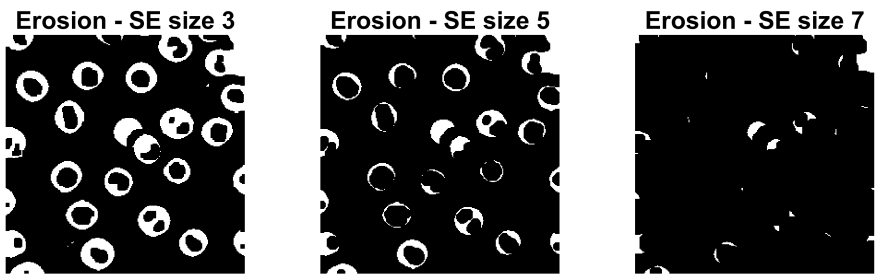
- Given that the image contains circular cells, the most appropriate structuring element (SE) to use is a **disk-shaped SE.** This shape matches the geometry of the objects (circular cells and the small debris) in the image and is optimal for preserving their roundness during morphological operations.

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.
    IM2 = imdilate(IM,SE) dilates the grayscale, binary, or packed binary
    image IM, returning the dilated image, IM2.  SE is a structuring
    element object, or array of structuring element objects, returned by
    the STREL or OFFSETSTREL functions.

    If IM is logical (binary), then the structuring element must be flat
    and imdilate performs binary dilation.  Otherwise, it performs
    grayscale dilation.  If SE is an array of structuring element
    objects, imdilate performs multiple dilations, using each
    structuring element in SE in succession.

    IM2 = imdilate(IM,NHOOD) dilates the image IM, where NHOOD is a
    matrix of 0s and 1s that specifies the structuring element
    neighborhood.  This is equivalent to the syntax imdilate(IM,
    STREL(NHOOD)).  imdilate determines the center element of the
    neighborhood by FLOOR((SIZE(NHOOD) + 1)/2).

    IM2 = imdilate(IM,SE,PACKOPT) or imdilate(IM,NHOOD,PACKOPT) specifies
    whether IM is a packed binary image.  PACKOPT can have either of
    these values:

        'ispacked'    IM is treated as a packed binary image as produced
                      by BWPACK.  IM must be a 2-D uint32 array and SE
                      must be a flat 2-D structuring element.  If the
                      value of PACKOPT is 'ispacked', SHAPE must be
                      'same'.

        'notpacked'   IM is treated as a normal array.  This is the
                      default value.

    IM2 = imdilate(...,SHAPE) determines the size of the output image.
    SHAPE can have either of these values:

        'same'        Make the output image the same size as the input
                      image.  This is the default value.  If the value of
                      PACKOPT is 'ispacked', SHAPE must be 'same'.

        'full'        Compute the full dilation.

    Class Support
    -------------
    IM can be logical or numeric and must be real and nonsparse.  It can
    have any dimension.  The output has the same class as the input. If
    the input is packed binary, then the output is also packed binary.

    Example 1
    ---------
    % Dilate the binary image in text.png with a vertical line
        originalBW = imread('text.png');
        se = strel('line',11,90);
        dilatedBW = imdilate(originalBW,se);
        figure, imshow(originalBW), figure, imshow(dilatedBW)

    Example 2
    ---------
    % Dilate the grayscale image in cameraman.tif with a rolling ball
```

```
    originalI = imread('cameraman.tif');
    se = offsetstrel('ball',5,5);
    dilatedI = imdilate(originalI,se);
    figure, imshow(originalI), figure, imshow(dilatedI)

Example 3
---------
% Determine the domain of the composition of two flat structuring
% elements by dilating the scalar value 1 with both structuring
% elements in sequence, using the 'full' option
    se1 = strel('line',3,0);
    se2 = strel('line',3,90);
    composition = imdilate(1,[se1 se2],'full');

Example 4
---------
% Dilate two points in 3D space using spherical structuring elements,
% so that they take the shape of spheres
    % Create a logical 3D volume with two points.
    BW = false(100,100,100);
    BW(25,25,25) = true;
    BW(75,75,75) = true;

    % Dilate the 3D volume by a spherical structuring element
    se = strel('sphere',25);
    dilatedBW = imdilate(BW, se);

    % Visualize the dilated image volume
    figure, isosurface(dilatedBW)

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

Documentation for imdilate
Other uses of imdilate
```

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

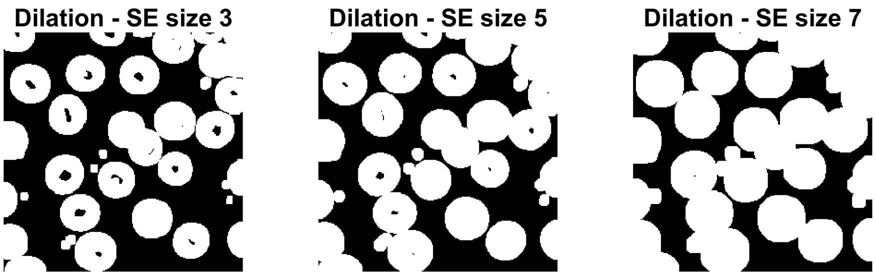
- For the task at hand, where the objects of interest are circular cells, the same is true that the most suitable structuring element (SE) is a **disk-shaped SE.** Dilation, here, helps to close the holes in the binary image.

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.`

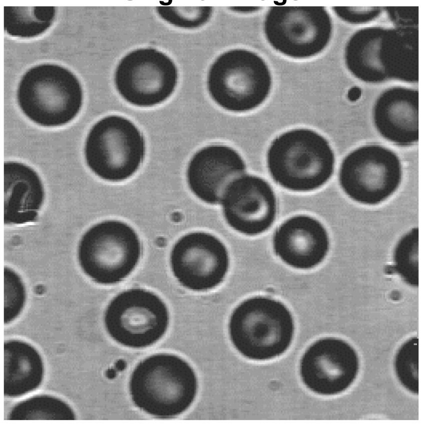
```
dbtype('preProcess.m')
```

```
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
```

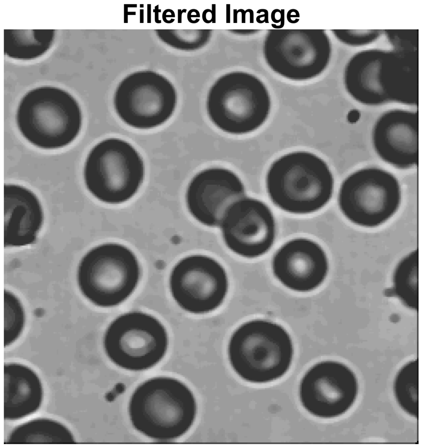
Call the function.

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

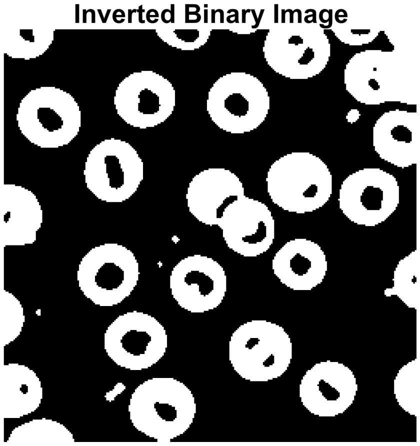
Original Image



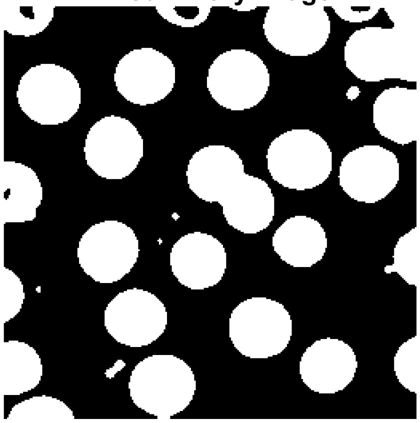
Filtered Image



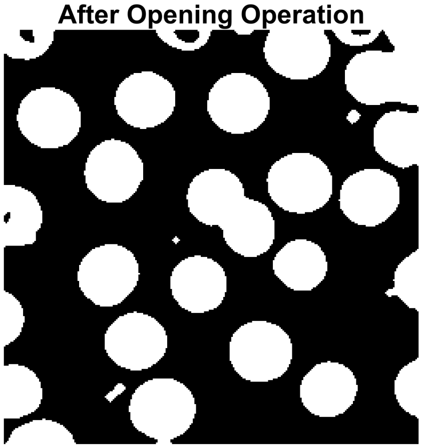
Inverted Binary Image



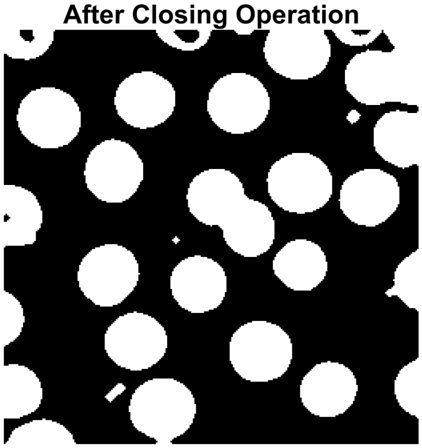
Filled Binary Image



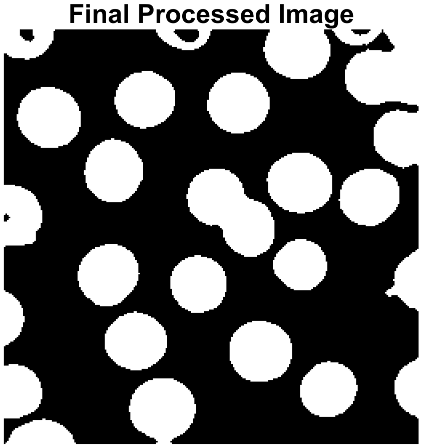
After Opening Operation



After Closing Operation



Final Processed Image



Label connected components and counting

Find the appropriate function for labeling the image objects.

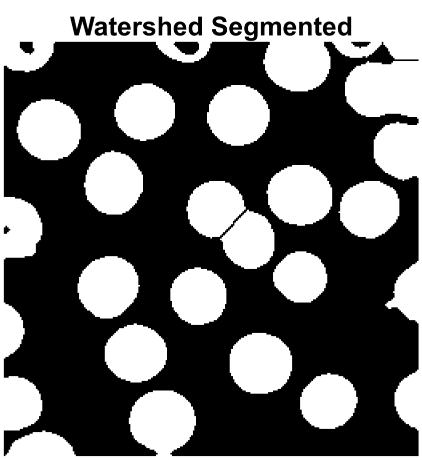
```
help bwlabel
```

```
 bwlabel Label connected components in 2-D binary image.
    L = bwlabel(BW,N) returns a matrix L, of the same size as BW,
    containing labels for the connected components in BW. N can have a
    value of either 4 or 8, where 4 specifies 4-connected objects and 8
    specifies 8-connected objects; if the argument is omitted, it defaults
    to 8.

    The elements of L are integer values greater than or equal to 0.  The
    pixels labeled 0 are the background.  The pixels labeled 1 make up one
    object, the pixels labeled 2 make up a second object, and so on.

    [L,NUM] = bwlabel(BW,N) returns in NUM the number of connected objects
    found in BW.

    Note: On the use of bwlabel, BWLABELN, BWCONNCOMP, and REGIONPROPS
    -----------------------------------------------------------------
    The functions bwlabel, BWLABELN, and BWCONNCOMP all compute connected
    components for binary images.  BWCONNCOMP is the most recent addition
    to the Image Processing Toolbox and is intended to replace the use
    of bwlabel and BWLABELN.  It uses significantly less memory and is
    sometimes faster than the older functions.

                 Input  Output            Memory   Connectivity
                 Dim    Form              Use
                 ----------------------------------------------
    bwlabel      2-D    Double-precision  High     4 or 8
                        label matrix

    BWLABELN     N-D    Double-precision  High     Any
                        label matrix

    BWCONNCOMP   N-D    CC struct         Low      Any

    To extract features from a binary image using REGIONPROPS using the
    default connectivity, just pass BW directly into REGIONPROPS, i.e.,
```

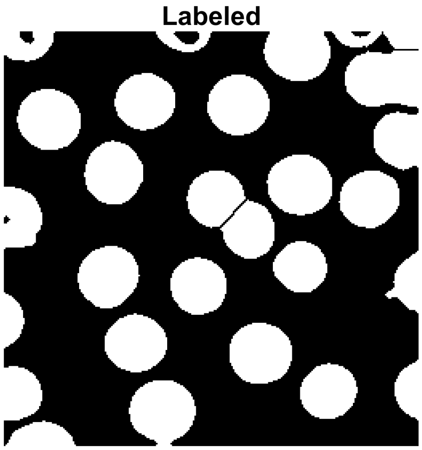
```
REGIONPROPS(BW).

To compute a label matrix having a more memory-efficient data type
(e.g., uint8 versus double), use the LABELMATRIX function on the output
of BWCONNCOMP. See the documentation for each function for more information.

Class Support
-------------
BW can be logical or numeric, and it must be real, 2-D, and nonsparse.
L is of class double.

Example
-------
    BW = logical([1 1 1 0 0 0 0 0
                  1 1 1 0 1 1 0 0
                  1 1 1 0 1 1 0 0
                  1 1 1 0 0 0 1 0
                  1 1 1 0 0 0 1 0
                  1 1 1 0 0 0 1 0
                  1 1 1 0 0 1 1 0
                  1 1 1 0 0 0 0 0]);
    L = bwlabel(BW,4)
    [r,c] = find(L == 2)

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

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')
```



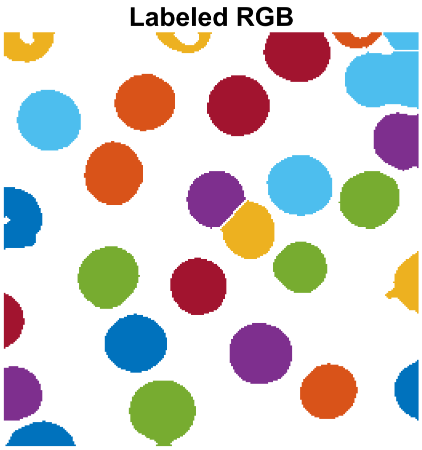
How many cells did you obtain?

```
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')
```

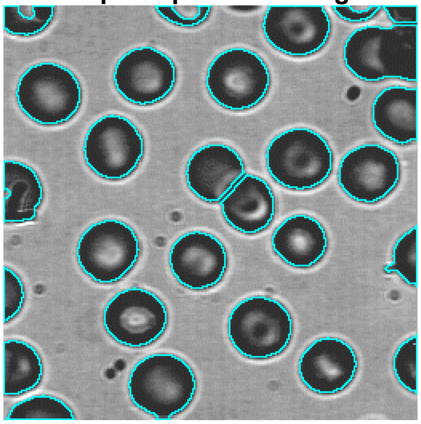


Extract the contours of all the segmented cells and superimpose them on the original image.

```
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.
    STATS = regionprops(BW,PROPERTIES) measures a set of properties for
    each connected component (object) in the binary image BW, which must be
    a logical array; it can have any dimension.

    STATS = regionprops(CC,PROPERTIES) measures a set of properties for
    each connected component (object) in CC, which is a structure returned
    by BWCONNCOMP.

    STATS = regionprops(L,PROPERTIES) measures a set of properties for each
    labeled region in the label matrix L. L can be numeric or categorical.
    When L is numeric, positive integer elements of L correspond to
    different regions. For example, the set of elements of L equal to 1
    corresponds to region 1; the set of elements of L equal to 2
    corresponds to region 2; and so on. When L is categorical, each
    category corresponds to different region.

    STATS = regionprops(...,I,PROPERTIES) measures a set of properties for
    each labeled region in the image I. The first input to regionprops (BW,
    CC, or L) identifies the regions in I.  The sizes must match: SIZE(I)
    must equal SIZE(BW), CC.ImageSize, or SIZE(L).

    STATS = regionprops(OUTPUT, ___) returns the output in a variable
    of type specified by OUTPUT. OUTPUT must be one of the
    following strings (string can be abbreviated):

    'struct' -  When 'struct' is specified, output STATS is an array of
                structures with length equal to the number of objects in
                BW, CC.NumObjects, or max(L(:)). The fields of the
                structure array denote different properties for each
                region, as specified by PROPERTIES. If OUTPUT is not
                specified, 'struct' is selected by default.

    'table' -   When 'table' is specified, output STATS is a MATLAB table
                with height (number of rows) equal to the number of objects
```

```
            in BW, CC.NumObjects, or max(L(:)). The variables of the
            table denote different properties for each region, as
            specified by PROPERTIES. See help for 'table' in MATLAB for
            additional methods for the table.

PROPERTIES can be a comma-separated list of strings or character
vectors, a cell array containing strings or character vectors,
'all', or 'basic'. The set of valid measurement strings or character
vectors includes:

Shape Measurements

  'Area'              'EulerNumber'        'MinorAxisLength'
  'BoundingBox'       'Extent'             'Orientation'
  'Centroid'          'Extrema'            'Perimeter'
  'ConvexArea'        'FilledArea'         'PixelIdxList'
  'ConvexHull'        'FilledImage'        'PixelList'
  'ConvexImage'       'Image'              'Solidity'
  'Circularity'       'MaxFeretProperties' 'SubarrayIdx'
  'Eccentricity'      'MinFeretProperties'
  'EquivDiameter'     'MajorAxisLength'

Pixel Value Measurements (requires grayscale image as an input)

  'MaxIntensity'
  'MeanIntensity'
  'MinIntensity'
  'PixelValues'
  'WeightedCentroid'

Property strings or character vectors are case insensitive and can be
abbreviated.

If PROPERTIES is set to 'all', regionprops returns all of the Shape
measurements. If called with a grayscale image, regionprops also
returns Pixel value measurements. If PROPERTIES is not specified or if
it is set to 'basic', these measurements are computed: 'Area',
'Centroid', and 'BoundingBox'.

Perimeter and Circularity should be used on a label matrix or binary
image with contiguous regions. Otherwise, 'Perimeter' gives unexpected
results on discontiguous regions which also affects the result of
'Circularity'. Circularity is also not recommended for very small
regions such as a 3x3 square, as in such cases the results might exceed
the circularity value for a perfect circle which is 1.

If 'MaxFeretProperties' is selected, the function will output the
following properties related to the maximum Feret Diameter -
   MaxFeretDiameter   - Maximum Feret diameter length.
   MaxFeretAngle      - Angle of maximum Feret diameter with respect to X
                        axis in degrees.
   MaxFeretCoordinates- Endpoint coordinates of maximum Feret diameter.

If 'MinFeretProperties' is selected, the function will output the
following properties related to the minimum Feret Diameter -
   MinFeretDiameter   - Minimum Feret diameter length.
   MinFeretAngle      - Angle of minimum Feret diameter with respect to X
                        axis in degrees.
   MinFeretCoordinates- Endpoint coordinates of minimum Feret diameter.

If the input is categorical, a property 'LabelName' is added to the
output along with any of the above selected properties.

Note that negative-valued pixels are treated as background
```

and pixels that are not integer-valued are rounded down.

Class Support

If the first input is BW, BW must be a logical array and it can have any dimension. If the first input is CC, CC must be a structure returned by BWCONNCOMP. If the first input is L, L must be categorical or numeric array which must be real, nonsparse, and contain integers. L can have any numeric class and any dimension.

Note on Terminology
----
regionprops can be used on contiguous regions and discontiguous regions.

Contiguous regions are also called "objects", "connected components", and "blobs". A label matrix containing contiguous regions might look like this:

```
    1 1 0 2 2 0 3 3
    1 1 0 2 2 0 3 3
```

Elements of L equal to 1 belong to the first contiguous region or connected component, elements of L equal to 2 belong to the second connected component, etc.

Discontiguous regions are regions that may contain multiple connected components. A label matrix containing discontiguous regions might look like this:

```
    1 1 0 1 1 0 2 2
    1 1 0 1 1 0 2 2
```

Elements of L equal to 1 belong to the first region, which is discontiguous and contains two connected components. Elements of L equal to 2 belong to the second region, which is a single connected component.

Example 1

% Estimate the center and radii of the circular objects in the image
% and plot the circles on the image.

```
      a = imread('circlesBrightDark.png');
      bw = a < 100;
      imshow(bw)
      title('Image with Circles')

      stats = regionprops('table',bw,'Centroid', ...
                          'MajorAxisLength','MinorAxisLength');

      % Get centers and radii of the circles
      centers = stats.Centroid;
      diameters = mean([stats.MajorAxisLength stats.MinorAxisLength],2);
      radii = diameters/2;

      % Plot the circles
      hold on
      viscircles(centers,radii);
      hold off
```

Example 2

% Label the connected pixel components in the text.png image, compute

```
% their centroids, and superimpose the centroid locations on the
% image.

      BW = imread('text.png');
      s  = regionprops(BW, 'centroid');
      centroids = cat(1, s.Centroid);
      imshow(BW)
      hold on
      plot(centroids(:,1), centroids(:,2), 'b*')
      hold off

See also bwconncomp,bwlabel,bwlabeln,bwpropfilt,ismember,labelmatrix,watershed,regionprops3.

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")
```

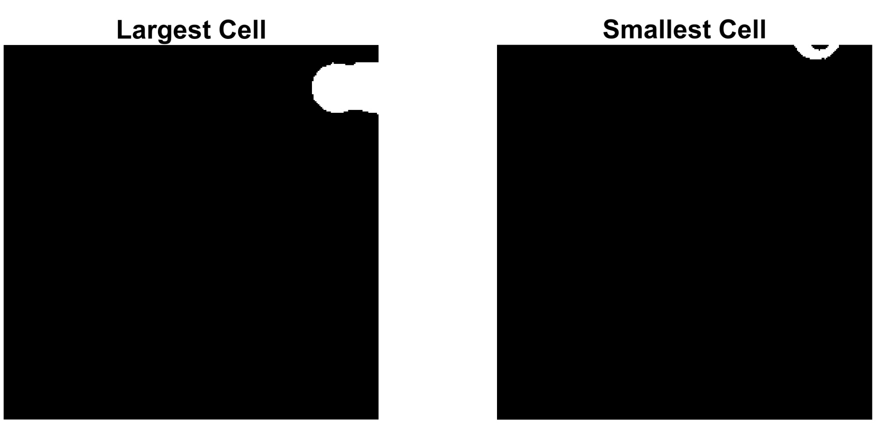
cellProps = 28×3 table

	Area	Centroid		Perimeter
1	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

	Area	Centroid		Perimeter
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.

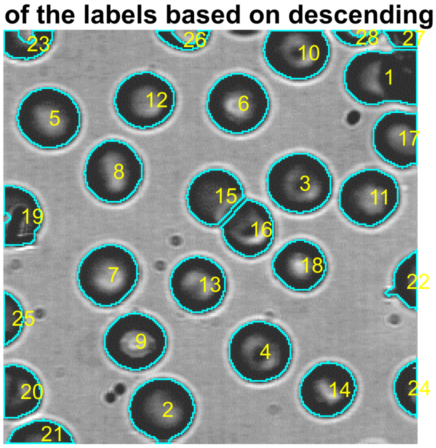
```
% Get the indices
allAreas = [cellProps.Area];
[~, largestIdx] = max(allAreas);
[~, smallestIdx] = min(allAreas);

% 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;
```