Image Segmentation Using Watershed Algorithm and its Application in Cell Analysis

Digital Image Processing Project
(CS1553)

V Semester

Submitted By:
Rajat Baxi
189301047
CSE-C

Department of Computer Science & Engineering

Introduction

Segmentation is one of the most important problem in image processing. It consists of constructing asymbolic representation of the image: the image is described as homogeneous areas according to one or several a priori attributes. In the literature, we can find various segmentation algorithms. The first method appeared during the sixties and then different algorithms have been constantly developed. The goal of the image segmentation process is to define areas within the image. Many image segmentation techniques have been developed. In Edge-based segmentation, image edges are detected and then linked into contours that represent the boundaries of image objects. In Clustering-based segmentation, image pixels are sorted in increasing order as a histogram according to their intensity values. In Region-based Segmentation, the goal is the detection of regions that satisfy a certain predefined homogeneity threshold. Two sub classes are Region Growing and Region Splitting/Merging. The watershed algorithm is based on region growing technique. The watershed algorithm can find contiguous edges in an image accurately but suffers from the over- segmentation problem.

There are three separate tasks that are done using watershed algorithm which are shown through this project:

I. The implementation of Watershed Algorithm that clearly shows how useful this algorithm is for segmenting images into background and foreground in situations that are difficult for other algorithms. A common example is the use of coins next to each other on a table.
II. The implementation of Watershed Algorithm by providing our own custom seeds that allow us to manually start where the valleys of the watersheds go.
III. The application of Watershed Algorithm in Cell Nuclei analysis by using an Osteosarcoma (bone cancer) cells image.

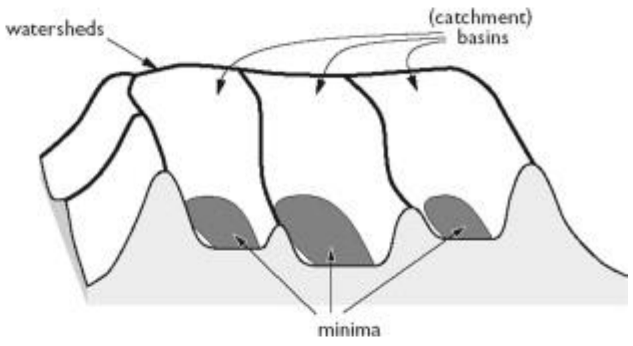
All image pixels are previously sorted according to the selected criterion. Then the Watershed Segmentation (WS) is applied on the sorted pixels to segment the image into as many regions as the number of *markers* interactively placed by the user.

To understand the watershed, one can think of an image as a surface where the bright pixels represent mountaintops and the dark pixels valleys. The surface is punctured in some of the valleys, and then slowly submerged into a water bath. The water will pour in each puncture and start to fill the valleys. However, the water from different punctures is not allowed to mix, and therefore the dams need to be built at the points of first contact. These dams are the boundaries of the water basins, and also the boundaries of image objects.

Literature Review

Watershed Algorithm

In watershed segmentation an image is regarded as a topographic landscape with ridges and valleys. The elevation values of the landscape are typically defined by the Gray values of the respective pixels or their gradient magnitude. Based on such a 3D representation the watershed transform decomposes an image into *catchment basins*. For each local minimum, a catchment basin comprises all points whose path of steepest descent terminates at this minimum. Watersheds separate basins from each other. The watershed transform decomposes an image completely and thus assigns each pixel either to a region or a watershed. With noisy medical image data, a large number of small regions arises. This is known as the "over-segmentation" problem.

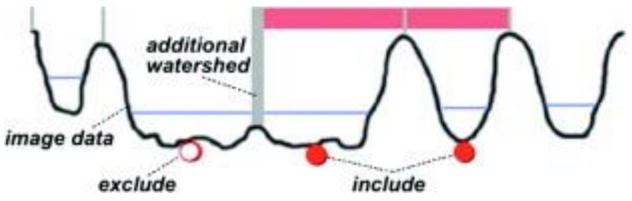


Over-segmentation: The over segmentation problem in the watershed transformation is mostly due to the noise and quantization error. To eliminate the effect of noise or quantization error on the final results, the watershed transformation is applied on the gradient of the original image. To decrease the over segmentation of watershed based techniques, several approaches have been proposed in the literature, we can cite for example techniques based on markers (Meyer and Beucher, 1990), region merging methods (Vincent and Soille, 1991), scale space approaches (Jackway, 1999), methods based on partial differential equations for image denoising or edge enhancement (Weickert, 2001), wavelet techniques combined with a watershed transformation(Jung and Scharcanski, 2005), etc.

The most widespread variant employs the gradient image as the basis for the watershed transform. Gradient magnitude, however, is strongly sensitive to image noise. Therefore, appropriate filtering is essential. There are many variants how the watershed transform may be used as a basis for a general segmentation approach. The "over-segmentation" problem may be solved by some criteria for merging regions. The user must be provided with some facilities to influence the merging process.

*Merging Basins***,** the decomposition of an image into regions is the basis for merging them. In the metaphorical sense of a landscape, catchment basins are merged at their watershed locations by flooding them. While some regions merge early (with low flooding level), other regions are merged later. In order to support interactive merging, Hahn and Peitgen introduced a *merge tree*. This tree consists of the original catchment basins as leaves and of intermediate nodes that represent *merging events*. A merging event is characterized by the nodes that are merged and by the flood level that is necessary for merging. As a first step, a certain amount of flooding may be applied ("pre-flooding" which may already be sufficient for segmenting the target structure).

Marker-based Watershed: Often however, no examined flooding levels sufficient to segment target structures. Therefore, the user may specify image locations that belong to the target structure (include points), or that do not belong the target structure (exclude points). If the user specifies an include point and an exclude point, an additional watershed is constructed at the maximum level between them. The merge tree is traversed such that each region contains either include points or exclude points but not both. This interaction style is called *marker-based watershed segmentation*. There are many variants of the watershed transform. For example, merging may consider also gradient information or other criteria for homogeneity. A frequently used variant is to merge regions where the difference of the mean Gray value is below a threshold. This process can be carried out iteratively and results also in a hierarchical merging tree.



Topological Gradient and Contour/Edge Detection

We consider in this section the problem of denoising an image and preserving features such as edges. According to the previous section, the topological asymptotic analysis provides the location of the edges as they are precisely defined as the most negative points of the topological gradient. For a better edge preservation, one has to threshold the topological gradient with a small enough coefficient. In the other case, if the thresholding coefficient is set to a large value, then the edges obtained will be thick, leading to an over smoothing and a loss of an important edge information and then a degradation of the restored image. Finally, to speed up the computations, a spectral method based on the discrete cosine transform has been used for the resolution of the direct and adjoint problems. Since the coefficient c is equal to a constant c0 except on edges, then the discrete cosine transform is a good preconditioner for the conjugate gradient method. The complexity of the restoration algorithm is O(NlogN) where N is the number of pixels of the image. Some comparisons about the computation times with other classical methods are presented in Jaafar Belaid et al. (2008).

Topological Gradient and WATERSHED

As mentioned previously, the topological gradient is much less sensitive to noise and small variations of the image, than the Euclidean and morphological gradients. This is due to the fact that the topological gradient evaluates in a global way whether a pixel is a part of an edge or not, compared to the Euclidean gradient which has more local properties. On the other hand, as the morphological gradient corresponds in a certain way to the modulus of the Euclidean gradient, then it will be easy to conclude that the topological gradient provides the best identification of the main edges of the processed image, and the over segmentation obtained.

Since the topological gradient is the best tool for preserving the most important edges and eliminating all the other insignificant ones, then the first idea was to replace the morphological gradient by a topological gradient in order to minimize the set of minima of I, leading to better segmentation results. Our algorithm is then composed of two different and separate steps: the first one consists of detecting the main edges of the image using the topological gradient restoration process. Then, the second step consists of applying the watershed algorithm using the topological gradient determined in the first step, instead of the morphological gradient classically used in watershed algorithms. The first results obtained are very promising. The computation times are then more or less similar between the two methods (this is due to the fact that the computation time may depends of the processed image). We recall here that the topological gradient algorithm is solved with a O(NlogN) complexity and that the watershed algorithm runs in a linear time with respect to the number N of pixels of the image.

Methodology

There are three separate tasks that are done using watershed algorithm which are shown through this project:

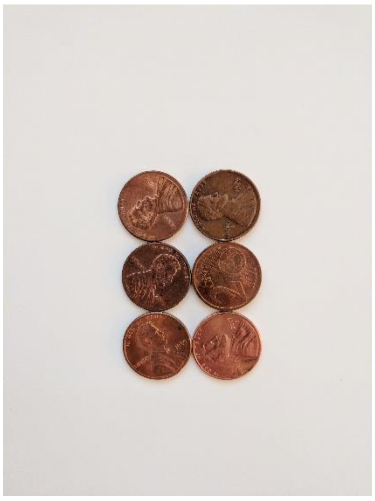
I. The implementation of Watershed Algorithm that clearly shows how useful this algorithm is for segmenting images into background and foreground in situations that are difficult for other algorithms. A common example is the use of coins next to each other on a table.

II. The implementation of Watershed Algorithm by providing our own custom seeds that allow us to manually start where the valleys of the watersheds go.

III. The application of Watershed Algorithm in Cell Nuclei analysis by using an Osteosarcoma (bone cancer) cells image.

I. Watershed when Touching/Overlapping Objects in Image

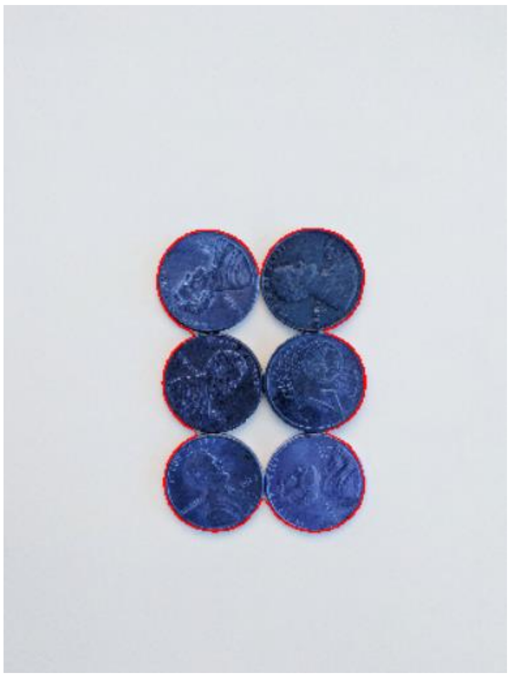
The **watershed algorithm** is a classic algorithm used for segmentation and is especially useful when extracting *touching* or *overlapping* objects in images, such as the coins in the image below:



Using traditional image processing methods such as thresholding and contour detection, we would be unable to extract each *individual* coin from the image — but by leveraging the watershed algorithm, we are able to detect and extract each coin without a problem.

We first see the problem with basic thresholding and contour detection if they are used on such images.

i. Convert image to grayscale image.
ii. We have too much detail in this image, including light, the face edges on the coins, and too much detail in the background. We use Median Blur Filtering to blur the image a bit, which will be useful later on when we threshold.
iii. Then we use Simple binary Thresholding on this image.
iv. And Finally, we find Contours and see the results obtained.



We can clearly see that this naïve approach is not able to detect any single coin correctly rather it treats all the six coins as one. The reason for this problem arises from the fact that coin borders are *touching* each other in the image — thus, the find Contours function only sees the coin groups as a *single* object when in fact they are *multiple, separate* coins.

This clearly indicates the problems with such an approach.

However, we can see that we get far better results when we use watershed algorithm.

Using Watershed Algorithm:

To apply the watershed algorithm, we need to define *markers* which correspond to the objects in our image. These markers can be either user-defined or we can apply image processing techniques (such as thresholding) to find the markers for us. When applying the watershed algorithm, it's absolutely critical that we obtain accurate markers.

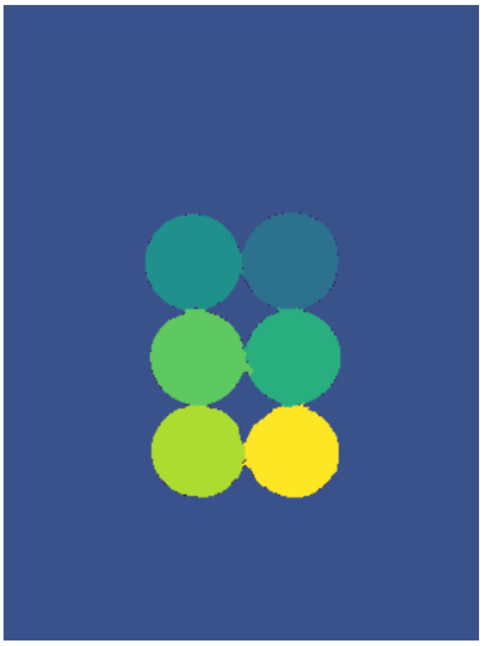
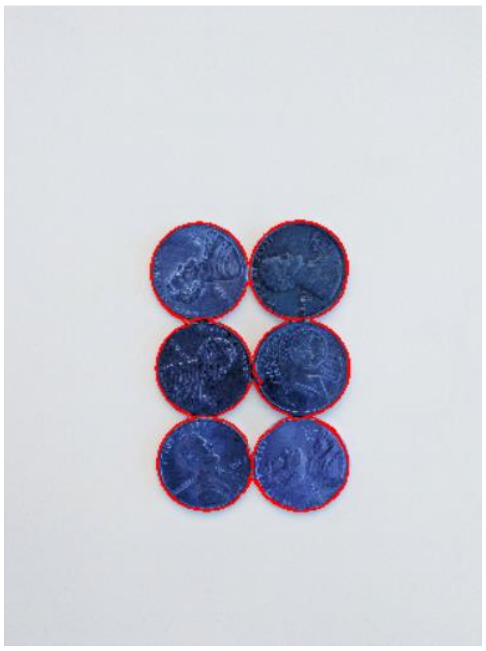
Given our markers, we can compute the Euclidean Distance Transform and pass the distance map to the watershed function itself, which "floods" valleys in the distance map, starting from the initial markers and moving outwards. Where the "pools" of water meet can be considered boundary lines in the segmentation process.

The output of the watershed algorithm is a set of labels, where each label corresponds to a unique object in the image. From there, all we need to do is loop over each of the labels individually and extract each object.

Steps are as follows:

i. Read the image and convert to grayscale.
ii. Apply Median Blur filtering to blur the extraneous details.
iii. Apply Threshold (Inverse Binary with Otsu thresholding) because Otsu has been tested out as the best thresholding technique for Watershed algorithm.
iv. Remove Noise using Morphological Operations.

v. Grab Background from the image that we are sure of using dilate function
vi. Now find the Sure Foreground by computing the Euclidean Distance Transform (EDT) via the distance transform function. As the name suggests, this function computes the Euclidean distance to the closest zero (i.e., background pixel) for each of the foreground pixels.
vii. Now we find the unknown regions by subtracting sure foreground and sure background.
viii. We Label Markers of the sure foreground and background.
ix. We apply Watershed algorithm for the markers
x. Finally, we find Contours and see the results.

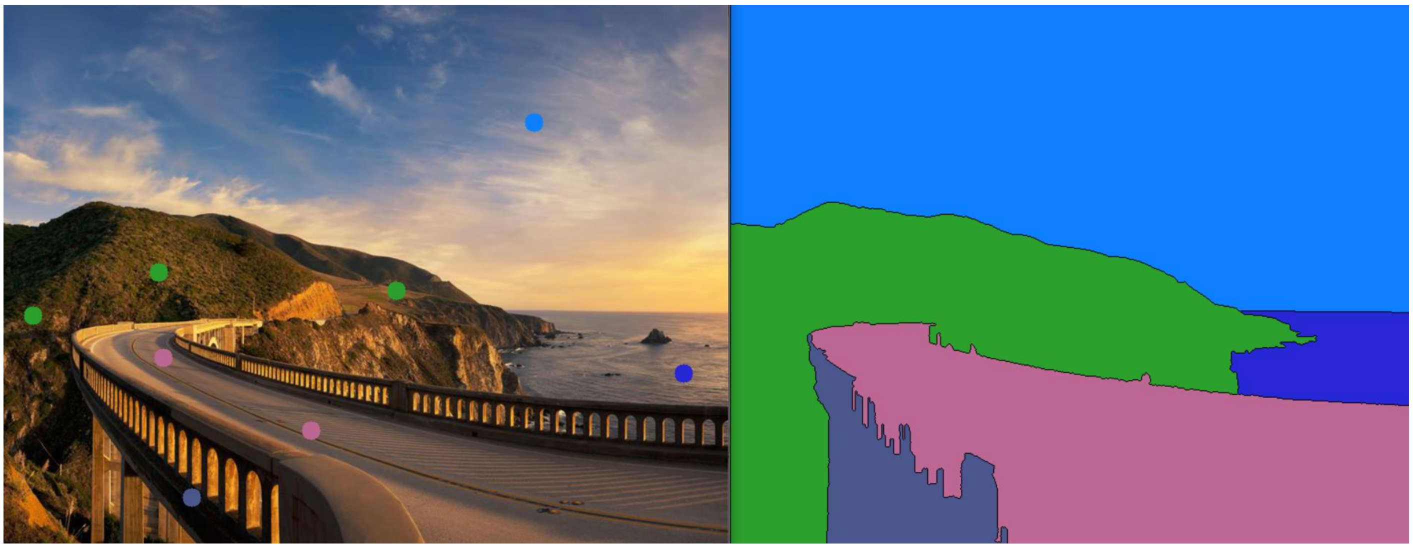



Therefore, the results clearly identify all the six coins separately and thus state the importance of the Watershed algorithm to detect and extract objects in images that are touching and/or overlapping.

II. Custom Seeds with Watershed Algorithm

Previously we set Markers to provide seeds to the Watershed Algorithm. However, we can provide the seeds ourselves also.

i. Read the image.
ii. Apply Median Blur filtering to blur the extraneous details.
iii. Create an Empty space for the results to be shown.
iv. Create Colors for the markers (can be done using color mappings).
v. Store the colors as one color for each single digit
vi. Set up a mouse call back function which is to be used when user interactively places the markers.
vii. Call the watershed algorithm for the chosen markers.
viii. Display the segmented regions on the result window as the user places the markers.

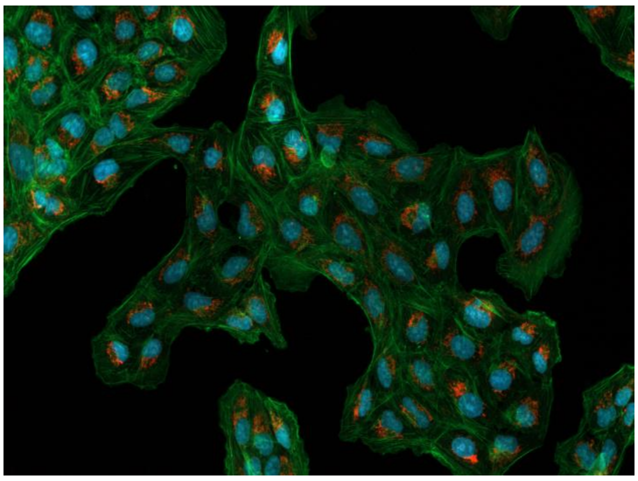


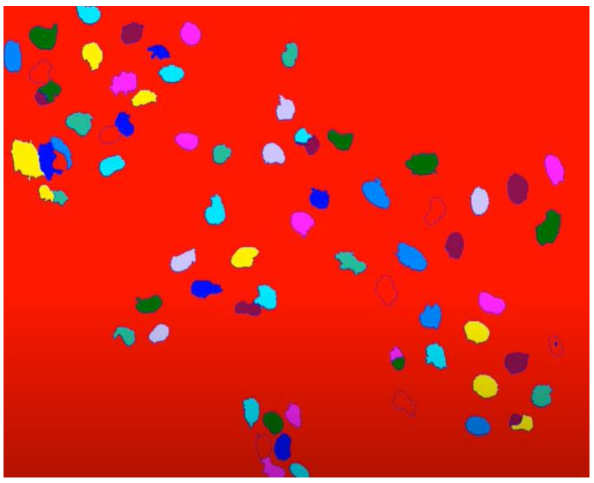
(The image shows the input image with markers on the left and watershed segments on the right)

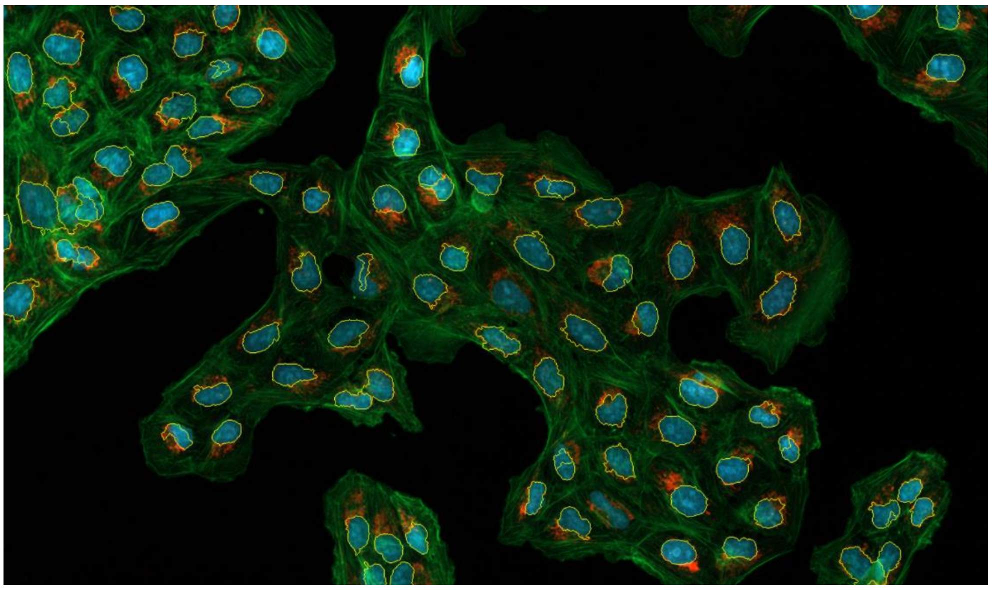
III. Cell Analysis using Watershed Algorithm

We work on Osteosarcoma Cells (Bone Cancer) and perform Watershed Segmentation on this image which can later be used to perform cell analysis based on properties like area, diameter, orientation, Axis Length and Perimeter.

i. Read the image.
ii. Apply Median Blur filtering to blur the extraneous details.
iii. Apply Threshold (Inverse Binary with Otsu thresholding).
iv. Remove Noise using Morphological Operations.
v. Grab Background from the image that we are sure of using dilate function
vi. Now find the Sure Foreground by computing the Euclidean Distance Transform (EDT) via the distance transform function.
vii. Now we find the unknown regions by subtracting sure foreground and sure background.
viii. We create a marker and label the regions inside, both foreground and background will be labeled with positive numbers while Unknown regions will be labeled 0.
ix. Apply Watershed for the markers.
x. Color the cell boundaries on the original image with a distinct color like yellow and see the results.



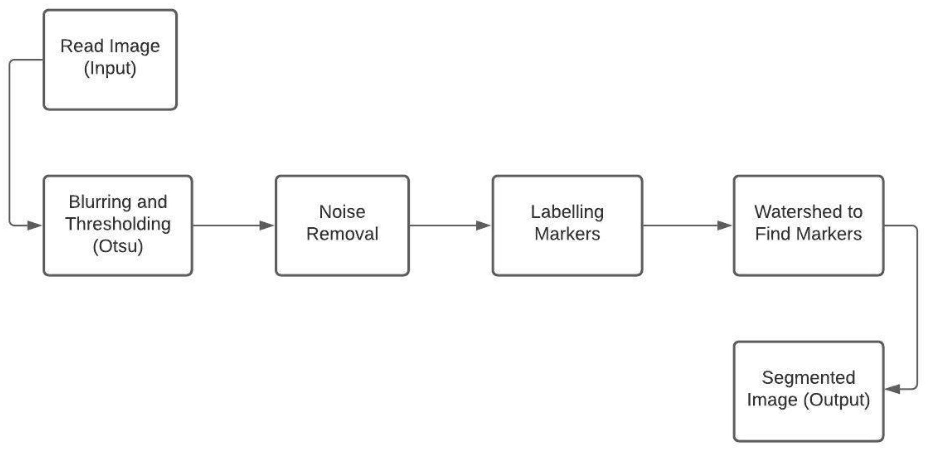




(The top left image is the original image; the top right image is the watershed segmented image; the bottom image is the enlarged version of original image where the cell boundary has been segmented and colored in yellow)

Block Diagram

The basic steps followed in watershed Segmentation can be represented by the block diagram as follows:



REFERENCES

1. De Andrade, M.C., 2004. An Interactive Algorithm for Image Smoothing and Segmentation, Electronic Letters on Computer Vision and Image Analysis Published by Computer Vision Center/Universität Autonoma de Barcelona, Barcelona, Spain, 4: 32-48.

2. Vincent, L. and P. Soille, 1991. Watersheds in Digital Space: An Efficient Algorithm based on Immersion Simulations, IEEE Transaction on Pattern Recognition and Machine Intelligence, 6: 583-598.

3. Yezm, A., 1998. Modified Curvature Motion for Image Smoothing and Enhancement, IEEE Transaction on Image Processing, 3: 345-352.

4. Yezm, A., 1998. Modified Curvature Motion for Image Smoothing and Enhancement, IEEE Transaction on Image Processing, 3: 345-352.

5. Beucher S, Rivest J.F, Soille P (1993). Morphological gradients. J Electron Imaging 2:326-36.

6. Amstutz S, Horchani I, Masmoudi M (2005). Crack detection by the topological gradient method. Control Cybern 34:119-38.

7. Aubert G, Kornprobst P (2001). Mathematical problems in image processing. New York: Springer-Verlag.

8. Auroux D, Masmoudi M (2006). A one-shot inpainting algorithm based on the topological asymptotic analysis. Comput Appl Math 25:1-17.

Code

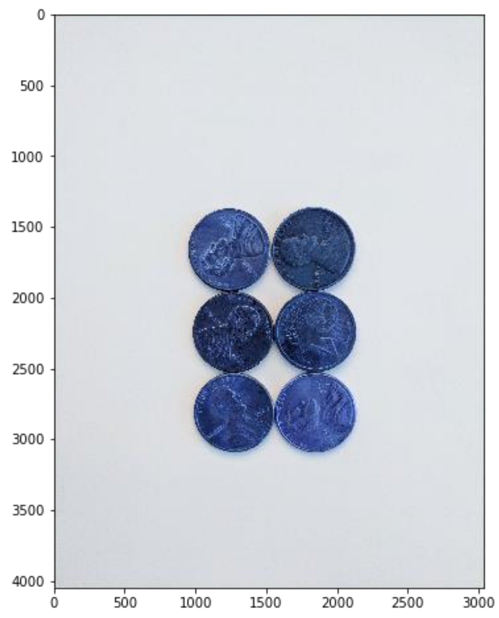
I. Watershed when Touching/Overlapping Objects in Image

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

def display(img,cmap=None):
    fig = plt.figure(figsize=(10,8))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap=cmap)

sep_coins =cv2.imread('OneDrive/Desktop/OnlineClass/DIP/Mask/pennies.jpg')
```

# Median Blurring

Since there is too much detail in this image, including light, the face edges on the coins, and too much detail in the background. use Median Blur Filtering to blur the image a bit, which will be useful later for thresholding.

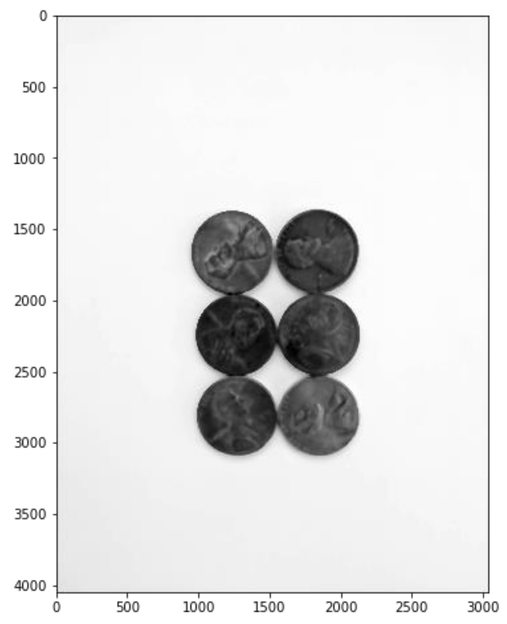
```
sep_blur = cv2.medianBlur(sep_coins,25)
display(sep_blur)
```



```
gray_sep_coins = cv2.cvtColor(sep_blur,cv2.COLOR_BGR2GRAY)
display(gray_sep_coins,cmap='gray')
```

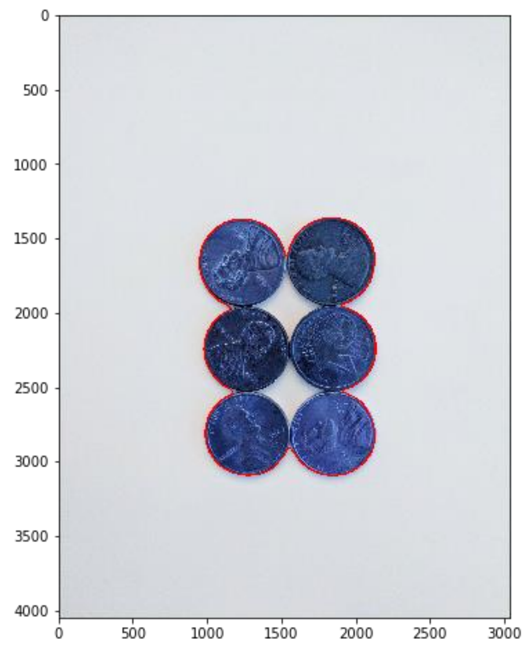
# Binary Threshold

```
ret, sep_thresh = cv2.threshold(gray_sep_coins,160,255,cv2.THRESH_BINARY_INV)
display(sep_thresh,cmap='gray')
```



# Find Contours

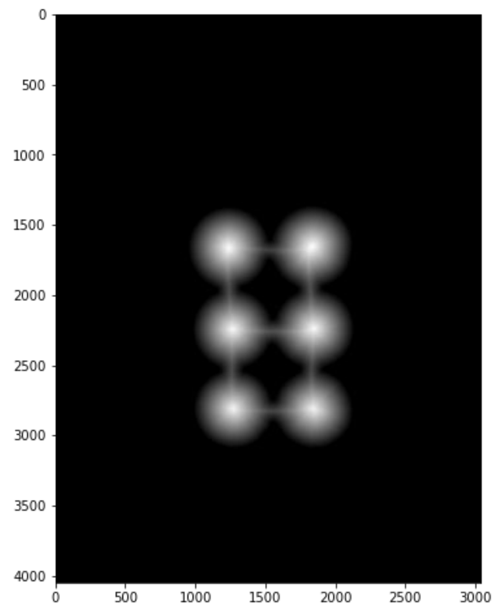
```
image, contours, hierarchy = cv2.findContours(sep_thresh.copy(), cv2.RETR_CCOMP, cv2.CHAIN_
APPROX_SIMPLE)
# For every entry in contours
for i in range(len(contours)):

    if hierarchy[0][i][3] == -1:

        # We can now draw the external contours from the list of contours
        cv2.drawContours(sep_coins, contours, i, (255, 0, 0), 10)

display(sep_coins)
```



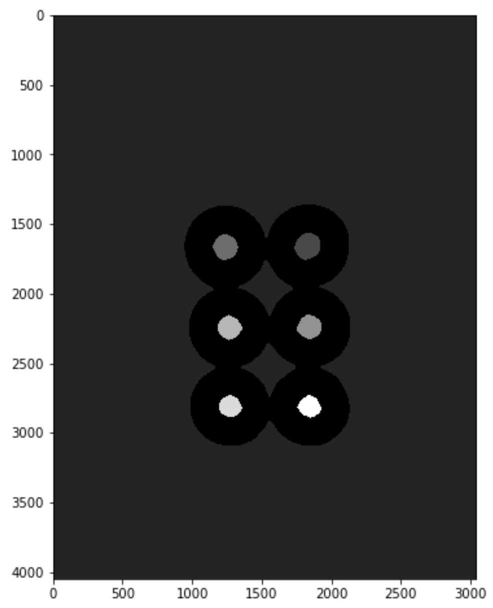
Watershed Algorithm

Now, the watershed algorithm approach to draw contours around the pennies.

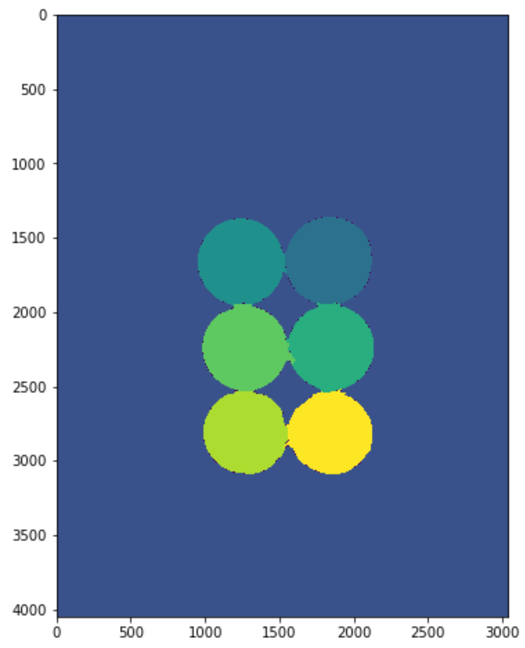
```
img = cv2.imread('OneDrive/Desktop/OnlineClass/DIP/Mask/pennies.jpg')
img = cv2.medianBlur(img,35)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,thresh=cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)


# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg =cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
display(dist_transform,cmap='gray')
```



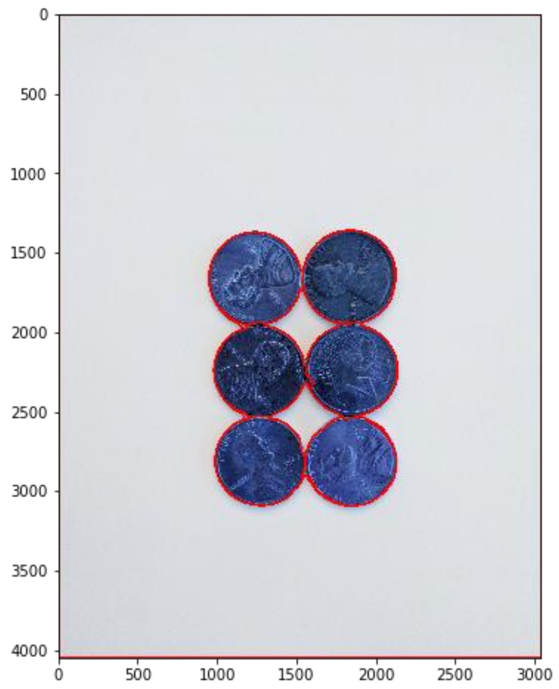
```
# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)

# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
# Now, mark the region of unknown with zero
markers[unknown==255] = 0
display(markers,cmap='gray')
```



```
markers = cv2.watershed(img,markers)
display(markers)
```



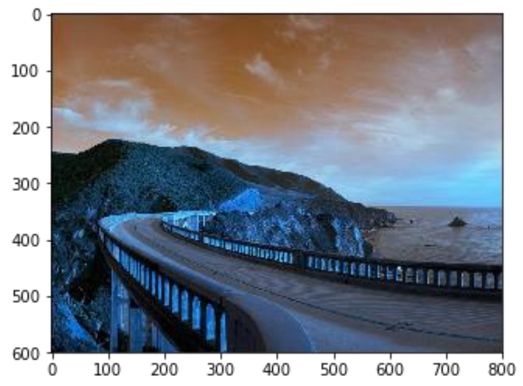
```
image, contours, hierarchy = cv2.findContours(markers.copy(), cv2.RETR_CCOMP, cv2.CHAIN_AP
PROX_SIMPLE)

# For every entry in contours
for i in range(len(contours)):

    if hierarchy[0][i][3] == -1:

        # We can now draw the external contours from the list of contours
        cv2.drawContours(sep_coins, contours, i, (255, 0, 0), 10)

display(sep_coins)
```



II. # Custom Seeds with the Watershed Algorithm

Previously we set Markers to provide seeds to the Watershed Algorithm. if we just provide seeds ourselves then:

```
road = cv2.imread('OneDrive/Desktop/OnlineClass/DIP/Mask/road_image.jpg')
road_copy = np.copy(road)
plt.imshow(road)
```

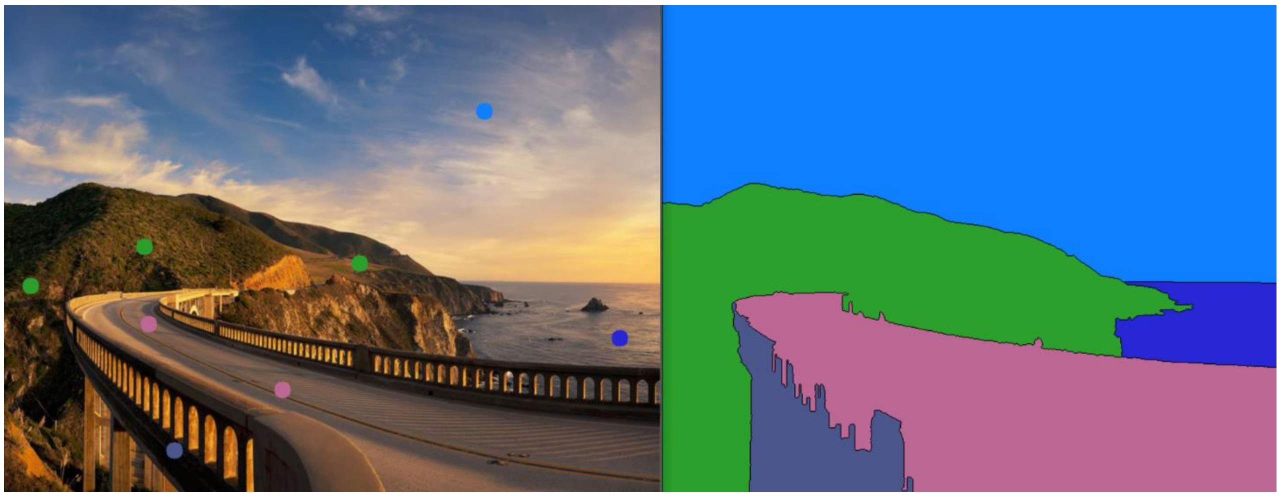


```
# to extract dimensions except channels
road.shape[:2]

marker_image = np.zeros(road.shape[:2],dtype=np.int32)
segments = np.zeros(road.shape,dtype=np.uint8)
segments.shape

from matplotlib import cm

np.array(cm.tab10(0))[:3]

array([0.12156863, 0.46666667, 0.70588235])
```

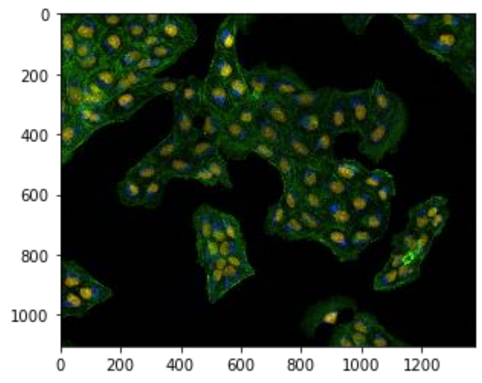
```
x = np.array(cm.tab10(0))[:3]*255
tuple(x.astype(int))

def create_rgb(i):
    x = np.array(cm.tab10(i))[:3]*255
    return tuple(x)

colors = []

# One color for each single digit
for i in range(10):
    colors.append(create_rgb(i))

# Numbers 0-9
n_markers = 10
# Default settings
current_marker = 1
marks_updated = False

def mouse_callback(event, x, y, flags, param):
    global marks_updated

    if event == cv2.EVENT_LBUTTONDOWN:

        # TRACKING FOR MARKERS
        cv2.circle(marker_image, (x, y), 10, (current_marker), -1)

        # DISPLAY ON USER IMAGE
        cv2.circle(road_copy, (x, y), 10, colors[current_marker], -1)
        marks_updated = True

cv2.namedWindow('Road Image')
cv2.setMouseCallback('Road Image', mouse_callback)


while True:

    # SHow the 2 windows
    cv2.imshow('WaterShed Segments', segments)
    cv2.imshow('Road Image', road_copy)
    # Close everything if Esc is pressed
    k = cv2.waitKey(1)
    if k == 27:
        break

    # Clear all colors and start over if 'c' is pressed
    elif k == ord('c'):
        road_copy = road.copy()
        marker_image = np.zeros(road.shape[0:2], dtype=np.int32)
        segments = np.zeros(road.shape,dtype=np.uint8)

    # If a number 0-9 is chosen index the color
    elif k > 0 and chr(k).isdigit():

        current_marker  = int(chr(k))

# If we clicked somewhere, call the watershed algorithm on chosen markers
```

```
        if marks_updated:

            marker_image_copy = marker_image.copy()
            cv2.watershed(road, marker_image_copy)

            segments = np.zeros(road.shape,dtype=np.uint8)

            for color_ind in range(n_markers):
                segments[marker_image_copy == (color_ind)] = colors[color_ind]

            marks_updated = False

    cv2.destroyAllWindows()
```

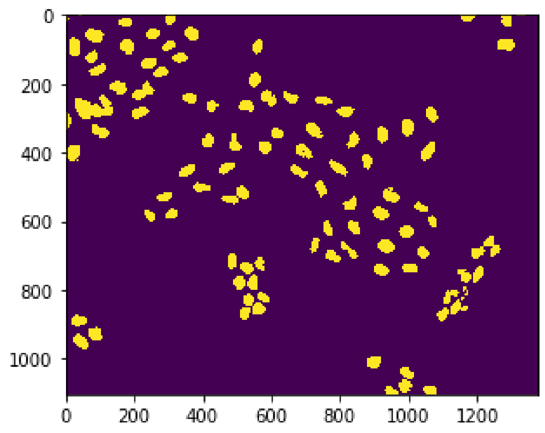
III. Cell Analysis



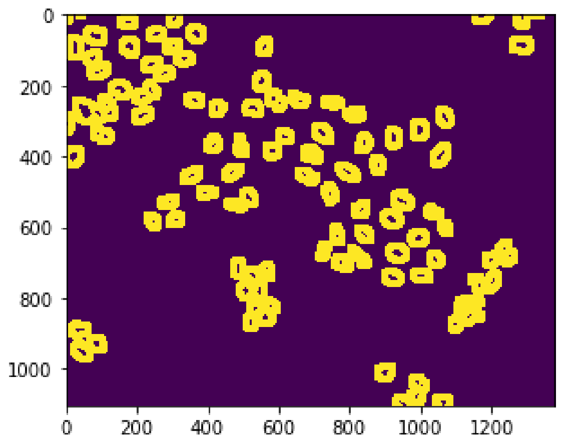
```
import cv2
import numpy as np
from scipy import ndimage
#from skimage import measure, color, io
import matplotlib.pyplot as plt
%matplotlib inline

def display(img,cmap=None):
    fig = plt.figure(figsize=(10,8))
    ax = fig.add_subplot(111)
    ax.imshow(img,cmap=cmap)

img=cv2.imread("OneDrive/Desktop/OnlineClass/DIP/Mask/Osteosarcoma_01.tif")
plt.imshow(img)
```



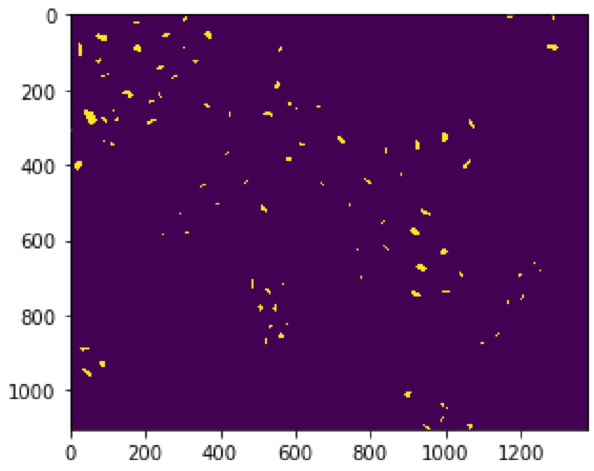
```
#Extract only blue channel as DAPI / nuclear (blue) staining is the best channel to perform cell count.
cells=img[:,:,0]
pixels_to_um = 0.454 # 1 pixel = 454 nm (got this from the metadata of original image)
#Threshold image to binary using OTSU. All threshold pixels will be set to 255
ret1, thresh = cv2.threshold(cells, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.imshow(thresh)
```



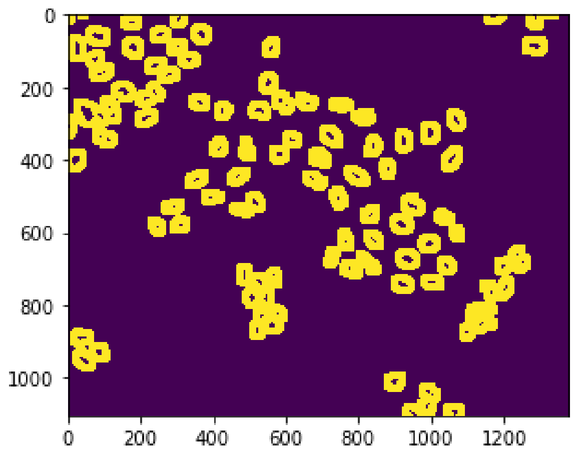
```
# Morphological operations to remove small noise - opening
#To remove holes we can use closing
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

sure_bg = cv2.dilate(opening,kernel,iterations=10)

plt.imshow(sure_bg)
```



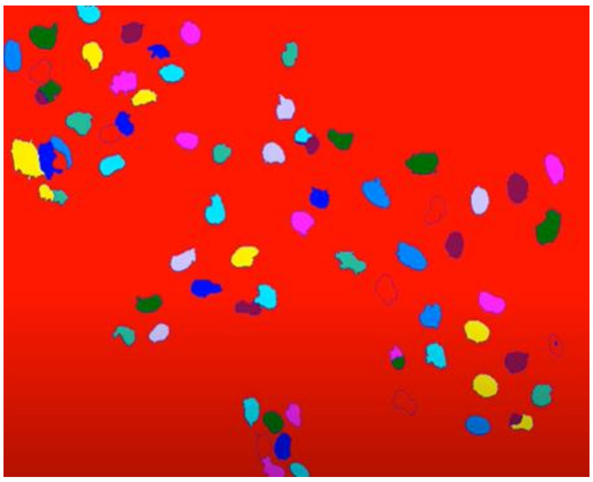
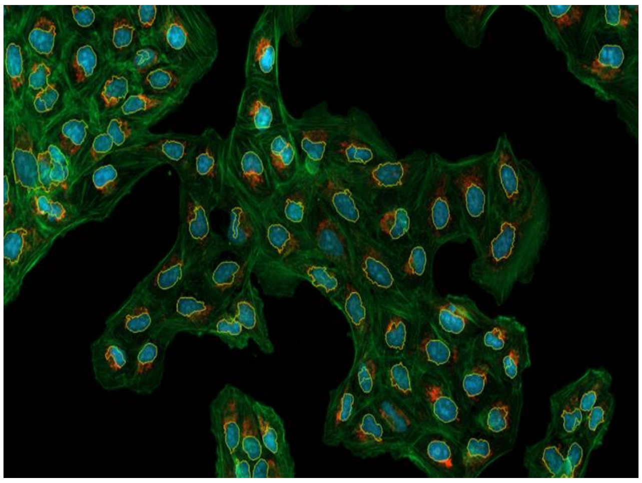
```
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)


ret2, sure_fg = cv2.threshold(dist_transform,0.5*dist_transform.max(),255,0)
plt.imshow(sure_fg)
```



```
# Unknown ambiguous region is nothing but background - foreground
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
ret3, markers = cv2.connectedComponents(sure_fg)
plt.imshow(unknown)
```



```
markers = markers+10
# Now, mark the region of unknown with zero
markers[unknown==255] = 0

markers = cv2.watershed(img,markers)
#color boundaries in yellow.
img[markers == -1] = [0,255,255]
cv2.imshow('Overlay on original image', img)
cv2.imshow('Colored cells', img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

(The left image is the watershed segmented image; the right image is the enlarged version of original image where the cell boundary has been segmented and colored in yellow)