Computer Vision Homework 3

Ian Sinclair
ENCE 3620-1

March 30, 2022
Dr. Mohammad H. Mahoor

Problem 1

Explain what would happen in binary erosion and dilation if the structuring element is a single point, valued 1. Give the reason for your answer.

Let A be a set in an image, and let B be a structuring element, $B \in \mathbb{R}^2$.
Where B contains a single point.

i) Erosion

Then the erosion of A by B is defined by,

$$A \ominus B = \{z : (B)_z \in A\}$$

For origin at the location of element z on the image, and $z \in B$.
In general this definition suggests that $A \ominus B$ contains every image element where B is completely contained in A.
Additionally, as a result of B being a single element consider the following cases.

i) $A \ominus B \subseteq A$

The following argument is trivial by definition.
Pick any element $z \in A \ominus B$, then $(B)_z \in A$, and $z \in B$.
Therefore, $z \in A$. Thus, $A \ominus B \subseteq A$

ii) $A \subseteq A \ominus B$

Pick any $a \in A$. Then, consider when set B is centered by its origin at a. $(B)_a$.
Because B is a single element and $(B)_a = a$, and so $(B)_a \in A$.
Therefore, $a \in A \ominus B$. And thus, $A \subseteq A \ominus B$.

Therefore, $A \ominus B = A$.

ii) Dilation

Now by dual, note the definition of the dilation of A by B.

$$A \oplus B = \{z : (B)_z \cap A \neq \emptyset\}$$

For set B's origin located at z on the image.
Then in general, if any element in $(B)_z$ is also in A, then $z \in A \oplus B$.
Following a symmetric argument to erosion. Pick any $z \in A \oplus B$, then there must exist $b \in (B)_z$ where $b \in A$.
But B is only a single element, so then $z = b \in A$. And so $A \oplus B \subseteq A$.
Lastly, the trivial argument, pick any $a \in A$, then $(B)_a$ has at least one element within A. And so, $A \supseteq A \oplus B$.
Therefore, $A \oplus B = A$.

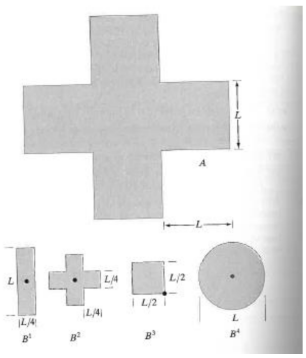
Solution

And so, if the structuring element, B is a single point valued 1, then the erosion and dilation of any set A with B would not change A.

$$A \ominus B = A$$
$$A \oplus B = A$$

Problem 2

Let A denote the set shaded in the following figure. Refer to the structuring elements shown below, sketch the results of the following morphological operations(do only two out of parts a-d):

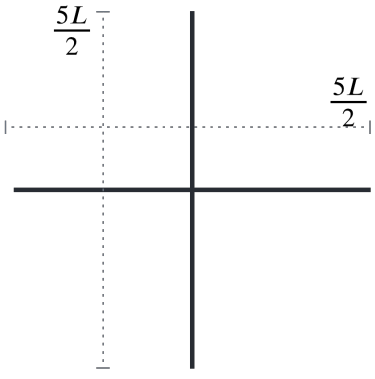


(a) $(A \ominus B^4) \oplus B^2$

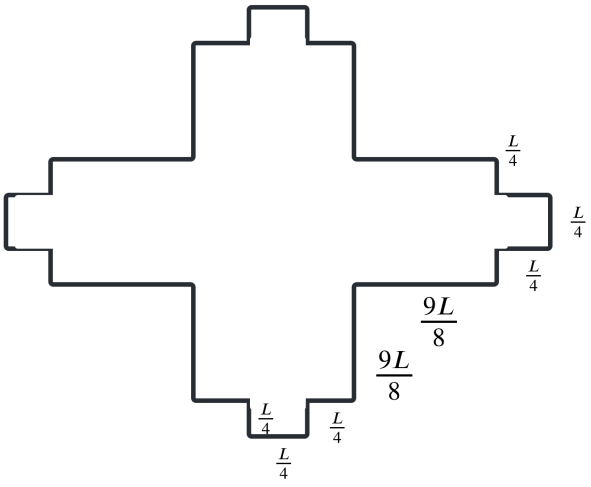
Consider the definition,

$$A \ominus B = \{z : (B)_z \in A\}$$
$$A \oplus B = \{z : (B)_z \cap A \neq \emptyset\}$$

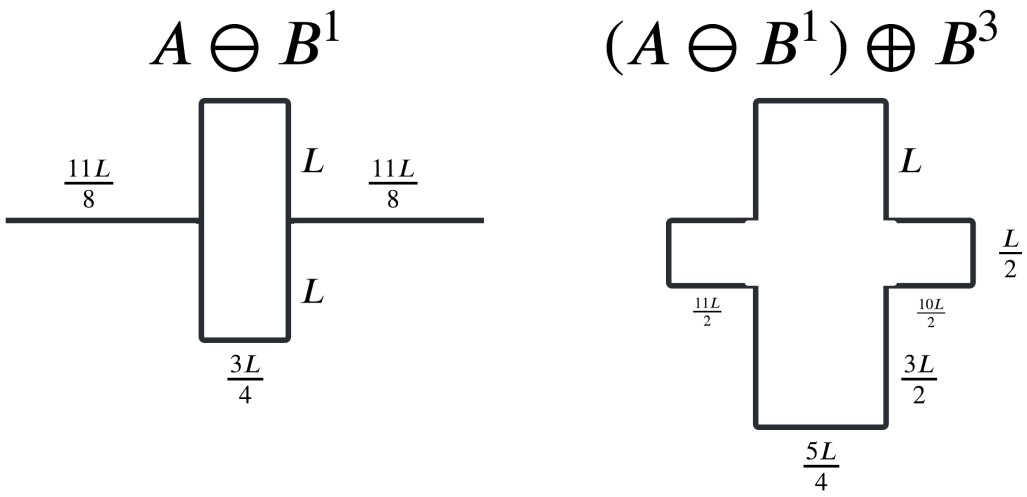
$A \ominus B^4$



$(A \ominus B^4) \oplus B^2$



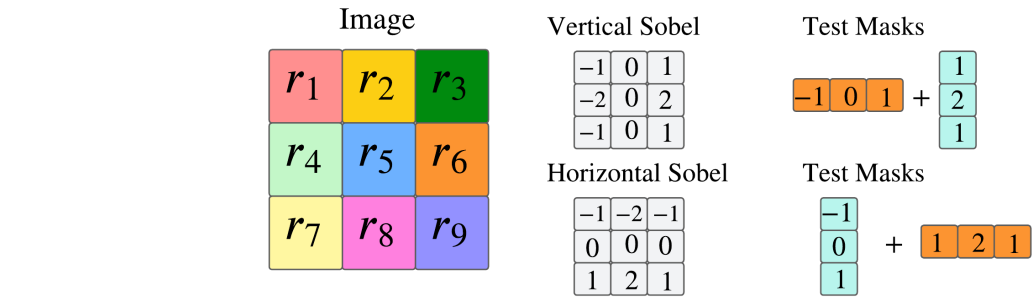
(b) $(A \ominus B^1) \oplus B^3$



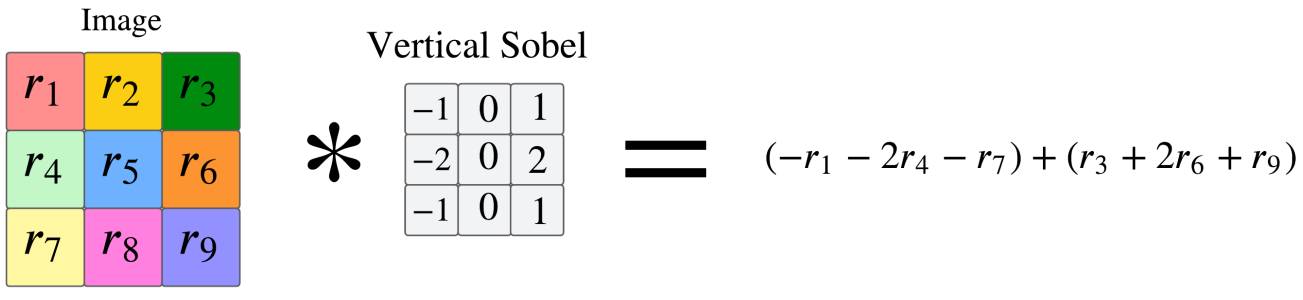
Problem 3

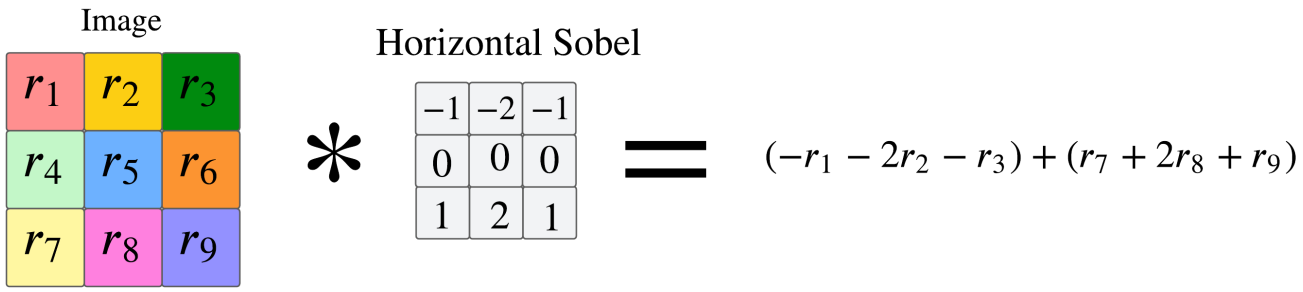
The results obtained by a single pass through an image of some $2D$ masks can be also achieved by two passes using $1D$ masks. For example the same result of using a $2D$ smoothing mask with coefficients $\frac{1}{9}$ can be obtained by a pass of the mask, $\begin{bmatrix}1 & 1 & 1\end{bmatrix}$ through an image. Then followed by a pass of the mask, $\begin{bmatrix}1\\1\\1\end{bmatrix}$. The final result is scaled by $\frac{1}{9}$. Show that the response of a 3×3 Sobel masks (fig 10.04). can be implemented similarly by one pass of the differencing mask, $\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$ followed by the smoothing mask, $\begin{bmatrix}1 & 2 & 1\end{bmatrix}$.

Consider the 3×3 image with 9 unique pixel intensity values r_i, $i = 1, 2, .., 9$. Along with the horizontal and vertical Sobel masks and finally, the sequence of test masks (shown below).



Then assume all Sobel masks are centered at the center of the image, r_5. Then the resulting images are.





And now convolving the image with each sequence of test masks.

Image

$$\begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -r_1 + r_3 \\ -r_4 + r_6 \\ -r_7 + r_9 \end{bmatrix}$$

$$\begin{bmatrix} -r_1 + r_3 \\ -r_4 + r_6 \\ -r_7 + r_9 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = (-r_1 - 2r_4 - r_7) + (r_3 + 2r_6 + r_9)$$

And so this sequence is equivalent to the vertical Sobel mask.

Image

$$\begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -r_1 + r_7 & -r_2 + r_8 & -r_3 + r_9 \end{bmatrix}$$

$$\begin{bmatrix} -r_1 + r_7 & -r_2 + r_8 & -r_3 + r_9 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = (-r_1 - 2r_2 - r_3) + (r_7 + 2r_8 + r_9)$$

This sequence is equivalent to the horizontal Sobel mask.

Mask Results	Sobel Mask	Differencing/Smoothing
Vertical	$(-r_1 - 2r_4 - r_7) + (r_3 + 2r_6 + r_9)$	$(-r_1 - 2r_4 - r_7) + (r_3 + 2r_6 + r_9)$
Horizontal	$(-r_1 - 2r_2 - r_3) + (r_7 + 2r_8 + r_9)$	$(-r_1 - 2r_2 - r_3) + (r_7 + 2r_8 + r_9)$

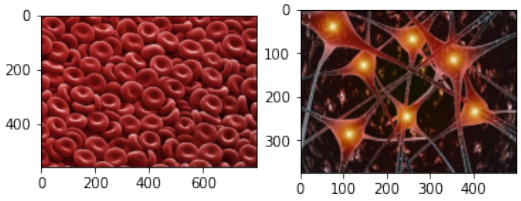
Solution

Therefore, the results of the 3×3 Sobel masks can be implemented similarly by one pass of the corresponding sequence of differencing and smoothing masks.

Problem 4

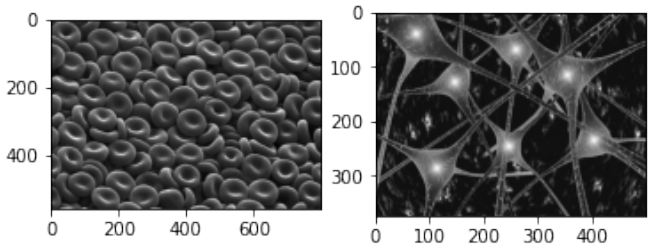
Use binary image processing techniques to count the number of objects (e.g. neurons or cells) in an image. You first need to convert the color image to gray scale and then binary using thresholding techniques. Then you should apply morphological operators to count to disconnect objects from each other and then apply connected component analysis to count the number of objects in the image. Also, find the center of detected neuron cells in image 1 and red cells in image 2. You are given two images for this problem. Work on both images and report the number of objects in each case. In your report, provide the output images after applying any operator or filter. Include your code in your report.

Consider the following raw images.



Then switching to gray scale.

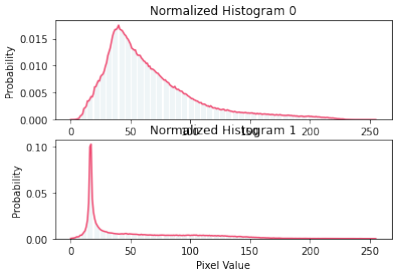
```
from traitlets.traitlets import List
def RGB_to_Greyscale( image ) :
  temp = image.copy()
  grey = np.zeros(( len(temp), len(temp[0]) ), dtype = int )

  for i in range(0 , len( image ) ) :
    for j in range(0 , len( image[i] ) ) :
      Filter =  0.299*temp[i][j][0] + 0.587*temp[i][j][1] + 0.114*temp[i][j][2]
      grey[i][j] = int( Filter )

  return grey
```



And generating the histograms of each image.

```
def generate_histogram( image ) :
  temp = image.copy()
  hist = []

  #Initialize an emply list on [0,255]
  for i in range( 0 , 256 ) :
    hist += [0]

  #Fill list with histogram values
  for i in range( 0 , len( image ) ) :
    for j in range( 0 , len( image[i] ) ) :
      intensity = int( image[i][j] )
      hist[intensity] += 1

  return hist

def generate_CDF( histogram ) :
  CDF = np.zeros( [256] )

  CDF[0] = histogram[0]
```

```
21
22    for i in range( 1 , len( histogram ) ) :
23      CDF[ i ] = CDF[ i -1] + histogram[ i ]
24
25    return CDF
26
27  def normalize_histogram( histogram ) :
28    temp = histogram.copy()
29    r = sum( temp )
30    for i in range( 0 , len( temp ) ) :
31      temp[ i ] = temp[ i ]/ r
32
33    return temp
```

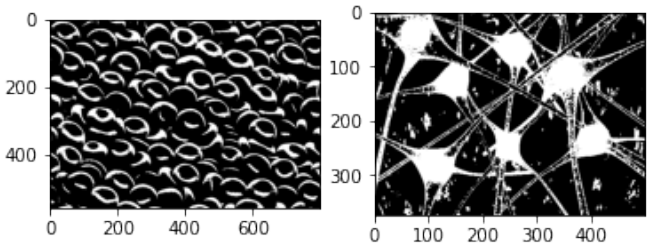


Now switching to a binary image using the Otsu

```
1   def Otsu( histogram ) :
2
3     sig_max = 0
4     t_best = 0
5     q1 = generate_CDF( histogram )
6
7     #mu = np.zeros(256,dtype = 'int')
8     mu = 0
9     for p in range( 0 , len( q1 ) ) :
10      mu += p * histogram[p]
11
12    mu1 = ((2)*histogram[2])/q1[2]
13    mu2 = (q1[2]-mu1)/(1-q1[2])
14
15    for t in range(1 , 255) :
16      sig = q1[t]*(1-q1[t]) *((mu1-mu2)**2)
17      if sig > sig_max :
18        t_best = t
19        sig_max = sig
20
21      mu1 = ((q1[t]*mu1) + (t+1)*histogram[t+1])/(q1[t+1])
22
23      mu2 = (mu-(q1[t+1]*mu1))/(1-q1[t+1])
24
25
26    return t_best
27
28  def Binary_Conversion( image, t : int) :
29    binary = np.zeros(( len(image), len(image[0]) ), dtype = int )
30
31    for i in range( 0 , len( image )) :
32      for j in range( 0 , len( image[i] )) :
33        if image[i][j] >= t :
34          binary[i][j] = 1
35        if image[i][j] < t :
36          binary[i][j] = 0
37
38    return binary
```

Which results in the thresholds for either image, Next now the morphological operators,

```
1   def erosion( image ) :
2     eroded_image = image.copy()
```

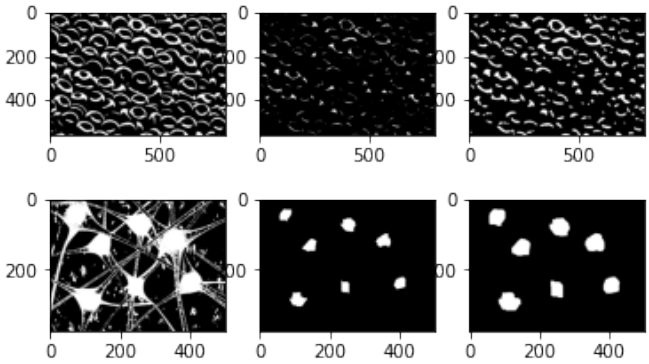


Number of Red Blood Cells	$T = 88$
Number of Neurons	$T = 70$

```
3
4     mask = [(1,0), (-1,0), (0,1), (0,-1), (1,1), (-1,1), (-1,-1), (-1,1)]
5
6     for i in range(0 , len( image ) ) :
7       for j in range(0 , len( image[i] ) ) :
8         eroded_image[i][j] = 1
9         for k in mask :
10          if 0 <= i+k[0] < len( image ) and 0 <= j+k[1] < len( image[i] ) :
11            if image[i+k[0]][j+k[1]] != 1 :
12              eroded_image[i][j] = 0
13              break
14          else :
15            eroded_image[i][j] = 0
16            break
17
18    return eroded_image
19
20  def Dilation( image ) :
21    Dilated_image = image.copy()
22
23    temp_image = cv2.copyMakeBorder(image.copy(), 1, 1, 1, 1, cv2.BORDER_CONSTANT, value=0)
24
25    mask = [(1,0), (-1,0), (0,1), (0,-1), (1,1), (-1,1), (-1,-1), (-1,1)]
26
27    for i in range(0 , len( image ) ) :
28      for j in range(0 , len( image[i] ) ) :
29        Dilated_image[i][j] = 0
30        for k in mask :
31            if temp_image[i+k[0]][j+k[1]] == 1 :
32              Dilated_image[i][j] = 1
33              break
34
35    return Dilated_image
```

```
1   neuron_image = erosion( binary_scale[1] )
2   for i in range( 0 , 10 ) :
3     neuron_image = erosion( neuron_image )
4
5   neuron_image2 = Dilation( neuron_image )
6   for i in range( 0 , 5 ) :
7     neuron_image2 = Dilation( neuron_image2 )
8
9    blood_cells_image = erosion( binary_scale[0] )
10  for i in range( 0 , 2 ) :
11    blood_cells_image = erosion( blood_cells_image )
12
13  blood_cells_image2 = Dilation( blood_cells_image )
14  for i in range( 0 , 2 ) :
15    blood_cells_image2 = Dilation( blood_cells_image2 )
```

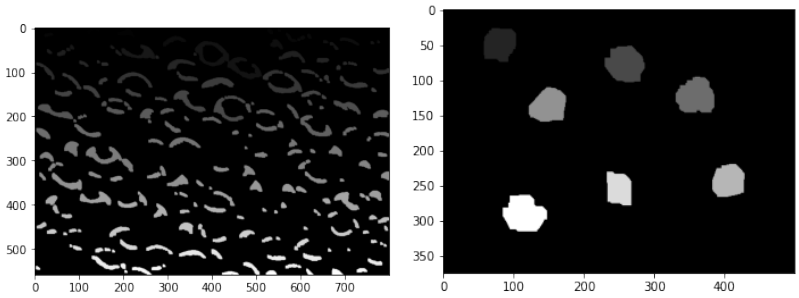
From here, opening is applied to both images. Resulting in,
Next, applying connected component analysis,



```
def negate( image ) :
  label = np.zeros(( len(image), len(image[0]) ), dtype = int )

  for i in range( 0 , len( image )) :
    for j in range( 0 , len( image[i] )) :
      if image[i][j] != 0 :
        label[i][j] = -1

  return label

def neighbors(LB, L, P) :
  N = []
  if L+1 < len( LB ) :
    N += [(L+1, P)]
  if P+1 < len( LB[L] ) :
    N += [(L, P+1)]
  if L-1 >= 0 :
    N += [(L-1, P)]
  if P-1 >= 0 :
    N += [(L, P-1)]
  return N

def search( LB, label, L, P) :
  LB[L][P] = label
  Nset = neighbors(LB, L, P)

  for m in Nset :
    if LB[m[0],m[1]] == -1 :
      search(LB, label, m[0],m[1])

def find_components( LB, label ) :
  for L in range( 0 , len( LB )) :
    for P in range( 0 , len( LB[L] )) :
      if LB[L][P] == -1 :
        label += 1
        search( LB, label, L , P)

def recursive_connected_components( B, LB ) :
  LB = negate( B )
  label = 0
  find_components( LB, label )

  return LB


L = recursive_connected_components( blood_cells_image2, blood_cells_image2.copy() )
```

Next, an image detailing the intensity change between classes.
Which results in the number of classes from each image.

Number of Red Blood Cells	195
Number of Neurons	7



Solution

And so the number of elements in either image is as follows.

Number of Red Blood Cells	195
Number of Neurons	7