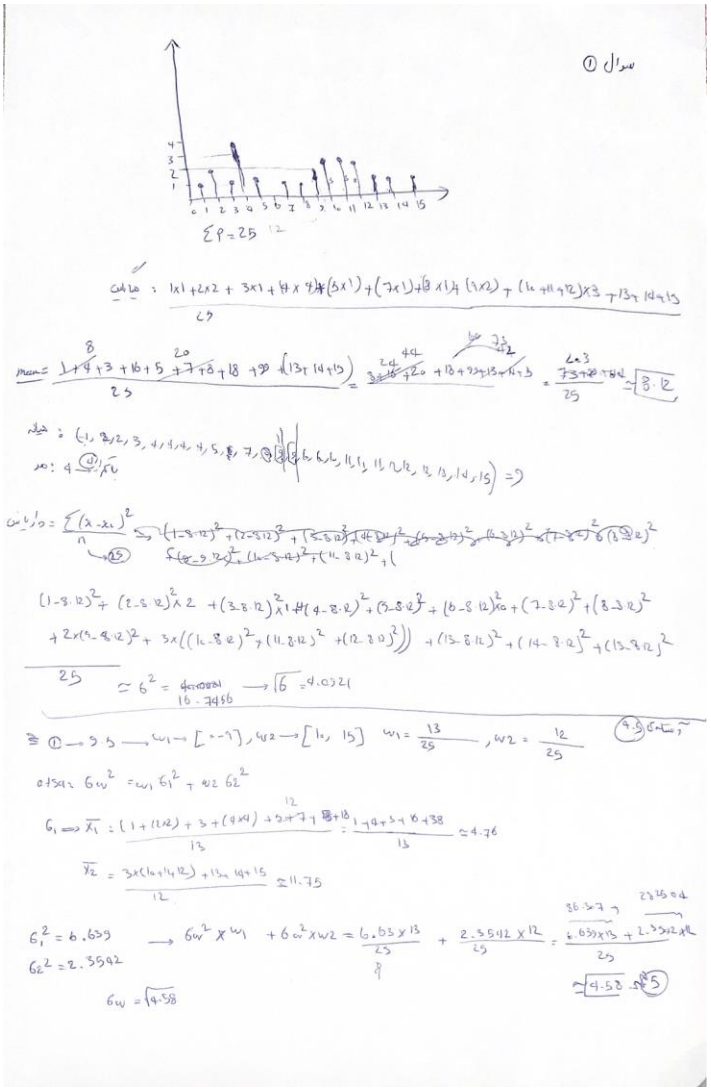
بنفشه قلی نژاد -تمرین پنجم - 98522328

سوال اول)



سوال دوم)

الف)از لحاظ سرعت، الگوریتم اوتسو سریع تر است چرا که محاسبات کم تری دارد و تنها یک بار روی تصویر اعمال میشود.

اما الگوریتم اوتسو گاوسی عملکرد دقیق تر و بهتری دارد و برای تصاویر با نویز بالا بهتر عمل می کند.

و همچنین مقدار حد آستانه را بهتر و دقیق تر به دست می آورد. اما محاسبات بیشتری دارد و در این محاسبات فیلتر گاوسی با کانولوشن را نیز اعمال می کند.

ب) طبق فرمول ذکر شده در متن، واریانس بین کلاسی معادل کم کردن واریانس درون کلاسی از واریانس کلی تصویر.

در نتیجه می دانیم که در منها کردن ، اکر عنصری که منها می کنیم کمینه شود، پاسخ اصلی بیشیته می شود در نتیجه مینیمم مقدار واریانس درون کلاسی معادل بیشیته مقدار واریانس بین کلاسی است.

سوال سوم)

در روش رشد ناحیه، از یک نقطه به نام نقطه ی seedشروغ می کنیم. با کمک الگوریتم جست و جوی اول سطح، تمام پیکسل های اطراف و متصل را بررسی می کنیم. در هر مرحله از این جست و جو، پیکسل های اطراف همسایگی را (به مدل ارتباط هشت تایبی) را در نظر گرفته و اختلاف مقدار هر کدام از پیکسل ها را با پیکسل وسط مقایسه می کنیم. (برای عکس های رنگی برای هر کدام از کانال های سبز، قرمز ، آبی، قدر مختلاف این اختلاف را محاسبه می کنیم. همچینین باید مقدار هر کدام از پیکسل های همسایه را با مقدار پیکسل seedمقایسه کنیم. در صورتی که مقدار احتلاف پیکسل های همسایه با پیکسل وسط از ترشولد اول ما کم تر باشد و مقدار هر کدام از این پیکسل ها از مقدار پیکسل seed کم تر باشد، (ترشولد دوم) آن پیکسل را به پیکسل های آن ناحیه اضافه می کنیم و به همین ترتیب پیش میرویم.

در پیاده سازی:

برای از بین بردن نویز های تصویر نیز ابتدا یک فیلتر گاوس را اعمال می کنیم.

ترشولد مقایسه بین همسایه ها 20 و اختلاف با نقطه seed 60 است.

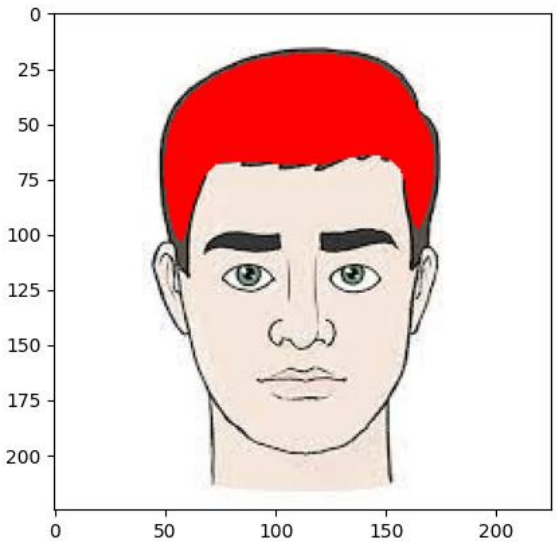
نقطه ی seed از قسمت سر شروع کرده است. و

هر نفطه که شرایط اختلاف مناسب را داشته باشد قرمز نشان داده می شود.

```
def segment(image)
    segmented_image = None
    image2 = image.copy()
    seedpoint = (35 , 100)
    queue = []
    visited = [seedpoint]
    queue.append(seedpoint)
    regions = []
    threshold = 20
    mainthreshold = 60
    image = cv2.GaussianBlur(image,(41,41),0)
    r, g, b = image[35,100,0] , image[35,100,1] , image[35,100,2]
    image2[35 , 100 , 0 ] = 255
    image2[35 , 100 , 1 ] = 0
    image2[35 , 100, 2 ] = 0

    while queue:
            s = queue.pop(0)
            print(s, end=" ")
            neighbors = []
            #put all conditions to add 8 neighbors(based on 8 connectivity)
            if(s[0] +1 < image.shape[1]):
                neighbors.append((s[0] +1 , s[1]))
            if(s[1] +1 < image.shape[0]):
                 neighbors.append((s[0] , s[1] + 1))
            if(s[0] -1 >= 0):
                 neighbors.append((s[0]-1 , s[1] ))
            if(s[1] -1 >= 0 ):
                 neighbors.append((s[0] , s[1] - 1))

            if( s[0] +1 < image.shape[1] and s[1] + 1 <image.shape[1]):
                 neighbors.append((s[0] + 1 , s[1] +1))
            if(s[0] +1 < image.shape[1] and s[1]-1 >= 0):
                 neighbors.append((s[0] + 1 , s[1] - 1 ))
            if(s[0] - 1 >= 0 and s[1] -1 >= 0 ):
                 neighbors.append((s[0] -1 , s[1] - 1))
            if( s[0] - 1 >= 0 and s[1] +1 < image.shape[0] ):
                 neighbors.append((s[0]-1 , s[1] + 1))
          #calculate differences (the abselute value) and for each channel
seperately
            rs , gs , bs = image[s[0] , s[1] , 0] , image[s[0] , s[1] , 1],
image[s[0] , s[1] , 2]
            for i in neighbors:
```

```
                if i not in visited:
                    r_p = image[i[0] , i[1] , 0]
                    g_p = image[i[0] , i[1] , 1]
                    b_p = image[i[0] , i[1] , 2]
                    dr = abs(r_p - rs)
                    dg = abs(g_p - gs)
                    db = abs(b_p - bs)
                    drm , dgm , dbm = abs(r_p - r) , abs(g_p - g) , abs(b_p - b)
                    if ((dr <= threshold and dg <= threshold and db <= threshold)
and (drm <= mainthreshold and dgm <= mainthreshold
                        and dbm <= mainthreshold)):
                         regions.append(i)
                         image2[i[0] , i[1] , 0 ] = 255
                         image2[i[0] , i[1] , 1 ] = 0
                         image2[i[0] , i[1] , 2 ] = 0
                         queue.append(i)
                         visited.append(i)
    return image2
```

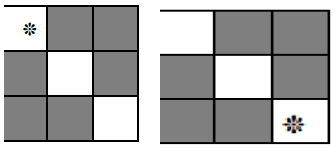
نتیجه الگوریتم پیاده سازی شده ی قسمت بالا



سوال 4) الف)

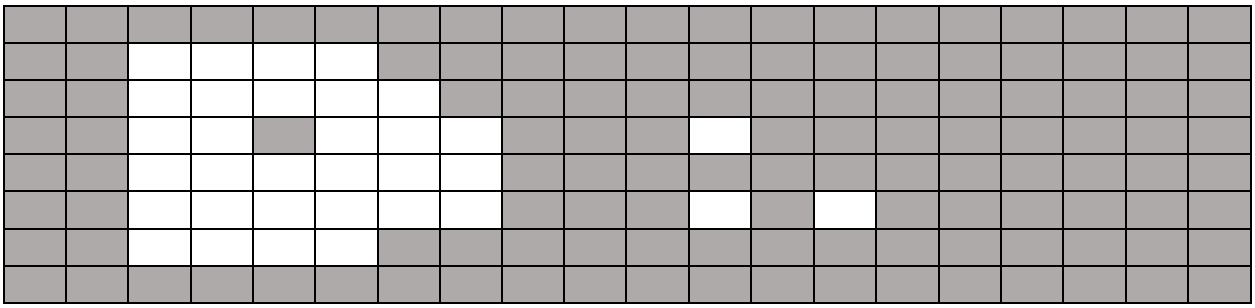
سوال 4) بخش ب:

Erode structure Dilate structure (with 180 degree rotation)

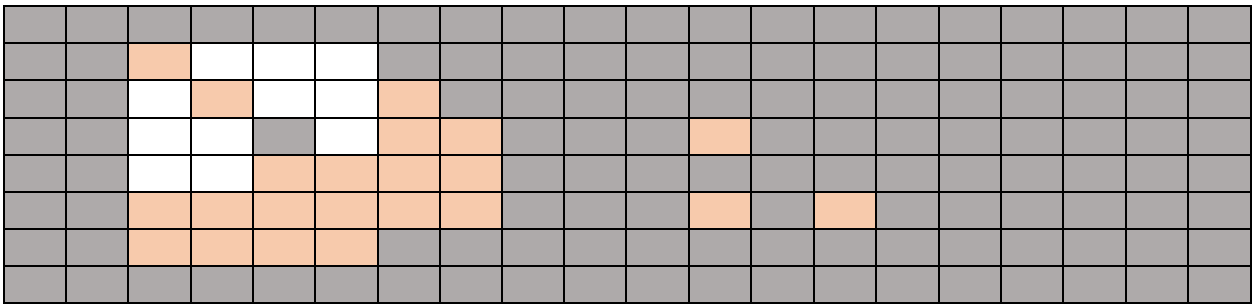


Opening = dilate(erode(image))

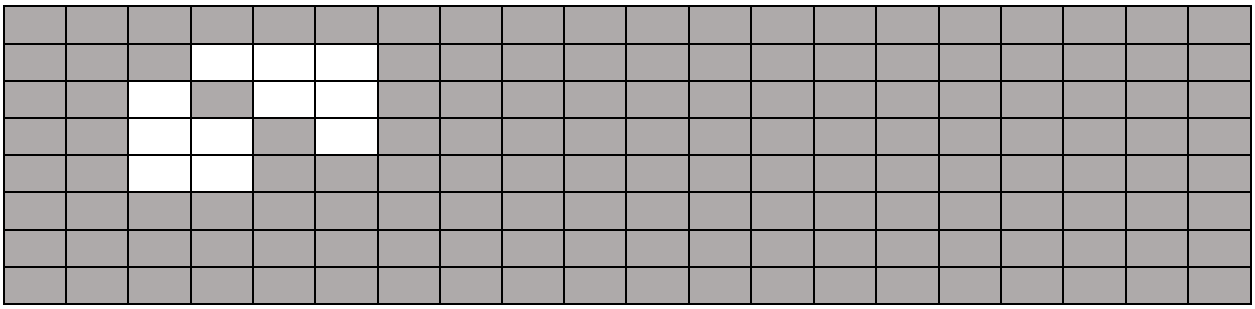
the removing ➔ orange , adding ➔ green



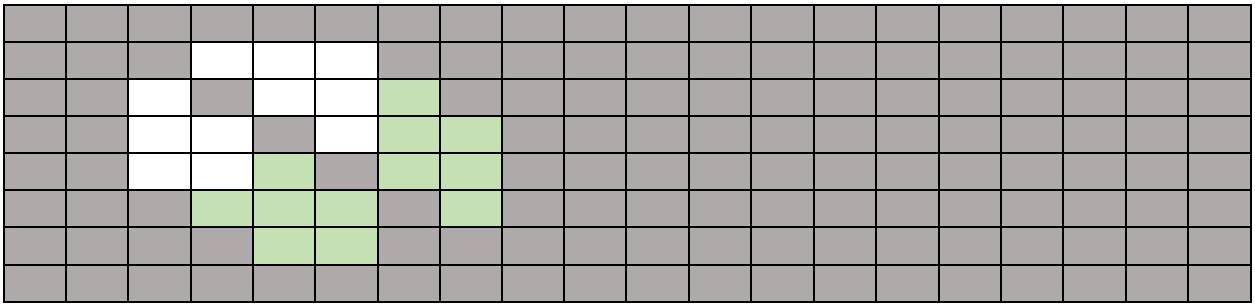
Erotion:



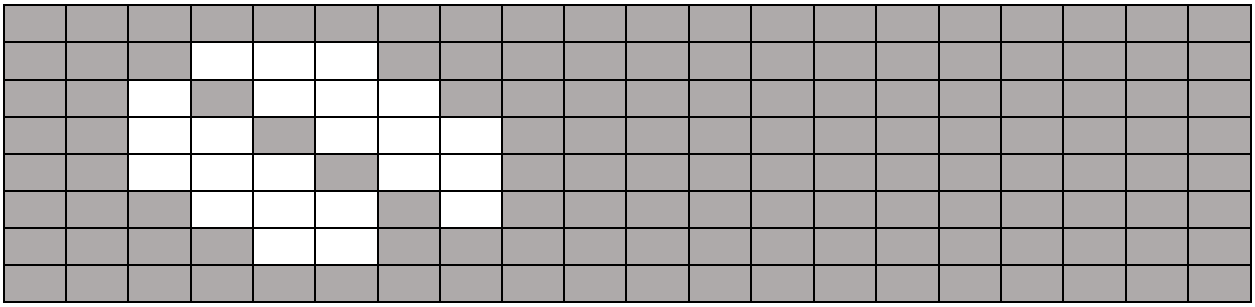
Erode Result:



Dilate

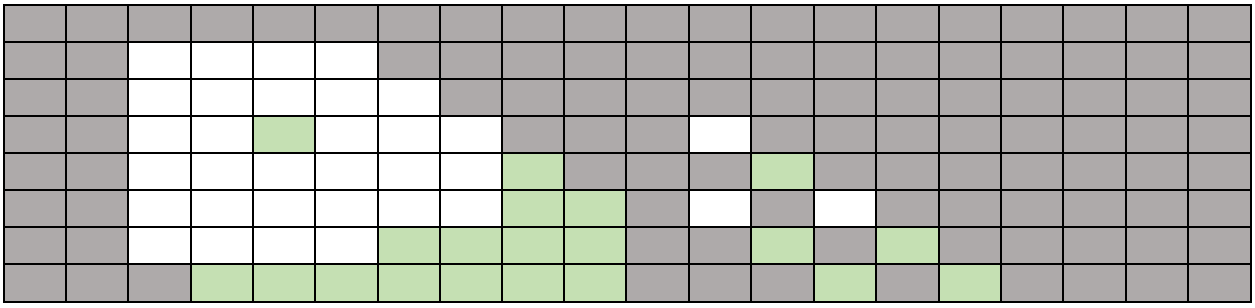


Opening result)

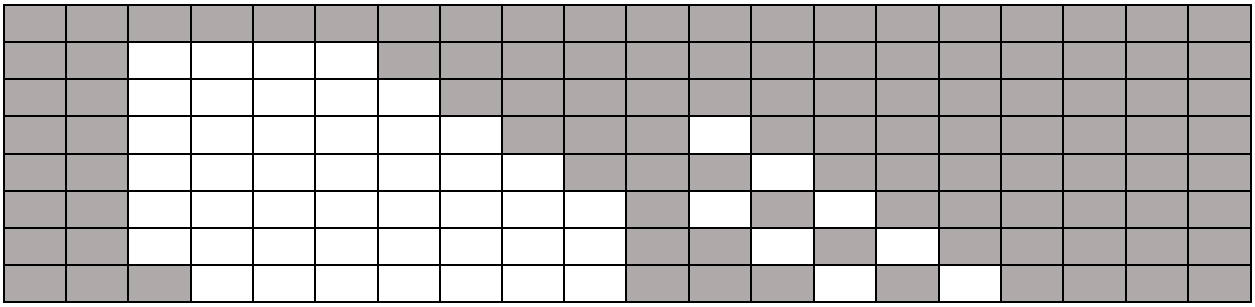


Closing = erode(dilate(image))

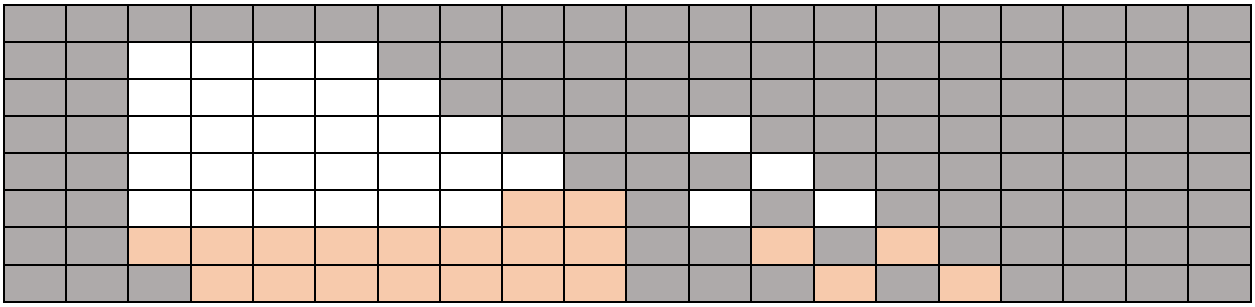
Dilate



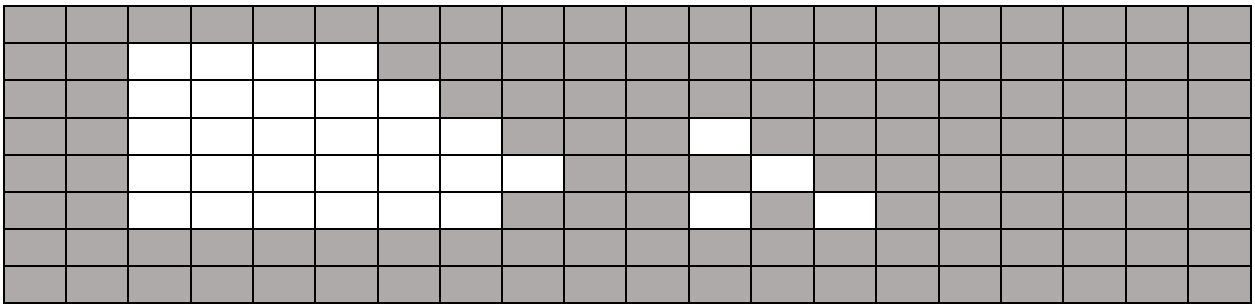
Dilate result)



Erotian) (with zero padding)



Closing result)

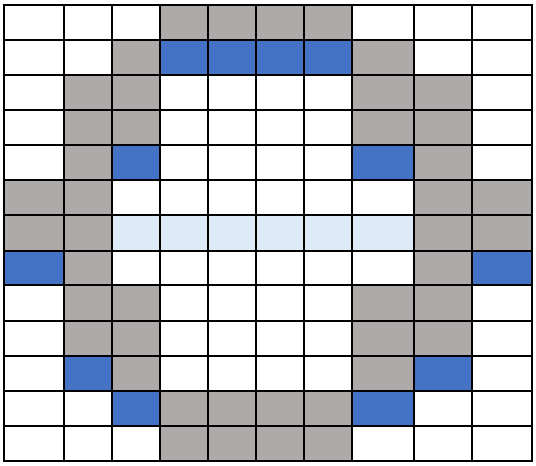


سوال 5) الف

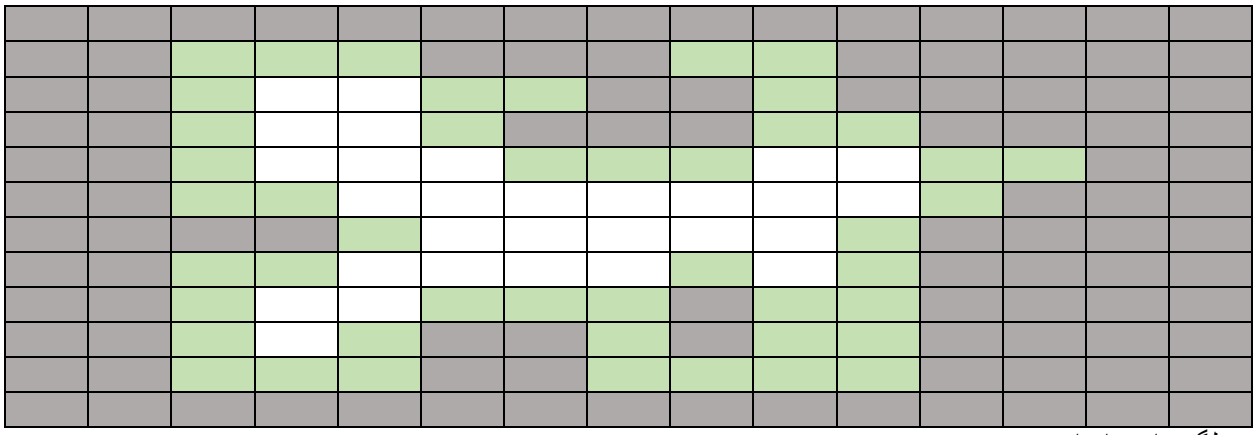
(پیکسل سفید ==> 0 پیکسل سیاه ==> 1)

عملگر: باز ، چرا که میخواهیم خطوط اضافی را حذف کنیم. عنصر ساختاری را انتخاب کرده و ابتدا با رنگ آبی روشن عملگر سایش و با ابی پررنگ عملگر افزایش را با اعمال 180 درجه چرخش کرنل اعمال می کنیم. در نتیجه خط وسط حذف می شود.

پیکسل های پررنگ در اثر افزایش به وجود امده اند و سیاه هستند و سلول های ابی کم رنگ در اثر سایش و به رنگ سفید است.



سوال 5 ب)



عملگر های ساختاری :

0	0	0
1-	1	0
0	0	0

0	0	0
0	1	-1
0	0	0

0	0	0
0	1	0
0	1-	0

0	1-	0
0	1	0
0	0	0

سوال 6)

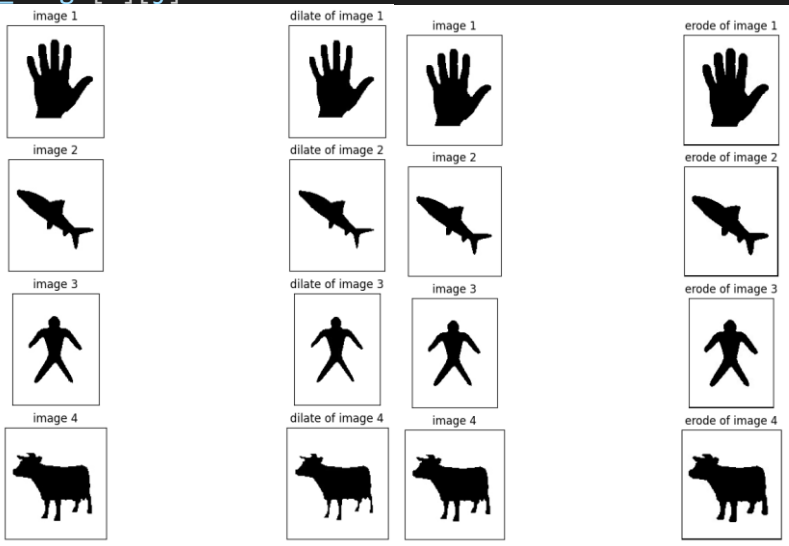
الف)

عملگرگسترش:

به صورت کلی ، عملکرد گسترش وجود حداقل یک پیکسل مشترک با ساختار است.

از روند کانولوشن استفاده می کنیم. ابتدا کرنل را 180 درجه چرخانده و در هر مرحله کانولوشن، ماکسیمم مقدار ضرب کرنل در پنجره تصویر را به عنوان مقدار هر پیکسل قرار می دهیم.

```
a = int((kernel.shape[0])/2)
   kernel = np.flipud(np.fliplr(kernel))
   s, t = kernel.shape

   image = cv2.copyMakeBorder(img, a, a, a ,a, cv2.BORDER_CONSTANT, value=0)
   w, h = image.shape
   new_image = np.zeros(( w - s + 1, h - t + 1))
   # print(new_image.shape , img.shape)
   for i in range( w - s + 1):
       for j in range(h-t+1):
           val = np.max(image[i:i+s, j:j+t]*kernel)
           new_image[i][j] = val
```



ب) عملگر کاهش)

عملگر کاهش به این صورت است که کرنل باید حتما زیر مجموعه ی تصویر باشد و در میان 1 ها مینیمم مقدار پیکسل به عنوان اندازه هر پیکسل قرار داده می شود.

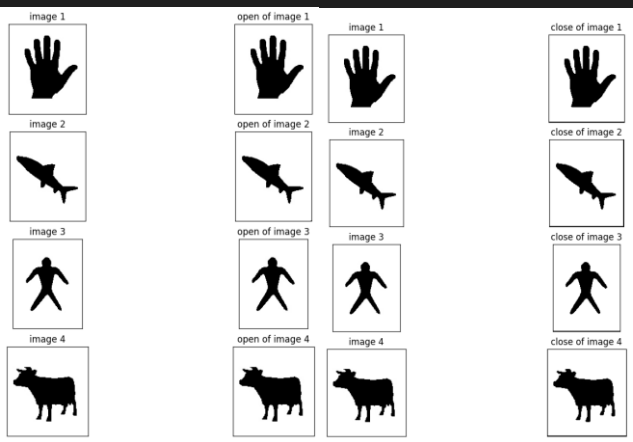
در کد همانند بالایی از کانولوشن استفاده می کند. با این تفاوت که برای مقدار گذاری هر پیکسل و در هر مرحله کانولوشن تعداد صفر های ضرب کرنل در پنجره نباید بیشتر از خود کرنل باشد(یعنی حالتی که زیر مجموعه باشد) در این حالت، مینیمم مقدار ضرب پنجره در کرنل را در نقاط غیر صفر به عنوان نتیجه بر میگردانیم.

```
    a = int((kernel.shape[0])/2)


    s, t = kernel.shape
    kernel_nz =  np.count_nonzero(kernel)
    image = cv2.copyMakeBorder(img, a, a, a ,a, cv2.BORDER_CONSTANT, value=0)
    kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
    w, h = image.shape
    new_image = np.zeros(( w - s + 1, h - t + 1))
    for i in range( w - s + 1):
        for j in range(h-t+1):
            arr = image[i:i+s, j:j+t]*kernel
            non_zeros = np.count_nonzero(arr)
            if(non_zeros  >= kernel_nz):
                new_image[i][j] = np.min(arr[np.nonzero(arr)])

    return new_image
```

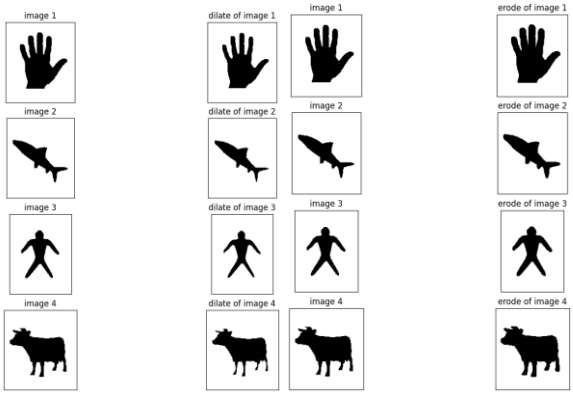
عملگر بازو عملگر بسته)

عملگر باز شامل ابتدا سایش و سپس گسترش تصویر و عملگر بسته ابتدا تصویر را گسترش و سپس کاهش می دهیم.

```
def open_morphology(img, kernel):

    eroded = erode(img , kernel)
    img_opened = dilate(eroded , kernel)
    return img_opened
def close_morphology(img, kernel):
    dilated = dilate(img , kernel)
    img_closed = erode(dilated , kernel)
    return img_closed
```



ب) نتایج OPENCV



همان طور که مشاهده میشود نتایج OPENCV و نتایج کد مشابهت دارد.

ج) (امتیازی)

ابتدا با کمک استانه گیری عکس را باینری و inverseمیکنیم.
یک حلقه ی وایل میزنیم و شرط اتمام آن خالی شدن صفحع توسط سایش است.
روی هر تصویر سایش شده عملگر باز میزنیم و از تصویر سایش شده کم می کنیم.
تصویر کم شده را داخل یک ارایه ریخته و آن را به نتیجه اضافه میکنیم.
تا زمانی که شرط براورده نشود الگوریتم به کار خود ادامه می دهد و در نهایت نتیجه مظلوب است.

$$S(A) = \bigcup_{k=0}^{K} S_k(A)$$

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$$

$$A \ominus kB = ((A \ominus B) \ominus B) \ominus \cdots)$$

$$K = max\{k | (A \ominus kB) \neq \emptyset\}$$

$$A = \bigcup_{k=0}^{K} S_k(A) \oplus kB$$

```
params = []
ret,thereshold = cv2.threshold(image,230,255,cv2.THRESH_BINARY_INV)
image= thereshold
while True:
    eroded= erode(image,kernel)
    opened=open_morphology(eroded , kernel)
    copy= eroded - opened
    result=result + copy
    image= eroded
    params.append(copy)
    if returnnonzero(eroded):
        break
return result
```