سوال1 و 4 مورد الف درamirrezavishtehHw4.pdf است

2)

توضیح روش:

این روش بر اساس واریانس بین کلاسی عمل میکند :

ما میانگین هر دسته را بدست آورده که اختلاف به توان دوی این میانگین‌ها را واریانس بین دسته‌ای میگوییم و هرچه این میانگین‌ها اختلاف بیشتری داشته باشند، نشان‌دهنده این است که عدد threshold ما عدد بهتری بوده و معیار Gaussian otsu عدد بزرگتری خواهد شد.

فرمول:

این روش بدین صورت است که وزنهای دو دسته که تعداد اعضای آن دسته نسبت به کل اعضا است، در هم ضرب شده و حاصل در توان دوی اختلاف میانگین دو دسته ضرب می‌شود. برای حالتی که میانگین هر دو دسته بالا باشند یعنی اختلافشان نزدیک به صفر شود، این معیار نزدیک صفر می‌شود که این یعنی مقدار خوبی برای threshold انتخاب نکردیم.

الف) ما درotsu باید واریانس را برای تمام 256 سطح روشنایی محاسبه کنیم پس سرعت Gaussian otsu همواره بیشتر است زیرا اگر threshold را تغییر دهیم، بدست آوردن میانگین جدید دسته‌ها با کم و زیاد کردن مقادیری که از دسته سمت راست کم شد و به دسته چپی اضافه شد، قابل محاسبه است.

دقت:معیار otsu معیار مناسب تری است زیرا حداقل واریانس بین دو گروه است اما دقت Gaussian otsu وابسته به تفاوت بیتر میانگین دو دسته است در حالتی که دو حالت یا دو mode داشته باشیم بیشتر است، منظور از mode محدوده‌ای است که هیستوگرام مقادیر زیادی دارند. همچنین اگر عکس ما دارای background و foreground باشد، الگوریتم otsu مقدار threshold صفر میدهد. و در باقی موارد یعنی اکثرا otsu بهتر است.

ب) بله، زیرا بیشینه شدن واریانس بین کلاسی بدین معناست که اختلاف میانگین دو دسته بیشتر است که این یعنی دو دسته ما با کاملا با این threshold از هم جدا شده اند یعنی یکسری دسته ای با مقادیر بزرگ یا روشن اند و دیگری با مقدار کم یا تیره اند و چون thresholding به درست است پس هر دسته داده پرت ندارد و در نتیجه واریانس درون آن کلاس کمینه است.

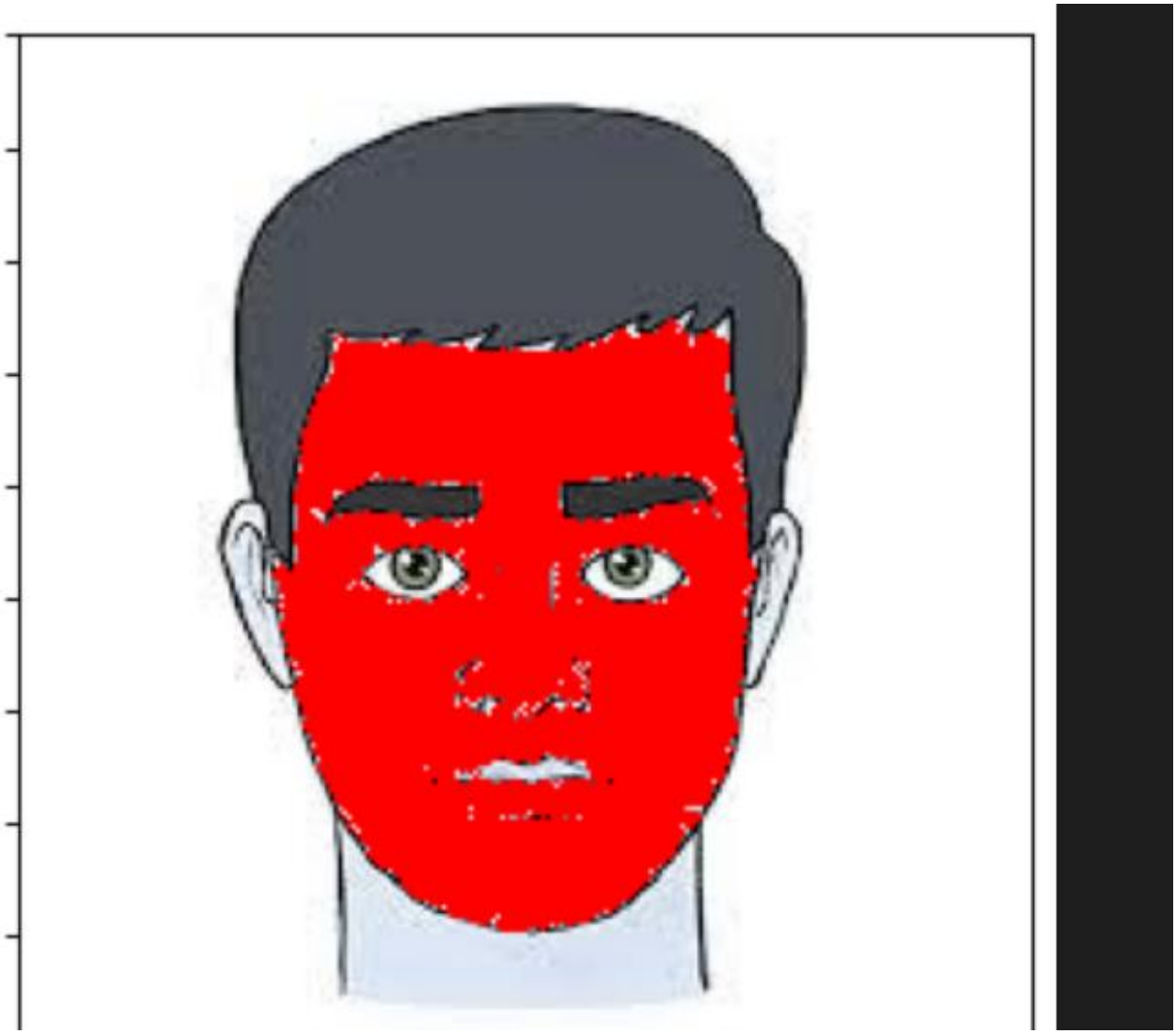
3)

در این سوال ما ابتدا یک seed میگیریم و سپس با الگوریتم dfs هماسایگی 8 تایی انرا تا جایی که نقطه شبیه ان در استک وجود دارد گسترش میدهیم معیار شباهت را در تابع دیگری اندازه گرفتم که در واقع اختلاف میانگین rgb نقاط است

```python
def compare(seed,pixel):
    a = np.mean(seed)
    b = np.mean(pixel)
    t = 10
    if max(a,b) - min(a,b) <= t:
        return True
    return False
```

```python
while len(stack)>0:
    currentNode = stack.pop()  # Get the top node from the stack
    # visited.add(currentNode)
    visited[currentNode[0], currentNode[1]]=1
    if compare(image[seed[0],seed[1]], image[currentNode[0], currentNode[1]]):
        cv2.circle(segmented_image, [currentNode[1],currentNode[0]], 1, (255,0,0), 1)
        neighbors = [[currentNode[0]-1, currentNode[1]], [currentNode[0]+1,currentNode[1]],
                     [currentNode[0], currentNode[1]-1],[currentNode[0],currentNode[1]+1],
                     [currentNode[0]-1, currentNode[1]-1],[currentNode[0]+1, currentNode[1]+1
                     [currentNode[0]-1, currentNode[1]+1],[currentNode[0]+1, currentNode[1]-1
                     ] # Get the neighbors of the current node
```

نتیجه:



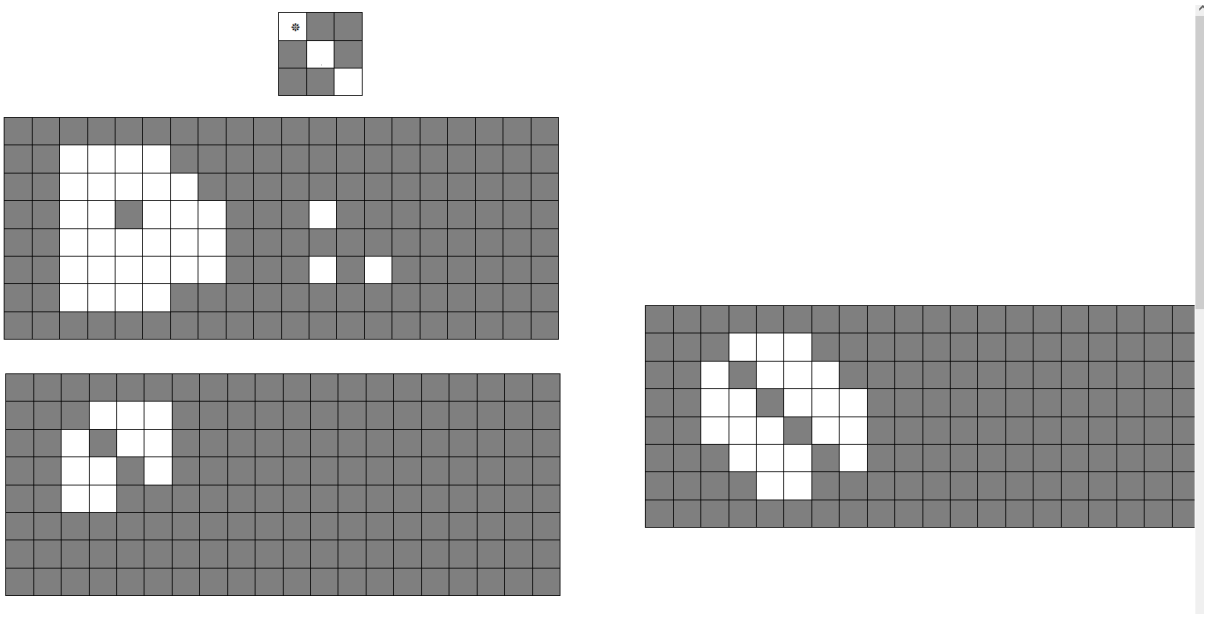
همانطور که دیده میشود چهره شخص به خوبی تشخیص داده شده

4)

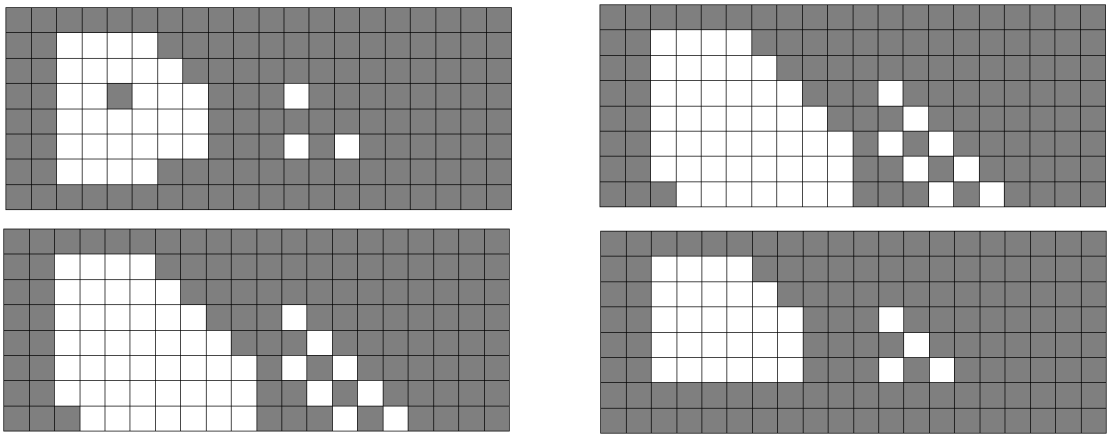
الف)

(ب

باز:



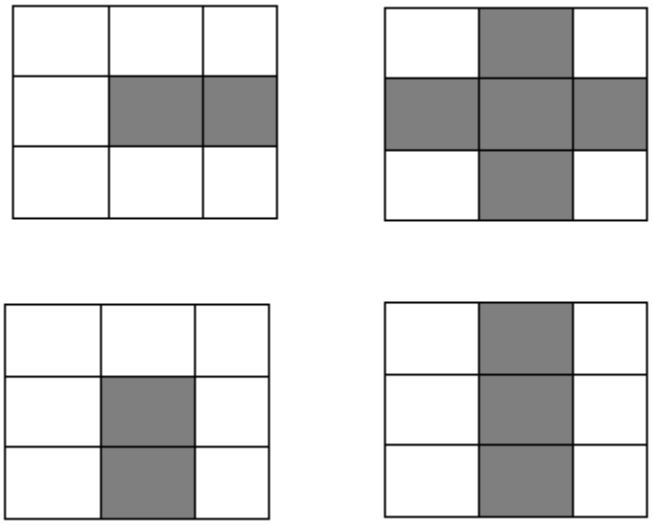
بسته:



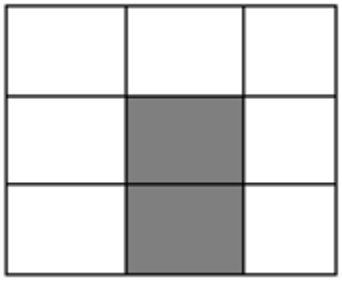
5)

الف:

در شکل من سیاه=1 و سفید =0 درنظر گرفتم

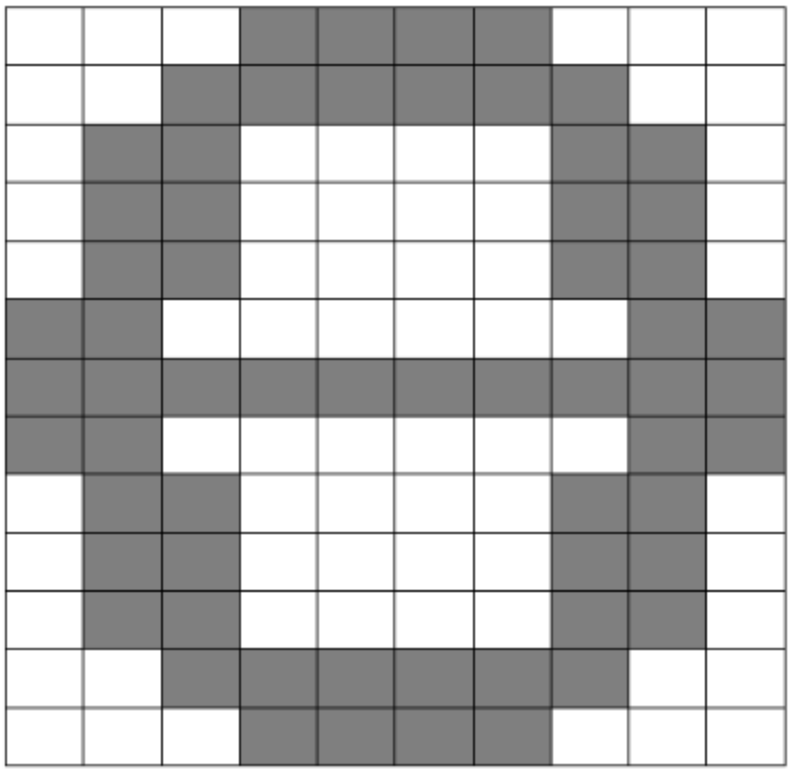


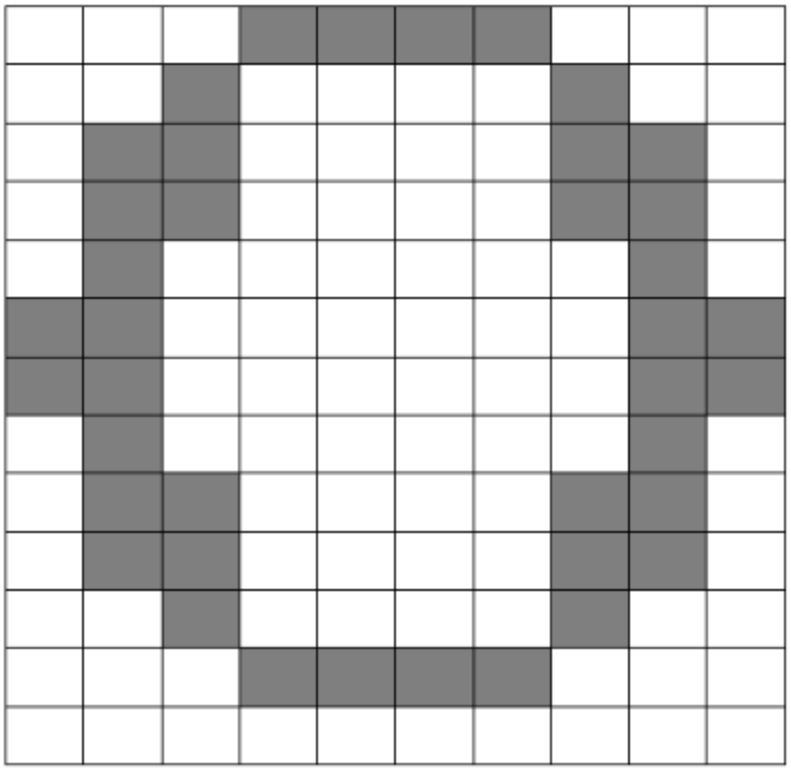
اول میدانیم که باید عملگر باز را استفاده کنیم زیرا میخواهیم پیکسل های اصلی باقی بماند و جزییات خط وسط حذف شود با ازمایش کردن کرنل ها میفهمیم کرنل زیر مناسب است



این به این علت است که در عملگر باز اگر کرنل زیر مجموعه ان بخش عکس باشد کل یک های کرنل در تصویر هم یک میماند پس این کرنل در واقع خط وسط را تنها حذف میکند :

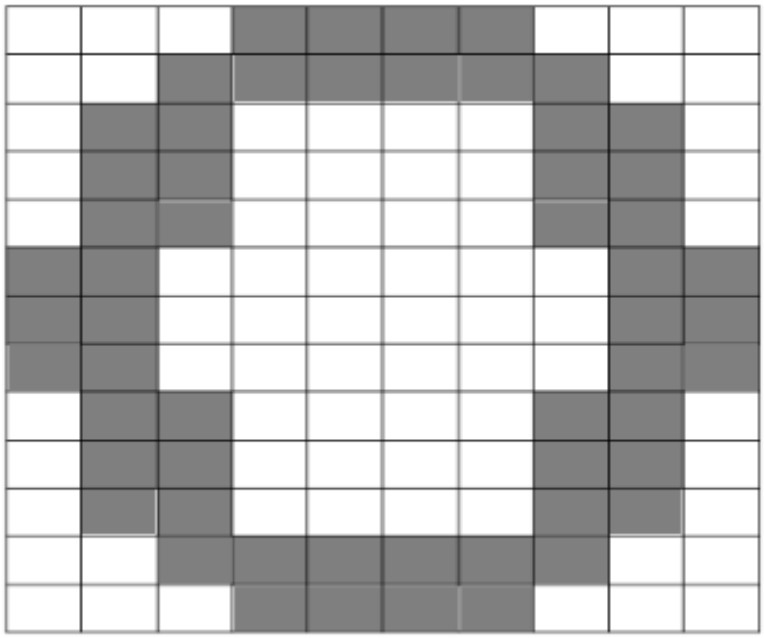
سایش :





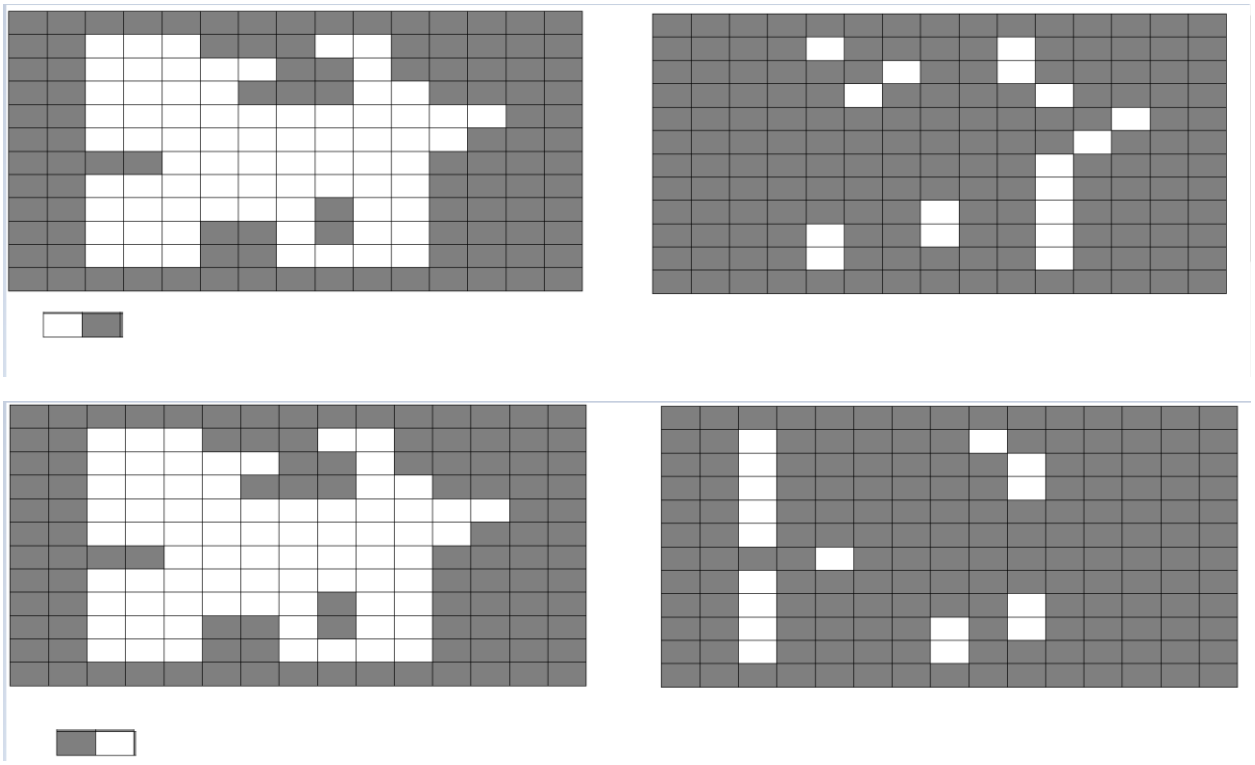
نتیجه سایش که در بالا امده را گسترش میدهیم:

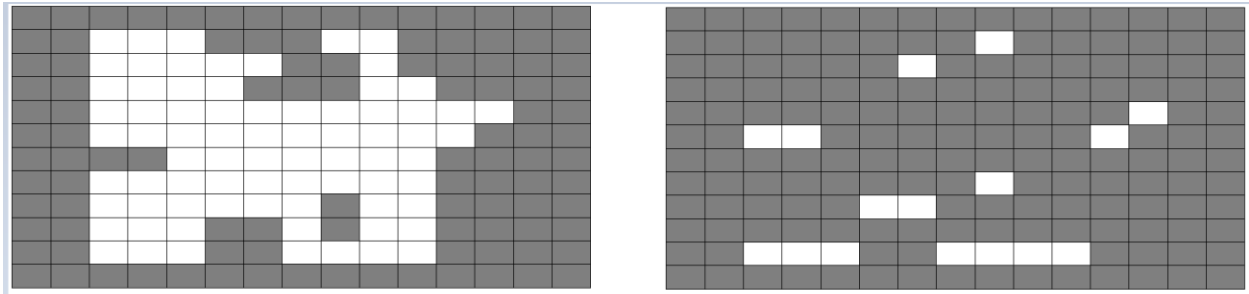
همانطور که میبینید خط وسط حذف میشود:

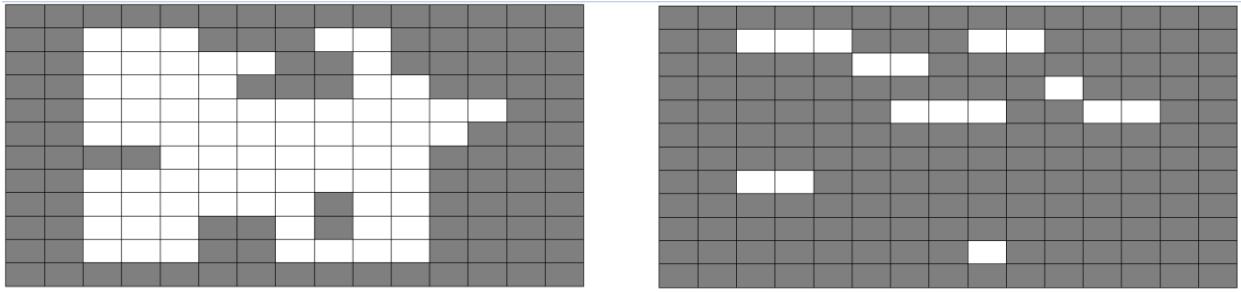


ب)

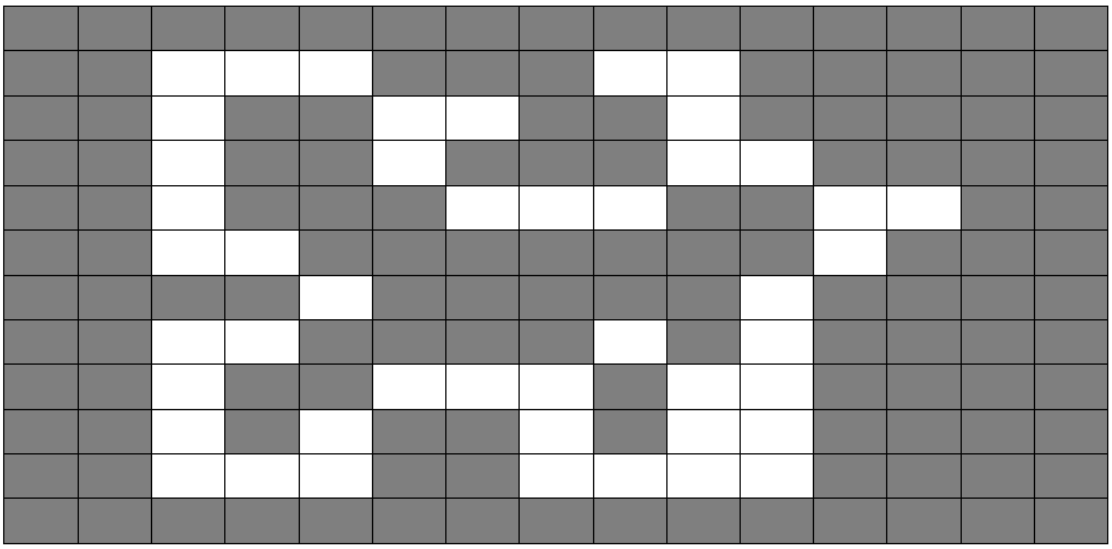
ابتدا تک تک مرز ها را می یابیم و بعد انها را اجتماع میگیریم:







نتیجه نهایی :



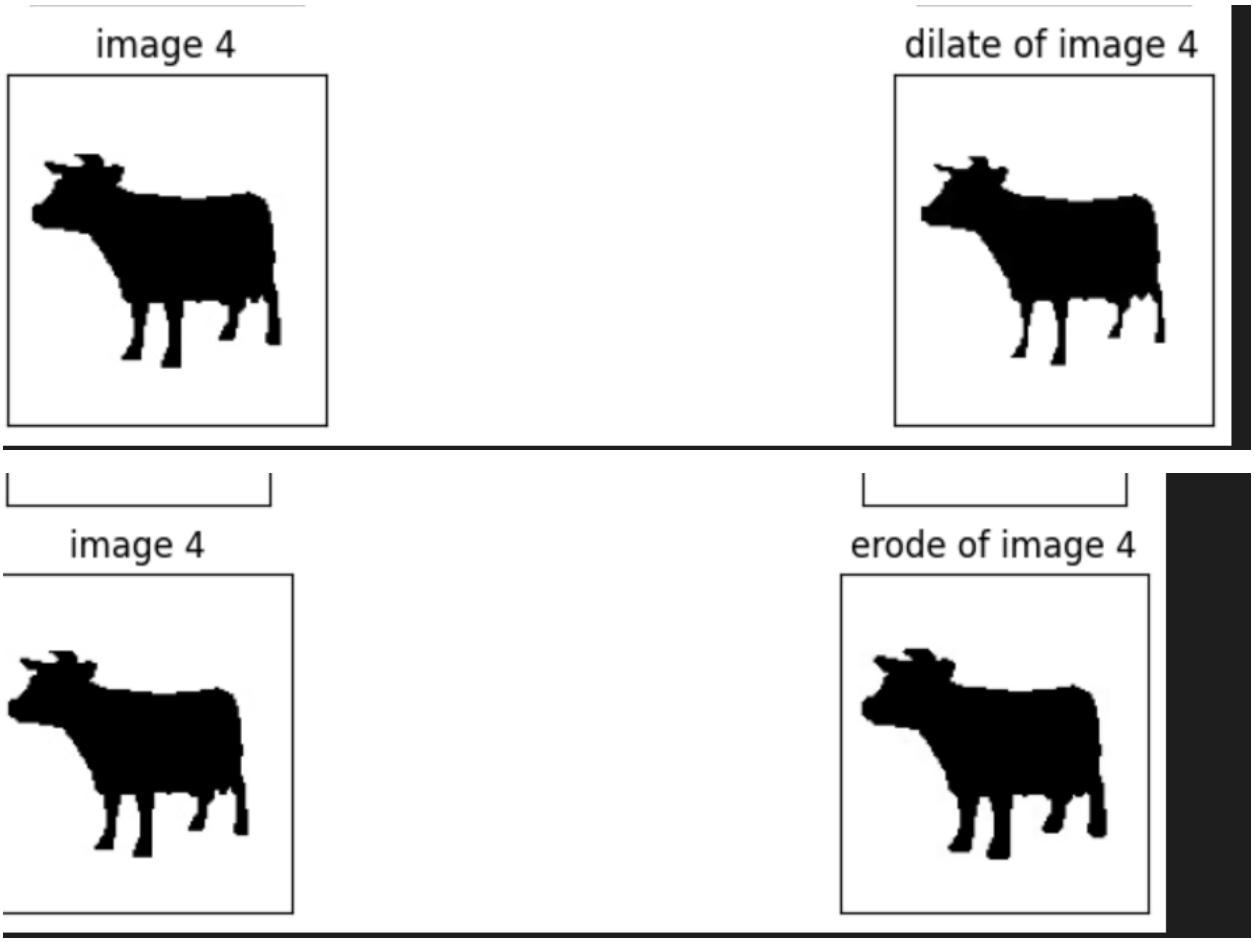
6)

```
'''
pad_height = kernel.shape[0] // 2
pad_width = kernel.shape[1] // 2

image=np.pad(img,((pad_height, pad_height), (pad_width, pad_width)), mode='reflect')
img_dialated = np.zeros(image.shape)
row_image,col_image= image.shape
kernel=np.rot90(kernel,2)
for i in range(1,row_image-pad_width):
    for j in range(1,col_image-pad_height):
        sample=image[i-pad_width:i+pad_width+1,j-pad_height:j+pad_height+1 ]
        dilation = np.max(np.multiply(sample, kernel))
        img_dialated[i,j]=dilation

return img_dialated[pad_width:row_image-pad_width,pad_height:col_image-pad_height]
```

در عملگر گسترش ما ابتدا کرنل را 180 درجه میچرخانیم و بعد کرنل را روی عکس دارای پدینگ حرکت میدهیم اگر بعد از ضرب شدن ان بخش تصویر و کرنل حداقل یک پیکسل یک بود یعنی یک اشتراک داشتند و باید مکس انها هم یک شود و بعد پیکسل مرکزی را یک میکینیم

```
pad_height = kernel.shape[0] // 2
pad_width = kernel.shape[1] // 2

image=np.pad(img,((pad_height, pad_height), (pad_width, pad_width)), mode='reflect')
img_erod = np.zeros(image.shape)
row_image,col_image= image.shape
for i in range(1,row_image-pad_width):
    for j in range(1,col_image-pad_height):

        sample=image[i-pad_width:i+pad_width+1,j-pad_height:j+pad_height+1 ]
        erod =np.min(sample[kernel==1])

        img_erod[i,j]=erod

return img_erod[pad_width:row_image-pad_width,pad_height:col_image-pad_height]
```

در سایش باید کرنل و عکس دقیقا برابر باشند برای اعمال این مسِله بین پیکسل هایی که در کرنل یک اند در ان بخش عکس مینیم میگیریم با این کار اگز حتی یک پیکسل هم متفاوت باشد حاصل صفر میشود



در گسترش پیکسل های سفید گسترش می یابند که باعث کوچک تر شدن ناحیه سفید تصویر میشود و در سایش ناحیه سیاه کمی بزرگتر میشود