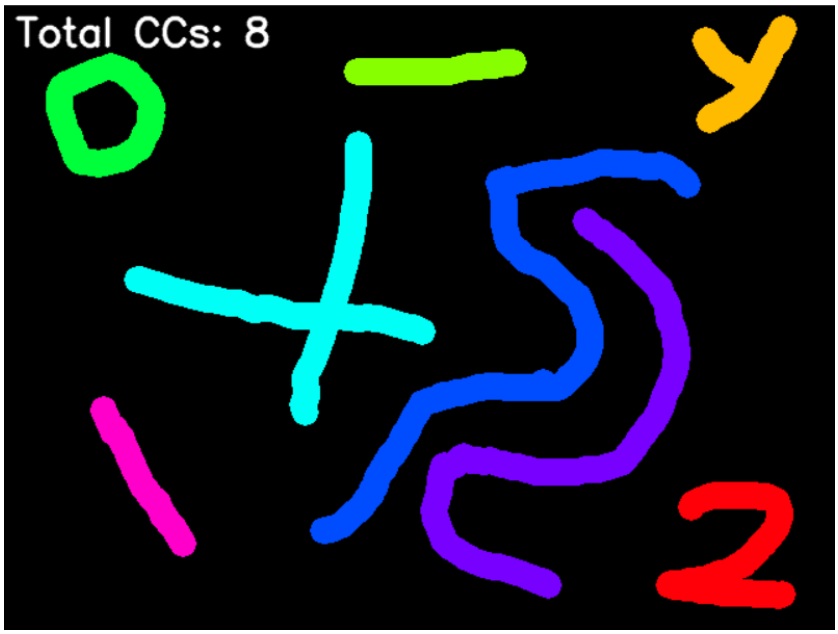
به نام خالق رنگین کمان

ستاره باباجانی – گزارش تمرین سری 5

سوال 1: کد خواسته شده زده شد و برای هر خط آن کامنت مناسب برای درک بهتر آن، قرار داده شد. حال خروجی به دست آمده را در گزارش قرار میدهم:



سوال 2:

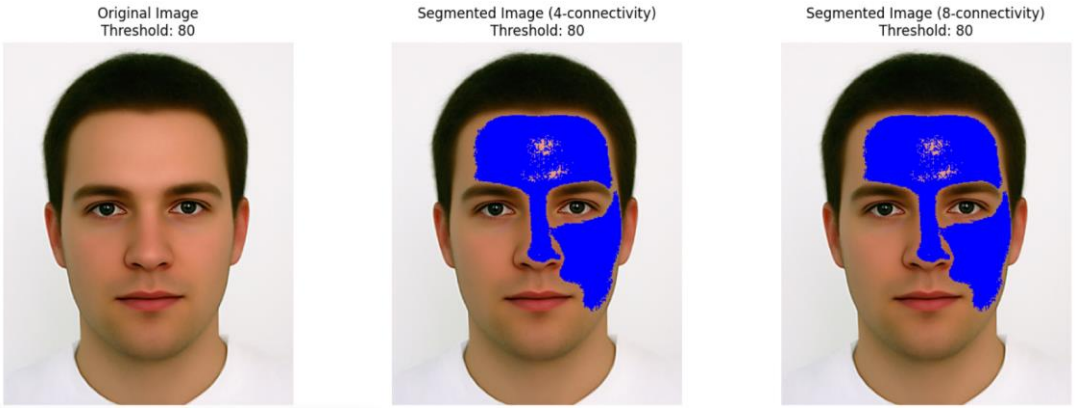
- کنترل آستانه: مقدار آستانه برای کنترل حساسیت الگوریتم در حال رشد منطقه به تفاوت رنگ بسیار مهم است. آستانه‌های پایین‌تر منجر به تقسیم‌بندی محافظه‌کارانه‌تر می‌شوند و مناطقی را که از نظر رنگ بسیار

شبیه به نقطه بذر هستند، می‌گیرند. آستانه های بالاتر به الگوریتم اجازه می دهد تا مناطقی با تفاوت رنگ بیشتر را شامل شود.

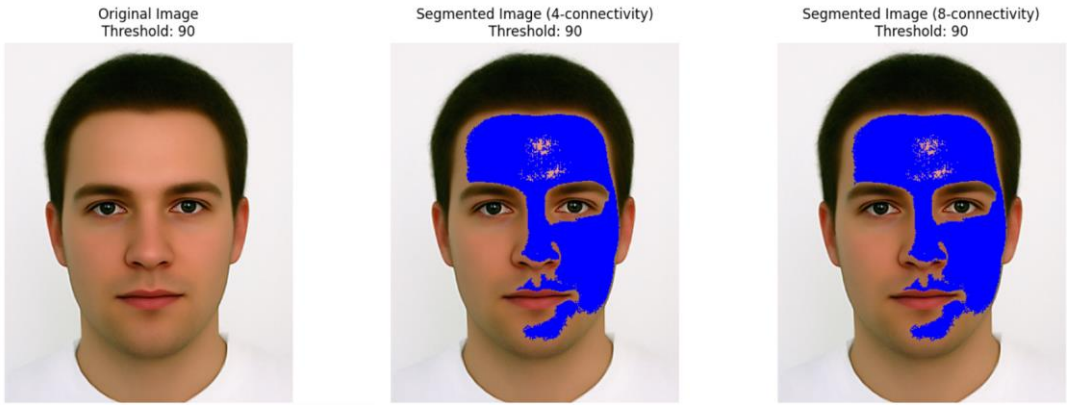
- 4-Connectivity: 8-Connectivity در مقابل 4-Connectivity محدودتر است، فقط پیکسل های مجاور را می گیرد. این حالت می تواند منجر به مناطق دقیق تر اما بالقوه کوچکتر شود.8-Connectivity فراگیرتراست. این حالت اغلب منجر به ایجاد مناطق تقسیم‌بندی شده بزرگ‌تر، ثبت جزئیات دقیق‌تر و مناطق وسیع‌تر می‌شود.

حال به بررسی نتایج آستانه های مختلف میپردازیم:

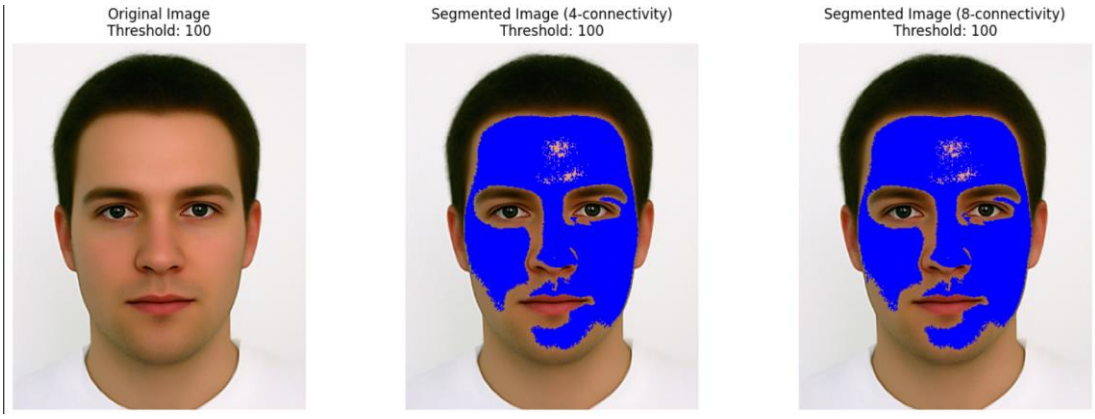
1. آستانه 80:



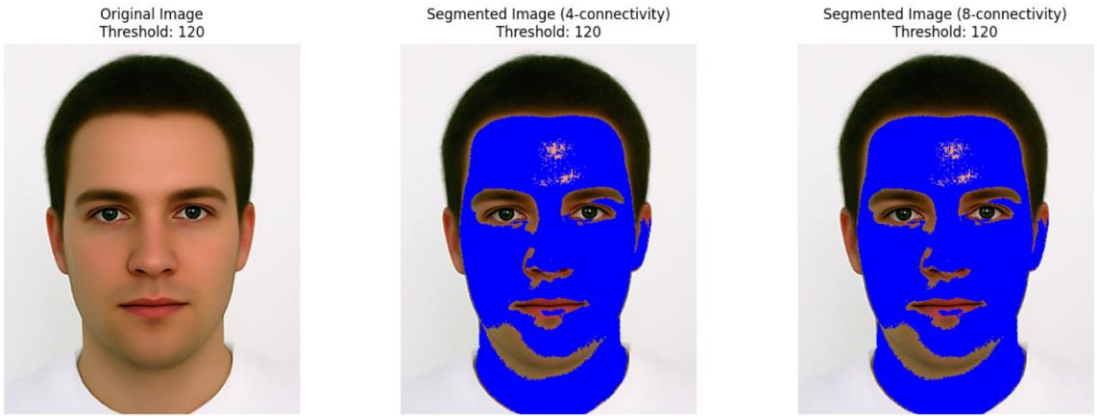
2. آستانه 90:



3. آستانه 100:



4. آستانه 120:



سوال 3: ابتدا بصورت رندوم یک تصویر با شرایط گفته شده ایجاد میکنیم:

2	12	7	9	1
6	10	5	8	4
3	14	15	11	7
13	2	8	5	3
12	6	9	7	10

سپس مراحل الگوریتم otsu را در پیش میگیریم:

- محاسبه هیستوگرام:

```
Value:     1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
Frequency: 1  2  2  1  2  2  3  2  2  2  1  2  1  1  1
```

- Probability distribution: تقسیم frequency بر تعداد کل پیکسل ها(25):

P(1) = 0.04, P(2) = 0.08, P(3) = 0.08, P(4) = 0.04, P(5) = 0.08, P(6) = 0.08, P(7) = 0.12, P(8) = 0.08, P(9) = 0.08, P(10) = 0.08, P(11) = 0.04, P(12) = 0.08, P(13) = 0.04, P(14) = 0.04, P(15) = 0.04

- Class Probabilities and Means:

1. حد آستانه 6: میدانیم:

Background (class 0): Values from 1 to 6

Foreground (class 1): Values from 7 to 15

پس خواهیم داشت:

W0 = Sum(P[0:6]) = 0.04 + 0.08 + 0.08 + 0.04 + 0.08 + 0.08 = 0.4

W1 = Sum(P[7:15]) = 1 - W0 = 0.6

پس:

μ0 = (1*0.04 + 2*0.08 + 3*0.08 + 4*0.04 + 5*0.08 + 6*0.08) / 0.4

= (0.04 + 0.16 + 0.24 + 0.16 + 0.4 + 0.48) / 0.4

= 3.7

μ1 = (7*0.12 + 8*0.08 + 9*0.08 + 10*0.08 + 11*0.04 + 12*0.08 + 13*0.04 + 14*0.04 + 15*0.04) / 0.6

= (0.84 + 0.64 + 0.72 + 0.8 + 0.44 + 0.96 + 0.52 + 0.56 + 0.6) / 0.6

= 10.8

2. حد آستانه 10: میدانیم:

Background (class 0): Values from 1 to 10

Foreground (class 1): Values from 11 to 15

پس خواهیم داشت:

W0 = Sum(P[0:10]) = 0.04 + 0.08 + 0.08 + 0.04 + 0.08 + 0.08 + 0.12 + 0.08 + 0.08 + 0.08 = 0.76

W1 = Sum(P[11:15]) = 1 - W0 = 0.24

پس:

μ0 = (1*0.04 + 2*0.08 + 3*0.08 + 4*0.04 + 5*0.08 + 6*0.08 + 7*0.12 + 8*0.08 + 9*0.08 + 10*0.08) / 0.76

= (0.04 + 0.16 + 0.24 + 0.16 + 0.4 + 0.48 + 0.84 + 0.64 + 0.72 + 0.8) / 0.76

= 5.89

μ1 = (11*0.04 + 12*0.08 + 13*0.04 + 14*0.04 + 15*0.04) / 0.24

= (0.44 + 0.96 + 0.52 + 0.56 + 0.6) / 0.24

= 12.83

- Inter-class Variance: اگر داشته باشیم:

$$\sigma_B^2(t) = W_0(t) \cdot \mathrm{Var}_0(t) + W_1(t) \cdot \mathrm{Var}_1(t)$$

$$\text{Var}_0 = \frac{1}{W_0} \sum_{i=0}^{t} P(i) \cdot (i - \mu_0)^2$$
$$\text{Var}_1 = \frac{1}{W_1} \sum_{i=t+1}^{L-1} P(i) \cdot (i - \mu_1)^2$$

آنگاه برای هر حد آستانه خواهیم داشت:

1. حد آستانه 6:

Var0 = 2.81
Var1=6.83
⇨ 0.4 * 2.81 + 0.6 * 6.83 = 5.222

2. حد آستانه 10:

Var0=7.45
Var1=1.81
⇨ 0.76 * 7.45 + 0.24 * 1.81 = 6.096

که چون Inter-class Variance برای حد آستانه 6 کمتر است، پس حد آستانه بهتری میباشد.

سوال 4: همان طور که میدانیم آستانه گذاری افقی با cv2.adaptiveThreshold انجام میشود که ورودی های زیر را دارد:

- thresholdType: دو مقدار دارد که بیانگر سفید بودن یا نبودن رنگ پس زمینه است.
- BlockSize: ابعاد پنجره مورد استفاده در محاسبه میانگین. (هر چه بزرگتر باشد، آستانه گذاری بیشتر سراسری میشود.)
- C: مورد استفاده در شیفت دادن آستانه نسبت به میانگین پنجره.

حال به بررسی هر کدام از عکس ها میپردازیم:

1. عکس q4_1: همان طور که در تصویر مشاهده میشود، نوشته های سمت چپ پایین مشخص نیستند(به خوبی binary نشده اند) پس blockSize بزرگی مثل 41 دارد و چون در مقایسه با بقیه کمی روشن تر است پس احتمالا C آن کوچکتر (مثل 5) دارد. همچنین thresholdType آن THRESH_BINARY میباشد.
2. عکس q4_2: همان طور که در تصویر مشاهده میشود، اکثر نواحی به دقت خوبی binary شده اند و واضح اند پس blockSize کوچکی مثل 21 دارد و چون کمی تیره تر از حالت عادی است پس احتمالا C آن بزرگ (مثل 30) است تا از حالت میانگین، تیره تر شود. همچنین thresholdType آن THRESH_BINARY میباشد.
3. عکس q4_3: همان طور که در تصویر مشاهده میشود، نوشته های سمت چپ مشخص نیستند(به خوبی binary نشده اند) پس blockSize بزرگی مثل 41 دارد و همچنین خطوط مداد و مورب خیلی پررنگ تر از حالت عادی هستند که نشان دهنده C بزرگ(مثل 30) میباشد. همچنین thresholdType آن THRESH_BINARY میباشد.
4. عکس q4_4: همان طور که در تصویر مشاهده میشود، اکثر نواحی به دقت خوبی binary شده اند و واضح اند پس blockSize کوچکی مثل 21 دارد و همچنین در مقایسه با تصویر q4_2 خطوط مداد و مورب

عادی هستند(پررنگ تر یا کم رنگ تر از میانگین نیستند) پس احتمالا C مقدار کوچکی مثل 5 دارد. همچنین thresholdType آن THRESH_BINARY میباشد.

5. عکس q4_5: همان طور که در تصویر مشاهده میشود، نوشته های سمت چپ پایین مشخص نیستند(به خوبی binary نشده اند) پس blockSize بزرگی مثل 41 دارد و چون کمی روشن است پس احتمالا C آن کوچکتر (مثل 5) دارد. همچنین thresholdType آن THRESH_BINARY_INVمیباشد زیرا پس زمینه سیاه شده است.

(روشنایی های بیشتر از آستانه به صفر و کمتر از آستانه به 255، تناظر یافته اند)

سوال 5: ابتدا reflect padding را بر روی تصویر اولیه اعمال میکنیم:

22	22	22	22	33	22	22	33	22	22
22	22	22	22	33	22	22	33	22	22
22	22	33	33	33	33	33	33	22	22
22	22	22	22	33	22	33	44	22	22
22	22	22	33	44	22	33	22	22	22
22	22	22	44	22	22	44	33	22	22
33	33	22	44	22	44	33	33	22	22
33	33	33	33	33	33	22	33	22	22
33	33	33	44	33	22	44	22	44	44
33	33	33	44	33	22	44	22	44	44

در مرحله بعد عملگرهای سایش و گسترش را بر روی تصویر بالا اعمال میکنیم.

برای سایش عنصر ساختاری را بدون تغییر روی خانه به خانه تصویر اعمال میکنیم. تصویر حاصل از عملگر سایش:

22	22	22	22	33	22	22	33	22	22
22	22	22	22	22	22	22	22	22	22
22	22	22	22	22	22	22	22	22	22
22	22	22	22	22	33	22	22	22	22
22	22	22	22	22	22	22	22	22	22
22	22	22	22	22	22	22	22	22	22
33	22	22	22	22	22	22	22	22	22
33	22	22	22	22	22	22	22	22	22
33	33	33	33	33	22	22	22	22	44
33	33	33	44	33	22	44	22	44	44

برای گسترش باید در مرحله اول structure element را 180 درجه نسبت به مرکزش بچرخانیم و سپس آن را برروی تصویر اعمال کنیم. که حاصل این کار به ما این عنصر ساختاری را میدهد:

1	1	1
1	0	0
1	0	0

تصویر حاصل از عملگر گسترش:

22	22	22	22	33	22	22	33	22	22
22	33	33	33	33	33	33	33	33	22
22	33	33	33	33	33	44	44	44	22
22	33	33	44	44	44	44	33	22	22
22	22	44	44	44	44	44	44	33	22

22	33	44	44	44	44	44	33	33	22
33	33	44	33	44	44	33	33	33	22
33	33	44	44	44	44	44	44	44	22
33	33	44	44	44	44	44	44	44	44
33	33	33	44	33	22	44	22	44	44

سوال 6: ابتدا مرحله به مرحله محاسبات را انجام میدهیم:

- $(A \ominus B_1)$:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

- $(A^c \ominus B_2)$:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

- حال اشتراک آنها را به دست میاوریم:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

که این دو پیکسل پیدا میشوند(گوشه های سمت راست بالا را پیدا میکند.)

سوال 8: الف) مراحل ساخت اسکلت تصاویر به شرح زیر است:

- پیاده سازی توابع neighbours و transitions:

```python
def neighbours(x, y, image):
    """Return 8-neighbours of image point P1(image[x,y]), in a clockwise order"""
    img = image
    return [img[x-1, y], img[x-1, y+1], img[x, y+1], img[x+1, y+1],
            img[x+1, y], img[x+1, y-1], img[x, y-1], img[x-1, y-1]]

def transitions(neighbours):
    """Number of 0,1 patterns (transitions from 0 to 1) in the ordered sequence"""
    n = neighbours + neighbours[0:1]  # P2, P3, P4, P5, P6, P7, P8, P9, P2
    return sum((n1, n2) == (0, 1) for n1, n2 in zip(n, n[1:]))
```

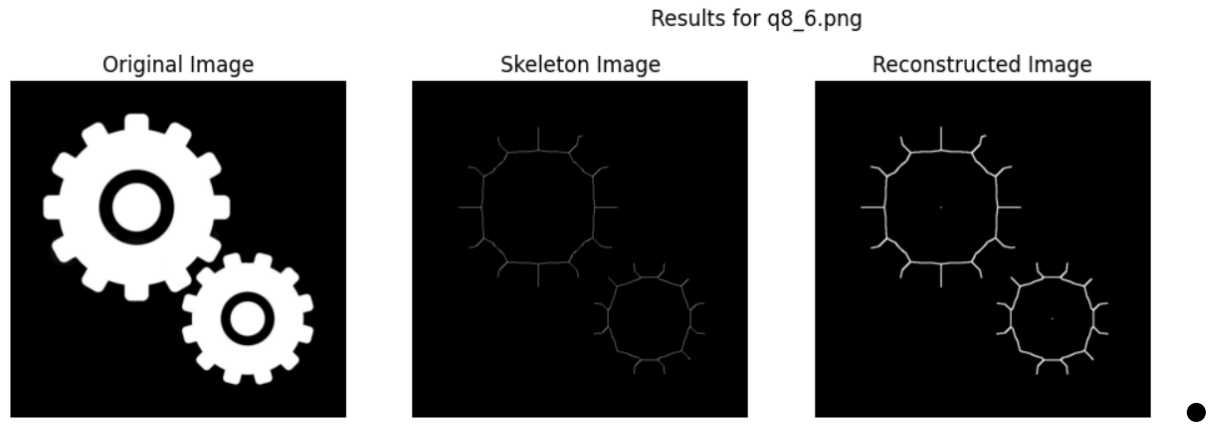
- تابع اصلی:

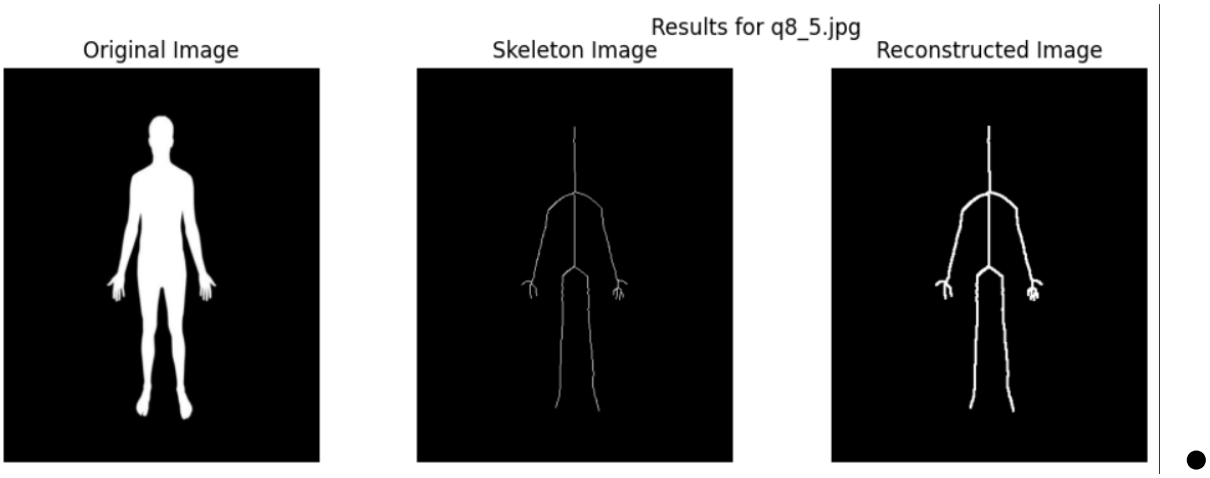
```python
def zhang_suen_iteration(image, iter):
    """Perform one iteration of the Zhang-Suen thinning algorithm"""
    changing = []
    rows, columns = image.shape
    for x in range(1, rows - 1):
        for y in range(1, columns - 1):
            P2, P3, P4, P5, P6, P7, P8, P9 = neighbours(x, y, image)
            if (image[x, y] == 1 and
                2 <= sum([P2, P3, P4, P5, P6, P7, P8, P9]) <= 6 and
                transitions([P2, P3, P4, P5, P6, P7, P8, P9]) == 1 and
                ((P2 * P4 * P6 == 0 and P4 * P6 * P8 == 0) if iter == 0 else
                 (P2 * P4 * P8 == 0 and P2 * P6 * P8 == 0))):
                changing.append((x, y))
    for x, y in changing:
        image[x, y] = 0
    return image, changing

img = image.copy()
prev_img = np.zeros(img.shape)
steps = []

while not np.array_equal(img, prev_img):
    prev_img = img.copy()
    img, changing = zhang_suen_iteration(img, 0)
    steps.append(changing)
    img, changing = zhang_suen_iteration(img, 1)
    steps.append(changing)

return img, steps
```

ب) بازسازی عکس اصلی:

```python
def skeleton_to_binary(skeleton_image):
    # Initialize an array of zeros with the same shape as the skeletonized image
    reconstructed_image = np.zeros_like(skeleton_image)
    rows, columns = skeleton_image.shape
    for x in range(1, rows - 1):
        for y in range(1, columns - 1):
            # Check if the current pixel is part of the skeleton
            if skeleton_image[x, y] == 1:
                # Set the current pixel and its neighbors to 1 in the reconstructed image
                reconstructed_image[x-1:x+2, y-1:y+2] = 1

    return reconstructed_image
```

خروجی ها به شرح زیر هستند:





سوال 9: با کمک عناصر ساختاری زیر، مرزهای 4 جهت راست، چپ، پایین، بالا، را بدست می آوریم. سپس بعد از اعمال hit or miss هر یک از عناصر را با هم اجتماع گرفته و نقاط مرزی نهایی را بدست می آوریم:

0	-1	0
0	1	0
0	0	0

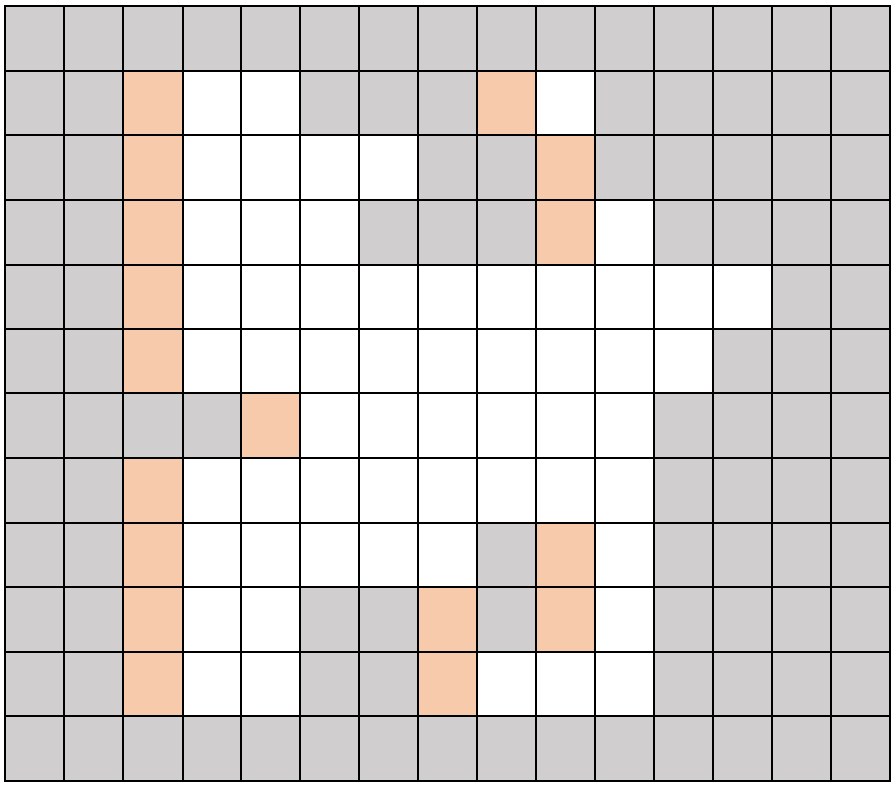
0	0	0
0	1	0
0	-1	0

0	0	0
0	1	-1
0	0	0

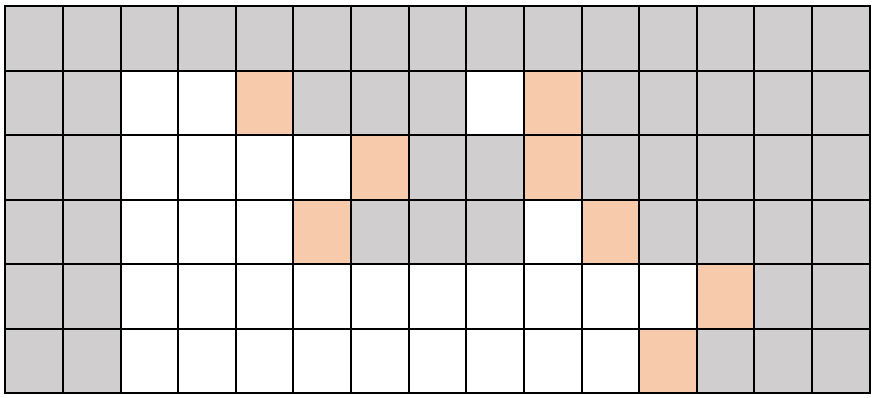
0	0	0

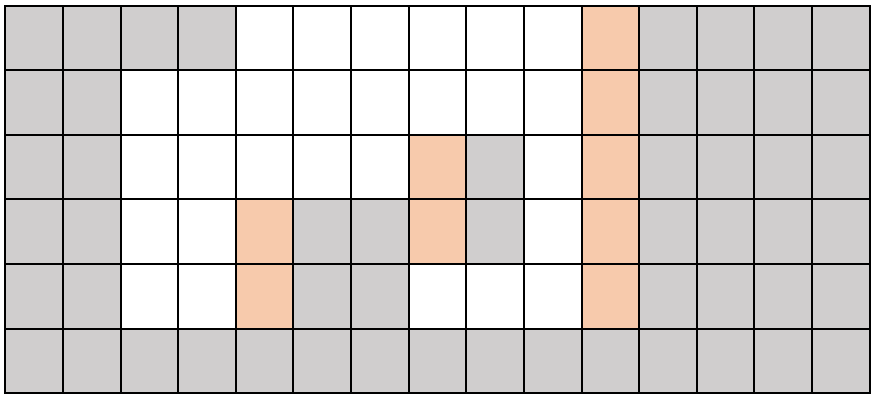
-1	1	0
0	0	0

- مرز چپ:

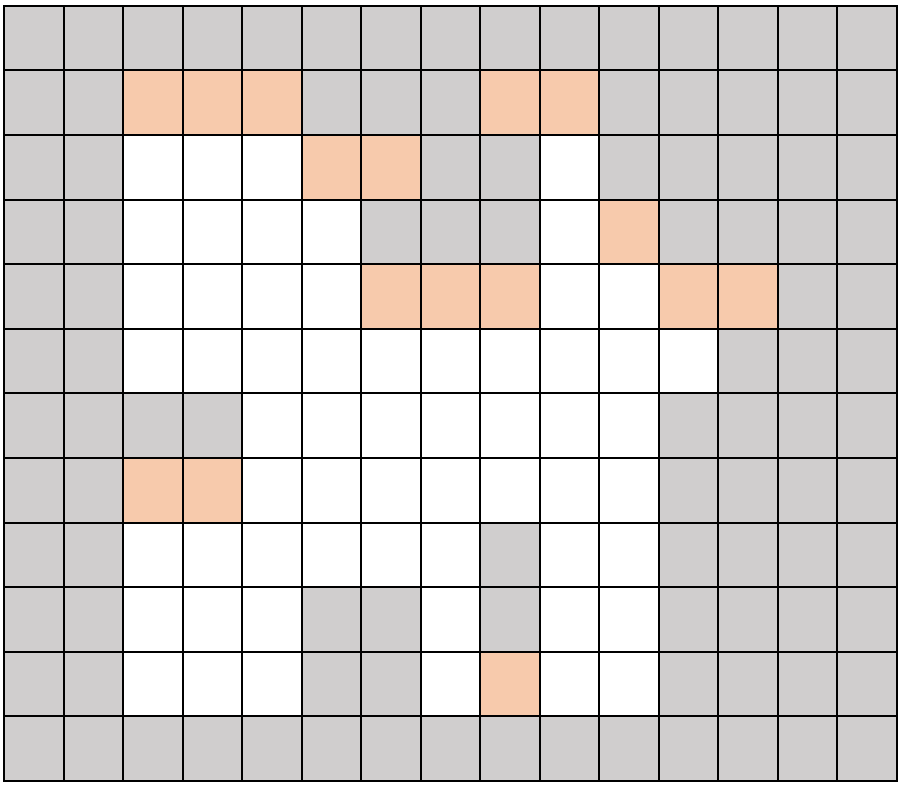


- مرز راست:

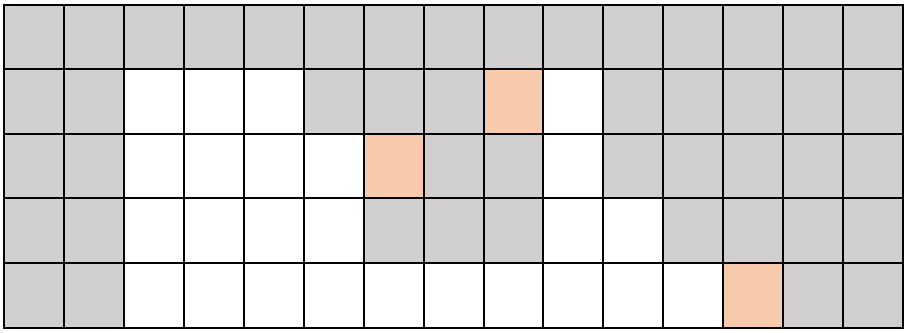


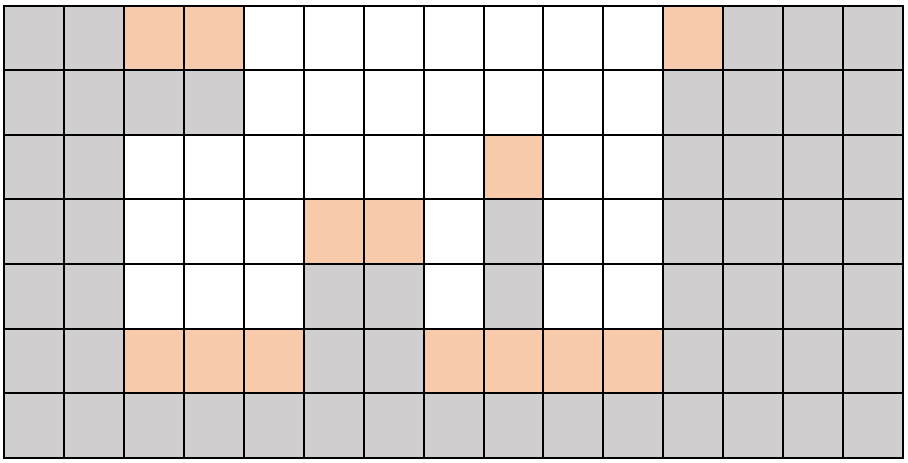


- مرز بالا:



- مرز پایین:





تصویر نهایی حاصل از اجتماع 4 حالت مرزی بالا: که خانه های زرد رنگ جواب نهایی بوده و نقاط مرزی تصویر هستند.