تمرین سری پنجم بینایی کامپیوتر

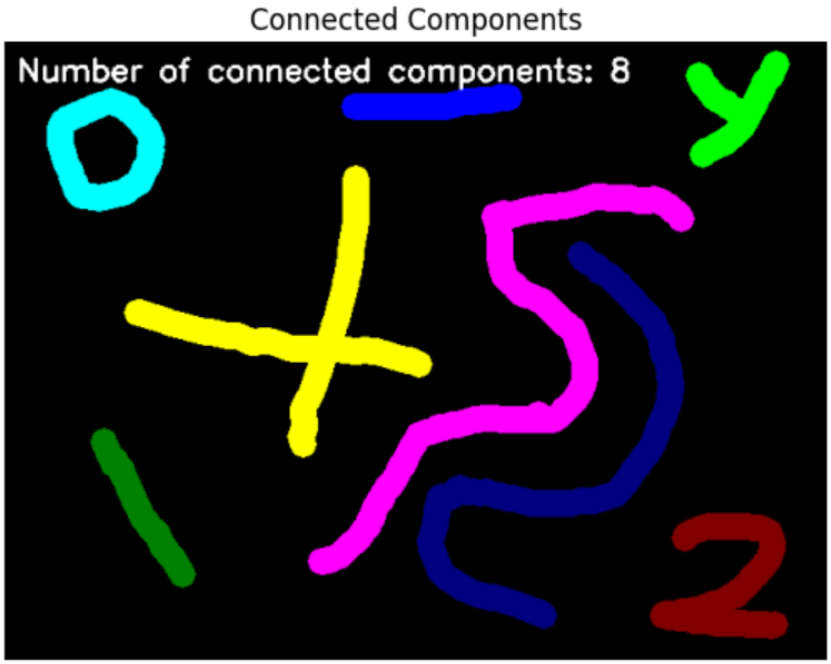
دکتر محمدی

هلیا شمس زاده

۴۰۰۵۲۱۴۸۶

سوال اول)

پس از اینکه با دستور cv2.imread، تصویر را خواندیم، تابع cv2.threshold را با پارامترهای cv2.THRESH_Binary و cv2.THRESH_OTSU صدا می‌زنیم تا حد آستانهٔ مناسب برای یافتن اجزا را پیدا کند. این پارامترها می‌گویند مقادیر بزرگتر از حد آستانهٔ Otsu برابر ۲۵۵ و مقادیر کمتر از حد آستانهٔ Otsu برابر صفر باشند. سپس تصویر به دست آمده از مرحلهٔ قبل را به دست آمده را به تابع cv2.connectedComponentsWithStat داده neighbor mode را برابر ۴ می‌گذاریم تا فقط ۴ تا همسایه را مورد بررسی قرار دهد. سپس هر label (همان component) پیدا شده را به رنگی متفاوت رنگ می‌کنیم و در خروجی نمایش می‌دهیم و تعداد را هم روی تصویر چاپ می‌کنیم. نتیجه:

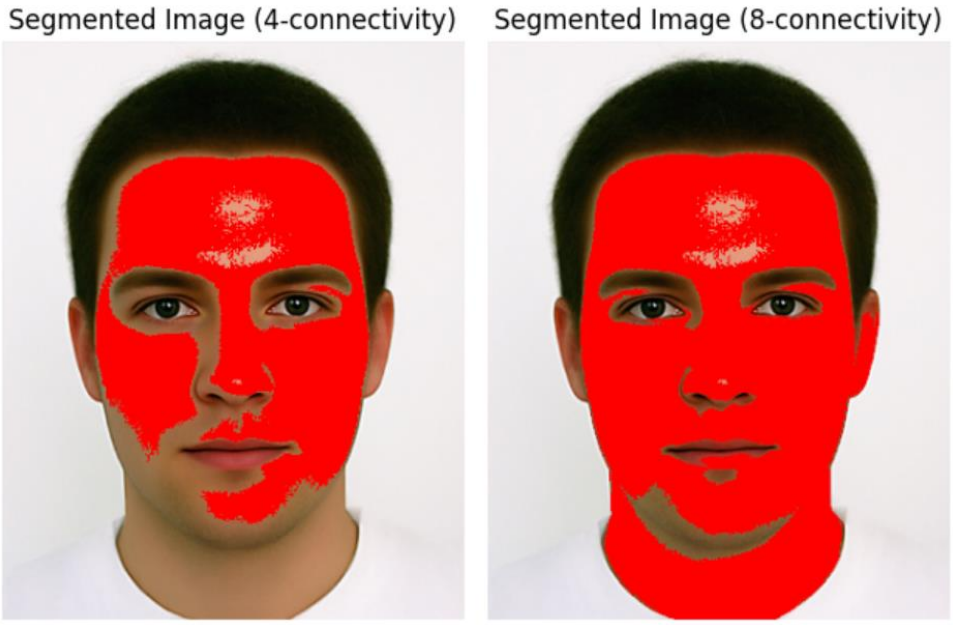


سوال دوم)

در ابتدا با دستور cv2.imread، تصویر را خوانده و سسپس با cv2.cv2Color، تصویر را به فرمت RGB تبدیل می‌کنیم و آن را نمایش می‌دهیم. نتیجه:

سپس تابع region_growing را کامل می‌کنیم. ابتدا یک کپی از تصویر ورودی گرفته و سپس بر اساس neighbor_mode، پوزیشن همسایه‌ها را تعیین می‌کنیم. (اگر صفر باشد یعنی 4-connectivity است و فقط پیکسل‌های بالا، پایین، چپ و راست به عنوان همسایه هستند و اگر یک باشد یعنی از نوع 8-connectivity است و تمام پیکسل‌های موجود در یک پنجرهٔ 3*3 به عنوان همسایه مد نظر است) پس از آن بر حسب تصویر ورودی طول و ارتفاع را پیدا می‌کنیم. نقطهٔ بذر را در داخل پشتهٔ region_point که سگمنت نهایی در آن قرار می‌گیرد را به عنوان نقطهٔ شروع در این لیست قرار می‌دهیم. سپس الگوریتم اصلی اجرا می‌شود. آخرین نقطهٔ موجود در استک (در ابتدا همان نقطهٔ یذر) پاپ شده، فلگ visited آن True شده و به ازای همسایه‌های دور آن در صورتی که اختلافش با پیکسل بذر و همسایه‌هایش از threshold کمتر باشد، به داخل استک اضافه شده و visited آن true می‌شود. و دوباره یک نقطه از استک پاپ شده و همسایه‌هایش پردازش می‌شوند و این روند تا خالی شدن استک ادامه می‌یابد. حد آستانه نیز به صورت آزمایشی انتخاب شده است.

نتیجه به ازای mode صفر و یک:



فرق 4-connectivity و 8-connectivity:

در حالت 4-connectivity ناحیه کندتر رشد می‌کند و تصویر نهایی حالت مربعی تری می‌گیرد چون همسایه‌های گوشه بررسی نمی‌شوند. 8-connectivity به سرعت رشدکرده و مرز ناحیه یکنواخت‌تر و بهتر می‌شود و پیکسل‌های بیشتری بررسی می‌شوند.

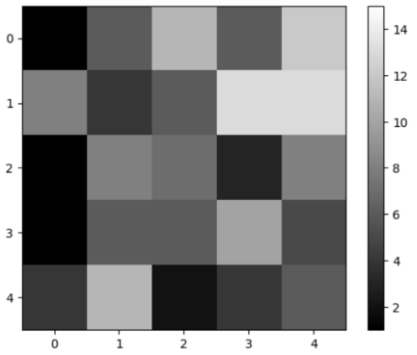
سوال سوم)

با استفاده از کد زیر، ماتریس رندوم تولید می‌شود:

```
random_image = np.random.randint(1, 16, size=(5, 5))
print(random_image)
plt.imshow(random_image, cmap='gray', vmin=1, vmax=15)
plt.colorbar()
plt.show()
```

ماتریس رندوم به دست آمده:

$$matrix = \begin{bmatrix} 1 & 6 & 11 & 6 & 12 \\ 8 & 4 & 6 & 13 & 13 \\ 1 & 8 & 7 & 3 & 8 \\ 1 & 6 & 6 & 10 & 5 \\ 4 & 11 & 2 & 4 & 6 \end{bmatrix}$$



فرمول الگوریتم Ostu:

$$\sigma_w^2 = w_1\sigma_1^2 + w_2\sigma_2^2$$

w_i تعداد پیکسل‌های کلاس i، و σ_i^2 واریانس پیکسل‌های آن کلاس است

فرمول واریانس:

$$\sigma^2 = \frac{\sum_{i=0}^{n} x_i - \mu}{n}$$

اعمال Otsu برای حد آستانهٔ ۶:

به این منظور، پیکسل‌هایی که مقدارشان از ۶ کمتر است در دستهٔ اول قرار می‌گیرند، و پیکسل‌هایی که مقدرشان بزرگتر یا مساوی ۶ است در دستهٔ دوم قرار می‌گیرند. طبق این آستانه، ماتریس رندوم تولید شده به شکل زیر دسته‌بندی می‌شود (پیکسل‌های آبی در دستهٔ اول و پیکسل‌های زرد در دستهٔ دوم قرار می‌گیرند):

$$matrix = \begin{bmatrix} 1 & 6 & 11 & 6 & 12 \\ 8 & 4 & 6 & 13 & 13 \\ 1 & 8 & 7 & 3 & 8 \\ 1 & 6 & 6 & 10 & 5 \\ 4 & 11 & 2 & 4 & 6 \end{bmatrix}$$

تعداد پیکسل‌های دستهٔ اول (آبی) برابر است با ۹ و تعداد پیکسل‌های دستهٔ دوم (زرد) برابر است با ۱۶.

- حال میانگین و واریانس دستهٔ اول را حساب می‌کنیم:

$$\mu_1 = \frac{1+4+1+1+4+3+2+4+5}{9} = \frac{25}{9} \approx 2.78$$

$$\sigma_1 = \frac{3\times(1-2.77)^2+(2-2.77)^2+(3-2.77)^2+3\times(4-2.77)^2+(5-2.77)^2}{9} \approx 2.17$$

- حال میانگین و واریانس دستهٔ دوم را حساب می‌کنیم:

$$\mu_2 = \frac{6\times 6+7+3\times 8+10+2\times 11+12+2\times 13}{16} = \frac{137}{16} \approx 8.56$$

$$\sigma_2 = \frac{6\times(6-8.56)^2+(7-8.56)^2+3\times(8-8.56)^2+(10-8.56)^2+2\times(11-8.56)^2+(12-8.56)^2+2\times(13-8.56)}{16} \approx 6.75$$

- حال واریانس‌های به دست آمده را در فرمول Otsu جایگذاری می‌کنیم:

$$\sigma_w^2 = w_1\sigma_1 + w_2\sigma_2 = \frac{9}{25}\times 2.17 + \frac{16}{25}\times 6.75 = 0.7812 + 4.32 = 5.1012$$

مقدار Otsu برای حد آستانهٔ ۶ به دست آمد.

اعمال Otsu برای حد آستانهٔ ۱۰:

به این منظور، پیکسل‌هایی که مقدارشان از ۱۰ کمتر است در دستهٔ اول قرار می‌گیرند، و پیکسل‌هایی که مقدرشان بزرگتر یا مساوی ۱۰ است در دستهٔ دوم قرار می‌گیرند. طبق این آستانه، ماتریس رندوم تولید شده به شکل زیر دسته‌بندی می‌شود (پیکسل‌های آبی در دستهٔ اول و پیکسل‌های زرد در دستهٔ دوم قرار می‌گیرند)

$$matrix = \begin{bmatrix} 1 & 6 & 11 & 6 & 12 \\ 8 & 4 & 6 & 13 & 13 \\ 1 & 8 & 7 & 3 & 8 \\ 1 & 6 & 6 & 10 & 5 \\ 4 & 11 & 2 & 4 & 6 \end{bmatrix}$$

تعداد پیکسل‌های دستهٔ اول (آبی) برابر است با ۱۹ و تعداد پیکسل‌های دستهٔ دوم (زرد) برابر است با ۶.

- حال میانگین و واریانس دستهٔ اول را حساب می‌کنیم:

$$\mu_1 = \frac{3 \times 1 + 2 + 3 + 3 \times 4 + 5 + 6 \times 6 + 7 + 3 \times 8}{19} = \frac{92}{19} \approx 4.84$$

$$\sigma_1 = \frac{3 \times (1-4.84)^2 + (2-4.84)^2 + (3-4.84)^2 + 3 \times (4-4.84)^2 + (5-4.84)^2 + 6 \times (6-4.84)^2 + (7-4.84)^2 + 3 \times (8-4.84)^2}{19} \approx 5.29$$

- حال میانگین و واریانس دستهٔ دوم را حساب می‌کنیم:

$$\mu_2 = \frac{10 + 2 \times 11 + 12 + 2 \times 13}{6} = \frac{70}{6} \approx 11.67$$

$$\sigma_2 = \frac{(10-11.67)^2 + 2 \times (11-11.67)^2 + (12-11.67)^2 + 2 \times (13-11.67)}{6} \approx 1.22$$

- حال واریانس‌های به دست آمده را در فرمول Otsu جایگذاری می‌کنیم:

$$\sigma_w^2 = w_1\sigma_1 + w_2\sigma_2 = \frac{19}{25} \times 5.29 + \frac{6}{25} \times 1.22 = 4.0204 + 0.2928 = 4.0204$$

مقدار Otsu برای حد آستانهٔ ۱۰ به دست آمد.

می‌دانیم هر چه مقدار به دست آمده برای Otsu کوچکتر باشد، یعنی دسته‌بندی‌های ما بهتر و یکنواخت‌تر هستند. طبق مقایسه برای حد آستانهٔ ۶ و ۱۰ متوجه می‌شویم مقدار آن برای حد آستانهٔ ۱۰ کمتر است، پس حد آستانهٔ ۱۰ برای این ماتریس مناسب‌تر است.

سوال چهارم)

مقدار c برابر است با عدد ثابتی که از میانگین یا میانگین وزن‌دار کم می‌شود.

Block size برابر است با سایز همسایگی یک پیکسل که برای محاسبهٔ threshold وفقی در نظر گرفته می‌شود.

هر چه مقدار c کوچک‌تر باشد، یعنی آستانه‌گذاری دقیق‌تر است، چون اختلاف از میانگین جزئی‌تر است و یک اختلاف رنگ جزئی سبب تغییر دسته‌بندی پیکسل می‌شود.

Threshold: در تصویر q4_5 به دلیل سیاه بودن پس‌زمینهٔ تصویر و سفید بودن نوشته‌ها متوجه می‌شویم که از threshold = cv2.THRESH_BINARY_INV استفاده شده است، چون به دلیل روشن‌تر بودن پس‌زمینهٔ نوشته بعد از آستانه‌گذاری معمولی مقدار ۲۵۵ گرفته و باید سفید باشد، ولی در این تصویر بر عکس بوده است. تصاویر q4_1، q4_2، q4_3 و q4_4 از cv2.THRESH_BINARY استفاده کرده اند.

Block size: در تصاویر q4_1 و q4_3 و q4_5 می‌بینیم که نوشته‌های گوشهٔ پایین صفحه واضح نیستند و این یعنی آستانه‌گذاری وفقی در block size درستی صورت نگرفته و به حالت سراسری نزدیک‌تر است، پس از block size بزرگتر یعنی ۴۱ استفاده کرده اند. اما تصویر q4_2 و q4_4 به دلیل استفاده از block size کوچکتر یعنی ۲۱ باعث شده است که نوشته‌ها در گوشهٔ تصویر که احتمالا سایه داشته ایم، به درستی انجام شود.

Constant: مقدار c کوچکتر سبب آستانه‌گذاری دقیق‌تری می‌شود که در تصویر q4_1 این وضوح وجود دارد و نویز کمتری داریم، پس از c=5 استفاده شده است. در q4_3 هم نسبت به بقیه بهتر است پس c = 5 است. ولی در q4_2 و q4_4 و q4_5 خطوط مورب یعنی آستانه‌گذاری دقیق انجام نشده است و برای همین از c=30 استفاده شده است.

سوال پنجم)

ماتریس ورودی بعد از padding = reflect برابر ماتریس زیر می‌شود.

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

عملگر گسترش برای تصویر خاکستری به شکل زیر است:

$$dst(x,y) = \max_{(x',y') \in SE} src(x+x', y+y')$$

عملگر سایش برای تصویر خاکستری به شکل زیر است:

$$dst(x,y) = \min_{(x',y') \in SE} src(x+x', y+y')$$

در این روش، یک پنجره به اندازهٔ SE دور هر پیکسل در نظر می‌گیریم و برای گسترش بین مقادیری که در مکان‌هایی قرار دارند که در عنصر ساختاری برابر ۱ هستند، مقدار ماکسیمم را جایگزین پیکسل مورد بررسی

قرار می‌دهیم و برای سایش بین مقادیری که در مکان‌هایی قرار دارند که در عنصر ساختاری برابر ۱ هستند، مقدار مینیمم را جایگزین پیکسل مورد بررسی می‌کنیم.

نتیجهٔ اعمال عملگر گسترش:

عنصر ساختاری مورد بررسی برابر ماتریس رو به رو است:

1	1	1
1	0	0
1	0	0

یعنی در هر پنجرهٔ مورد بررسی، بین مکان‌های ۱ آن باید ماکسیمم بگیریم.

مثلا اعمال این عنصر ساختاری روی ماتریس A، به صورت زیر است:

ماتریس A

10	20	30
15	25	35
20	30	40

ماتریس A با در نظر گرفتن SE

10	20	30
15	25	35
20	30	40

پیکسل خروجی برای این پنجره برابر ۳۰ می‌باشد (max)

نتیجهٔ نهایی پس از اعمال گسترش (۸ در ۸):

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

نتیجهٔ اعمال عملگر سایش:

عنصر ساختاری مورد بررسی برابر ماتریس رو به رو است:

1	1	1
1	0	0
1	0	0

یعنی در هر پنجرهٔ مورد بررسی، بین مکان‌های ۱ آن باید مینیمم بگیریم.

مثلا اعمال این عنصر ساختاری روی ماتریس A، به صورت زیر است:

ماتریس A

10	20	30
15	25	35
20	30	40

ماتریس A با در نظر گرفتن SE

10	20	30
15	25	35
20	30	40

پیکسل خروجی برای این پنجره برابر ۱۰ می‌باشد (min)

نتیجهٔ نهایی پس از اعمال سایش (۸ در ۸):

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	22	22	22
22	22	22	22	22	22	22	22
33	33	33	33	22	22	22	22

سوال ششم)

این عملگر در واقع همان عملگر hit-or-miss است که عنصر B_1 در واقع عناصر hit را نشان می‌دهد (مکان‌های ۱ آن یعنی باید در پنجرهٔ مورد بررسی ۱ باشند) و عنصر B_2 عملگر miss را نشان می‌دهد (یعنی مکان‌های ۱ آن در پنجرهٔ مورد بررسی باید صفر باشند).

$$(A \circledast B) = (A \ominus X) \cap (A^c \ominus (W - X))$$
$$(A \circledast B) = (A \ominus B_1) \cap (A^c \ominus B_2)$$

با ترکیب این دو عنصر ساختاری، عنصر ساختاری B به دست می‌آید که به شکل زیر است (مکان‌های ۱ یعنی باید ۱ باشند، مکان‌های ۱- یعنی باید صفر باشند و مکان‌های صفر یعنی فرقی نمی‌کند صفر باشند یا ۱):

0	-1	-1
1	1	-1
0	1	0

باید ماتریس را با تصویر بالا hit-or-miss کنیم. که یعنی در مکان‌هایی که دقیقا این الگو وجود دارد، پیکسل وسط برابر ۱ می‌شود.

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

نتیجه:

فقط دو پیکسل زرد دقیقا این الگو را در یک پنجرهٔ ۳ در ۳ اطرافشان دارند. این پیکسل‌ها در خروجی برابر ۱ هستند و بقیهٔ پیکسل‌ها در خروجی برابر صفر هستند بکه یعنی این عملگر گوشه‌های سمت چپ و بالای تصویر را می‌یابد.

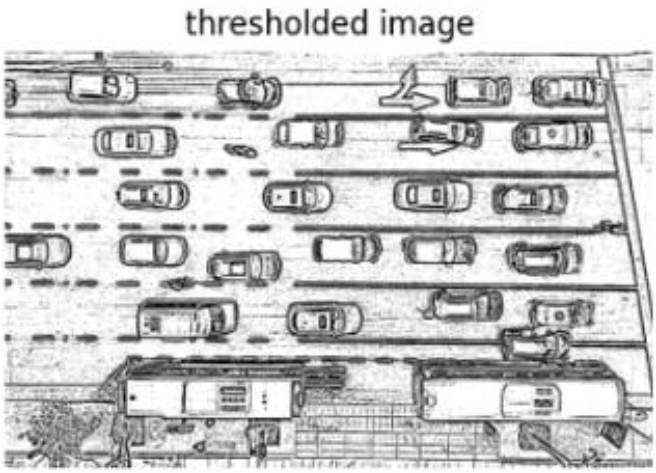
سوال هفتم)

الف)

ابتدا تصویر را خوانده و با cv2.cvtColor آن را به RGB تبدیل می‌کنیم. سپس adapriveThreshold را با threshold های باینری و gaussian اعمال می‌کنیم تا به یک تصویر صفر و یکی برسیم. در تصویر خروجی مشخص است که یک سری نویز داریم که اینها می‌توانند باعث خطا شوند. پس با عملگر مورفولوژی opening این نویزها را حذف می‌کنیم (یکی از کاربردهای opening حذف نویزها است). سپس از تابع `cv2.findContours()` برای یافتن لبه‌ها در تصویر استفاده می‌شود. سپس با استفاده از `cv2.RETR_TREE` به عنوان پارامتر انتخاب شده برای بازیابی ساختار شجره‌ای لبه‌ها، و `cv2.CHAIN_APPROX_SIMPLE` به عنوان روش تخمین برای فشرده‌سازی لبه‌ها، لیستی از لبه‌ها به همراه ساختار شجره‌ای آن‌ها به دست می‌آید. سپس، با گذاشتن لیست حاصل از لبه‌ها در متغیر `contours`، ابتدا تمامی لبه‌ها را به تعداد مشخص شده درون یک حلقه مورد بررسی قرار می‌دهیم. در هر

مرحله، اگر مساحت محصول فعلی بیشتر از ۵۰۰۰ واحد باشد (آزمایشی)، آن لبه به لیست `considerable_contours` اضافه می‌شود.

در نهایت، طول لیست `considerable_contours` به معنای تعداد خودروهای موجود در تصویر است و آن را چاپ می‌کنیم. مراحل و نتیجه:

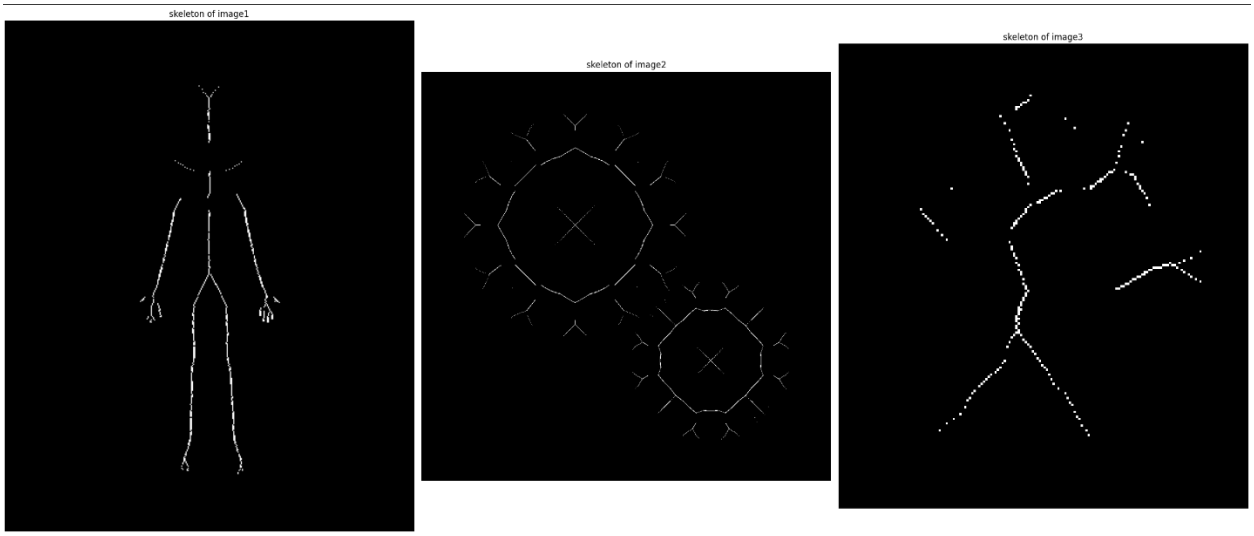



```
There are 23 cars in this image!
```

سوال هشتم)

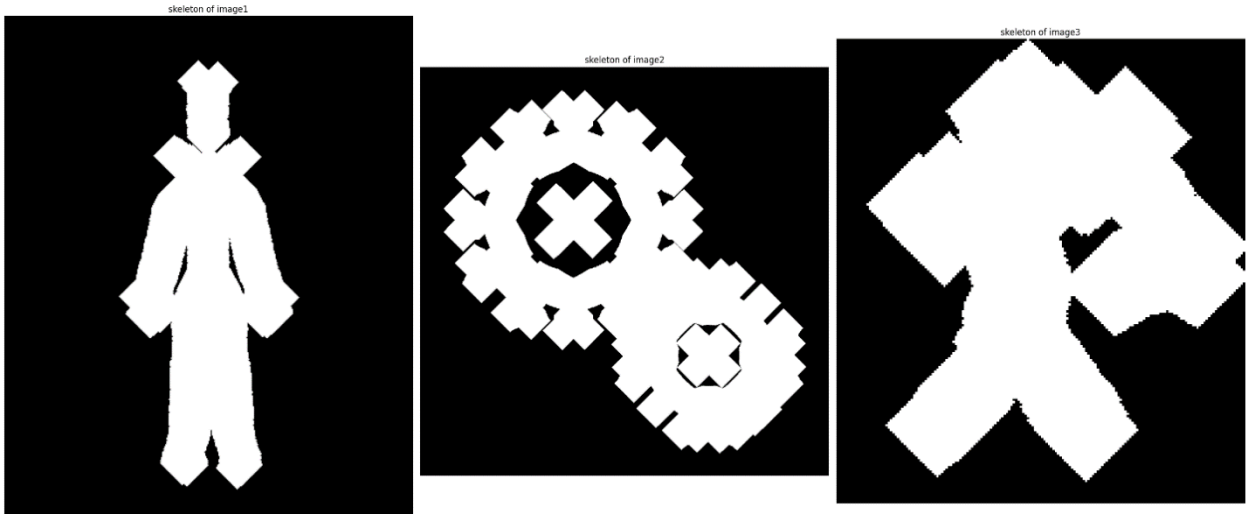
الف)

در تابع morphological_skeleton ابتدا یک متغیر به اسم count تعریف می‌شود که تعداد دفعات opening را ذخیره می‌کند. سپس تصویر را به صورت باینری در می‌آوریم. یک عنصر ساختاری برایش تعریف می‌کنیم که ۳ در ۳ و از نوع MORPH_CROSS است. طبق مراحل گرفته شده برای یافتن اسکلت تصویر، به ازای هر بار اجرای حلقه count را یکی زیاد می‌کنیم و یکبار روی تصویر سایش انجام می‌دهیم و سپس روی نتیجه گسترش می‌زنیم (opening) و تصویر open را از تصویر قبلی کم می‌کنیم و با اسکلت فعلی or می‌کنیم. سپس تصویر فعلی را برابر با تصویر سایش شده قرار داده و مقادیر پیکسل‌های غیر صفر آن را می‌شماریم. اگر صفر شده باشد یعنی اسکلت پیدا شده است. در غیر اینصورت دوباره مراحل را تکرار می‌کنیم.



ب)

برای reconstruct، به تعداد دفعاتی که opening انجام شده است روی اسکلت گسترش انجام می‌دهیم و نتیجه را برمی‌گردانیم. البته در یافتن اسکلت بخشی زیادی از اطلاعات تصویر از دست رفته است برای همین تصویر reconstructed دقیقا مثل تصویر اولیه نخواهد شد. نتایج:



سوال نهم)

عملگرهای مناسب برای تشخیص لبه به شکل‌های زیر هستند:

Element1

0	0	0
0	1	-1
0	0	0

Element2

0	-1	0
0	1	0
0	0	0

Element3

0	0	0
-1	1	0
0	0	0

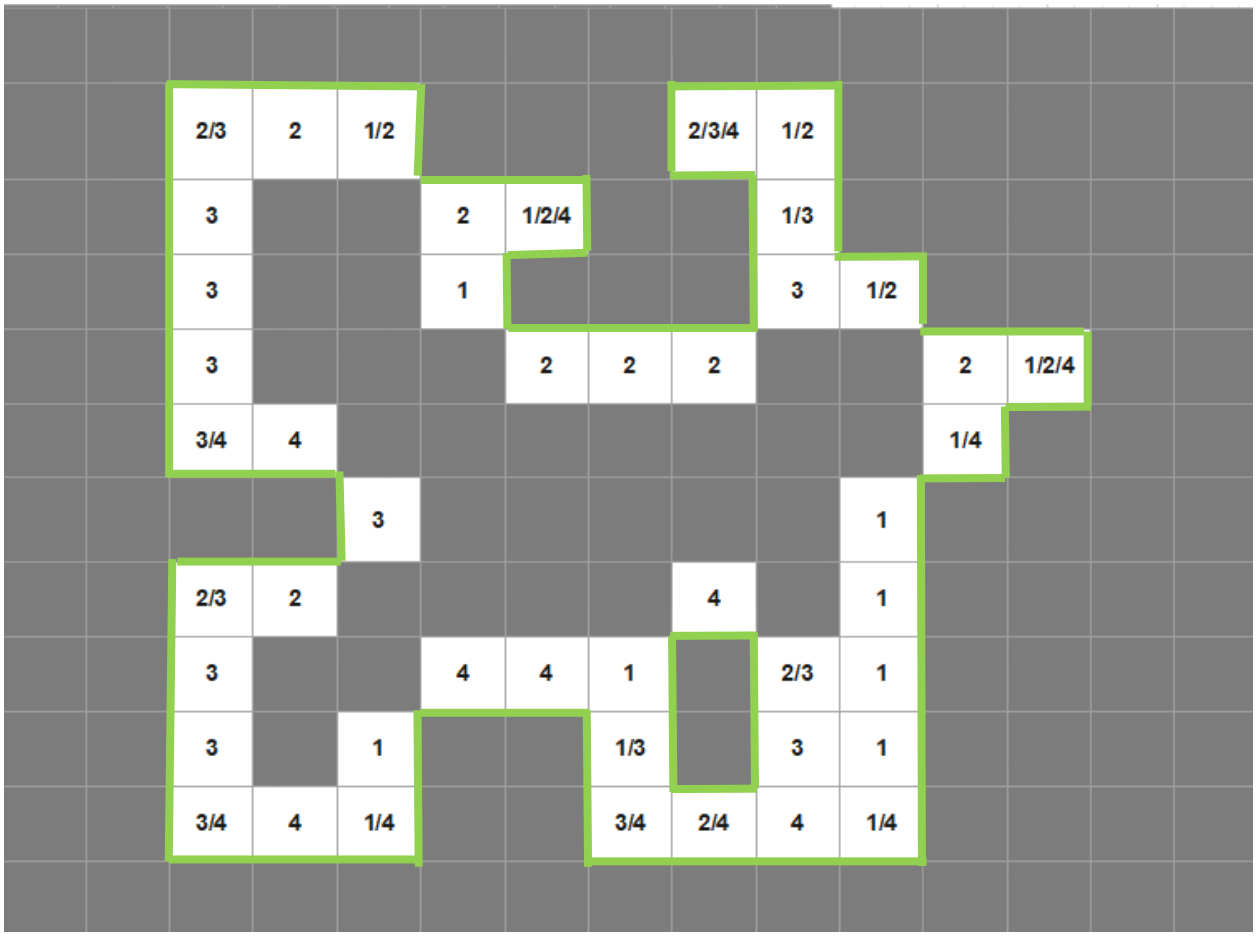
Element4

0	0	0
0	1	0
0	-1	0

در هر یک از این عناصر ساختاری، مکان‌های ۱ یعنی باید حتما ۱ دیده شود، مکان‌های ۱- یعنی باید حتما صفر دیده شود و مکان‌های صفر یعنی فرقی نمی‌کند صفر دیده شود یا ۱.

طبق الگوریتم hit-or-miss، در پیکسل‌هایی که در یک همسایگی ۳*۳ آن‌ها یکی از این عناصر ساختاری دقیقا دیده شود، آن پیکسل در خروجی برابر ۱ گذاشته می‌شود. اما اگر هیچ کدام از این عناصر دقیقا در آن نباشد، برابر صفر گذاشته می‌شود.

نتیجهٔ این عملیات با عناصر ساختاری نوشته شده برابر زیر خواهد شد که پیکسل‌های مرزی پیدا شده و به عنوان مرز در نظر گرفته می‌شوند:



منابع:

1. Hit-or-Miss
2. Skeleton