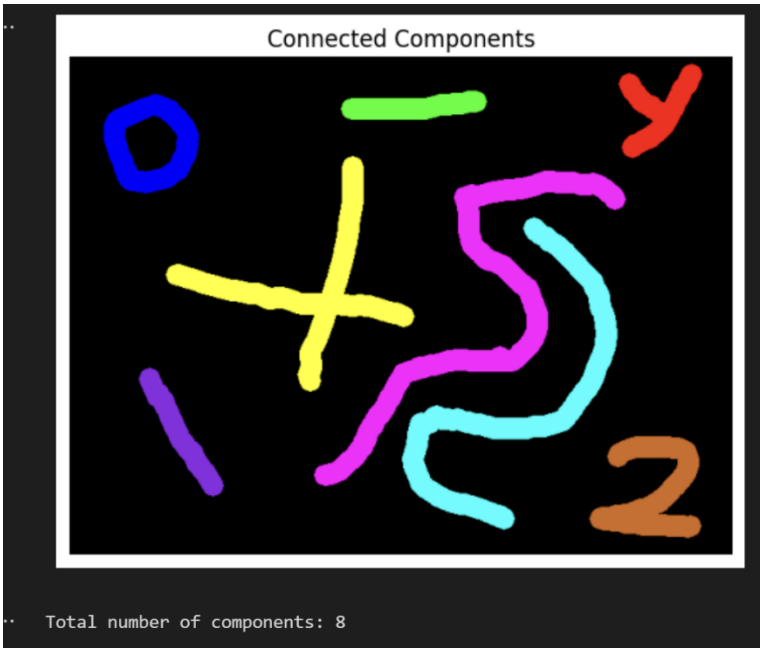
به نام خدا

تمرین پنجم – بینایی کامپیوتر **نیایش خانی ۹۹۵۲۱۲۳۵**

سوال ۱) ابتدا تصویر مورد نظر را به حالت خاکستری درآورده و سپس آن را با استفاده از یک آستانه (threshold) به تصویر باینری تبدیل می‌کنیم. سپس با استفاده از تابع **connectedComponentsWithStats**، اجزاء متصل در تصویر باینری را پیدا می‌کنیم. حال لیستی از رنگ‌های دلخواه را برای نمایش اجزاء متصل تعریف می‌کنیم و به هر جزء متصل یک رنگ اختصاص می دهیم. در آخر تصویر خروجی را نمایش می دهیم. خروجی :



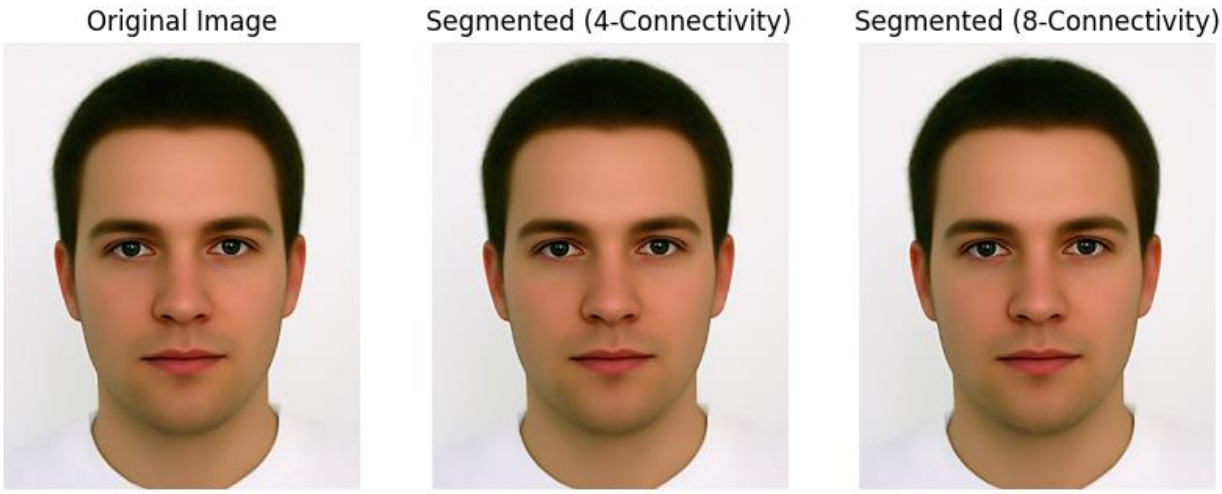
تعداد نواحی متصل ۸ بوده که هر کدام با یک رنگ مشخص شده اند.

سوال ۲) ابتدا، تصویر ورودی را می خوانیم و رنگ‌های تصویر را به RGB تغییر می‌دهیم (زیرا OpenCV تصویر را در فرمت BGR می‌خواند). سپس تابع Segment را به این صورت پیاده سازی می کنیم :

این تابع به عنوان ورودی تصویر، نقطه شروع، مقدار آستانه (threshold) و حالت همسایگی (که مشخص می‌کند که از چه نوع همسایگی استفاده شود، ۴ یا ۸) را می‌گیرد. این تابع با استفاده از مقدار آستانه و همسایگی مشخص، بخشی از تصویر را که با نقطه شروع مشخص شده مرتبط است، جدا می‌کند. برای این کار از یک ماسک مربعی استفاده می‌کند که ابعاد آن با ابعاد تصویر همخوانی دارد. سپس با استفاده از الگوریتم جستجوی اول سطح (BFS)، بخش مربوطه تشخیص داده می‌شود.

در نهایت، تصاویر اصلی و تصاویری که بر اساس تقسیم‌بندی بدست آمده‌اند، نمایش داده می‌شوند.

برای بدست اوردن نقطه seed در تشخیص چهره معمولا نقطه ای روی پیشانی یا چشم شخص انتخاب می شود. برای مثال در seed_point = (60, 170) ، خروجی به این صورت خواهد بود :



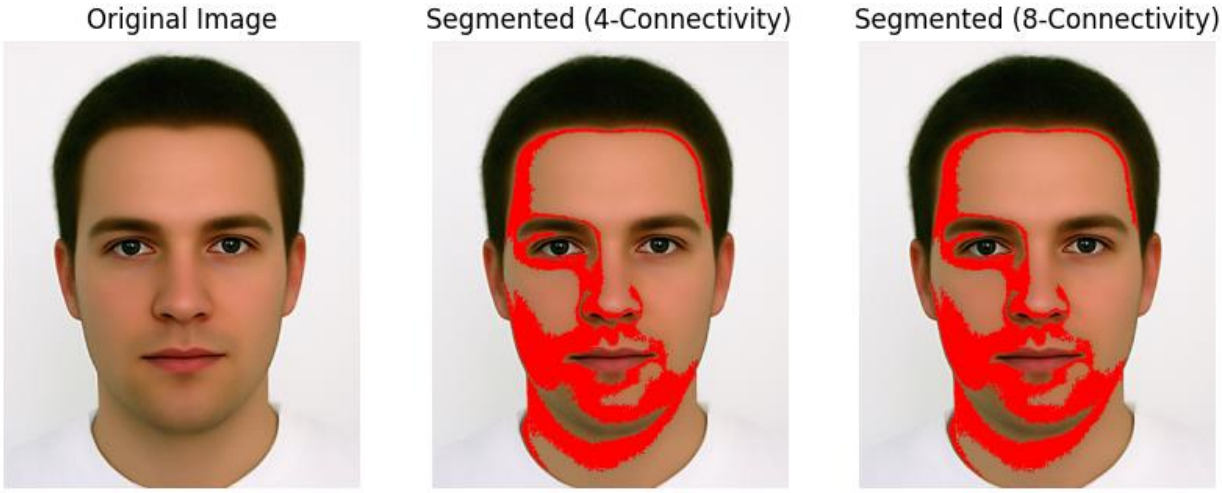
که مشاهده می شود که هیچ نقطه گسترشی نداشته ایم.

در حالت همسایگی ۴ هر پیکسل با ۴ پیکسل مجاور خود (بالا، پایین، چپ و راست) متصل است. این بدان معناست که در جستجوی سطح، تنها ۴ پیکسل مجاور یک پیکسل خاص بررسی می‌شوند. اما در حالت همسایگی ۸، هر پیکسل با ۸ پیکسل مجاور خود متصل است، شامل همه جهات: بالا، پایین، چپ، راست و همه چهار راستای مورب. برای تشخیص چهره، استفاده از حالت همسایگی ۸ معمولاً بهتر است، زیرا این حالت اجازه می‌دهد که بخش‌های متصل در همه جهات (شامل راستای مورب) تشخیص داده شوند، که می‌تواند باعث تشخیص دقیق‌تر و کامل‌تر چهره‌ها شود.

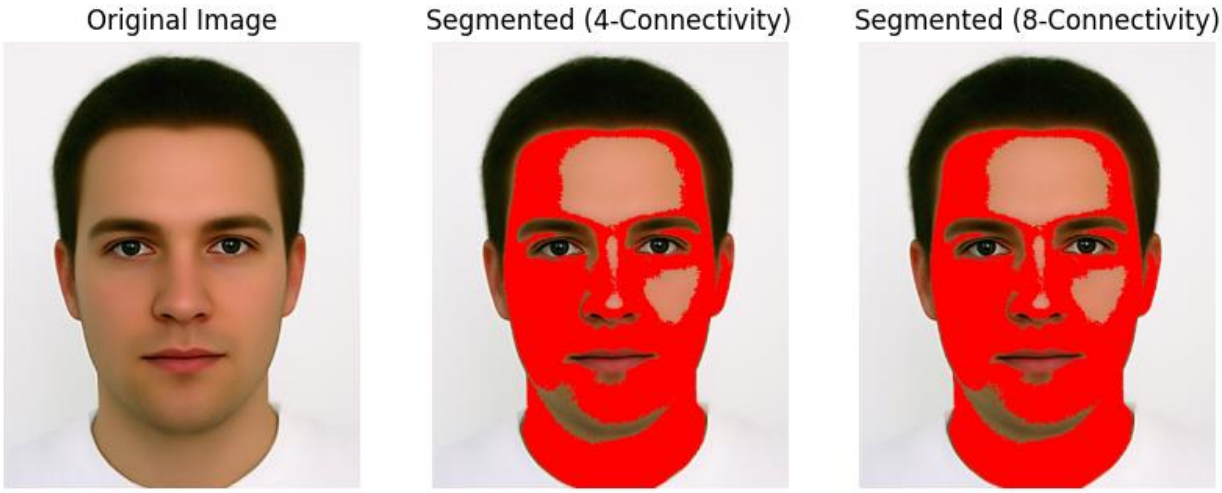
در تشخیص چهره، بخش‌های مختلفی از چهره مانند چشم، بینی، دهان و موهای اطراف نیاز به تشخیص دارند که ممکن است در جهات مختلف قرار گرفته باشند. حالت همسایگی ۸ این امکان را فراهم می‌کند که این بخش‌ها را در تمام جهات ممکن تشخیص دهد.

برای بدست آوردن مقدار threshold باید آزمون و خطا کنیم تا به عدد خوبی برسیم.

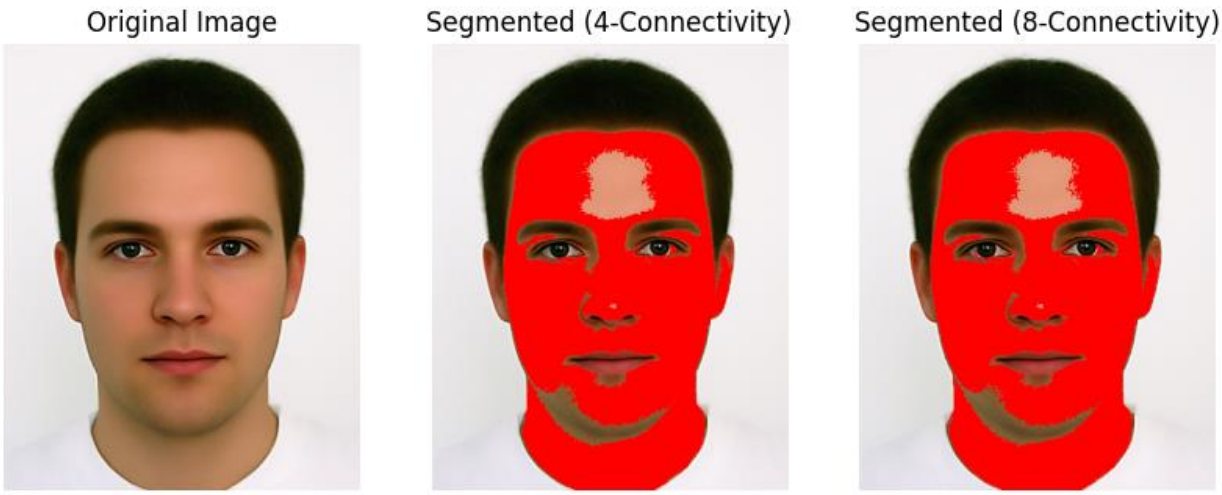
در حالت threshold = 60 به خروجی زیر میرسیم که تصویر مناسبی نیست و برای گسترش بیشتر می‌فهمیم که باید آن را افزایش دهیم.



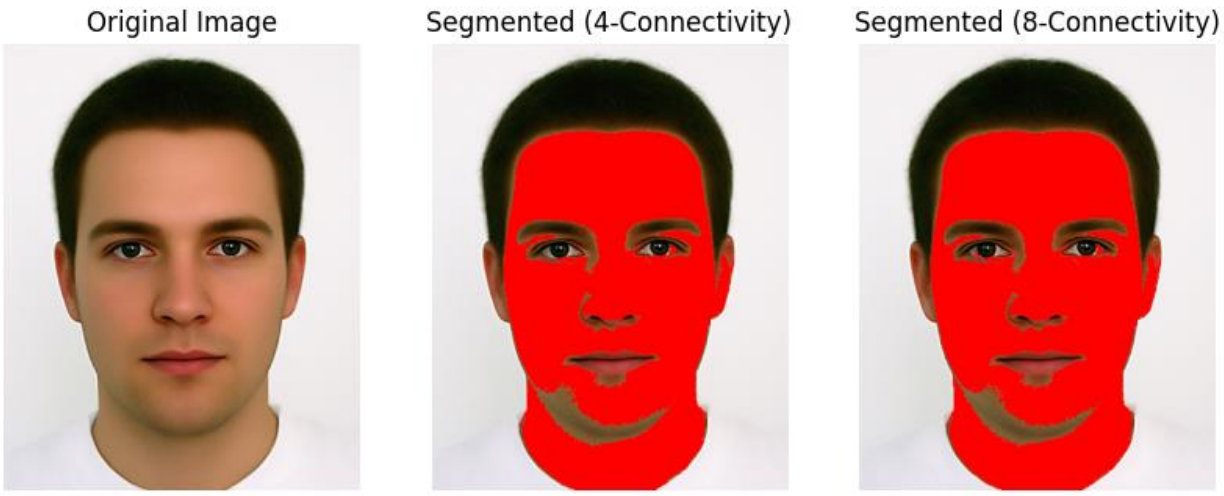
به ازای threshold = 100 به خروجی زیر می رسیم که نسبتا بهتر شده است اما هنوز قادر به تشخیص کل چهره نیستیم.



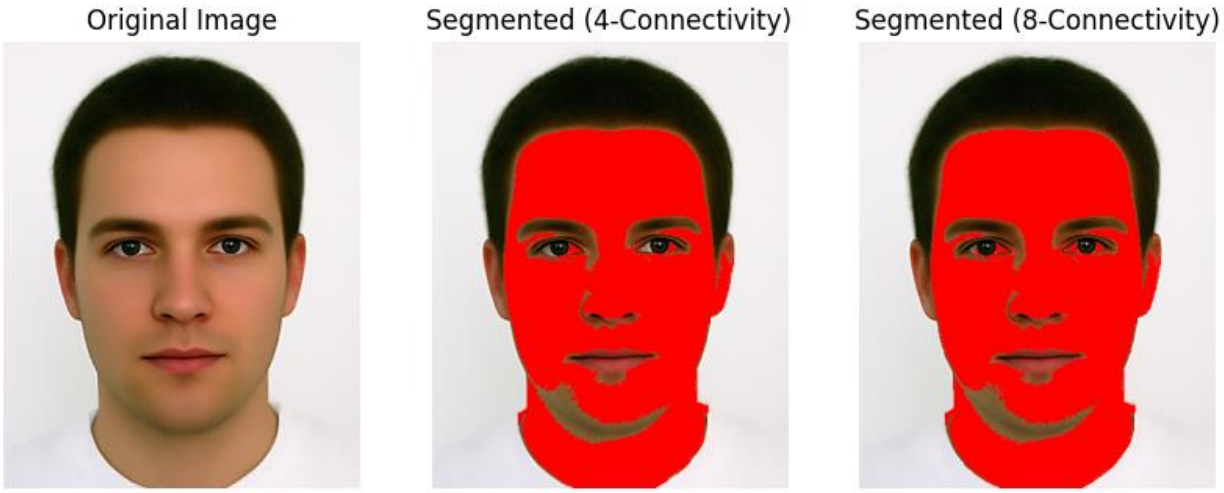
با افزایش آن به 130 به تصویر زیر خواهیم رسید که هنوز برای کامل شدن جا دارد.



با انتخاب threshold = 150 به خروجی تقریبا دلخواهی می رسیم.



البته ما قصد داریم که نقاطی مانند داخل چشم، ابروها، لب و ... شامل تشخیص ما نشوند بنابراین از افزایش threshold خودداری می کنیم. به عنوان مثال برای threshold = 200 خواهیم داشت :

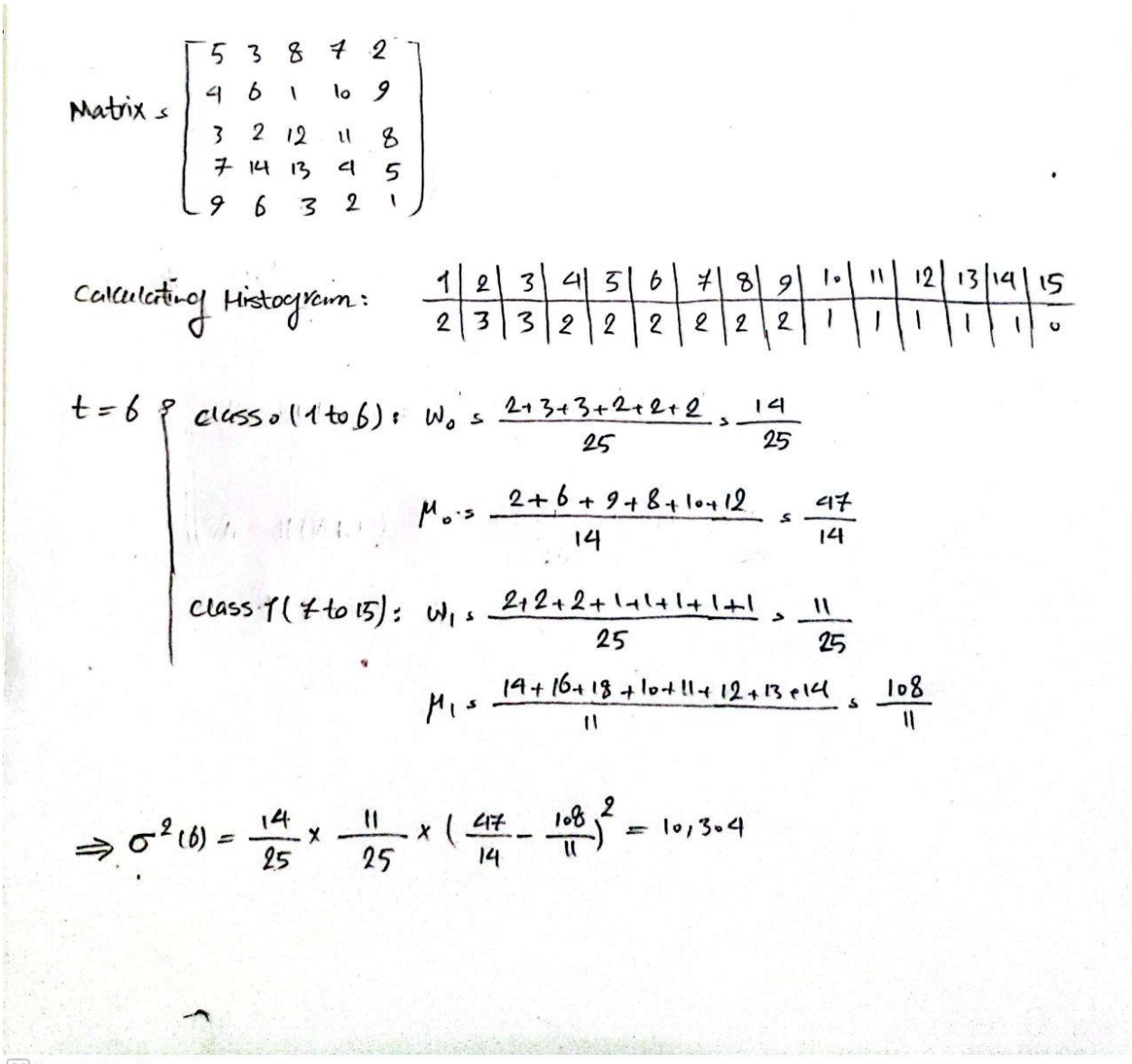


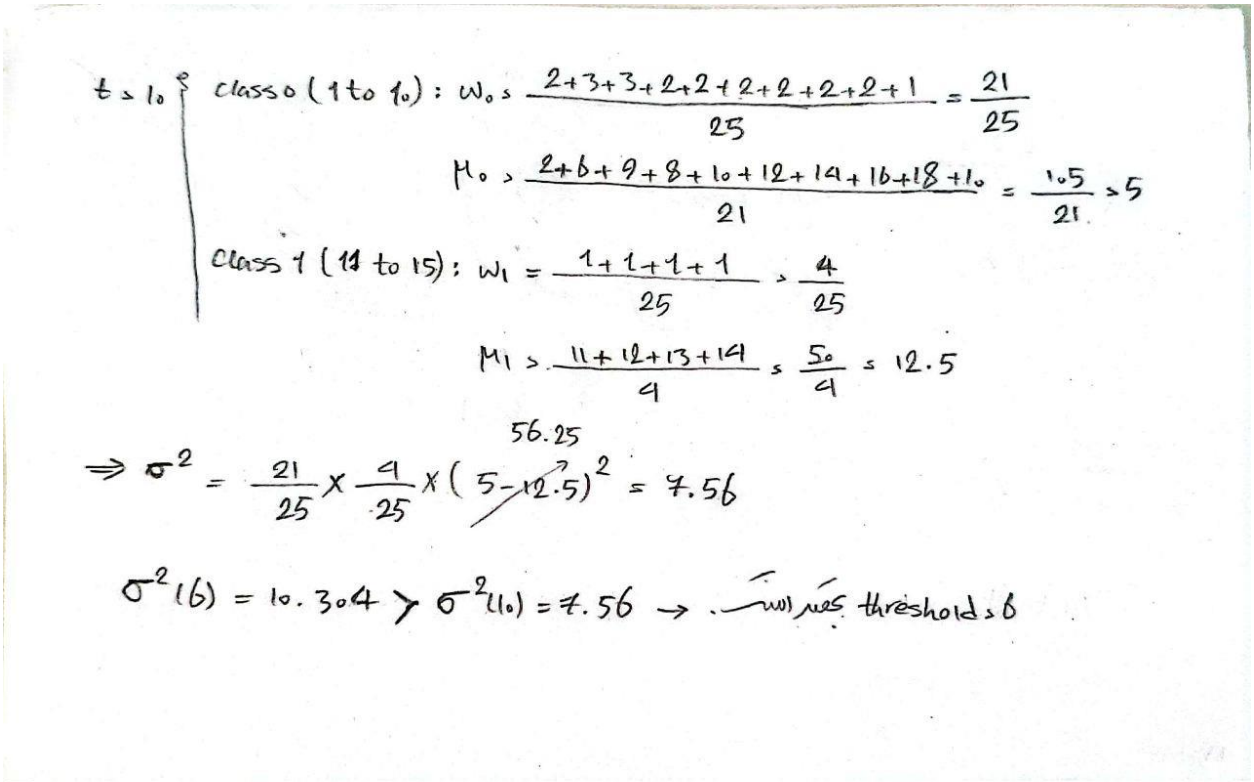
که نقاطی مانند نواحی داخل چشم نیز تشخیص داده شده اند که خروجی دلخواهی نیست.

سوال ۳) برای اجرای الگوریتم Otsu، ابتدا یک ماتریس 5*5 در نظر گرفته و هیستوگرام آن را محاسبه می‌کنیم.

سپس برای هر یک از حد آستانه ها دو کلاس در نظر می‌گیریم که یکی کمتر از حد آستانه و دیگری بیشتر از آن است.

برای هر یک از این کلاس ها احتمال w را محاسبه می کنیم به صورتی که مجموع تعداد پیکسل ها را بر تعداد کل پیکسل ها تقسیم می‌کنیم. سپس میانگین را نیز محاسبه کرده و در آخر واریانس آن کلاس را بدست می آوریم.





حد آستانه ای که واریانس بیشتری داشته باشد، بهتر است بنابراین threshold = 6 انتخاب بهتری است.

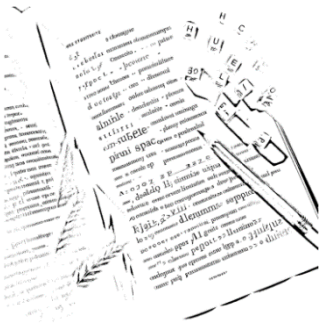
سوال ۴) در الگوریتم آستانه‌گذاری وفقی (Adaptive Thresholding)، پارامترهای مختلفی وجود دارند که نقش مهمی در عملکرد و نتیجه نهایی دارند. این پارامترها شامل blocksize ، c و thresholdType هستند.

blockSize اندازه ناحیه (بلاک) محلی که در آن آستانه محاسبه می‌شود. که اگر مقدار آن کوچک باشد، تصویر دقیق‌تر آستانه‌گذاری می‌شود، اما ممکن است نویزهای محلی بیشتری ظاهر شوند و جزئیات ریزتر به حساب بیایند و اگر بزرگ باشد، تصویر نرم‌تر و یکنواخت‌تر آستانه‌گذاری می‌شود، اما ممکن است جزئیات مهم نادیده گرفته شوند و نواحی بزرگ‌تری تحت تاثیر قرار گیرند. هرچه این مقدار بیشتر باشد، به آستانه گذاری سراسری نزدیکتر می شود.

پارامتر بعدی c مقدار ثابتی که از مقدار میانگین محاسبه‌شده آستانه کم می‌شود. این پارامتر کنترل می‌کند که آستانه‌گذاری چقدر سخت یا نرم باشد.اگر بزرگ باشد، آستانه‌گذاری سخت‌تر می‌شود، و پیکسل‌های بیشتری به مقدار ۰ تبدیل می‌شوند و در نتیجه تصویر تیره‌تر می‌شود. از طرفی با کوچکتر شدن، آستانه‌گذاری نرم‌تر می‌شود و پیکسل‌های بیشتری به مقدار ۲۵۵ تبدیل می‌شوند بنابراین تصویر روشن‌تر می‌شود.

thresholdType :

- `cv2.THRESH_BINARY` : اگر مقدار پیکسل بیشتر از آستانه باشد، پیکسل ۲۵۵ می‌شود، وگرنه ۰ بنابراین نواحی روشن تصویر به سفید و نواحی تیره به سیاه تبدیل می‌شوند.
- `cv2.THRESH_BINARY_INV`: برعکس حالت قبلی است، اگر مقدار پیکسل بیشتر از آستانه باشد، پیکسل ۰ می‌شود، وگرنه ۲۵۵ بنابراین نواحی روشن تصویر به سیاه و نواحی تیره به سفید تبدیل می‌شوند.



۱) همانطور که مشاهده می‌شود، قسمت هایی از تصویر واضح نیست و جزئیات تصویر مشخص نیستند بنابراین می توان نتیجه گرفت که از blocksize بزرگی استفاده کردیم (۴۱). همچنین به دلیل روشن بودن تصویر می توان گفت که c مقدار کوچکتری (۵) داشته است که از مقدار محاسبه شده آستانه کم می شود و در نتیجه تصویر روشن تر شده است. از طرفی از THRESH_BINARY استفاده شده است زیرا نواحی روشن تصویر به سفید و نواحی تیره به سیاه شده‌اند.

۲)تصویر به خوبی باینری شده است و متن موجود در تصویر به خوبی قابل مشاهده است بنابراین می توان گفت که از blocksize کوچکتری استفاده شده است (۲۱) . همچنین می بینیم که قسمت های سیاه تصویر بیشتر شده است حتی خطوط موربی روی تصویر ظاهر شده‌اند بنابراین c بزرگ (۳۰) است و پیکسل‌های بیشتری به مقدار ۰ تبدیل شده‌اند. همچنین به دلیل قبل، از THRESH_BINARY استفاده شده است.

۳)مانند تصویر اول، جزئیات تصویر به خوبی مشخص نیست و متن ناواضح است بنابراین blocksize بزرگی به کار رفته است (۴۱). از طرفی پیکسل‌های نسبتا کمتری نسبت به تصویر قبل به مقدار ۰ تبدیل شده‌اند پس می توان گفت از c نسبتا کوچکتری استفاده شده است اما این c از تصویر اول بزرگتر است. در کل می توان گفت c مقدار بزرگ (۳۰) دارد. از THRESH_BINARY نیز استفاده شده است.

۴) در این تصویر جزئیات تصویر به خوبی قابل مشاهده است بنابراین blocksize کوچکی (۲۱) استفاده شده است. از طرفی در مقایسه با تصویر اول، همچنان خطوط موربی دیده می شوند که نباید در تصویر واضح باشند. در این صورت می توان گفت از c کوچکی (۵) استفاده شده است. مانند قبل از THRESH_BINARY استفاده شده است.

۵) در این تصویر بر خلاف تصاویر قبل از THRESH_BINARY_INV استفاده شده است زیرا نواحی روشن تصویر به سیاه و نواحی تیره به سفید تبدیل شده اند.

از طرفی به دلیل واضح نبودن جزئیات می توان گفت از blocksize بزرگی استفاده شده است و همچنین پیکسل‌های نسبتا کمتری ۰ تبدیل شده‌اند (قبل از THRESH_BINARY_INV) پس می توان گفت از c کوچکتری (۵) استفاده شده است.

سوال ۵) برای اعمال عملگر سایش باید عنصر ساختاری را بر روی هر پیکسل اعمال کنیم بنابراین نیاز داریم که ازreflect padding استفاده کنیم.

۲۲	۲۲	۲۲	۲۲	۳۳	۲۲	۲۲	۳۳	۲۲	۲۲
۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲
۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲
۲۲	۲۲	۲۲	۲۲	۲۲	۳۳	۲۲	۲۲	۲۲	۲۲
۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲
۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲
۳۳	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲
۳۳	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲	۲۲
۳۳	۳۳	۳۳	۳۳	۳۳	۲۲	۲۲	۲۲	۲۲	۴۴
۳۳	۳۳	۳۳	۴۴	۳۳	۲۲	۴۴	۲۲	۴۴	۴۴

با اعمال عنصر ساختاری، آن پنجره از پیکسل ها در ۰ و ۱ ضرب می شوند و از میان مقادیر باقیمانده، مقدار مینیمم به جای پیکسل وسط قرار می گیرد. بنابراین خواهیم داشت :

۲۲	۲۲	۲۲	۲۲	۳۳	۲۲	۲۲	۳۳	۲۲	۲۲
۲۲	۲۲	۲۲	۲۲	۳۳	۲۲	۲۲	۳۳	۲۲	۲۲
۲۲	۲۲	۳۳	۳۳	۳۳	۳۳	۳۳	۳۳	۲۲	۲۲
۲۲	۲۲	۲۲	۲۲	۳۳	۲۲	۳۳	۴۴	۲۲	۲۲
۲۲	۲۲	۲۲	۳۳	۴۴	۲۲	۳۳	۲۲	۲۲	۲۲
۲۲	۲۲	۲۲	۴۴	۲۲	۲۲	۴۴	۳۳	۲۲	۲۲
۳۳	۳۳	۲۲	۴۴	۲۲	۴۴	۳۳	۳۳	۲۲	۲۲
۳۳	۳۳	۳۳	۳۳	۳۳	۳۳	۲۲	۳۳	۲۲	۲۲
۳۳	۳۳	۳۳	۴۴	۳۳	۲۲	۴۴	۲۲	۴۴	۴۴
۳۳	۳۳	۳۳	۴۴	۳۳	۲۲	۴۴	۲۲	۴۴	۴۴

در مرحله بعد برای اعمال عملگر گسترش باید ابتدا عنصر ساختاری را ۱۸۰ درجه چرخانده و سپس برای هر پیکسل آن را اعمال کنیم و بین مقادیر ماکسیمم بگیریم که خروجی به این صورت خواهد بود :

۲۲	۲۲	۲۲	۲۲	۳۳	۲۲	۲۲	۳۳	۲۲	۲۲
۲۲	۳۳	۳۳	۳۳	۳۳	۳۳	۳۳	۳۳	۳۳	۲۲
۲۲	۳۳	۳۳	۳۳	۳۳	۳۳	۴۴	۴۴	۴۴	۲۲
۲۲	۳۳	۳۳	۴۴	۴۴	۴۴	۴۴	۳۳	۲۲	۲۲
۲۲	۲۲	۴۴	۴۴	۴۴	۴۴	۴۴	۴۴	۳۳	۲۲
۲۲	۳۳	۴۴	۴۴	۴۴	۴۴	۴۴	۳۳	۳۳	۲۲
۳۳	۳۳	۴۴	۳۳	۴۴	۴۴	۳۳	۳۳	۳۳	۲۲
۳۳	۳۳	۴۴	۴۴	۴۴	۴۴	۴۴	۴۴	۴۴	۲۲
۳۳	۳۳	۴۴	۴۴	۴۴	۴۴	۴۴	۴۴	۴۴	۴۴
۳۳	۳۳	۳۳	۴۴	۳۳	۲۲	۴۴	۲۲	۴۴	۴۴

سوال ۶) برای سایش تصویر A با B1 ابتدا به آن zero padding می دهیم. (به دلیل بی تاثیر بودن در نتیجه، اینجا آورده نشده است.) نتیجه به صورت زیر خواهد بود :

$(A \ominus B_1)$

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

Ac نیز با معکوس کردن A بدست می آید که در نهایت به ماتریس زیر میرسیم.

پیکسل هایی که با رنگ قرمز مشخص شده اند حاصل

سایش Ac با B2 و پیکسل های سبز حاصل سایش A با B1 هستند. اشتراک این دو، تنها دو پیکسل مشخص شده با رنگ زرد است. نتیجه نهایی این پردازش تصویری است که فقط شامل بخش‌هایی از تصویر اصلی است که هم پس از سایش تصویر اصلی با B1 و هم پس از سایش مکمل تصویر باB2 باقی مانده‌اند.

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

سوال ۸)

الف) این کد الگوریتم اسکلت‌سازی تصویر را با استفاده از روش Zhang-Suen اجرا می‌کند. الگوریتم Zhang-Suen از دو مرحله اصلی تشکیل شده است که به صورت مکرر اجرا می‌شوند تا تصویر به حالتی بهینه تبدیل شود.

تابع `get_neighborhood`: این تابع همسایه‌های هشتگانه هر پیکسل در تصویر را استخراج می‌کند.

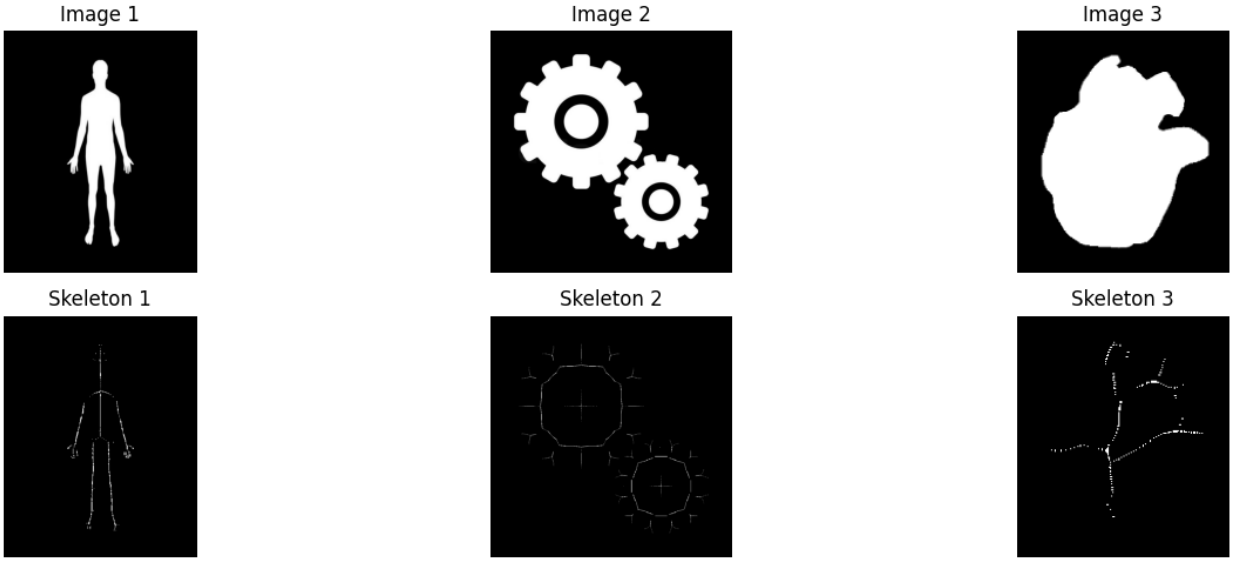
تابع `zhang_suen_thinning` : این تابع الگوریتم اسکلت‌سازی Zhang-Suen را اجرا می‌کند. در اینجا، برای هر پیکسل در تصویر، همسایه‌های هشتگانه آن بررسی می‌شوند و با استفاده از شرایط خاصی که الگوریتم نرمالیزه شده ارائه می‌دهد، تصمیم‌گیری می‌شود که آیا پیکسل مورد نظر حذف شود یا خیر.

تابع `thinning`: این تابع الگوریتم اسکلت‌سازی را به صورت مکرر اجرا می‌کند تا تصویر به حالتی بهینه تبدیل شود. در اینجا، تابع `zhang_suen_thinning` به صورت مکرر فراخوانی می‌شود تا تا زمانی که هیچ تغییری در تصویر حاصل نشود ادامه یابد.

توابع `erode` و `dilate`: این دو تابع به ترتیب فرایند کاهش حجم و گسترش بخش‌های سفید در تصویر را انجام می‌دهند. این فرایندها برای پیاده‌سازی بخش‌هایی از الگوریتم اسکلت‌سازی لازم است. (طبق اسلاید ها)

تابع `skeletoniztion`: این تابع تصاویر را به سکلت‌های آن‌ها تبدیل می‌کند. ابتدا تصویر به صورت خاکستری تبدیل شده و سپس به صورت باینری برای اجرای الگوریتم اسکلت‌سازی آماده می‌شود. سپس فرایند کاهش حجم و گسترش بخش‌های سفید اجرا می‌شود و سپس الگوریتم سکلت‌سازی Zhang-Suen اعمال می‌شود.

در نهایت، تصاویر اصلی و اسکلت‌های حاصل نمایش داده می‌شوند. خروجی :



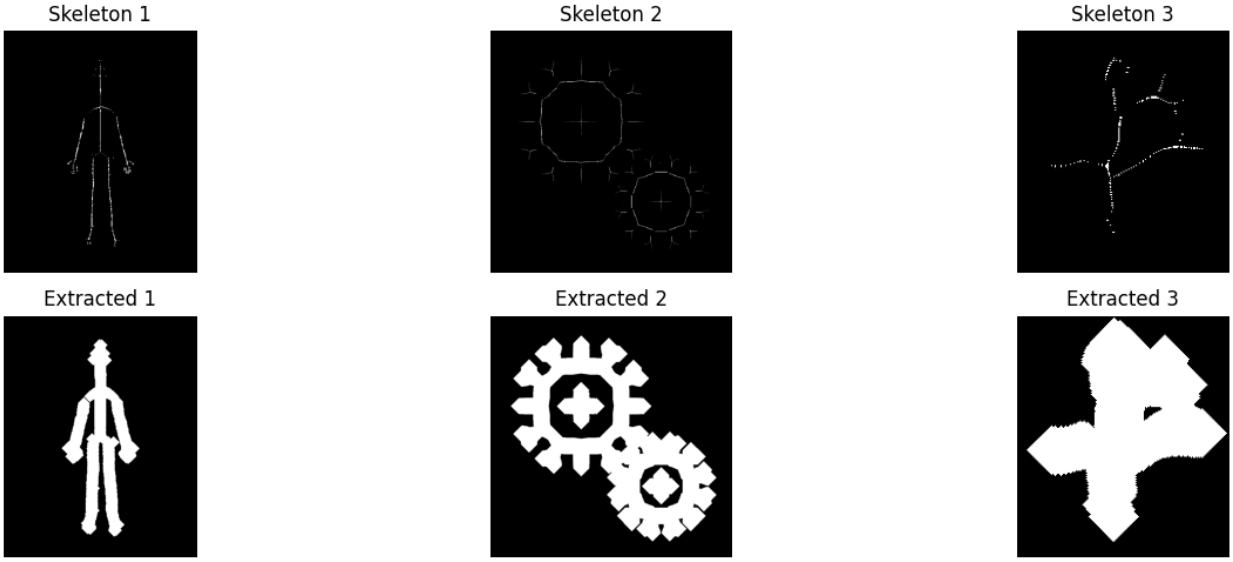
ب)

واضح است که برای بازیابی تصویر از اسکلت آن باید از عملگر گسترش (dilation) استفاده کنیم. فرایند گسترش به این معنا است که هر پیکسل سفید در سکلت به همراه پیکسل‌های همسایه‌اش گسترش می‌یابد و بزرگ‌تر می‌شود. این عملیات می‌تواند به منظور اتصال بخش‌های مجزا از اسکلت و افزایش ضخامت خطوط ، بخشی از تصویر را بازیابی کند.

iterations تعداد مراحل گسترش است که به ازای هر مرحله، عملیات گسترش بر روی تصویر اجرا می‌شود.

این گسترش با یک عنصر ساختاری مرکز متقارن (MORPH_CROSS) به ابعاد (۳، ۳) اعمال می شود.

در نهایت، تصویر گسترش یافته (extracted) به عنوان خروجی تابع برگردانده می‌شود و خروجی هر یک را می توان مشاهده کرد :

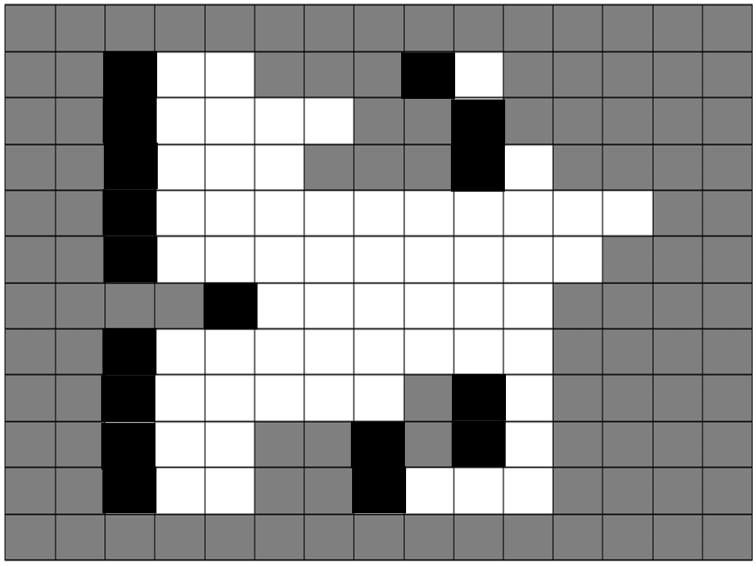


از آنجایی که ضخامت مورد نظر هر یک از تصاویر متفاوت است، مقدار iteration آنها نیز باید با هم متفاوت باشد. برای تصویر اول که خروجی مورد نظر ضخامت زیادی ندارد، iteration کمتر، برای تصویر دوم کمی بیشتر تا جایی که از فرم مورد دلخواه خارج نشود و برای تصویر سوم نیز تا جایی که ممکن است آن را افزایش می دهیم تا تقریبا شبیه شکل اولیه شود.

سوال ۹)

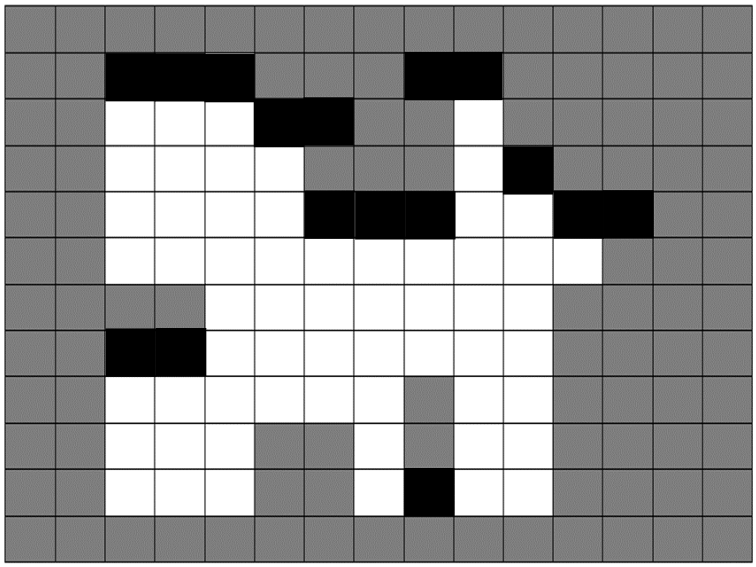
به دست آوردن مرز سمت چپ با اعمال عنصر زیر و نتیجه حاصل :

۰	۰	۰
۰	۱	-۱
۰	۰	۰



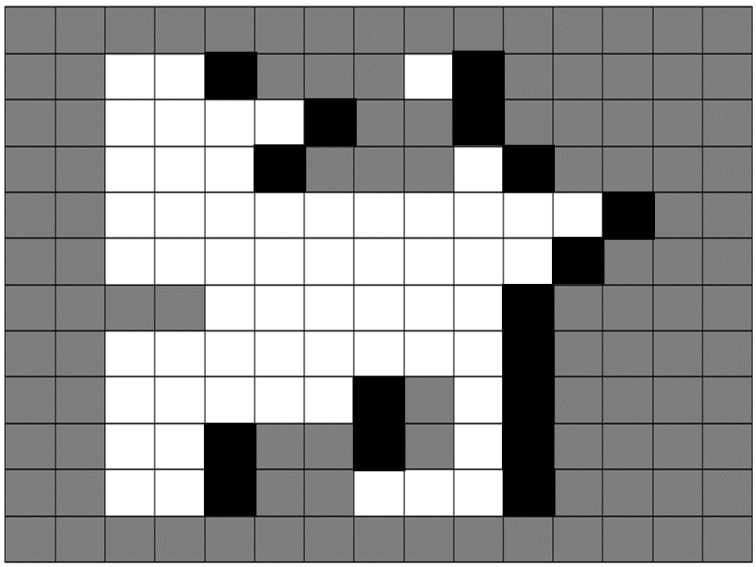
به دست آوردن مرز بالا با اعمال عنصر زیر و نتیجه حاصل :

۰	-۱	۰
۰	۱	۰
۰	۰	۰



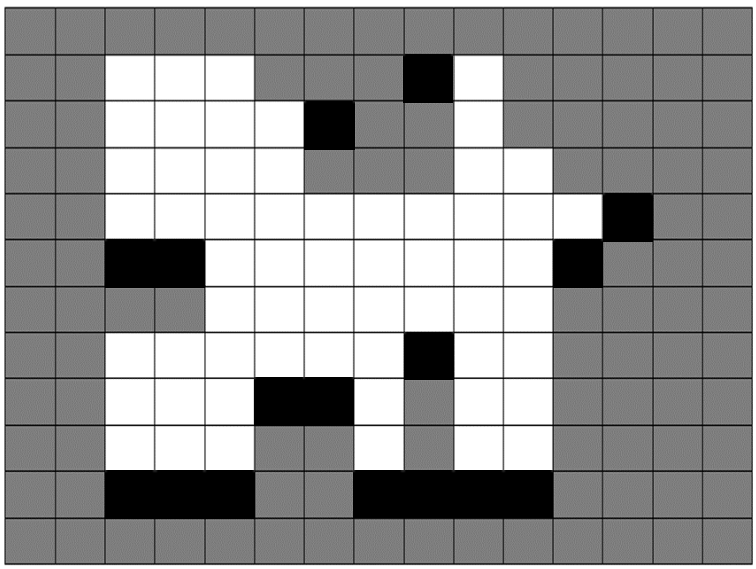
به دست آوردن مرز سمت راست با اعمال عنصر زیر و نتیجه حاصل :

۰	۰	۰
۰	۱	-۱
۰	۰	۰



به دست آوردن مرز پایین با اعمال عنصر زیر و نتیجه حاصل :

۰	۰	۰
۰	۱	۰
۰	-۱	۰



با اجتماع این عناصر، کل نقاط مرزی به صورت زیر حاصل می شود.