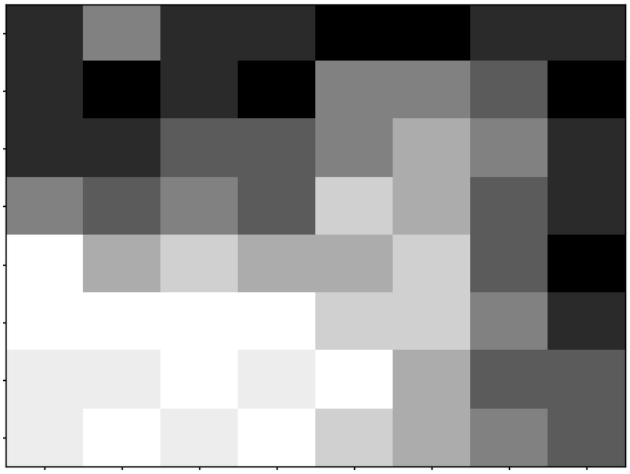
.1در هر مرحله، مربع را به 4 مربع کوچک‌تر تقسیم می‌کنیم و بررسی می‌کنیم که این آستانه در هرکدام از مربع‌های کوچک رعایت شده است یا نه. در صورت برقراری، پیکسل‌های در مربع merge می‌شوند و در غیر این صورت مربع دوباره به 4 قسمت تقسیم می‌شود و مراحل بالا تکرار می‌شود.

در هر مربع اگر شرط زیر برای همه پیکسل‌ها برقرار بود، آن پیکسل‌ها merge می‌شوند :

$$|m_1 - m_2| < T\,,$$

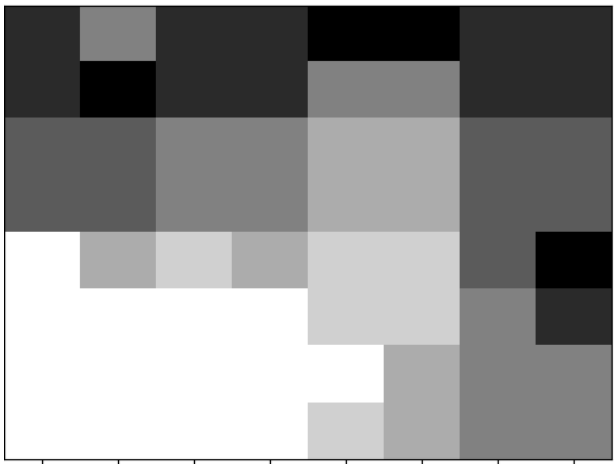
در اینجا بعد از merge کردن یک ناحیه، جای پیکسل‌های این مربع، میانگین از خانه را جای‌گذاری می‌کنیم:

شکل اولیه ما به‌صورت زیر است:



به‌صورت بازگشتی مسئله را حل می‌کنیم. در ابتدا مربع را به 4 بخش تقسیم می‌کنیم. در هر قسمت بررسی می‌کنیم که merge داریم و یا نه. اگر merge داشتیم سراغ قسمت بعدی میریم و در غیر این صورت برای همین قسمت دوباره فرایند را چک می‌کنیم تا جایی که به پیکسل برسیم و در این مرحله مستقیم پیکسل را بازگشت می‌دهیم:

تصویر نهایی به فرمت زیر است (کد این فرایند در پوشه Q1 قرار دارد):



حال سوال را با فرض متفاوتی برعکس بالا حل میکنیم :

6	4	6	6	7	7	6	6
6	7	6	7	4	4	5	7
6	6	5	5	3	2	4	6
4	5	4	5	2	3	5	6
0	3	2	4	3	2	5	7
0	0	0	0	2	2	4	6
1	1	0	1	0	3	5	5
1	0	1	0	1	3	4	5

ابتدا ناحیه بالا را به 4 قسمت تقسیم میکنیم :

6	4	6	6	7	7	6	6
6	7	6	7	4	4	5	7
6	6	5	5	3	2	4	6
4	5	4	5	2	3	5	6
0	3	2	4	3	2	5	7
0	0	0	0	2	2	4	6
1	1	0	1	0	3	5	5
1	0	1	0	1	3	4	5

حال بررسی میکنیم در هر ناحیه اختلاف ماکسیمم با مینمم کمتر مساوی با threshold است یا خیر. در صورت کمتر مساوی بودن ناحیه را تغییر نمیدهیم و اگر بیشتر بود، همان ناحیه را به 4 قسمت تقسیم میکنیم :

6	4	6	6	7	7	6	6
6	7	6	7	4	4	5	7
6	6	5	5	3	2	4	6
4	5	4	5	2	3	5	6
0	3	2	4	3	2	5	7
0	0	0	0	2	2	4	6
1	1	0	1	0	3	5	5
1	0	1	0	1	3	4	5

حال پس از split کرد ناحیه ها، باید مناطق را merge کنیم. در صورتی که اختلاف جفت max و min 2 ناحیه کنار هم کمتر مساوی 3 بود، ان 2 ناحیه با هم merge خواهند شد. و ناحیه های کلی به صورت زیر خواهد بود :

6	4	6	6	7	7	6	6
6	7	6	7	4	4	5	7
6	6	5	5	3	2	4	6
4	5	4	5	2	3	5	6
0	3	2	4	3	2	5	7
0	0	0	0	2	2	4	6
1	1	0	1	0	3	5	5
1	0	1	0	1	3	4	5

دو رویکرد متفاوت برای تقسیم‌بندی منطقه گرا وجود دارد:

- منطقه درحال‌رشد با تجمع پیکسل (**Region Growing**) :

رشد منطقه روشی است که پیکسل‌ها یا زیر منطقه‌ها را در مناطق بزرگ‌تر گروه‌بندی می‌کند.

روند تجمع پیکسل‌ها با مجموعه‌ای از نقاط بذر شروع می‌شود و از این ناحیه رشد می‌کند و برای هر نقطه دانه آن پیکسل‌های مجاور که نسبت مشابهی دارند اضافه می‌شود.

تصاویر تقسیم‌بندی شده بر اساس روش‌های رشد منطقه، اغلب شامل مناطق بسیار زیاد (کم رشد) یا مناطق بسیار کمی (زیاد رشد) در نتیجه تنظیم پارامتر غیربهینه هستند. بسیاری از پس پردازشگرها برای بهبود طبقه‌بندی توسعه داده شده‌اند. پس پردازشگرهای ساده، تعداد مناطق کوچک را در تصویر تقسیم شده کاهش می‌دهند. پس‌پردازش پیچیده‌تر ممکن است اطلاعات تقسیم‌بندی به‌دست‌آمده از رشد منطقه و تقسیم‌بندی مبتنی بر لبه را با هم ترکیب کند.

- تقسیم و ادغام منطقه (**Splitting & Merging**) :

در این روش یک تصویر ابتدا به مجموعه‌ای از ناحیه‌های دلخواه ازهم‌گسسته تقسیم می‌شود و سپس مناطق را ادغام و/یا تقسیم می‌کند.

اجازه دهید R کل منطقه تصویر را نشان دهد و سپس یک محمول P را انتخاب کنید.

برای تصویر، یکی از روش‌های تقسیم‌بندی R این است که آن را به‌صورت متوالی به ناحیه ربع کوچک‌تر و کوچک‌تر تقسیم کنیم، به‌طوری‌که برای هر منطقه Ri، Predicate(Ri) = True.

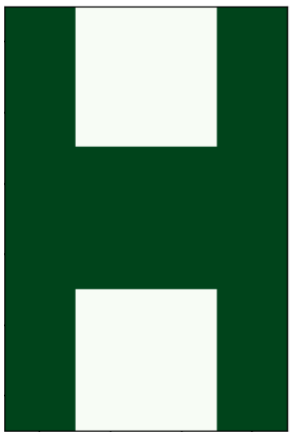
اگر Predicate(Region) = False تصویر را به ربع تقسیم کنید.

اگر Predicate(Region) = False برای هر ربع، آن ربع را به ربع فرعی و غیره تقسیم کنید.

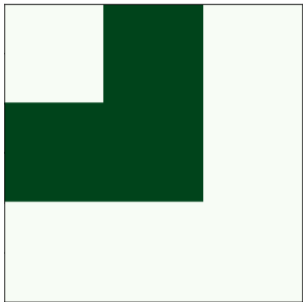
در حالت کلی رشد تصویر این‌گونه است که ما از یک پیکسل شروع می‌کنیم و پیکسل‌های همسان آن را انتخاب می‌کنیم و اگر پیکسلی داشتیم که حتی با مرز یک پیکسل هم به یک‌بخشی وصل شده بود هم می‌تواند جزو رشد ما باشد درحالی‌که در تقسیم و ادغام منطقه این‌طور نخواهد بود و ما در یک کرنل به‌خصوص پیکسل‌ها را بررسی می‌کنیم و ممکن است 2 پیکسل یکسان در کنار همچون در 2 کرنل متفاوت قرار دارند با هم رشد نکنند. البته از لحاظ سرعت بسیار سریع‌تر از تقسیم و ادغام خواهد بود؛ چون به‌طورکلی ما همه پیکسل‌ها را چک نمی‌کنیم.

2. عملگر باز برای حذف نویز و جزئیات کوچک در تصویر استفاده می‌شود.

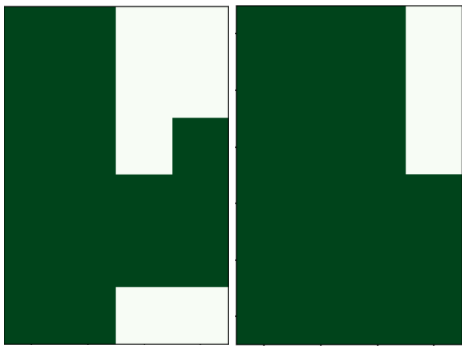
تصویر اولیه ما به‌صورت زیر است:



حال ما می‌خواهیم عملگر باز را بر روی تصویر بالا با کرنل زیر انجام دهیم (کرنل ما 3 در 3 خواهد بود):

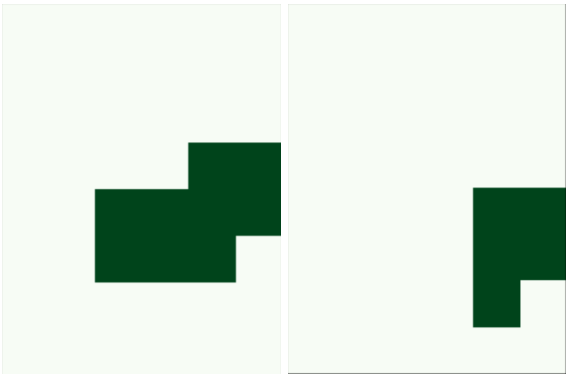


نتیجه استفاده از عملگر باز (تصویر سمت راست برای استفاده 2 بار از این عملگر و تصویر است چپ برای استفاده از یک‌بار از این عملگر است):

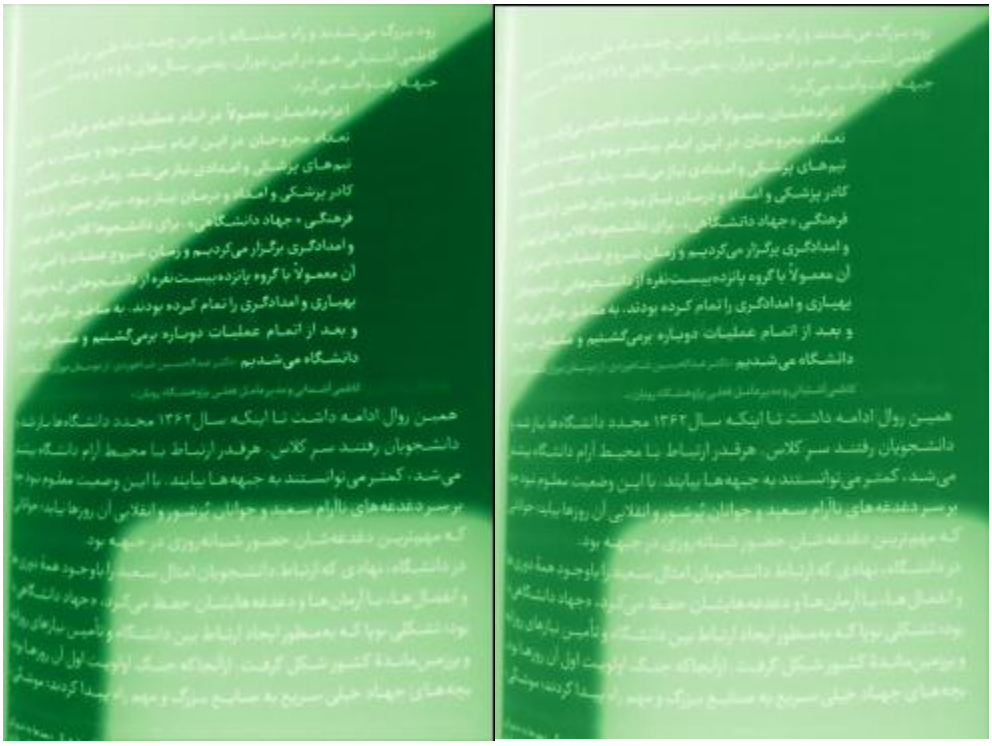


حال اگر عملیات بالا را همراه با padding به‌اندازه 1 واحد روی تصویر ورودی در همه جهت‌ها انجام دهیم نتیجه ما به‌صورت زیر خواهد شد:

نتیجه استفاده از عملگر باز همراه با padding (تصویر سمت راست برای استفاده 2 بار از این عملگر و تصویر است چپ برای استفاده از یک‌بار از این عملگر است):



در حالت کلی ما از این عملگر برای کم‌کردن نویز استفاده می‌کنیم. با افزایش میزان تکرارها، گسترش ما در حالت کلی بیشتر خواهد بود و انگار جزئیات اصلی تصویر bold تر می‌شوند.



به طور مثال در تصویر سمت چپ ما 10 بار این عملیات را انجام دادیم و نسبت به تصویر سمت راست که 2 بار عملگر باز رو استفاده کردیم عبارت‌های ما bold تر شده‌اند.

3. ما برای تعیین سطح استانه خاکستری، نیاز به الگوریتمی داشتیم که otsu و adaptive threshold برای این کار توسعه داده شده‌اند.

Otsu یک الگوریتم تعیین سطح خاکستری بر اساس مشخصات آماری است. به این‌گونه عمل می‌کند که ما هیستوگرام تصویر موردنظر را می‌کشیم و تک‌تک مقادیر ممکن را از بین 255 حالت به‌عنوان مرز انتخاب می‌کنیم. این مرزها هیستوگرام را به 2 بخش تقسیم می‌کنند و ما در تلاشیم برای هرکدام از این حالات، عبارت زیر را محاسبه کنیم و مینیمم مقدار را به‌عنوان پاسخ Otsu پیدا کنیم :

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

در اینجا wها نشان‌دهنده پیکسل هر کلاس و σ نشان‌دهنده واریانس پیکسل‌های این کلاس است.

الگوریتم Otsu در خیلی از حالت مانند تصاویری که میزان روشنایی متفاوتی به‌خاطر 7 نور محیط دارند نتایج مناسبی را درست نمی‌کند برای همین ما نیاز داریم که تصویر را ناحیه‌بندی کنیم و برای هر بخش Otsu بزنیم (که بازم نتیجه خوبی ندارد) و یا برای هر پیکسل یک سطح آستانه تعریف کنیم (adaptiveThreshold) که این الگوریتم نتیجه بهتری را برای ما فراهم می‌کند.

adaptiveThreshold به این صورت کار می‌کند که برای هر پیکسل، سطح آستانه را پیدا می‌کند و این کار را با کمک پیکسل‌ها همسایه خود انجام می‌دهد و به همین دلیل الگوریتم بسیار سنگینی خواهد بود؛ چون به‌ازای تک‌تک پیکسل‌های تصویر ما باید محاسباتی را انجام دهیم (مراحل Otsu)، بنا بر همین سعی می‌کنیم میانگین پیکسل‌های اطراف پیکسل موردنظر خود را به‌عنوان سطح آستانه معرفی کنیم که هزینه محاسبه کمتر شود.

adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)

src : منبع تصویر تک‌کاناله 8 بیتی
maxValue : مقدار غیر صفر اختصاص‌داده‌شده به پیکسل‌هایی که شرط برای آنها برآورده شده است.
adaptiveMethod : الگوریتم آستانه تطبیقی برای استفاده MEAN یا GAUSSIAN
thresholdType : نوع آستانه که باید THRESH_BINARY یا THRESH_BINARY_INV باشد
blockSize : اندازه همسایگی پیکسلی که برای محاسبه مقدار آستانه استفاده می‌شود
C : ثابت کسر شده از میانگین یا میانگین وزنی

4. هر دوی الگوریتم‌ها جواب‌های دقیق را محاسبه نکردند:

Otsu بخشی خیلی بزرگی از متن را به خاطر مشکلی که دارد از دست داده است ولی باقی جملات را نسبتاً درست پیدا کرده. adaptiveThreshold برعکس Otsu توانسته است کل تصویر را را درستی تبدیل کند و حجم متن چاپی بیشتر است ولی خیلی از جملات پیدا شده، ارزش ادبیاتی ندارند.