به نام خدا

نام: فاطمه زهرا بخشنده **استاد: دکتر محمدرضا محمدی**

شماره دانشجویی: 98522157 **درس مبانی بینایی کامپیوتر**

گزارش تمرین 8:

سوال سوم:

الف)

Otsu یک الگوریتم تعیین سطح مقدار آستانه بر حسب مشخصه های آماری است. خلاصه الگوریتم این است که سطح آستانه ای را انتخاب کنیم که واریانس بین پیکسل های هر کلاس کمینه شود. پیکسل هایی که پایین تر از Threshold باشند گروه اول و پیکسل های بالاتر از Threshold در گروه دوم قرار میگیرند.

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

از آن جایی کهاین تابع، تابع خوش تعریفی برای مشتق نیست، برای پیدا کردن مینیمم این تابع، Threshold را 255 مقدار مختلف گذاشته و بررسی می کنیم کدام مقدار بهتر است.

نقاط قوت otsu: سرعت بالایی دارد. مبتنی بر مشخصه های آماری تصویر است.

نقاط ضعف otsu: در این روش اگر تصویر ما دارای سایه یا نویزی باشد و یا کنتراست و شدت نور محیط نامناسب داشته باشد، چون ویژگی های محلی را در نظر نمی گیریم و برای کل تصویر یک Threshold در نظر می گیریم، ممکن است بخشی از اطلاعات از بین برود.

برای حل این چالش بهتر است برای تصاویر نویزی و یا با کنتراست و شدت نور محیط نامناسب، از Adaptive Thresholding استفاده کنیم و برای کل تصویر تنها یک Threshold در نظر نگیریم.

در این حالت باید برای هر ناحیه از تصویر یک آستانه متناسب تعریف شود. در حالت حدی میتوان برای هر پیکسل یک آستانه تعریف کرد. البته این محاسبات پیچیده برای هر پیکسل هزینه بر است. می توان میانگین پیکسل های اطراف هر ناحیه را به عنوان معیاری برای مقدار آستانه محاسبه کرد.

نقاط قوت Adaptive Thresholding: مبتنی بر ویژگی های محلی است. برای تصاویری که دارای سایه یا نویزی باشد و یا کنتراست و شدت نور محیط نا مناسب داشته باشند و بخش های تصویر قایل جدا کردن با یک Threshold نیستند خیلی خوب است.

نقاط ضعف Adaptive Thresholding: پیدا کردن Threshold برخلاف روش قبلی چالش بیشتری دارد. مشخصه های کل تصویر را در نظر نمیگیرد.

مقایسه عملکرد: عملکرد Otsu مبتنی بر مشخصه های آماری تصویر توسط هیستوگرام تصویر است. عملکرد Adaptive Thresholding بر اساس مشخصه های آماری همسایه های هر پیکسل به طور جداگانه و محلی است.

مقایسه سرعت: Otsu سرعت بالایی دارد .یک بار هیستوگرام محاسبه می شود و سپس 255 محاسبه انجام می دهد. اما سرعت Adaptive Thresholding بستگی به همسایگی ای که انتخاب میشود و تعداد پیکسل ها دارد. سرعت عمل convolution خوب است اما این روش در کل از Otsu کند تر است چون به تعداد پیکسل ها بستگی دارد.

ب)

این تابع به صورت است:

```
dst = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)
```

توضیح هر پارامتر:

src: تصویر ورودی که grayscale است.

MaxValue: بیشترین مقداری که به پیکسل های یک ناحیه متصل در تصویر خروجی assign می شود.

AdaptiveMethod: تابع میانگین یا گاوسی که Adaptive Thresholding توسط آن انجام می شود. تابع میانگین همسایه blocksize در blocksize برای هر پیکسل را حساب میکند و منهای C میکند. تابع گاوسی این کار را به صورت وزن دار انجام می دهد.

thresholdType: تایپ Thresholdind که می تواند Thresh_binary_inv یا Thresh_binary باشد. تفاوت در انتخاب مقدار پیکسل با توجه به شرط است. اگر Thresh_binary_inv باشد:

$$\mathtt{dst}(x,y) = \begin{cases} 0 \ if \ \mathtt{src}\,(x,y) > \mathtt{T(x,y)} \\ \mathtt{maxValue} \ otherwise \end{cases}$$

اگر Thresh_binary باشد:

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxValue} \; if \, \texttt{src}\,(x,y) > \texttt{T(x,y)} \\ 0 \; otherwise \end{cases}$$

blockSize: سایز همسایگی که در نظر می گیرد و روی آن عملیات را انجام می دهد و پیکسل را با میانگین ساده یا وزن دار همسایگی اش مقایسه می کند که اگر کوچکتر بود 0 و اگر بزرگتر بود 1 می‌گذارد. اگر همسایگی خیلی بزرگ باشد به سمت threshold global می رود و اگر خیلی کوچک باشد نتیجه نویزی میشود.

C: یک عدد ثابت. ممکن است در تصویر مقدار پیکسل ها کمی بخاطر نویز یا شرایط نور متفاوت شوند در صورتی که تقریبا یک رنگ بودند، اما یکی بالای میانگین بیفتد و یکی پایین میانگین. در این حالت اگر C در نظر نگیریم یکی از آن ها 0 و یکی 1 می شود. و تصویر خروجی نویزی می شود. پس برای حل این چالش، یک عدد C در نظر می گیریم. و سپس شرط 1 بودن پیکسل این است که از یه حدی روشن تر باشد، یعنی از میانگین به اضافه C بیشتر شود. و شرط 0 بودن این است که از یه حدی تیره تر باشد، یعنی از میانگین به اضافه C کمتر شود.

dst: تصویر خروجی باینری شده هم تایپ و هم سایز src.

مراحل اجرای الگوریتم: در این الگوریتم باید برای هر پیکسل با توجه به اندازه blockSize، همسایگی پیکسل را در نظر گرفته و میانگین پیکسل ها را در این همسایگی محاسبه می کنیم. این کار را با استفاده از convolve کردن کل تصویر با کرنل میانگین یا کرنل گاوسی میانگین وزن دار می توان انجام داد. kernel را نیز با توجه به اندازه blockSize می سازیم.

سپس هر پیکسل با میانگین پیکسل های همسایه اش مقایسه می شود. اگر مقدار پیکسل از میانگین به اضافه عدد ثابت C بیشتر بود مقدار آن 1 و اگر کمتر بود مقدار آن 0 می شود.

این کار را نیز می تواند طی یک مقایسه ساده کل تصویر با نتیجه convolution تصویر با کرنل میانگین، انجام داد. نتیجه مقایسه همان تصویر باینری خروجی است.

سودوکد این الگوریتم را طی یک خط می توان نوشت:

dst = src ≥ conv(src, mean_kernel) + C

جزئیات بیشتر هر پارامتر و نحوه استفاده از آن نیز در بالاتر شرح داده شد.