به نام خدا

عنوان

بخش چهارم از تکلیف اول درس پردازش تصویر رقمی

استاد

دکتر منصوری

دانشجو

محمدعلی مجتهدسلیمانی

۴۰۳۳۹۰۴۵۰۴

تاریخ

۱۴۰۴/۰۲/۵

Table of Contents

سوال چهار

این سوال در ۳ قسمت توضیح داده شده است و هر بخش گزارش کار پیاده سازی شده آن آمده است.

بخش الف

۲ فیلتر نام برده شده هر دور بر اساس گرادیان لبه ها را تشخیص میدهند این کار با تقریب زدن گرادیان توسط شدت/کمیت عکس صورت میگیرد.

فیلتر sobel

این فیلتر تخمین گرادیان عکس را به صورت جهت عمودی و افقی محاسبه میکند. از کانولوشن کمک میگیرد برای هر جهت که معمولا یک کرنل ۳ در ۳ دارد.

جهت افقی:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

جهت عمودی:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

کرنل افقی شدت تغییرات در جهت افقی را در سرتاسر پیکسل مرکزی اندازه گیری میکند و به سطر مرکزی بیشترین وزن را میدهد. کرنل عمودی دقیقا همین کار را به صورت عمودی میکند. کرنل ها بر روی عکس ورودی پیمایش انجام میدهند و در هر موقعیت یک ضرب نقطه ای محاسبه میکنند که بین کرنل و مجموع پیکسل های عکس است تا یک مقداری که متناظر با پیکسل در خروجی گرادیان تصویر است تولید شود. روش وزن دهی گفته شده باعث میشود که فیلتر گفته شده کمتر به نویز حساس باشد، کرنل هر چه بزرگتر باشد همسایگی بیشتری در نظر میگیرد و سبب میشود اون نواحی بیشتر هموار شوند یعنی نویز کمتر شوند البته که ممکن است این کار باعث **blurring** و از دست رفته لبه ها بشود.

فیلتر scharr

مانند **sobel** این فیلتر هم تخمین گرادیان را محاسبه میکند. این فیلتر برای بهبود فیلتر سوبل آمده است زیرا فیلتر سوبل حرکت دورانی ندارد و این سبب میشود که واکنش آن نسبت به لبه ها بر اساس جهت لبه ها باشد و فیلتر **scharr** سعی میکند اینکار را بهبود ببخشید.

کرنل افقی:

$$\begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}$$

کرنل عمودی:

$$\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

گذاشتن وزن ۱۰ در مرکز سبب میشود که این فیلتر بیشتر به شدت تغییرات حساس شود که باعث میشود بهتر لبه ها را پیدا کند. همچنین وزن های (۳و۱۰و۳) باعث شده است تا تخمین بهتر انجام شود زیرا به صورت چرخشی حرکت میکند. سایز کرنل آن معمولا ۳ در ۳ است.

گزارش کار

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
```
✓ 0.9s Python

از کتابخانه **cv2** برای اعمال عملیات ها مختلف مانند خواندن عکس‌ها استفاده میکنیم.

برای نشان دادن خروجی از **matplotlib** استفاده کرده‌ایم. همچنین تصاویر مورد نیاز را از **scikit-image** تحت عنوان **skimage** بارگذاری کرده‌ایم تا بتوانیم یک **benchmark** معیار را برای سنجش فیلترهای پیاده سازی شده خودمان داشته باشیم و نتایج را بررسی کنیم.

```
gray_img = data.page()

if gray_img is None:
    print(f"Error: Could not load image skimage.data.page()")
else:
    print("Benchmark image 'page' loaded successfully.")
    print(f"Image shape: {gray_img.shape}, Data type: {gray_img.dtype}")
```
[3] ✓ 0.0s Python

```
Benchmark image 'page' loaded successfully.
Image shape: (191, 384), Data type: uint8
```

در این قسمت عکس **benchmark** را بارگذاری کرده ایم و مطمئن میشویم کد به درستی کار میکند.

```
sobelx_64f = cv2.Sobel(gray_img, cv2.CV_64F, 1, 0, ksize=3)
sobely_64f = cv2.Sobel(gray_img, cv2.CV_64F, 0, 1, ksize=3)
sobel_magnitude = cv2.magnitude(sobelx_64f, sobely_64f)

sobel_final = cv2.convertScaleAbs(sobel_magnitude)
print("Sobel filter applied.")
```
[4] ✓ 0.0s Python

```
Sobel filter applied.
```

همانطور که توضیح دادیم با کمک کتابخانه **cv2** فیلتر سوبل را بر روی تصویر اعمال میکنیم. گرادیان را در ۲ جهت **x** و **y** محاسبه میکنیم با دقت ۶۴ بیت. اندازه کرنل را ۳ در نظر گرفتیم. در

نهایت اندازه گرادیان را بدست آوردیم و بعد اندازه را به خروجی ۸ بیت بدون علامت تبدیل کردیم تا بتوانیم آن را نمایش بدهیم.

```
scharrx_64f = cv2.Scharr(gray_img, cv2.CV_64F, 1, 0)
scharry_64f = cv2.Scharr(gray_img, cv2.CV_64F, 0, 1)

scharr_magnitude = cv2.magnitude(scharrx_64f, scharry_64f)

scharr_final = cv2.convertScaleAbs(scharr_magnitude)
print("Scharr filter applied.")
```

✓ 0.0s Python

```
Scharr filter applied.
```

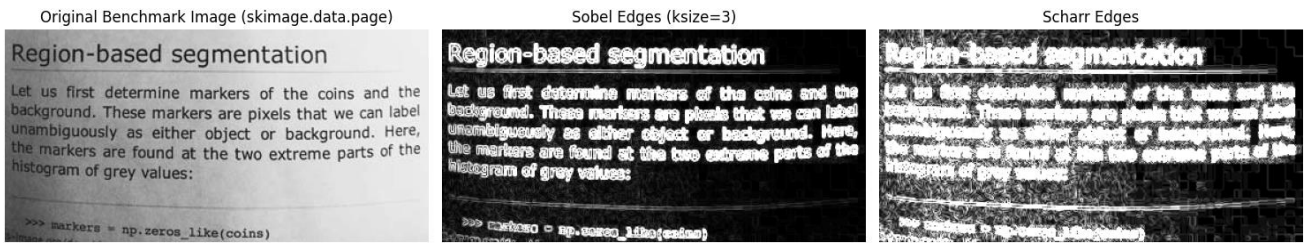
همانطور که توضیح دادیم حالا فیلتر **scharr** را اعمال میکنیم، به صورت پیشفرض اندازه کرنل ۳ هست پس نیازی به تعریف متغیر **ksize** نخواهد بود. بعد اندازه گرادیان را محاسبه میکنیم و در نهایت مانند سوبل خروجی را به ۸ بیت بدون علامت تبدیل میکنیم تا بتوانیم نمایش بدهیم.

```
plt.figure(figsize=(15, 7))

plt.subplot(1, 3, 1)
plt.imshow(gray_img, cmap='gray')
plt.title('Original Benchmark Image (skimage.data.page)')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(sobel_final, cmap='gray')
plt.title('Sobel Edges (ksize=3)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(scharr_final, cmap='gray')
plt.title('Scharr Edges')
plt.axis('off')

plt.tight_layout()
plt.show()
```

✓ 0.5s Python

در نهایت خروجی را نمایش میدهیم.

خروجی



تحلیل

همانطور که بالاتر توضیح دادیم فیلتر سوبل توانسته outline های کاراکتر به خوبی تشخیص دهد و لبه ها واضح هستند. فیلتر scharr توانسته نیز به خوبی outline ها را مشخص کند. به صورت کلی خروجی scharr بسیار روشن تر است نسبت به سوبل این بخاطر این هست که scharr وزن ها بزرگتری مورد استفاده قرار داده است که باعث میشود قوی تر در اطراف مرز کاراکتر ها عمل کند نسبت به سوبل. در کل به صورت کلی هر ۲ فیلتر خوب عمل کردند.

بخش ب

برای پیاده سازی اینکار ما یک آستانه تعریف میکنیم تا بتوانیم به کمک آن یک ماسک باینری بسازیم. با تعریف ماسک باینری پیکسل ها در مجموع ۲ مقدار فقط میگیرند یا سیاه هستند ۰ و یا سفید هستند (۲۵۵). در کدی که پیاده سازی میکنیم متن ها در عکس اصلی سفید خواهند بود و پس زمینه سیاه خواهد بود. برای تعریف کردن آستانه نیاز داریم که مقادیر پیکسل هایی که کمیت آنها از آستانه بالاتر بود سفید شوند و یک مقدار بگیرند و همینطور همینکار را برای پیکسل هایی که زیر آستانه هستند انجام میدهیم. به جای اینکه دستی خودمان آستانه تعریف بکنیم از روش Ostu's Binarization استفاده میکنیم که به صورت خودکار و الگوریتمی میتوانیم آستانه بهینه را تعریف کنیم تا پیکسل ها به ۲ کلاس تقسیم شوند. این روش با کاهش واریانس درون هر کلاس صورت میگیرد. بعد از اعمال این روش حالا میتوانیم ماسک خودمان را بسازیم و بعد از آن ماسک را روی تصویر اعمال میکنیم. از عملیات AND به صورت bitwise استفاده میکنیم تا بتوانیم ماسک را روی تصویر اعمال کنیم نحوه عمل کردن این AND به این صورت هست که به هر پیکسل نگاه میکنیم اگر پیکسل متناظر آن در ماسک سفید باشد، مقدار خودش را نگه میدارد. اگر پیکسل متناظر سیاه باشد، پیکسل مقدار خودش را از دست میدهد و مقدار جدید میگیرد.

گزارش کار

عمده موارد مانند قسمت الف است، در این قسمت فقط موارد جدید اضافه شده را بررسی میکنیم.

```
    otsu_threshold, binary_mask = cv2.threshold(
        gray_img, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU
    )
    print(f"Otsu's threshold calculated: {otsu_threshold}")
    print("Binary mask created.")
```

[5] ✓ 0.0s Python

```
Otsu's threshold calculated: 157.0
Binary mask created.
```

همانطور که در بالا توضیح دادیم ابتدا یک ماسک میسازیم و بعد از آن روش Ostu's را پیاده سازی میکنیم. این کار با کمک کتابخانه open cv انجام میشود، پیکسل هایی که زیر آستانه قرار بگیرند سفید یعنی ۲۵۵ و آنهایی که بالا باشند ۰ میگیرند. از آنجایی که از THRESH_BINARY_IMV استفاده میکنیم باعث میشود متن که سیاه است تبدیل به سفید شود.

```
text_only = cv2.bitwise_and(gray_img, gray_img, mask=binary_mask)
print("Text separated from background using the mask.")
```

[6] ✓ 0.0s Python

```
Text separated from background using the mask.
```

با کمک ماسک میایم و متن را جدا میکنیم با استفاده از اعمال ماسک روی تصویر اولیه که به صورت grayscale است. جایی که ماسک ۲۵۵ باشد مقدار پیکسل همان اولیه باقی میماند. اگر ماسک صفر باشد مقدار پیکسل در آنجا ۰ میشود.

```
inverse_mask = cv2.bitwise_not(binary_mask)
background_only = cv2.bitwise_and(gray_img, gray_img, mask=inverse_mask)
print("Background separated from text using the inverse mask.")
```

[7] ✓ 0.0s Python

```
Background separated from text using the inverse mask.
```

یک ماسک معکوس ساخته ایم تا بتوانیم پس زمینه را جدا کنیم.

در ادامه با کمک کتابخانه matplotlib خروجی را نمایش دادیم.

خروجی

بخش ج

همانطور که گفته شد الگوریتم **sobel, scharr** عمدتا بر روی تغییرات شدید تکیه میکنند تا بتوانند لبه ها را تشخیص دهند. در عکس های **low-contrast** تفاوت بین پس زمینه و محتویات عکس خیلی کم است. نتیجه میشود گرادیان ضعیف و خیلی سخت میشود تا لبه ها را تشخیص داد و یا ممکن است لبه ها از دست بروند.

روش رایج در این مواقع این است که ما بیایم **contrast** تصویر را بهبود بدهیم قبل از اینکه بخواهیم فیلتر ها را اعمال کنیم. با افزایش شدت تغییرات بین متن و پس زمینه، گرادیان بسیار زیادتر میشود و فیلتر های گفته میشوند به صورت موثر لبه ها را تشخیص دهند.

برای پیاده سازی این روش از تکنیک **CLAHE** استفاده میکنیم که باعث میشود که هیستوگرام تصویر را باز کند و مقادیر شدید تری در سراسر عکس پخش کند تا به صورت سراسری **contrast** بالا برود. این روش به صورت خالص ممکن است باعث تقویت نویز ها شود به خصوص در نواحی مرتبط به همدیگر بنابراین از **AHE** استفاده میکنیم تا بتوانیم به صورت سراسری بهبود ببخشیم به وسیله اعمال آن روی نواحی محلی. **CLAHE** یک روش بهبود یافته از **AHE** است که میزان

افزایش **contrast** را محدود میکند با اعمال کردن **cliplimit** روی نواحی محلی تا از تقویت زیاد نویز ها جلوگیری کند. این روش چندین هیستوگرام متناظر با بخش های مختلف تولید میکند. نحوه پیاده سازی آن در گزارش کار مشخص است.

گزارش کار

بخش عمده ای از کد مانند قسمت های قبلی است پس از توضیح آنها صرف نظر میکنیم.

اعمال CLAHE

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_img = clahe.apply(gray_img)
print("CLAHE applied to enhance contrast.")
```

```
✓ 0.0s Python
CLAHE applied to enhance contrast.
```

با کمک کتابخانه **open cv** میتوانیم **CLAHE** را به برنامه خودمان اضافه کنیم همچنین با کمک متغیر **clipLimit** یک آستانه ای برای محدود کردن میزان **contrast** قرار میدهیم هر چه قدر مقدار آن بیشتر باشد یعنی **contrast** بیشتری دارد. **tileGridSize** میاد و سایز هر کدام از آنها هیستوگرام هایی که گفتیم که به صورت محلی بودن را مشخص میکند.

دقت شود که ما یکبار دو فیلتر گفته شده را روی تصویر اصلی اعمال میکنیم و یکبار میایم و فیلتر **scharr** را روی عکس **CLAHE** اعمال میکنیم.

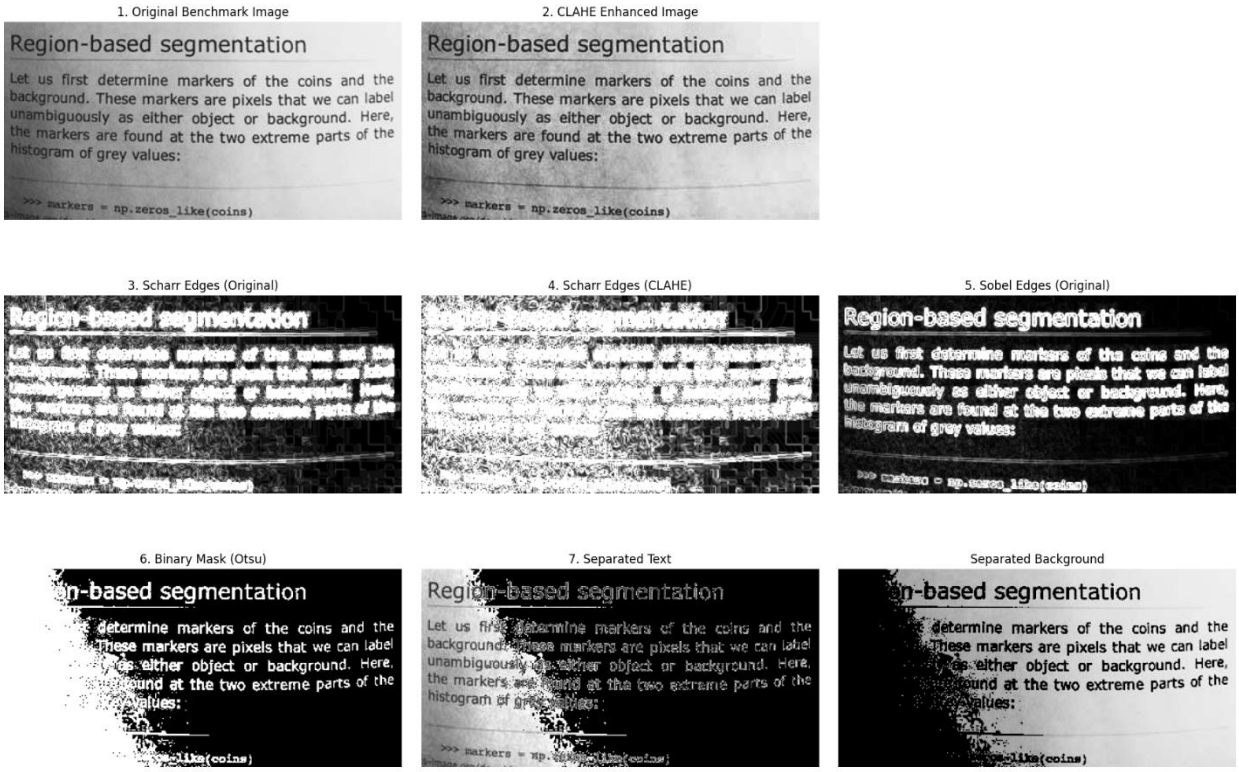
اعمال فیلتر scharr روی تصویر CLAHE

```
scharrx_64f_clahe = cv2.Scharr(clahe_img, cv2.CV_64F, 1, 0)
scharry_64f_clahe = cv2.Scharr(clahe_img, cv2.CV_64F, 0, 1)
scharr_mag_clahe = cv2.magnitude(scharrx_64f_clahe, scharry_64f_clahe)
scharr_clahe = cv2.convertScaleAbs(scharr_mag_clahe)
print("Scharr filter applied to CLAHE-enhanced image.")
```

```
✓ 0.0s Python
Scharr filter applied to CLAHE-enhanced image.
```

مانند قبل صرفا فیلتر اعمال کردیم و بعد خروجی را نمایش میدهیم.

خروجی



همانطور که قابل مشاهده است لبه ها در تصویر تولید شده توسط CLAHE برجسته تر و واضح تر هستند نسبت به عکس اصلی.