به نام خدا

عنوان

بخش سوم از تکلیف اول درس پردازش تصویر رقمی

استاد

دکتر منصوری

دانشجو

محمدعلی مجتهدسلیمانی

۴۰۳۳۹۰۴۵۰۴

تاریخ

۱۴۰۴/۰۲/۵

Table of Contents

سوال ۵

این سوال در ۳ بخش حل شده است که قسمت هایی که نیاز به پیاده سازی دارند در گزارش کار بخش مربوطه آن توضیحات پیاده سازی آمده است.

بخش الف

در این بخش تفاوت بین ۲ دسته از فیلترهای تقویتی و فیلترهای هموارسازی را بررسی میکنیم:

فیلترهای تقویتی

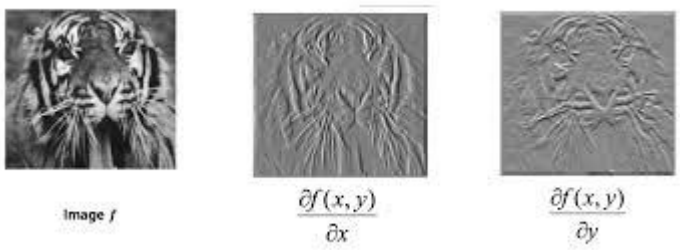
هدف این فیلتر ها بهبود جزئیات و واضح تر کردن لبه ها یا برجسته کردن آنها است. در واقع هدف آنها افزایش کنتراست محلی در امتداد لبه ها است. فیلترهای تقویتی معمولا با تاکید بر تفاوت بین مقادیر پیکسل مجاور کار میکنند. روش های رایج عبارتند از:

روش unsharp masking: این روش که یکی از رایج ترین روش ها است که ۳ مرحله دارد: اول یک نسخه تار (blur) از نسخه اصلی تولید میکند (اغلب به کمک gaussian blur) دوم نسخه تار را از نسخه اصلی کم میکنند که باعث میشود جزئیات فرکانس بالا مانند لبه ها را جدا کند. سوم افزودن این جزئیات به تصویر اصلی است که به طور موثر کنتراست را در لبه ها افزایش میدهد.

روش فیلتر لاپلاسین: این یکی از روش های تقویتی است که بر اساس مشتق دوم کار میکند که نواحی که در آن تغییر شدت بالا و سریع است مانند لبه ها یا نویز را بر جسته میکند. افزودن خروجی لاپلاس به تصویر اصلی این نواحی را بهبود میبخشد.

در نهایت تاثیر این فیلترها لبه ها را برجسته تر میکند و بافت ها را بهبود میبخشد و میتواند تصویر را تازه تر یا صاف تر نشان دهند. با این حال به تقویت نویز موجود نیز کمک میکند.

تصویر پایین بیانگر خروجی فیلتر گرادیان و فیلتر لاپلاسین است:



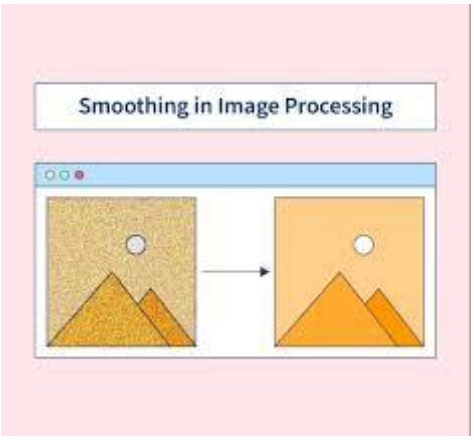
فیلترهای هموار ساز

این فیلتر ها مانند **gaussian blur**، به دنبال کاهش نویز و جزئیات هستند تا بتوانند ظاهری صاف تر و هموار تر ایجاد کنند. هدف آنها کاهش تفاوت بین پیکسل های مجاور است.

فیلتر های هموار ساز مقدار هر پیکسل را با میانگین وزنی پیکسل های مجاور آن جایگزین میکنند.

Gaussian blur: در این روش از یک طرح وزن دهی خاصی بر اساس تابع گوسی استفاده میکند. پیکسل های نزدیک تر به مرکز هسته فیلتر نسبت به پیکسل های دورتر تاثیر بیشتری بر میانگین دارند. این منجر به تاری بسیار هموار و طبیعی در مقایسه با تاری که با کمک میانگین یکنواخت بدست میاد بشود. میزان تاری توسط سیگما تابع گوسی کنترل میشود. سیگما بزرگتر به معنای تاری بیشتر است.

این فیلتر ها باعث میشوند که جزئیات که تیز تر هستند یا تند تر هستند را کاهش دهند و لبه ها را نرم تر کنند. به طور موثر فرکانس های بالا مانند نویز گوسی و نویز نمک و فلفل را کاهش دهند. این باعث میشود تصویر با جزئیات کمتر به نظر برسد.



فیلترهای هموار ساز	فیلترهای تقویتی	ویژگی
کاهش نویز و جزئیات	بهبود لبه ها و جزئیات	هدف
میانگین وزن دار همسایگی لحاظ میکند	اهمیت دادن به تغییر مقادیر پیکسل	مکانیزم
مانند فیلتر low-pass رفتار میکند	مانند فیلتر high-pass رفتار میکند	فرکانس
لبه ها را نرم تر میکنند و کنتراست را کاهش میدهند	باعث میشود لبه ها برجسته تر شوند و کنتراست را زیاد میکند	تاثیر بر لبه
جزئیات را کاهش میدهند	باعث افزایش جزئیات میشود	تاثیر بر جزئیات
نویز را کاهش میدهند	باعث تقویت نویز میشوند	تاثیر بر نویز

جدول بالا به طور خلاصه تفاوت این ۲ روش را توضیح میدهد.

بخش ب

در این بخش ما از ۲ فیلتر high-pass به نام sobel gradient sharpening و unsharp mask استفاده کردیم که توضیحات هر یک را بالاتر داده ایم، در ادامه به جزئیات پیاده سازی میپردازیم:

گزارش کار

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage import io
from skimage import color
from skimage import filters
from skimage.util import img_as_float, img_as_ubyte
```

✓ 2.7s Python

از **matplotlib** برای نمایش خروجی استفاده کردیم.

با کمک کتابخانه **scikit** هم برای تصویر ورودی و هم برای اعمال فیلتر ها و هم برای تبدیل ها استفاده کردیم. میتوانستیم همین کار را با **open cv** نیز انجام دهیم.

تنظیم پارامتر

```python
benchmark_image_name = 'camera'
sobel_sharpen_amount = 0.4
unsharp_radius = 1.5
unsharp_amount = 1.5
```

✓ 0.0s Python

تصویر ورودی را مشخص کردیم به عنوان **benchmark**.

در ادامه پارامتر های فیلتر **unsharp** را مشخص کردیم. **Radius** مشخص میکند تخمین سیگما گاوسی را که میزان هموار شدن جزئیات را کنترل میکند. **Amount** نیز قدرت نواحی تیز و تند را مشخص میکند یعنی به چه میزان باید وزن تخصیص کنیم. هر چه بیشتر باشد تیز تر خواهد بود.

بارگذاری تصویر

```python
try:
    original_image = getattr(data, benchmark_image_name)()
    print(f"Loaded benchmark image: '{benchmark_image_name}'")
except AttributeError:
    print(f"Error: Benchmark image '{benchmark_image_name}' not found in skimage.data.")

    original_image = data.camera()
    benchmark_image_name = 'camera'
```

✓ 0.0s Python

```
Loaded benchmark image: 'camera'
```

در این قسمت صرفا تصویر را بارگذاری کردیم و بررسی کردیم تصویر به درستی کار بکند.

سپس

```python
image_float = img_as_float(original_image)

if image_float.ndim == 3:
    gray_image_float = color.rgb2gray(image_float)
    print("Converted color image to grayscale.")
else:
    gray_image_float = image_float
```

0.0s Python

اول تصویر را به **float** بازنویسی میکنیم زیرا برای کتابخانه **skimage** ظاهرا طبق پیاده سازی آن نیاز است.

بعد از آن مطمئن میشویم تصویر حتما **grayscale** است و ۳ کاناله نیست.

پیاده سازی اول: **unsharp masking**

```python
sharpened_unsharp = filters.unsharp_mask(
    gray_image_float,
    radius=unsharp_radius,
    amount=unsharp_amount,
    channel_axis=None
)
sharpened_unsharp = np.clip(sharpened_unsharp, 0.0, 1.0)
```

0.0s Python

توضیحات مربوط به **unsharp masking** را بالاتر داده ایم و در اینجا صرفا موارد گفته شده را اضافه کردیم. برای اعمال این فیلتر از **skimage** کمک گرفتیم.

پیاده سازی دوم: sobel gradient sharpening

```
gradient_magnitude = filters.sobel(gray_image_float)
sharpened_sobel = gray_image_float + sobel_sharpen_amount * gradient_magnitude
sharpened_sobel = np.clip(sharpened_sobel, 0.0, 1.0)
```

✓ 0.0s Python

در این روش ابتدا روی جهت افقی و عمودی **sobel** را اعمال میکنیم. سپس اندازه گرادیان مقیاس داده شده را به تصویر اصلی اضافه میکنیم. در نهایت خروجی را محدود میکنیم با کمک **clip**. در نهایت خروجی را نشان دادیم، البته قبل از آن تصویر را به ۸ بیت تبدیل کردیم تا بتوانیم آن را نمایش بدهیم.

خروجی

مقایسه

به دنبال این هستیم که با برجسته کردن بیشتر لبه ها و جزئیات کوچک، تصویر واضح تر به نظر برسد.

ابتدا **unsharp masking**:

این روش معمولا برای تیز و برجسته کردن تصویر به کار میرود و متداول است که ابتدا یک نسخه کمی تار از تصویر ایجاد میکند که با **radius** میزان تار شدن آن را کنترل میکنیم. سپس متوجه میشود که چه جزئیاتی هنگام ایجاد نسخه تار شده از بین رفته است و این جزئیات گم شده بیشتر

لبه ها و بافت ها هستند، در نهایت این جزئیات از دست رفته را به تصویر اصلی اضافه میکند، اما آنها را قوی و برجسته تر میکند. تنظیم مقدار میزان قوی و تیز تر شدن آن را **amount** کنترل میکند. این کار سبب میشود نتایج نرم تر و کنترل بیشتری داشته باشیم نسبت به روش **sobel sharpening**.

در گام دوم **sobel gradient sharpening** را پیاده سازی کردیم، این روش با مشاهده سرعت تغییر روشنایی در تصویر، لبه ها را پیدا میکند. جایی که روشنایی به سرعت تغییر میکند احتمالا یک لبه است و آن را تشخیص میدهد. سپس با افزودن روشنایی اضافی به این نواحی لبه، تصویر را واضح تر میکند. میزان روشنایی اضافی توسط **amount** تنظیم و کنترل میشود. این کار لبه ها را قوی تر میکند، اما مانند سایر روش ها، دانه های نویز را که از نوع نویز **speckle** هستند نیز تقویت میکند و آنها را نیز بیشتر در تصویر برجسته میکند.

به نظر میرسد هر دو تصویر را واضح تر میکنند. **sobel** ممکن است گاهی اوقات لبه ها را کمی ضخیم تر یا پر رنگ تر نشان دهد. هر دو روش نویز مخصوصا از نوع نویز **speckle** تقویت میکنند. نحوه ظاهر شدن نویز در این ممکن است کمی تغییر کند.

به نظر میرسد **unsharp masking** راه های بیشتری برای تنظیم خروجی در اختیار قرار میدهد یعنی **radius** برای تاری و **amount** برای تنظیم میزان تیز تر شدن لبه ها، **sobel** عمدتا فقط امکان **amount** را برای تنظیم میدهد.

به طور کلی میتوان گفت که برای شارپ کردن عکس ها **unsharp masking** معمولا انتخاب بهتری است، انعطاف پذیر است و کنترل خوبی میدهد و اغلب طبیعی تر بنظر میرسد.

بخش ج

توضیحات مربوط به روش لاپلاسین در بخش الف به طور کامل گفته شده است و اینجا به جزئیات پیاده سازی میپردازیم و نتایج را در نهایت تحلیل میکنیم.

گزارش کار

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage import io
from skimage import color
from skimage import filters
from skimage.util import img_as_float, img_as_ubyte, random_noise
```

✓ 0.0s Python

از **matplotlib** برای نمایش خروجی استفاده کردیم.

با کمک کتابخانه **scikit** هم برای تصویر ورودی و هم برای اعمال فیلتر ها و هم برای تبدیل ها استفاده کردیم. میتوانستیم همین کار را با **open cv** نیز انجام دهیم.

تنظیم پارامتر

```
benchmark_image_name = 'camera'
laplacian_sharpen_amount = 0.7
noise_mode = 'gaussian'
noise_parameter = 0.01
```

✓ 0.0s Python

تصویر ورودی را مشخص کردیم به عنوان **benchmark**.

در ادامه پارامتر لاپلاسین را مشخص کردیم که مشخص میکند چه مقدار از لاپلاسین باید کم شود از تصویر اصلی هر چه این مقدار بزرگتر باشد تصویر تیز تر و برجسته تر خواهد بود.

در ادامه نوع نویز را مشخص کردیم میتوانستیم از نویز نمک و فلفل نیز استفاده کنیم.

بعد از با کمک **Nosie parameter** مقدار واریانس برای نویز گاوسی را مشخص کردیم.

```
try:
    original_image = getattr(data, benchmark_image_name)()
    print(f"Loaded benchmark image: '{benchmark_image_name}'")
except AttributeError:
    print(f"Error: Benchmark image '{benchmark_image_name}' not found.")
    print("Falling back to 'camera'.")
    original_image = data.camera()
    benchmark_image_name = 'camera'
```
[4] ✓ 0.0s Python

عکس مورد نظر را بارگذاری کردیم و مطمئن شدیم درست کار میکند.

```
image_float = img_as_float(original_image)
if image_float.ndim == 3:
    gray_image_float = color.rgb2gray(image_float)
    print("Converted color image to grayscale.")
else:
    gray_image_float = image_float
```
[15] ✓ 0.0s Python

اول تصویر را به **float** بازنویسی میکنیم زیرا برای کتابخانه **skimage** ظاهرا طبق پیاده سازی آن نیاز است.

بعد از آن مطمئن میشویم تصویر حتما **grayscale** است و ۳ کاناله نیست.

```
if noise_mode == 'gaussian':
    if noise_parameter < 0:
            print(f"Warning: Noise variance ('noise_parameter') cannot be negative. Using absolute value: {abs(noise_parameter)}")
            noise_parameter = abs(noise_parameter)
    noisy_image_float = random_noise(gray_image_float, mode='gaussian', var=noise_parameter)
    print(f"Successfully added 'gaussian' noise with variance: {noise_parameter}")

elif noise_mode == 's&p':
    if not (0 <= noise_parameter <= 1):
            print(f"Warning: S&P noise amount ('noise_parameter') should be between 0 and 1. Clamping value to range.")
            noise_parameter = np.clip(noise_parameter, 0.0, 1.0)
    noisy_image_float = random_noise(gray_image_float, mode='s&p', amount=noise_parameter)
    print(f"Successfully added 's&p' noise with amount: {noise_parameter}")
```
[17] ✓ 0.0s Python

```
Successfully added 'gaussian' noise with variance: 0.01
```

در این مرحله میخواهیم نویز را به تصویر اعمال کنیم. این کار با ساختن یک نسخه نویزی از تصویر با کمک **skimage.util.random_noise** انجام میشود. همچنین پیشبینی کرده ایم که اگر کاربر بخواهد از نویز نمک و فلفل استفاده بکند نیز پوشش بدهیم.

```
laplacian_original = filters.laplace(gray_image_float)
laplacian_noisy = filters.laplace(noisy_image_float)
```
[18] ✓ 0.0s Python

بعد از آن فیلتر لاپلاس را اعمال کردیم.

هم بر روی تصویر اصلی و هم بر روی تصویر نویزی شده، فیلتر را اعمال کرده ایم.

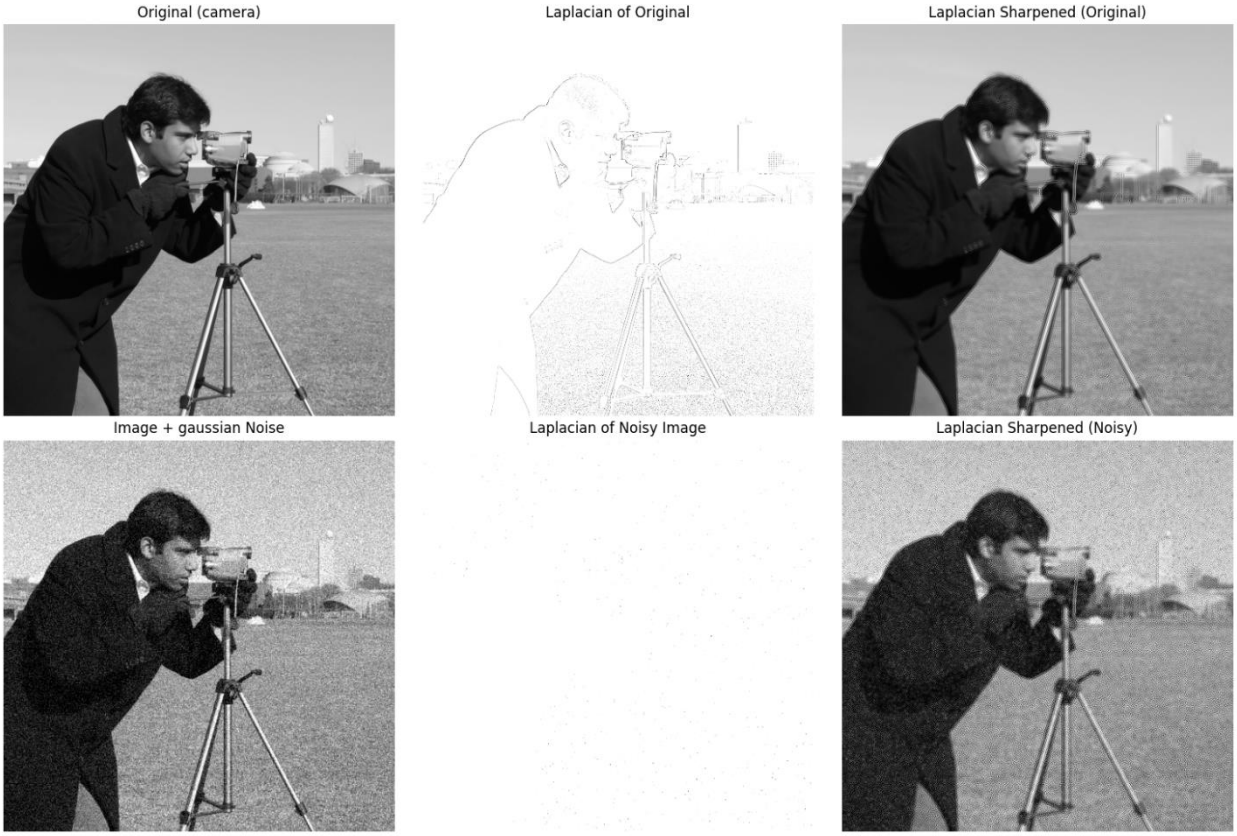
```
sharpened_original = gray_image_float - laplacian_sharpen_amount * laplacian_original
sharpened_noisy = noisy_image_float - laplacian_sharpen_amount * laplacian_noisy
```
[19] ✓ 0.0s Python

Sharpening را روی تصویر اعمال کردیم.

در نهایت خروجی را به ۸ بیت تبدیل کردیم تا قابل نمایش باشد. و بعد با کمک کتابخانه matplotlib آن را نمایش میدهیم.

```
gray_image_ubyte = img_as_ubyte(gray_image_float)
noisy_image_ubyte = img_as_ubyte(noisy_image_float)
laplacian_original_display = img_as_ubyte(np.clip(laplacian_original - np.min(laplacian_original), 0, 1))
laplacian_noisy_display = img_as_ubyte(np.clip(laplacian_noisy - np.min(laplacian_noisy), 0, 1))
sharpened_original_ubyte = img_as_ubyte(sharpened_original)
sharpened_noisy_ubyte = img_as_ubyte(sharpened_noisy)
```

خروجی



تحلیل

همانطور که در بخش اول گفتیم روش لاپلاسین میتواند به خوبی نقاطی که روشنایی به سرعت تغییر میکنند و نقاط تیز را پیدا کند یعنی همان جزئیات و لبه ها ولی مشکل این است که اینکار همراه با نویز معمولا از نوع **speckle** همراه است. برای اینکه عکس را تیز تر کنیم یعنی لبه ها را برجسته تر کنیم نیاز داریم یک **map** از نقاط تیز داشته باشیم و آن را از تصویر اصلی کم کنیم همانطوری که قبلا در بالا توضیح دادیم.

اگر تصویر ورودی بدون نویز باشد همانطور که قابل مشاهده است این روش خوب کار میکند، لبه ها را پیدا میکند و تصویر صاف تر و تیز تر تولید میکند.

اگر تصویر نویزی باشد، چون که ما یک سری نویز اضافه کردیم، این نویزها تیز هستند و در روشنایی تاثیر گذار هستند همانطور که قابل مشاهده است، روش لاپلاسین حساس خواهد شد و نویز ها را هم همانند لبه ها برجسته میکند، و وقتی از نگاشت نقاط تیز استفاده میکنیم، دانه های نویز را همانند لبه های واقعی تقویت میکنیم و وضعیت نویز بدتر میشود و خروجی به طور کلی بدتر میشود. اگرچه که لبه ها نیز تقویت میشوند و کمی برجسته تر میشوند اما به طور کلی خروجی بدتر میشود.