

به نام خدا

گزارش تمرین سری دوم - درس مبانی بینایی کامپیوتر

سید محمد علی خواری - شماره دانشجویی: 99521496

سوال اول)

الف) در ابتدا با استفاده از تابع نوشته شده آرایه ای دو بعدی با اندازه $n \times n$ که اعضای آن اعداد تصادفی بین 0 تا 255 هستند، میسازیم.

```
[ ] # generate random matrix with size n*n
def generateRandomMatrix(n):
    matrix = np.random.randint(0, 256, size=(n,n), dtype=np.uint8)
    return matrix
```

در مرحله بعد تابع کانوالو را تعریف میکنیم که کار آن محاسبه کانولوشن یک تصویر دو بعدی با یک کرنل دو بعدی است. برای این کار ابتدا باید به اندازه نصف طول و عرض کرنل برای تصویر ورودی border ایجاد کنیم. سپس بر روی تصویر ورودی پیایش کرده و از تصویر با حاشیه اضافه شده در هر پیایش به اندازه کرنل، جدا کرده و آن را در ماتریس کرنل ضرب میکنیم و حاصل جمع ماتریس به دست آمده را در خانه مرتبط با آن در ماتریس نهایی میریزیم.

پس از آن کرنل های افقی و عمودی عملگر سوبل را تعریف میکنیم. در نهایت با استفاده از توابع کامل شده، ابتدا یک آرایه 10×10 ساخته و اعضای آن را به صورت رندوم پر میکنیم و به صورت جداگانه آرایه به دست آمده را یک بار با کرنل افقی و یک بار با کرنل عمودی کانوالو میکنیم. پس از این کار با استفاده از توابع calc_magnitude و calc_direction به محاسبه ماتریس اندازه و ماتریس جهت گردیدیان میپردازیم.

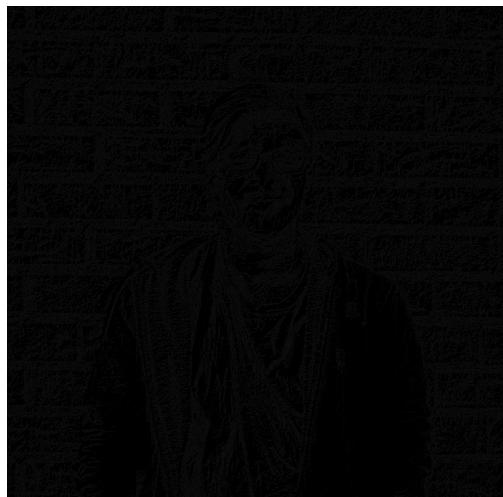
(ب)

- با توجه به اینکه از محیط colab برای حل تمرین استفاده کردم، مسیر خواندن تصویر را درون یک متغیر ذخیره کرده و از آن در تابع imread استفاده کرده ام.

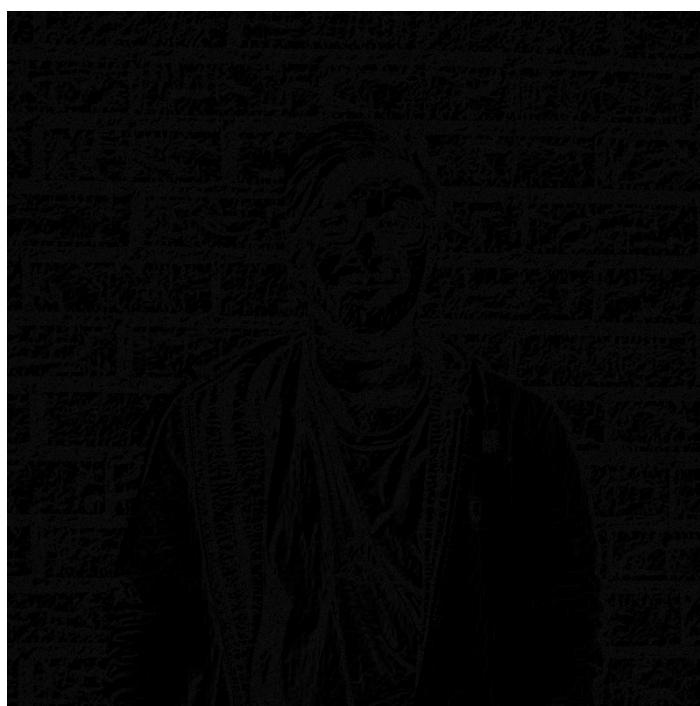
پس از خواندن تصویر، برای اینکه بتوانیم عملگر سوبل را بر روی آن اعمال کنیم، آن را به یک تصویر در حالت gray تبدیل میکنیم.

همانطور که گفته شد، عملگر سوبل حساس به نویز است. به همین دلیل برای اینکه بتوانیم خروجی بهتری داشته باشیم ابتدا با یکی از فیلتر های هموار ساز مثل فیلتر گاوین تصویر را هموار کرده و نویز های آن را کمتر میکنیم.

پس از اعمال فیلتر گاوین بر روی تصویر، عملگر سوبل را بر روی تصویر اعمال میکنیم. خروجی نهایی برای زمانی که از فیلتر هموار ساز برای حذف نویز استفاده نکنیم، به صورت زیر خواهد بود:



همانطور که مشخص است، نویز های موجود در تصویر باعث شده است که تغییرات رنگ در تصویر به سادگی قابل تشخیص نباشد. پس از اعمال یک فیلتر گاوی با اندازه 3×3 و سیگما 40، خروجی به صورت زیر خواهد بود:



همانطور که مشخص است، در این حالت جزئیات تغییر رنگ (لبه های افقی و عمودی) بهتر قابل تشخیص است.

(ج)

در این قسمت با استفاده از تابع آماده عملگر سوبل در کتابخانه opencv لبه های موجود در تصویر را پیدا میکنیم.

```
[ ] # do the operations in part b with OpenCV Sobel method and describe its parameters
sobelx = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=1, scale=1, delta=0)
sobely = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=1, scale=1, delta=0)

# Combine the results of the Sobel filter in both x and y directions
sobel = calc_magnitude(sobelx, sobely)
cv2_imshow(sobel)
```

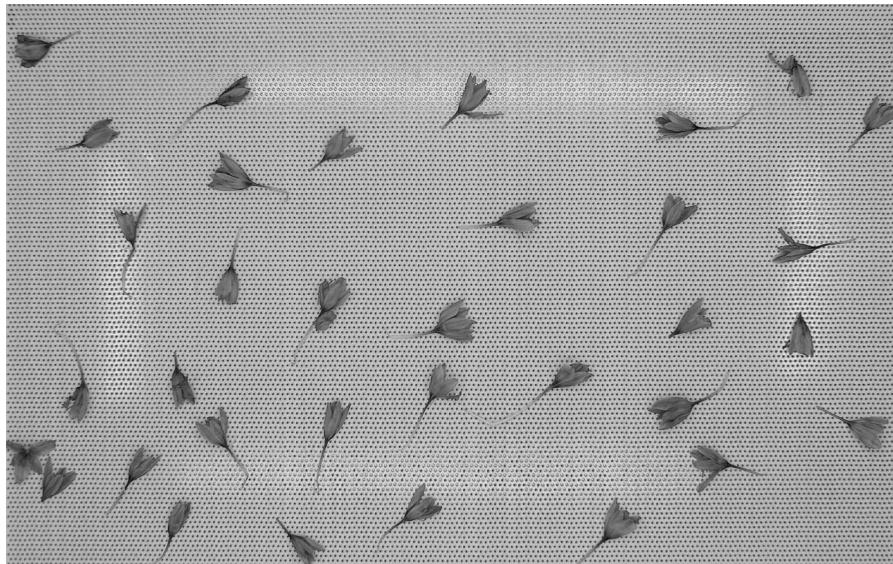
آرگومان های ورودی تابع sobel در کتابخانه opencv به ترتیب، تصویر gray scale که میخواهیم لبه های آن را تشخیص دهیم، عمق تصویر یا depth که در اینجا برای جلوگیری از overflow مقدار آن را برابر با 64f قرار دادم، درجهت x و درجهت y که مشخص میکند در چه جهتی میخواهیم عملگر سوبل را اعمال کنیم، ksize که اندازه کرنل عملگر افقی یا عمودی سوبل را مشخص میکند، پارامتر scale که به ترتیب مقادیر کرنل های افقی و عمودی عملگر را مشخص میکند که مقدار یک برابر با دیفالت بوده و مقادیر موجود در کرنل را تغییر نمیدهدنده.

خروجی نهایی به صورت زیر خواهد بود:



سوال دوم)

الف) ابتدا باید تصویر گفته شده را با استفاده از کتابخانه opencv خوانده و آن را به فرمت gray بیاریم، تصویر خروجی به صورت زیر خواهد بود:



همانطور که در سوال گفته شد باید نویز موجود در تصویر که همان روزنه های موجود در تصویر هستند را با استفاده از تبدیل فوریه از بین بیاریم.

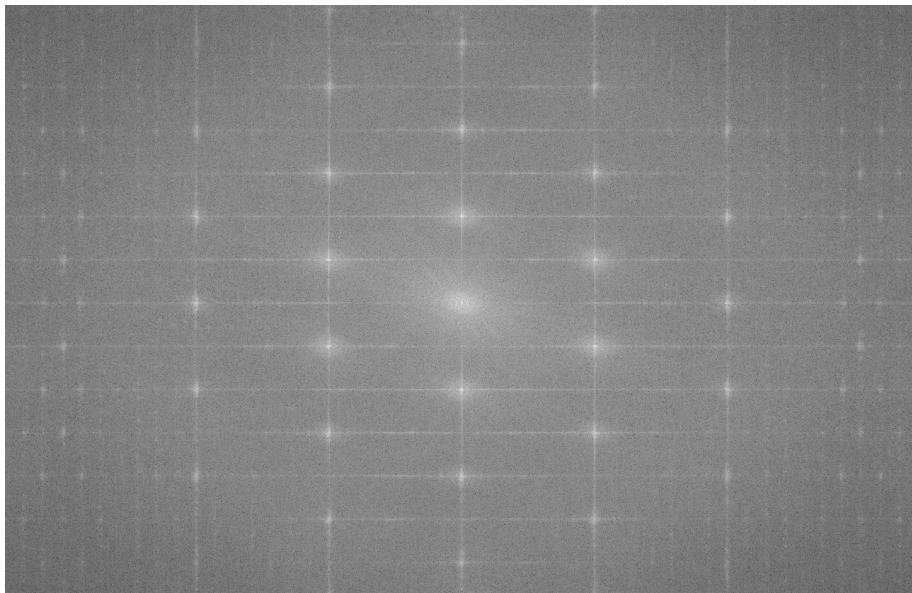
برای این کار ابتدا با استفاده از توابع موجود باید تبدیل فوریه تصویر داده شده را بدست بیاوریم.

تابع fft.fft2(img) از کتابخانه numpy این کار را انجام میدهد. که ورودی آن همان تصویر gray به دست آمده در مرحله قبل است. سپس برای آنکه بتوانیم فرکانس های مهم و تاثیر گذار بر تصویر را بهتر ببینیم باید فرکانس صفر تصویر که در مکان 0 و 0 از تصویر به دست آمده از فرکانس قرار دارد

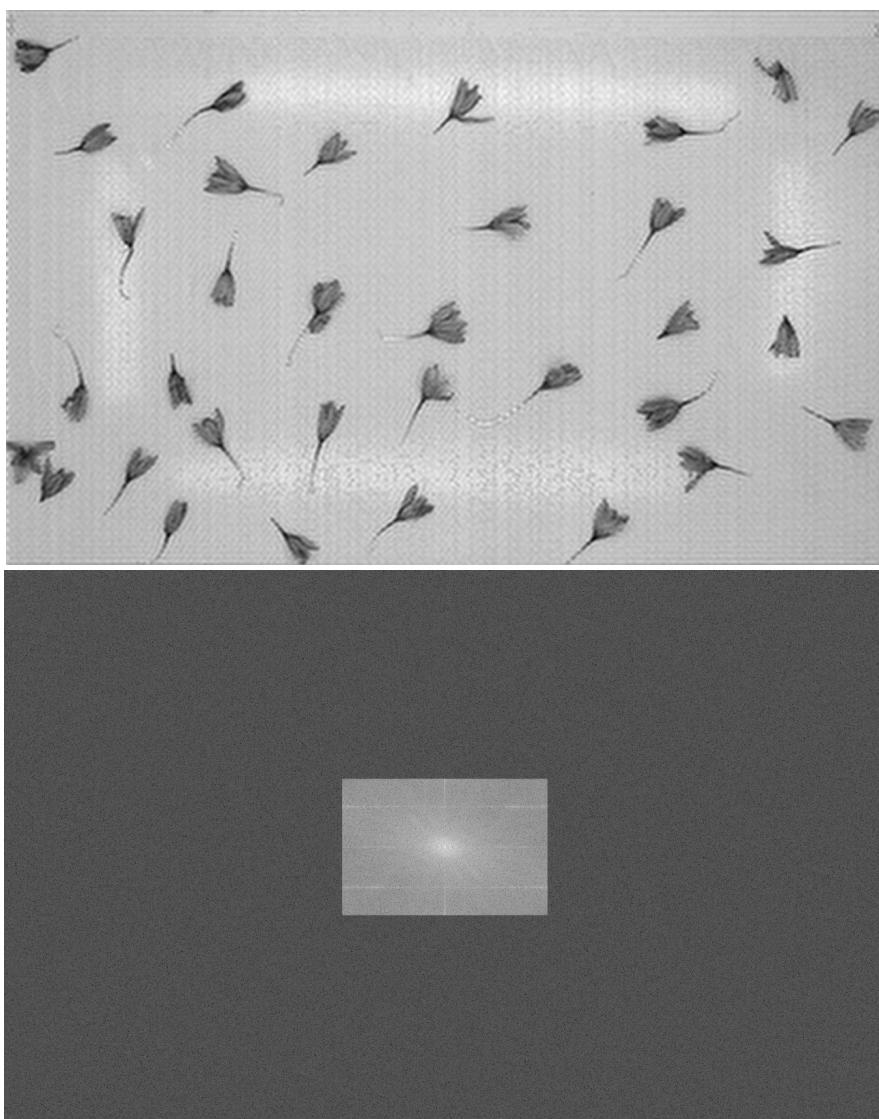
را به مرکز آن شیفت میدهیم، این کار هم با تابع fft.fftshift از کتابخانه numpy انجام میگیرد.

حال برای وضوح بهتر نتیجه و افزایش contrast تصویر از نتیجه به دست آمده لگاریتم گرفته و در

یک عدد ثابت ضرب میکنیم.



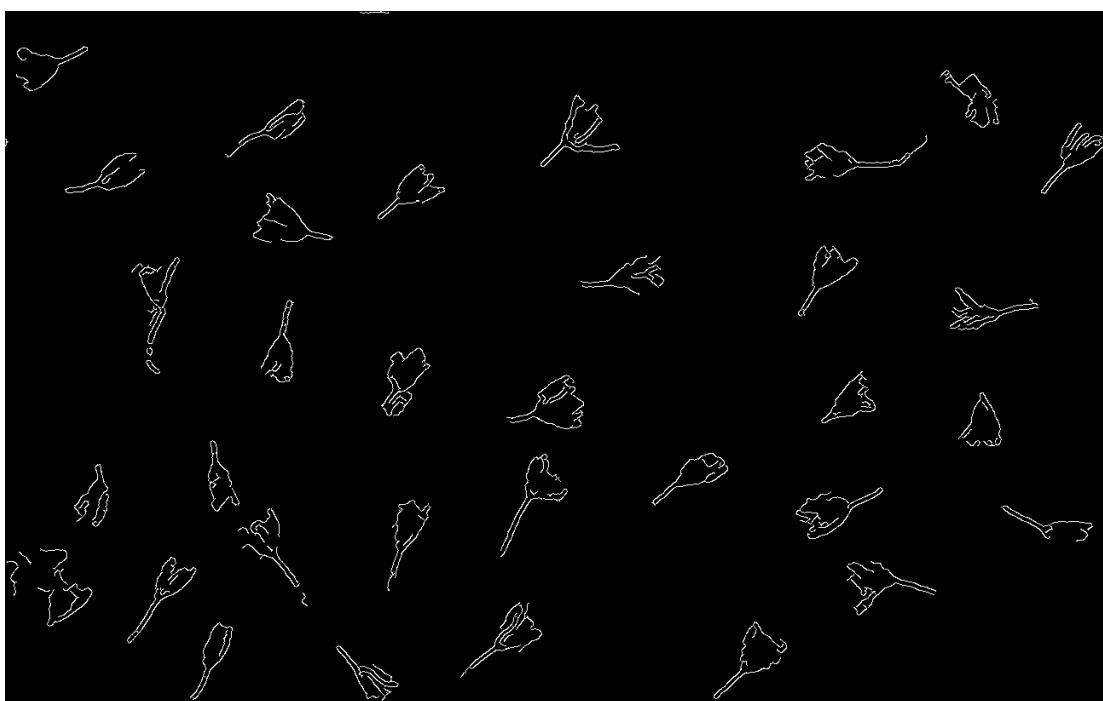
حال برای اینکه بتوانیم نویز های تصویر را از بین ببریم باید فرکанс هایی که موجب ایجاد این نویز متناوب شده اند را شناسایی کرده و آن ها را از بین ببریم. برای این کار راه های مختلفی برای حذف فرکанс های نویزی انجام دادم اما در نهایت بهترین حالتی که باعث حذف نویز شد، حالتی بود که تمام فرکанс ها به جز ناحیه $100^{\circ} \times 150^{\circ}$ مرکزی را از بین بردم. برای این کار تبدیل فرکанс ابتدایی را در تبدیل فرکانسی جدید ضرب کرده و مراحلی که طی کردیم تا تصویر را به حالت فرکانسی ببریم برعکس طی کرده تا بتوانیم تصویر نهایی و بدون نویز را به دست پاوریم.



ب) در این قسمت با استفاده از تابع canny لبه های موجود در تصویر نهایی را بدست می آوریم. برای این کار تابع canny را با پارامتر های زیر صدا زده و خروجی را نمایش میدهیم.

```
edges = cv2.Canny(img_filtered, 70, 200)  
cv2_imshow(edges)
```

سه پارامتر تابع canny به ترتیب عکسی که میخواهیم لبه های موجود در آن را پیدا کنیم و دو مقدار بعدی به عنوان threshold معین میکنیم، به این صورت که مقادیر بزرگتر از پارامتر سوم به عنوان حاشیه در نظر گرفته میشوند و مقادیر بین پارامتر دوم و سوم در صورتی که به مقادیر حاشیه متصل باشند، به عنوان حاشیه در نظر گرفته میشوند در غیر اینصورت حاشیه نیستند و مقادیر پایین تر از پارامتر دوم هم حاشیه نیستند. نتیجه نهایی به صورت زیر خواهد بود:



ج) در این قسمت با استفاده از عملگر sobel ماتریس اندازه گردانی و جهت آنها را بدست می آوریم.

(سوال پنجم)

(الف)

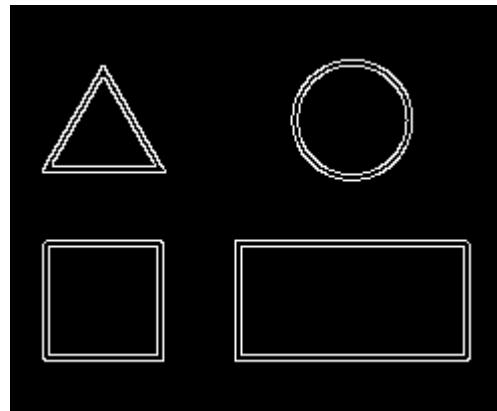
در این سوال ابتدا تصویر گفته شده را از محیط گوگل درایو خود خوانده و آن را به حالت gray در می آوریم.

(ب)

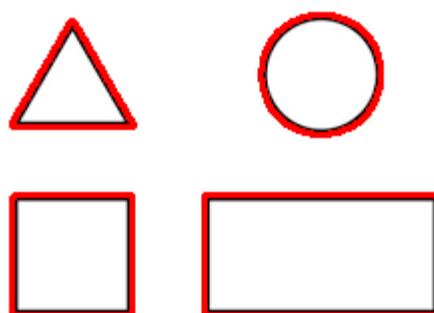
تابع findContours یکی از توابع موجود در کتابخانه opencv میباشد که از آن برای تشخیص خطوط اشیا در تصاویر باینری بکار میرود. روند کلی آن به این صورت است که تصویری به فرمت باینری را از ورودی گرفته و لیستی از خطوط موجود در آن را بر میگرداند.

```
[ ] edge = cv2.Canny(gray, 0, 800)
cv2_imshow(edge)
contours, hierarchy = cv2.findContours(edge, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img1, contours, -1, (0, 0, 255), 2)
cv2_imshow(img1)
```

همانطور که گفته شد ابتدا تصویر را از ورودی گرفته و آن را به حالت gray درآورده، سپس باید با استفاده از تابع threshold و یا لبه یاب canny تصویر را به صورت باینری تبدیل کنیم، یعنی محدوده ای تعیین کنیم که هر پیکسلی که بزرگتر از یک مقداری باشد مقداریک و اگر کمتر از آن باشد مقدار صفر بگیرد. لبه یاب canny این کار را برای ما انجام میدهد. خروجی اعمال لبه یاب canny به صورت زیر خواهد بود:

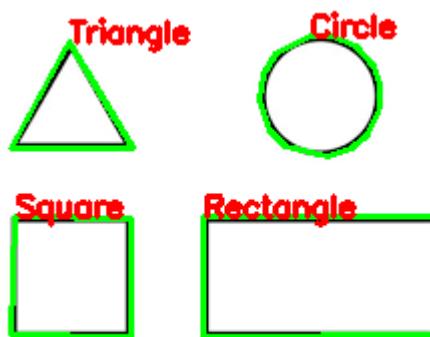


حال تابع findContours را بر روی تصویر باینری به دست آمده صدا زده و خطوط آن را به دست آورده و با استفاده از تابع drawContour و مشخص کدن رنگ این خطوط را مشخص میکنیم. خروجی به صورت زیر خواهد بود:



ج) در این قسمت با استفاده از تابع approxPolyDP نقاط گوشی اشکال موجود در تصویر را پیدا کرده و از روی آن نوع اشکال را تشخیص میدهیم. برای این کار بر روی خروجی تابع approxPolyDP که آرایه از خطوط موجود در تصویر میباشد پیاپیش کرده و تابع findContorus را برای هر کدام از اعضای آن آرایه صدا میزنیم.

پارامتر های ورودی تابع approxPolyDP به ترتیب خطوط شکلی که میخواهیم نوع آن را تشخیص دهیم، پارامتر بعدی epsilon است. این پارامتر مشخص میکند که دقت تقریبی تشخیص را مشخص میکند هرچه مقدار اپسیلون کوچکتر باشد تقریب بهتری خواهد داشت، پارامتر آخر closed میباشد که مشخص میکند چند ضلعی تقریبی باید بسته یا باز باشد که در اینجا مقدار آن را برابر با true قرار میدهیم. در نهایت از روی اندازه خروجی تابع approxPolyDP نوع شکل را تشخیص میدهیم که $1 \leq 3$ باشد، شکل مثلث، 4 باشد بررسی میکنیم که مربع هست یا مستطیل که این کار را با تابع boundingRect انجام میدهیم که چهار خروجی دارد: دو خروجی اول نقطه بالا-چپ شکل را خروجی میدهد و دو مقدار بعدی عرض و طول آن را خروجی میدهد که با تقسیم طول بر عرض در صورتی که مقدار نزدیک به یک داشته باشد آن را مربع تشخیص میدهیم، و اگر هیچ کدام از آنها نباشد شکل را دایره تشخیص میدهیم.



برای نوشتن نام اشکال بر روی تصویر از تابع opencv.putText کتابخانه استفاده میکنیم که میتوانیم فونت نوشتار، رنگ آن، اندازه آن را مشخص کنیم.

(سوال ششم)

الف) ابتدا تابع Reflect101 را پیاده سازی میکنیم که به عنوان ورودی تصویر و اندازه کرنل را گرفته و تصویر با حاشیه را بر میگرداند. این نوع حاشیه به این صورت عمل میکند که به اندازه نصف مقدار طول و عرض کرنل به طول و عرض تصویر ورودی مقادیر تصویر را آینه میکند. تنها تفاوت این نوع padding با refletct در این است که پیکسل مرزی در نظر گرفته نمیشود.

```

rows, cols = img.shape
padd_size = filter_size // 2
image = np.zeros((rows+2*padd_size, cols+2*padd_size), dtype=img.dtype)
# fill the top padded image
image[0:padd_size, padd_size:-padd_size] = np.flipud(img[1:padd_size+1, :])
# fill the down padded image
image[-padd_size:, padd_size:-padd_size] = np.flipud(img[-padd_size-1:-1, :])
# fill the right padded image
image[padd_size:-padd_size, -padd_size:] = np.fliplr(img[:, -padd_size-1:-1])
# fill the left padded image
image[padd_size:-padd_size, 0:padd_size] = np.fliplr(img[:, 1:padd_size+1])

# fill the corners of padded image
image[0:padd_size, 0:padd_size] = np.fliplr(np.flipud(img[1:padd_size+1, 1:padd_size+1]))
image[0:padd_size, -padd_size:] = np.fliplr(np.flipud(img[1:padd_size+1, -padd_size-1:-1]))
image[-padd_size:, 0:padd_size] = np.fliplr(np.flipud(img[-padd_size-1:-1, 1:padd_size+1]))
image[-padd_size:, -padd_size:] = np.fliplr(np.flipud(img[-padd_size-1:-1, -padd_size-1:-1]))

#fill the center of padded image
image[padd_size:-padd_size, padd_size:-padd_size] = img

return image

```

همانطور که در کد بالا مشخص است ابتدا بالای تصویر را با استفاده از تابع flipud که از کتابخانه numpy است را پر میکنیم. کارتای flipud این است که به عنوان ورودی یک آرایه دو بعدی گرفته و آن را در راستای افقی قرینه میکند. به همین ترتیب پایین تصویر را هم پرمیکنیم. پس از آن چپ و راست تصویر را با استفاده از تابع fliplr که همانند تابع flipud بوده و فقط آرایه دو بعدی گرفته شده را در راستای عمودی قرینه میکند. سپس گوشه های تصویر را با استفاده از تابع fliplr در نقاط مربوطه پرمیکنیم. در آخر هم باقی تصویر را به صورت معمولی در نتیجه نهایی قرار میدهیم.

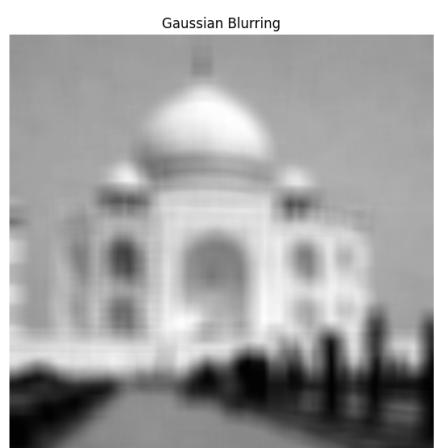
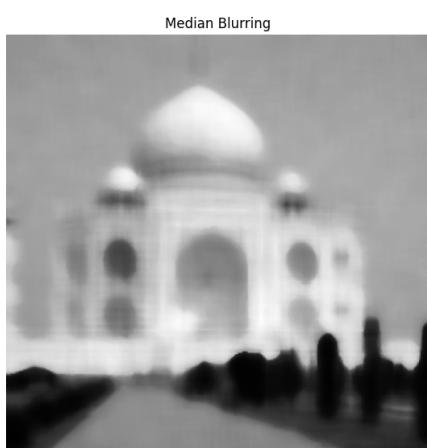
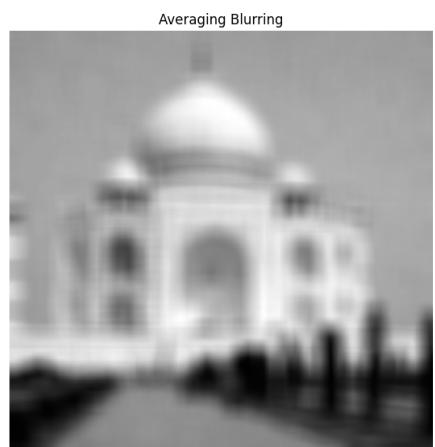
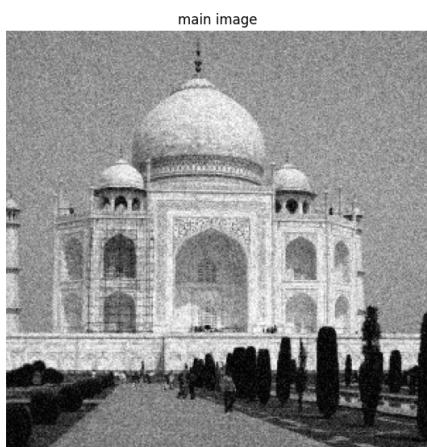
پس از پیاده سازی تابع بالا، تابع مربوط به فیلتر متوسط گیر را پیاده میکنیم. برای این کار کافی است که ابتدا تصویر ورودی را با تابع reflect101 حاشیه بدھیم و سپس کرنل میانگین گیری را ساخته و آن را با تصویر حاشیه داده شده کانوالو میکنیم.

تابع بعدی فیلتر میانه است. برای این کار ابتدا با تابع reflect101 تصویر را padd میدھیم و سپس بر روی تصویر ورودی پیاپیش کرده و مقدار هر خانه را برابر با مقدار میانه یک آرایه به اندازه کرنل ورودی قرار میدھیم.

در آخر فیلتر گاؤسین را پیاده میکنیم که برای این کار همانند فیلتر های قبلی ابتدا به تصویر padd داده و سپس با استفاده از فرمول موجود در اسلاید های درس کرنل گوسی را با توجه به مقدار سیگما محاسبه کرده و آن را با تصویر ورودی کانوالو میکنیم.

$$G(s, t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}} = Ke^{-\frac{r^2}{2\sigma^2}}$$

خروجی نهایی این قسمت به صورت زیر خواهد بود:

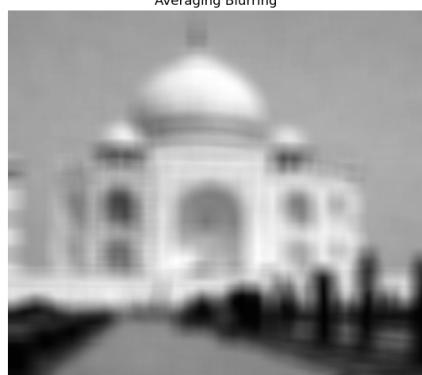


ب) برای پیاده سازی فیلتر دو طرفه منابع مختلفی را جستجو کردم که در نهایت فیلمی در یوتیوب پیدا کردم و از روی آن تابع آن را پیاده کردم اما به نتیجه مطلوبی نرسیدم و خروجی نهایی با خروجی تابع پیاده سازی شده در کتابخانه opencv تفاوت دارد. خروجی آن به صورت زیر میباشد:

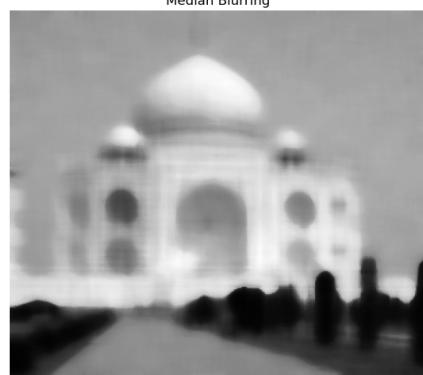


ج) در این قسمت با استفاده از توابع موجود در کتابخانه opencv تمای توابع نوشته شده در قسمت الف و ب را پیاده کرده و نتیجه آن را با خروجی تابع های پیاده شده مقایسه میکنیم که به جز نتیجه تابع فیلتر دو طرفه باقی توابع نتایج یکسانی داشتند.

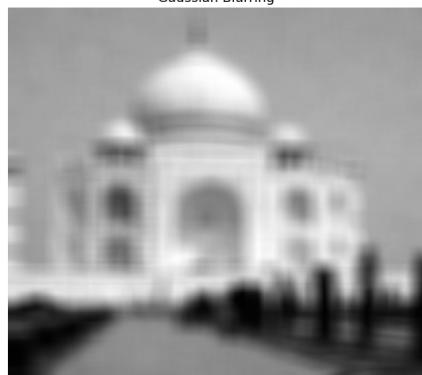
Averaging Blurring



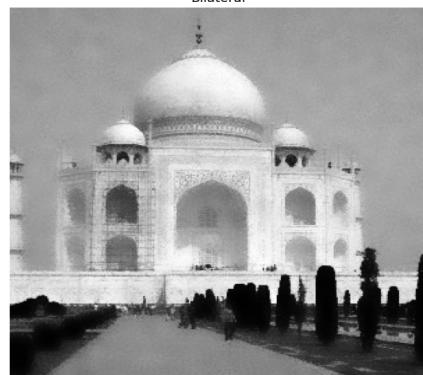
Median Blurring



Gaussian Blurring



Bilateral



منابع:

- سایت رسمی گابخانه opencv و numpy
- اسلاید ها و فیلم های کلاس درس
- فیلم های مربوط به پردازش تصویر در یوتیوب