

بسم الله الرحمن الرحيم



محمد عرفان زارع زردینی

۹۸۴۱۱۴۳۲

تمرین سری دوم درس بینایی ماشین

References:

<https://www.geeksforgeeks.org/how-to-create-a-matrix-of-random-integers-in-python/>

<https://www.javatpoint.com/python-random-randrange>

slides classes(about convoloution,kernels)

<https://www.geeksforgeeks.org/python-program-to-detect-the-edges-of-an-image-using-opencv-sobel-edge-detection/>

(الف)

ابتدا کتابخانه های مورد نیاز سوال 1 را ادد می کنیم. حال بوسیله کتابخانه هایی ادد شده، تابع تولید ماتریس که `generateRandomMatrix` است را که ورودی سایز ماتریس را می گیرید، و ماتریس $n*n$ می سازد را مینویسیم. (ورودی مقادیر، بازه 0 تا 255). در تابع `convolve` ابتدا با پدینگ با مقدار صفر می دهیم. در تابع `calc_dot_product` همانطور که از اسمش مشخصه، مناسبات ضرب نقطه ای در آن انجام می شود. سپس متغیر `cnv` که ماتریس نهایی ما در آن ذخیره می شود را تعریف کرده، سپس با حلقه تودرتو، ضرب داخلی ماتریس مان را با کرنل مناسبه و ذخیره می نماییم. سپس کرنل عمودی و افقی را با توجه به اسلاید کلاس، تعریف می نماییم.

حال با استفاده از کتابخانه افزوده شده و دستور `generateRandomMatrix`، یک ماتریس 10 در 10 با اعداد رندوم می سازیم. سپس مشتق عمودی و افقی آن را مناسبه کرده (با تابع `convolve`) و هر کدام را نمایش می دهیم. در نهایت هم سایز و جهت گرادیان را نمایش داده و با استفاده از مشتق های عمودی و افقی، مقداردهی انجام می شود و چاپ می شوند

```
for i in range(len(m)):
    for j in range(len(m[0])):
        c=pow(g_x[i,j],2)+pow(g_y[i,j],2)
        smat[i,j]=math.sqrt(c)
        direc[i,j]=math.atan2(g_x[i,j],g_y[i,j])
```

(ب)

ابتدا عکس را خوانده ، سپس کرنل گاوسی را تعریف نموده و بعد تصویر را با کرنل گاوسی کانوالو کرده و بدین گونه کمی از نویز تصویر می کاهیم.(متغیر **gau_img** سپس برای تصویر اولیه و تصویر جدید، مشتق افقی و عمودی را مناسبه می نماییم و سپس جهت و اندازه گرادیان را میاییم و چاپ می نماییم.

در نتایج آنچه مشخص هست این است که تصویر کانوالو و **smooth** شده ، لبه ها را بیشتر نمایش می دهد و خطوطی که حالت نویز داشته از تصویر حذف شده اند.

(ج)

در بخش ج، تصویر سیاه سفید را، با فیلتر گاوسی تغییر داده و با استفاده از دستور **sobel** مسیر قبلی را پیش رفته و فروبی را چاپ می نماییم.همچنین پارامتر های دستور به مانند زیر می باشند.

```
cv2.Sobel(original_image,ddepth,xorder,yorder,kernelsize)
```

به ترتیب از چپ به راست می شود:عکس اولیه، مقدار عمق تصویر فروبی(یعنی تا چه حد مشتق دقیق باشد)،برای مشتق افقی، برای مشتق عمودی(هر کدوم یک باشد مشتق آن را میگیرد)،سایز کرنلی که روی تصویر اعمال میشود.

(د)

Refrences:

https://matplotlib.org/stable/api/as_gen/matplotlib.colors.LogNorm.html

<https://numpy.org/doc/stable/reference/generated/numpy.fft.fft2.html>

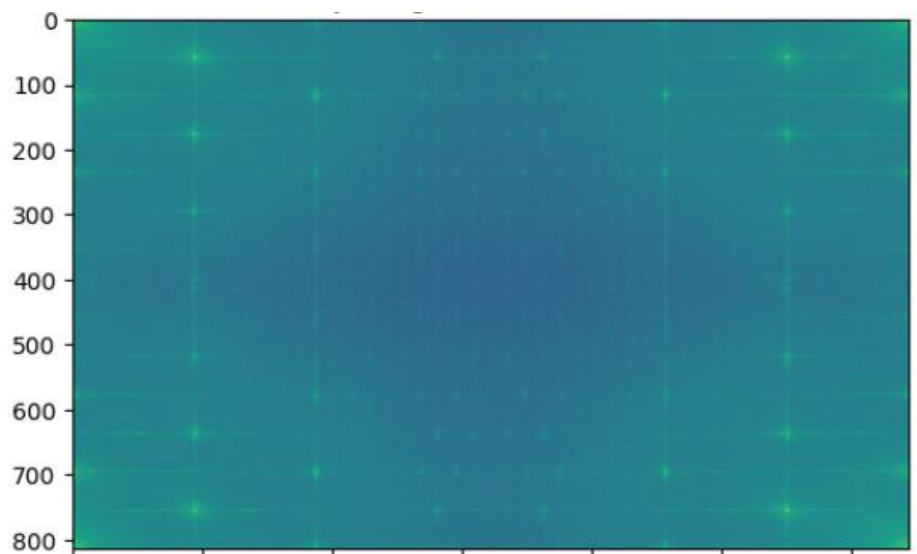
<https://numpy.org/doc/stable/reference/generated/numpy.fft.ifft2.html>

classes slides(canny code(cv_08)

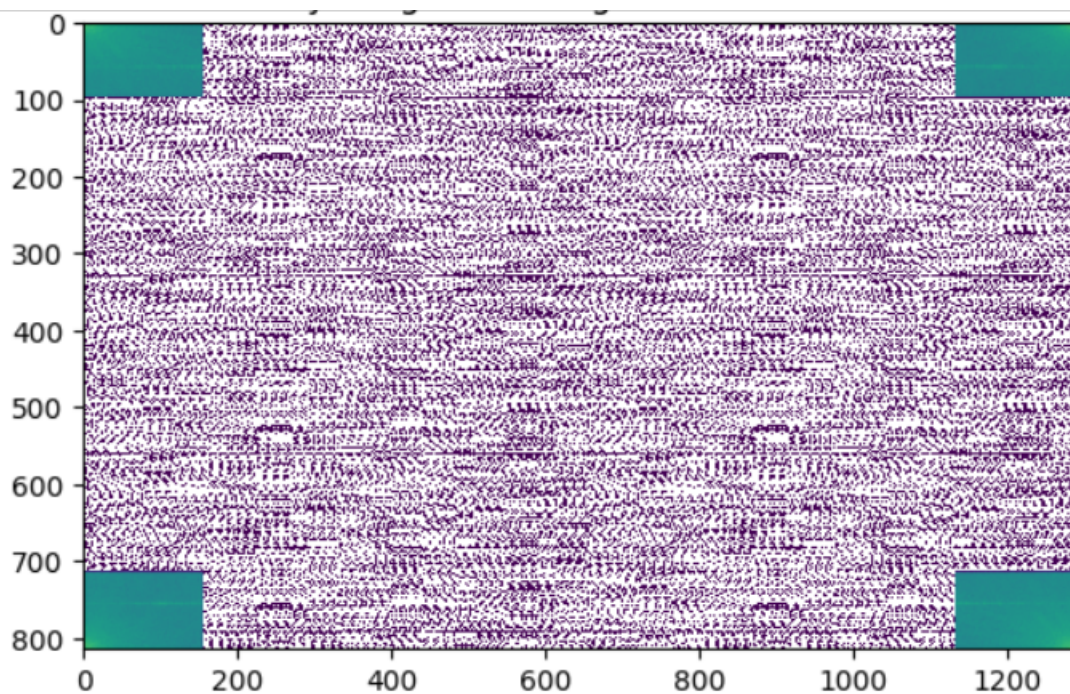
<https://learnopencv.com/contour-detection-using-opencv-python-c/>

(الف)

ابتدا با استفاده از ماژول `numpy.fft` و تابع `fft2` که تبدیل فوریه دو بعدی می باشد، تصویرمان را وارد فضای فرکانسی کرده، سپس برای نمایش و متمایز شدن نقاط خروچی ، تصویرمان را در فرمت لوگاریتمی بردیم که نتیجه تصویر حاصل در زیر است که تفاوت فرکانس های مختلف، نامیان است. سپس 76 درصد فرکانس های میانی حذف شود با توجه به اینکه ماهیت فرکانس به علت یکسان بودن 0 و 2p، حالت پرفششی دارند پس 12 درصد از بالا و پایین فرکانس ها نگه می داریم و مابقی حذف می شوند. این عمل در هر دو راستای عمودی و افقی انجام میشود تا بخش هایی با فرکانس بالا که احتمالا نمایانگر تغییرات شدید اند، حذف شود. حال با اسکیل لوگاریتمی، شدت روشنایی فرکانس های مختلف را نمایش داده که عکس شماره 2، نمایانگر آن می باشد که همانطور که مشفص است گوشه های تصویر فرکانس های نگه داشته شده اند و قسمت های حذف شده هم نمایش داده شده است .

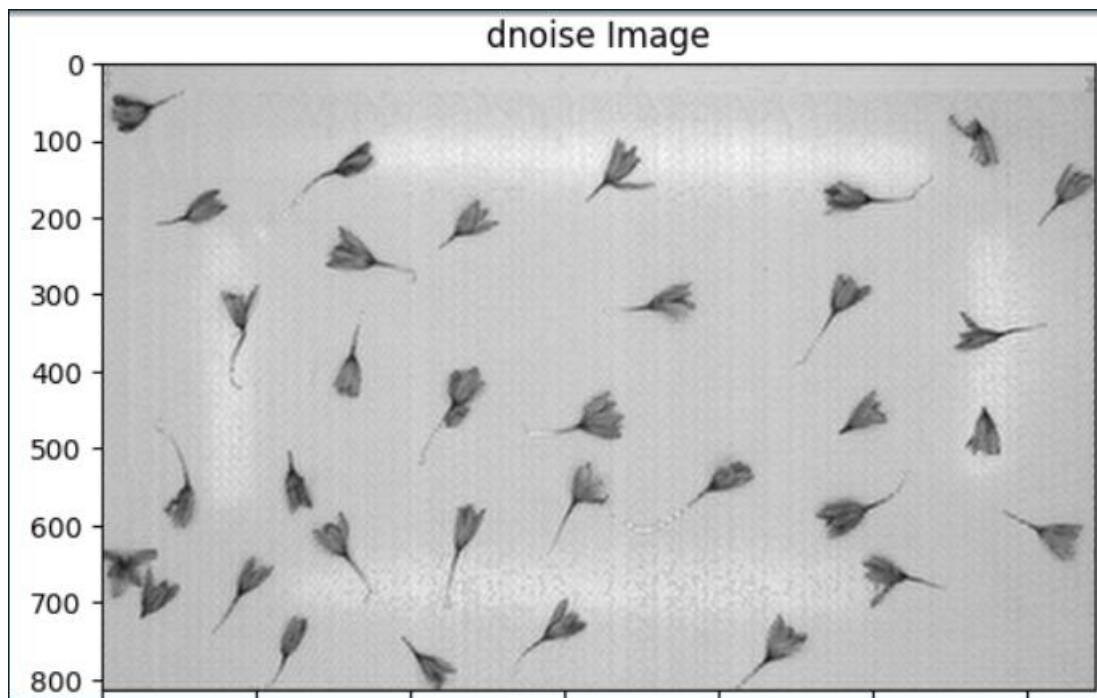


تصویر 1



تصویر 2

سپس تصاویر را با دستور برعکس اولیه، از فضای فرکانس به مکان آورده، و فروجی را چاپ می نماییم که عکس سمت چپ می شود. هر چقدر درصد فرکانس نگه داشته شده کمتر باشد، نویز پشت زعفران ها کمتر می شود ولی جزئیات تصویر هم مقدار بیشتری کاهش میابد.



(ب)

ابتدا با استفاده از لبه یاب **Canny** لبه های تصویر را میابیم. تصویر (که خروجی قسمت الف هست رو) را اگر همانگونه بدهیم ارور میخوریم ولی با تغییر فرمت آن پاس دادن به تابع، مشکل حل می شود. دو ورودی دیگر آن عدد استانه های مرحله گذاری در لبه یاب کنی اند. هر پیکسلی که اندازه گرادیان آن کوچکتر از $T1$ باشد به عنوان غیر لبه معرفی میشود، هر پیکسلی که اندازه گرادیان آن بزرگتر از $T2$ باشد به عنوان لبه معرفی میشود. پیکسلهایی که اندازه گرادیان آنها بین $T1$ و $T2$ باشد تنها در صورتی به عنوان لبه معرفی میشوند که به یک پیکسل لبه به صورت مستقیم یا از طریق پیکسلهایی که اندازه گرادیان آنها بین $T1$ و $T2$ است متصل باشند (بر اساس اسلایدها). اینجا دو مقدار 20 و 130 را به عنوان پارامتر ها با آزمون خطا در نظر گرفتیم که قواعد بالا روی آن صادق است.

(ج)

برای محاسبه گرادیان تصویر، ابتدا با استفاده از **sobel**، مشتق افقی و عمودی تصویر (خروجی مرحله قبل) را محاسبه و سپس چون فقط جهت گرادیان را میخواهد، با استفاده از **arctan2** مشتق افقی و عمودی، حاصل را میابیم و نمایش می دهیم.

(د)

برای پیدا کردن نقطه برش کافی است که به تقاطی که جهت گرادیان تغییر میکند و مثال گرادیان به سمت بالا است و نقطه ای به صورت دو شافه طور در می آید توجه کنیم. در تقاطی که جهت گرادیان به شکل ملموس نسبت به گرادیان های نزدیکش تغییر میکند، در واقع همان نقطه جدا کننده ساقه و گلبرگ زعفران می باشد. (در واقع وقتی گرادیان ها در یک توالی یکرسمت مشخص را نشان می دهند ولی در نقطه ای جهتشان به شکل ملموس متمایز می شود، نقطه مدنظر را میتوان یافت.)

(ط)

References:

(الف)

<https://shahaab-co.com/mag/edu/machine-vision/spatial-filtering-image-processing-in-matlab/#h--3>

<https://gisman.ir/low-and-high-pass-filters-samples/>

<https://srs->

[gis.ir/spectral and spatial filtering remotely sensed data/](https://gis.ir/spectral-and-spatial-filtering-remotely-sensed-data/)

<https://www.javatpoint.com/dip-high-pass-vs-low-pass-filters>

ج)

Slides classes(cv(08 ..10))

<https://farsgraphic.com/70438/article-noise-on-photo/>

[chatgpt](#)

د)

<https://farsgraphic.com/70438/article-noise-on-photo/>

https://en.wikipedia.org/wiki/Contraharmonic_mean

https://en.wikipedia.org/wiki/Salt-and-pepper_noise#:~:text=Salt%2Dand%2Dpepper%20noise%2C,occurring%20white%20and%20black%20pixels.

<https://www.frontiersin.org/articles/10.3389/fams.2022.918357/full>

(الف)

اگر میزان تغییرات نور را در نظر بگیریم، در قسمت هایی که نور به سرعت تغییر میکند، فرکانس بالا و قسمت هایی که نور با شیب کمی تغییر میکند، فرکانس پایین هست. فیلترهای پایین گذر یا همان **lpf(low pass filter)** فیلترهایی هستند که روی سیگنال اثر گذاشته و فرکانس های پایین را عبور می دهند ولی فرکانس های بالا را اجازه عبور نمی دهند و روی آن تغییر ایجاد می کنند و در واقع تغییرات با شیب آهسته(کلیات و نواحی همگن تصویر) را نگه می دارد و فقط می نماید و از تاثیر تغییرات با شیب تند را که در لبه های تصاویر و اجسام هست(جزئیات ظریف تصویر)



می‌کاهد و فرکانس‌های آن‌ها را کاهش می‌دهد. فیلتر اشاره شده برای هموارسازی و نرم کردن (smoothing) و مات کردن تصویر یا کاهش نویز بیشتر کاربرد دارد. فیلترهای میانگین، میان و مد از فیلترهای این دسته هستند. معمولاً فیلترهای پایین‌گذر فضا را با کانولوشن دوبعدی با ضرایب غیرمنفی پیاده‌سازی نمود.

فیلترهای بالا‌گذر (hpf) مقابل فیلتر پایین‌گذر است، پیکسل‌های با فرکانس بالا را عبور داده و بر روی پیکسل‌ها با فرکانس پایین، تغییر ایجاد می‌کنند و تیزتر می‌کند و در واقع تصویر حاصل جزئیات بیشتری دارد. این فیلتر همچنین اثرات پیکسل‌های نویزی نیز حفظ یا بهبود می‌دهد. اجزای فرکانس بالا شامل جزئیات، نقاط، خطوط و لبه‌هاست که در واقع این فیلتر تغییرات شدت روشنایی تصویر را برجسته می‌نماید. فیلتر مذکور، جزو فیلترهای لبه‌است. از جمله فیلترهای لبه می‌توان به laplacian, sobel اشاره کرد که در تشخیص لبه کاربرد دارد. به این فیلتر، آشکارکننده لبه نیز گویند. این فیلتر تصویر را sharp می‌کند و در واقع کنتراست تصویر را زیاد می‌نماید.

(ب)

با توجه به نکات و توضیحات بیان شده در قسمت الف، تصویر سمت راست حاصل اعمال فیلتر hpf یا همان فیلتر بالا‌گذر است که دلایل آن جزئیات تصویر و لبه‌ها آن بولد شده است و همچنین کنتراست بیشتر شده است.

(ج)

هر دو نویز مد نظر متناوب اند.

نویز جمع شونده (additive noise) نویزهایی جدا از تصویر اند و گویی به تصویر اضافه می‌شوند.

مدل نویز جمع‌شونده:

$$g(x, y) = f(x, y) + n(x, y)$$



در این جا n همان نویز ، f تصویر اولیه و g تصویر حاصل است. از جمله مثال های آن ، نویز گاوسی، نویز سفید است برای حذف نویز جمع شونده ،میتوان از تصویر نهایی فوریه گرفت و سپس نویز را از تصویر نهایی کم کرد تا برسیم به شکل اولیه. با افزایش شدت نویز، امکان تشخیص سیگنال و داده کاهش میابد. مثال هایی از آن مانند نویز هوایی، نویز تصادفی سفید و نویز امواج الکتريکی، نویز شات ، نویز کوانزاسیون.از دیگر روش ها برای حذف این نوع نویز می توان به فیلترینگ ، کاهش فرکانس و نرمال سازی و تصحیح خطا اشاره کرد که بیشتر برای کاهش نویز در فرکانس بالا مفید است.

این درحالی است که نویز های ضرب شونده به گونه ای هستند که وجود سیگنالشان، وابسته به حالت تصویر اصلی است و درواقع: نویز * تصویر اصلی = تصویر نویزی

این نوع نویز بیشتر با خطاهای اندازه گیری و نویزهای فنی مرتبط است.برای حذف این نویز،نرمال سازی، کالیبراسیون ، پردازش سیگنال و رویکرد های مبتنی بر مدل سازی کاربرد دارد که بیشتر برای کاهش نویز در فرکانس پایین و حذف تاثیر نویز روی سیگنال های پیچیده ،کاربرد دارد.

نکته مهم این است که در اغلب موارد امکان حذف کامل نویز ضرب شونده نیست و هدف تنها کاهش آن است

مثال های آن نویز **speckle** ، رگرسیون، نویز گوسی و نویز ضربه ای

حذف نویز جمع شونده به مراتب آسان تر از ضرب شونده می باشد زیرا برعسب بزرگی سیگنال یا داده تغییر میکند.

مقایسه:علاوه بر نحوه افزوده شدنشان به عکس و نکات بالا،روش حذف نویز ضرب شونده برای

حذف نویز در فرکانس پایین و درمیان سیگنال های پیچیده است. درحالی که روش های حذف نویز جمع شونده برای کاهش نویز در فرکانس های بالا کاربرد دارد. جمع شونده یک بار برداری ثابت دارد



به اندازه **amplitude** سیگنال یا داده بستگی ندارد. در حالی که نویز ضربی به طور متناسب با **amplitude** سیگنال یا داده است و با افزایش **amplitude** نویز نیز به طور متناسب زیاد می شود.

(د)

این نویز به عنوان نویز ضربه ای هم شناخته می شود. این نویز در تصاویر دیجیتال دیده می شود و می تواند به علت اختلالات شدید و ناگهانی در سیگنال تصویر، عملکرد دستگاه هنگام انتقال و دریافت تصویر ایجاد می شود. این نویز خود را به صورت پیکسل های سیاه و سفید نمایش می دهد. پیکسل های سیاه مربوط به فلفل معمولاً در نواحی روشن تر تصویر ظاهر می شوند و پیکسل های سفید که مربوط به نویز نمک اند معمولاً در نواحی تیره تصویر ظاهر می شوند و این ها به صورت تصادفی توزیع می شوند. این نویز برای استخراج لبه تصویر، ترمیم و نظایر این ها مناسب نیست. به دلیل اینکه مقدار پیکسل نقطه نویز کاملاً با مقدار پیکسل همسایه هایش متفاوت است، می توان آن را با فیلتر میانه حذف نمود. احتمال رخ دادن این نویز فقط در دو مقدار 0 یا 255 (در تصویر 8بیتی) است و یا یک سیگنال را صفر میکند و یا یک سیگنال را یک می نماید (یا کامل سیاه یا کامل سفید) و میان این دو را ندارد. همچنین مقدار نویز را با مقدار سیگنال جایگزین می نماید. همانطور که گفتیم برای حذف نویز نمک و فلفل، از فیلتر میانه استفاده می نماییم. فیلتر میانه که فیلتری پایین گذر است که توانایی زیادی در حذف چنین نویزی دارد. کار این فیلتر بدین گونه است که تمامی همسایه های یک پیکسل مرکزی را به صورت صعودی مرتب میکند و مقدار میانی را جایگزین پیکسل مرکزی می نماید. در این روش تصویر را از پنجره ای 3 در 3 عبور داده و آن عمل توزیع داده شده را اعمال می کنیم. این روش جزو **Geometric Enhancement** است که لبه ها را بارز کرده و در واقع کیفیت بصری تصویر را افزایش می دهد و نویز حذف می شود. این نویز عمدتاً در فرآیند های انتقال اطلاعات استفاده می شود.

همچنین روش کنتر هارمونی میانگین که میانگین حاصل جمع مجذور اعداد تقسیم بر میانگین حسابی اعداد تقسیم کرد.

$$C(x_1, x_2, \dots, x_n) = \frac{\frac{1}{n} (x_1^2 + x_2^2 + \dots + x_n^2)}{\frac{1}{n} (x_1 + x_2 + \dots + x_n)},$$

$$= \frac{x_1^2 + x_2^2 + \dots + x_n^2}{x_1 + x_2 + \dots + x_n}.$$

(۱۰)

References: Slides classes (cv_08)

(الف)

سؤال ۱۴

الف)

با توجه به فصول های موجود در اسلاید جلسه ۸ (cv-8) در بخش فصول تبدیل فوری به سیستم 2D داریم:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^{-j2\pi \left(\frac{ux}{M} + \frac{vy}{N} \right)}$$

نقطه‌ای
مبدأ (0,0) $\Rightarrow F(0,0) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^0 = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)$

همان تصویر اصلی مان

ب)

طبقه اولی در امتحان جلسه 8 که در بحث الف ام نوشته شده است بایستی

فقطه داشته باشد، چنانچه این برده و بررسی می کنیم، ابتدا تصویر را در مختصات $f(x,y)$

مختصات یکای ما را می بینیم

$$f(x,y) = \begin{matrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{matrix} = \begin{matrix} 1 & 0 \\ 1 & 1 \end{matrix}$$

$$\Rightarrow F(0,0) = \sum_{x=0}^1 \sum_{y=0}^1 f(x,y) e^{j0} = 1 + 0 + 1 + 1 = 3$$

$$F(0,1) = \sum_{x=0}^1 \sum_{y=0}^1 f(x,y) e^{-j\pi(\frac{0x}{2} + \frac{y}{2})}$$

$$= \sum_{x=0}^1 \sum_{y=0}^1 f(x,y) e^{-j\pi y}$$

$$= 1 + 0 + 1 + 1 e^{-j\pi} = 0$$

$$F(1,0) = \sum_{x=0}^1 \sum_{y=0}^1 f(x,y) e^{-j\pi x}$$

$$= 1 + 0 + 1 + (-1) = 1$$

$$F(1,1) = \sum_{x=0}^1 \sum_{y=0}^1 f(x,y) e^{-j\pi(x+y)}$$

$$= 1 - 1 + 1 = 1$$

د)

References:

https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html

<https://www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/>

<https://stackoverflow.com/questions/62274412/cv2-approxpolydp-cv2-arclength-how-these-works>

https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html

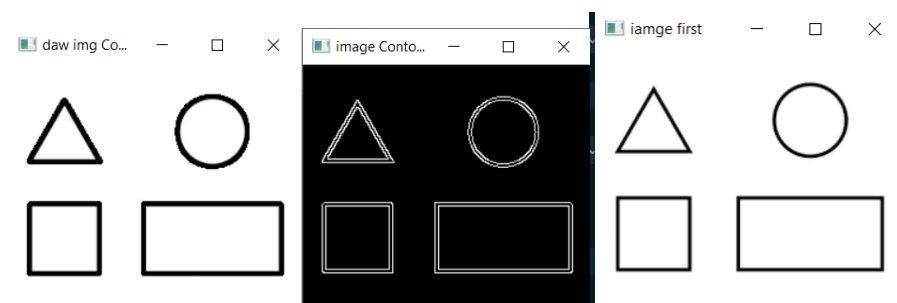
<https://www.geeksforgeeks.org/python-opencv-cv2-puttext-method/>

(الف)

ابتدا عکس را خوانده و با فرمت سیاه سفید خوانده و چاپ می نمایم .

(ب)

تابع **findconours** که از توابع کتابخانه **opencv** است ، مقتضات نقاط لبه را میابد. 3 پارامتر اصلی به عنوان ورودی دارد که به ترتیب: تصویر اولیه ، روش بازیابی لبه ، روش پیشبینی لبه است. در خروجی دستور، لبه ها و **hierarchy** داده می شود. برای پیاده سازی آن بهتره ورودی به فرمت باینری ومشتق گرفته باشد. پس ابتدا **Canny** را روی تصویر می اندازیم و سپس خروجی آن را به تابع می دهیم و نتیجه را نمایش میدیم. از راه های نمایش نتیجه میتوان به دستور **drawcontours** اشاره نمود.



(ج)

این بخش یک حلقه روی contours میزنیم و سپس برای هر شکل، با arclength ضریب اپسیلون را میابیم. سپس تابع approxPolyDP را صدا زده تا تعداد گوشه ها را بیابیم و بر آن اساس با نوشتن شرط تعیین کنیم شکل از چه نوعی است و در نهایت با دستور puttext نام شکل مورد نظر را روی تصویر بنویسیم و در نهایت چاپ می نماییم. ورودی های puttext به ترتیب: عکس اولیه، متن نوشتار، محل نوشتار، فونت نوشتار، سایز فونت، رنگ، کلفتی نوشته می باشد.

(د)

جهت دست بندی اشکال می شود از روش های ماشین لرنینگ همچون پرسپترون تک لایه استفاده نمود. بدین سان که تعدادی مدل برای هر نوع متفاوت را train میکنیم. به علت آن که روش مد نظر supervised است، پس نیاز به داده زیاد با ابعاد یکسان داریم تا آموزش انجام شود. در واقع پس از آموزش میتوان به بررسی اشکال پرداخت و کلاستر انجام داد. پس با توجه به داده های کم موجود دچار چالش می شویم و به داده های بیشتری برای بررسی آن نیازمندیم.

(ع)

References:

<https://www.geeksforgeeks.org/python-bilateral-filtering/>

<https://www.geeksforgeeks.org/python-image-blurring-using-opencv/>

https://docs.opencv.org/3.4/dc/dd3/tutorial_gaussian_median_blur_bilateral_filter.html

[https://sparkbyexamples.com/numpy/numpy-fill-function-with-examples/#:~:text=NumPy%20fill\(\)%20function%20in,position%20to%20the%20end%20position.](https://sparkbyexamples.com/numpy/numpy-fill-function-with-examples/#:~:text=NumPy%20fill()%20function%20in,position%20to%20the%20end%20position.)



الف) در تابع اول بدون استفاده از حلقه و با بازه بندی، پدینگ مورد نظر را برای هر عکس مشخص می کنیم و تصویر مورد نیاز را در تابع (**reflect101**) میسازیم. سپس در تابع بلور میانگین یا همان **Averaging_Blurring**، ابتدا تابع **reflect101** و **result** را تعریف کردیم. سپس در حلقه با استفاده از دستور میانگین، فیلتر مدنظر را پیاده می کنیم. تابع **median** هم بدین گونه است و تنها تفاوت آن در حلقه و فراخوانی تابع است. در تابع گاوسی، ابتدا یک آرایه از کرنل تعریف میکنیم. سپس با توجه به فرمول گاوسی به مناسبه آن میپردازیم. **std** یک ضریب در فرمول است.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

در این فیلتر ها، هرچه اندازه فیلتر بیشتر باشد، فروبی بیشتر تار هست و برعکس.


ب)

فیلاتر **bilateral**، فیلتر غیر خطی است که مقداری از بلور شدن لبه ها جلوگیری می کند. فیلتر مدنظر برای کاهش نویز است و هر پیکسل با یک کرنل میانگین گیر وزن دار مناسبه می شود که براساس توزیع گاوسی دارد. در این روش در تشکیل رابطه اصلی از ایده گاوسی استفاده می شود ولی بدین گونه است که به طوری پیاده سازی می شود که به حفظ لبه ها کمک شود. رابطه گاوسی یکبار برای فاصله با نقطه اصلی و یکبار برای تفاوت رنگ پیکسل ها

رابطه **bilateral** به شکل زیر است:

Bilateral Filter: an Additional Edge Term

The bilateral filter can be formulated as follows:

$$BF[I]_p = \underbrace{\frac{1}{W_p}}_{\text{Normalization Factor}} \sum_{q \in S} \underbrace{G_{\sigma_s}(\|p - q\|)}_{\text{Space Weight}} \underbrace{G_{\sigma_r}(|I_p - I_q|)}_{\text{Range Weight}} I_q$$


(تصویر از [geeksforgeeks](https://www.geeksforgeeks.org/bilateral-filter/))

فیلتر در کد زده شده که از سایت ها و منابع متفاوت در الگوریتم آن استفاده شده، به همین سبب فرمت و ساختار ورودی های آن براساس فرمتی است که به ذهنم رسید و به نتیجه رسیدم.

ابتدا ابعاد پنجره و کرنل گاوسی را می سازیم. سپس کرنل گاوسی را مقدار می دهیم. سپس در حلقه تودرتو، کرنل گاوسی برای دامنه رنگ ها را می سازیم. سپس کرنلی حاصل از ضرب دو کرنل ساخته و نرمالایز می نماییم و بر آن اساس تصویر نهایی را آپدیت می نماییم تا به نتیجه حاصل برسیم. در واقع در این کد براساس فرمول و قواعد این فیلتر، عناصر را در نظر گرفته و آپدیت می کنیم.

نکته: تصویر این قسمت باتوجه به چاپ با فرمت **cv2**، هنگام ران کردن موجود است. فقط با توجه به اینکه فرمت ورودی آن با فرمت تابع آماده متفاوت است، تصویر با آن مقادیر تابع آماده، عکسی یکسان نسافته.

ج)

در این قسمت دستورات نوشته شده در دو قسمت قبل را به کمک توابع آماده نمایش می دهیم.
مشاهده می شود نتایج حاصل از محاسبات دو قسمت قبل ، با این قسمت ، یکسان می شود.