Computer vision

Homework 5

Parsa Eissazadeh 97412364

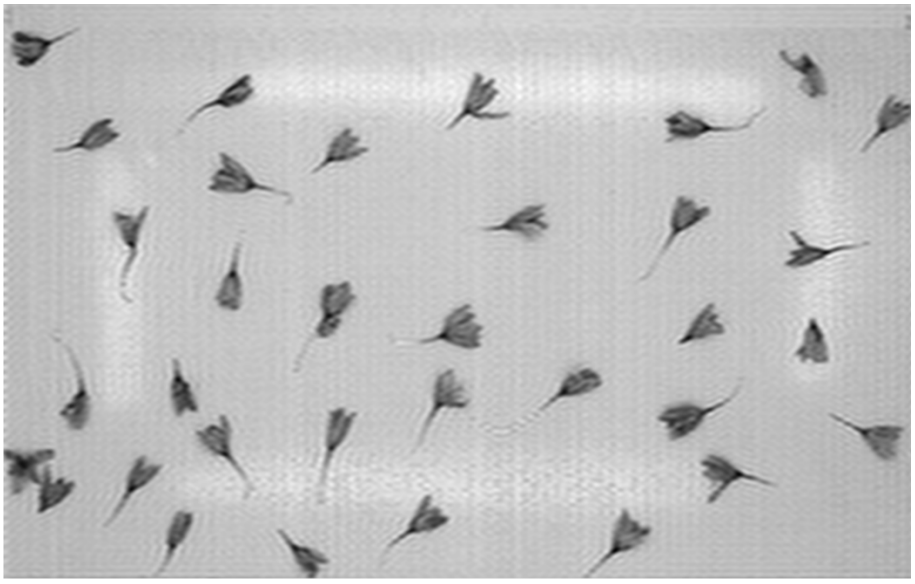
سوال 1

همانند تمرین قبل که رفع نویز کردیم ابتدا تبدیل فوریه را میگیریم و شیفت می دهیم و سپس دستی ماسکش میکنیم :

به خاطر اینکه shape عکس (815,1290) بود ، ابتدا اعداد زیر را انتخاب کردیم :

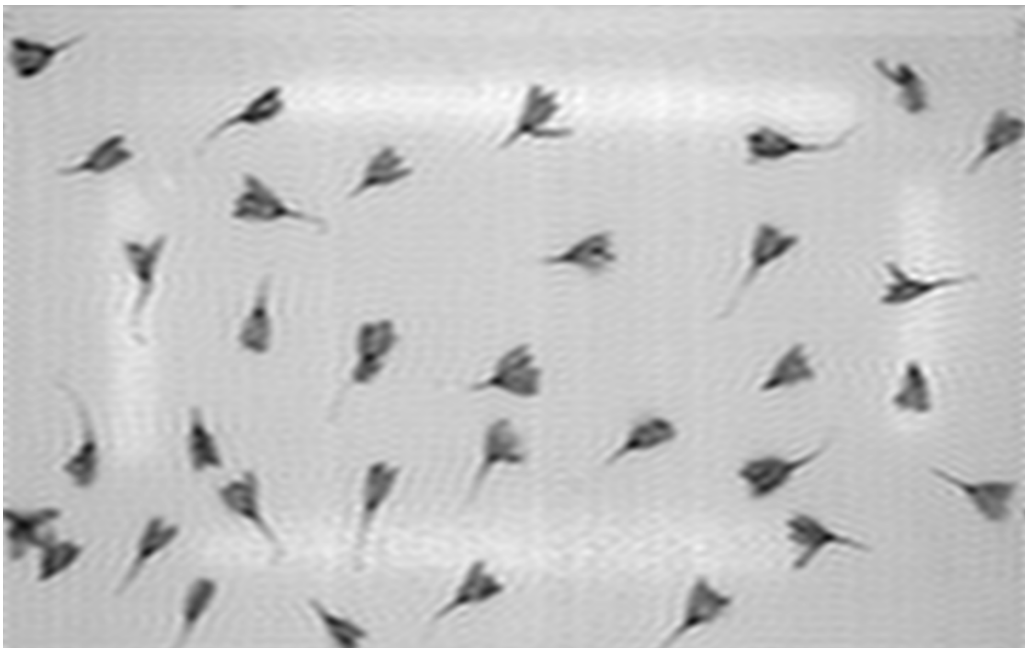
```
dft_shifted[:,735:] = 0
dft_shifted[:,:555] = 0
dft_shifted[497:,:] = 0
dft_shifted[:317,:] = 0
```

مختصات مرکز عکس : (407,645) در ابتدا تا شعاع 90 پیکسلی را نگه داشتم بقیه را صفر کردم . خروجی :



نتیجه نسبتا خوب شد و روزنه ها حذف شدند . حال هر بار اندکی شعاع را کمتر میکنم تا به نتیجه بهتری برسم .هر چه شعاع کمتر باشد عکس تار تر می شود ، در نتیجه نویز ها حذف می شوند ولی در عوض کیفیت از دست می رود .

با صحیح و خطا به عکس زیر رسیدم (شعاع 60) :



لبه یاب Canny

از متد زیر استفاده میکنیم :

```
edges = cv2.Canny(im,200,250)
```

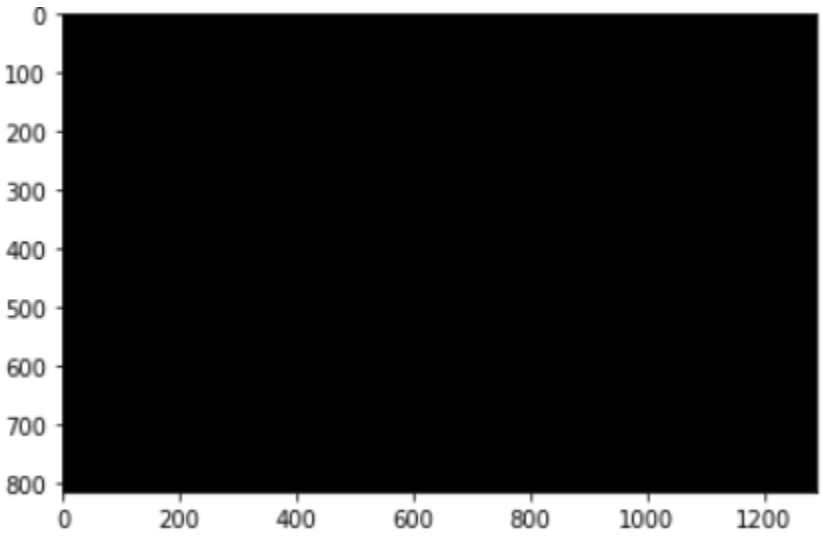
این متد 3 پارامتر میپذیرد که دو پارامتر آخر آستانه های دو مرحله ای ما هستند . این آستانه ها با مقدار گرادیان مقایسه می شوند .

از آنجایی که اختلاف روشنایی دو پیکسل مجاور حداکثر 255 است (یکی سفید باشد که می شود 255 و دیگری سیاه باشد که می شود 0) ، مشتق هم حداکثر می شود 255 .

همچنین از آنجایی که در canny برای محاسبه گرادیان از sobel استفاده می شود ، گرادیان ها چیزی حدود 8 برابر حالت عادی (بدون استفاده از sobel و فیلتر مشتق گیر) هستند .

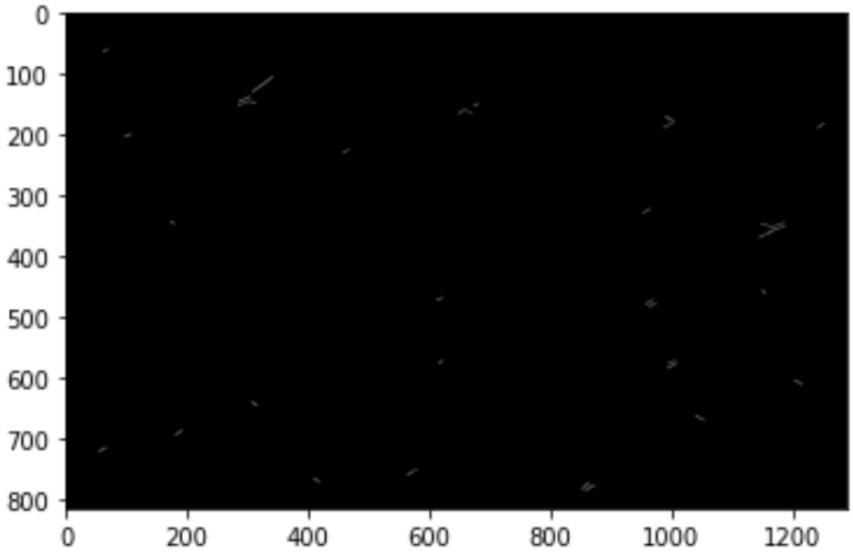
در نتیجه این پارامتر ها حداکثر مقداری حدود 2000 می توانند داشته باشند .

بعد از اینکه متد canny را فراخوانی کردیم ، نتیجه به شکل زیر شد :

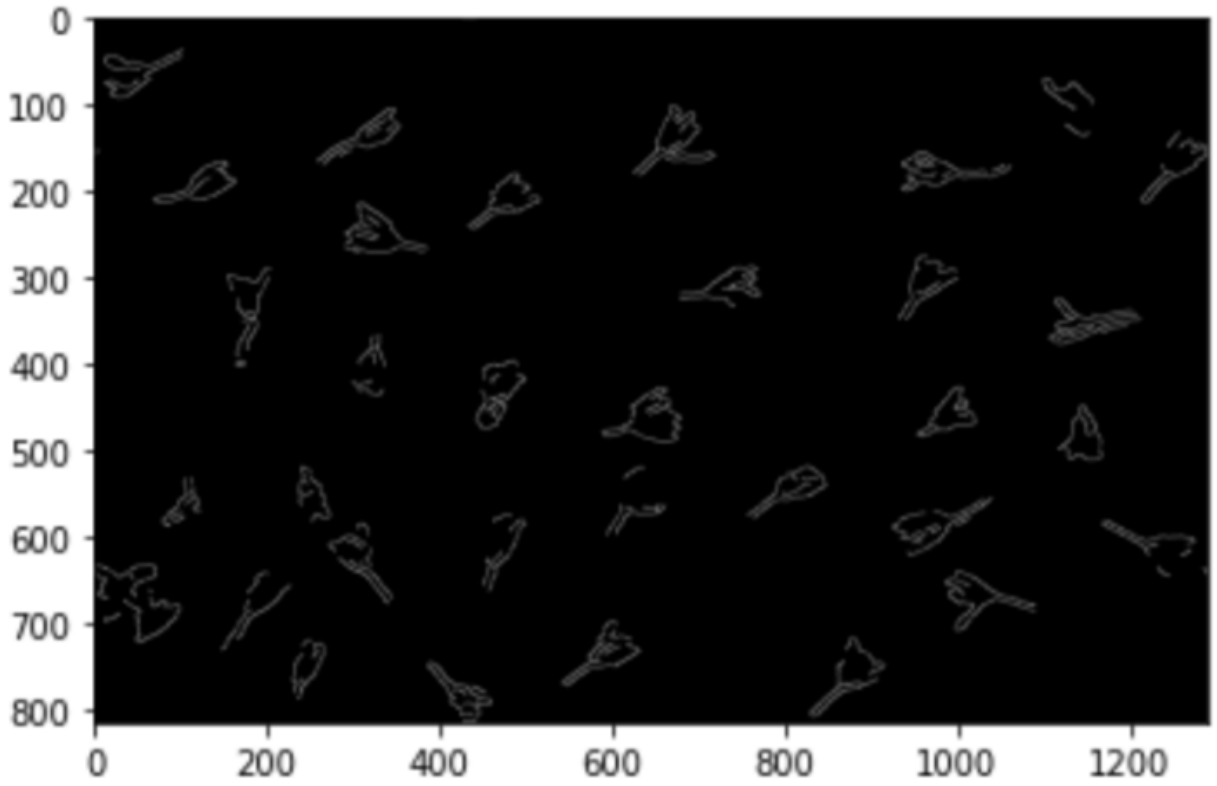


ظاهرا لبه ها خیلی نا واضح شدند . به حالت اول (شعاع 90 باز میگردم) :

همچنان ناواضح :



تصویر همچنان واضح نشد برای همین پارامتر های متد کنی را عوض کردم و نتیجه بهتر شد :



نتیجه گیری : هر چه شعاع حذف کننده روی تبدیل فوریه کمتر باشد کیفیت بیشتری از دست می رود ، در نتیجه لبه ها ضعیف تر می شوند . برای بازیابی لبه ها در این وضعیت ، باید آستانه های متد کنی را پایین تر در نظر بگیریم .

برای محاسبه گرادیان از یک فیلتر مشتق گیر استفاده می کنیم . بسته به اینکه مشتق در جهت y است یا در جهت x ، فیلتر تغییر می کند . برای گرادیان در دو جهت x و y گرادیان میگیریم . برای گرادیان از فیلتر سوبل استفاده میکنیم .
سپس کانوالو میکنیم .

برای convolve از متد آماده ی filter2D استفاده کردم .
مرجع : https://learnopencv.com/image-filtering-using-convolution-in-opencv/

کد و نتیجه به شکل زیر شد :

```
def get_gradient_tetha(image):
  Ix = cv2.filter2D(image, -1, derivative_kernel_x_axis)
  Iy = cv2.filter2D(image, -1, derivative_kernel_y_axis)

  arctan = np.arctan2(Iy, Ix)

  return arctan
```

```
get_gradient_tetha(im)
```

```
array([[0.    , 0.    , 0.    , ..., 0.    , 0.    , 0.    ],
       [1.57  , 1.57  , 1.57  , ..., 0.    , 0.    , 0.    ],
       [1.57  , 1.57  , 1.57  , ..., 0.6577, 0.4216, 1.57  ],
       ...,
       [0.    , 0.    , 1.57  , ..., 0.    , 0.    , 0.    ],
       [1.57  , 1.57  , 1.57  , ..., 0.1517, 0.0813, 1.57  ],
       [0.    , 0.    , 0.    , ..., 0.    , 0.    , 0.    ]],
      dtype=float16)
```

الان اعداد بر حسب رادیان هستند و غیر قابل لمس . برای همین از متد آماده rad2deg استفاده میکنیم .

مرجع : https://numpy.org/doc/stable/reference/generated/numpy.rad2deg.html

حال جهت گرادیان در هر پیکسل ملموس تر است :

```
[111] def get_gradient_tetha(image):
        Ix = cv2.filter2D(image, -1, derivative_kernel_x_axis)
        Iy = cv2.filter2D(image, -1, derivative_kernel_y_axis)

        arctan = np.arctan2(Iy, Ix)
        theta = np.array([np.rad2deg(x) for x in arctan])

        return theta
```

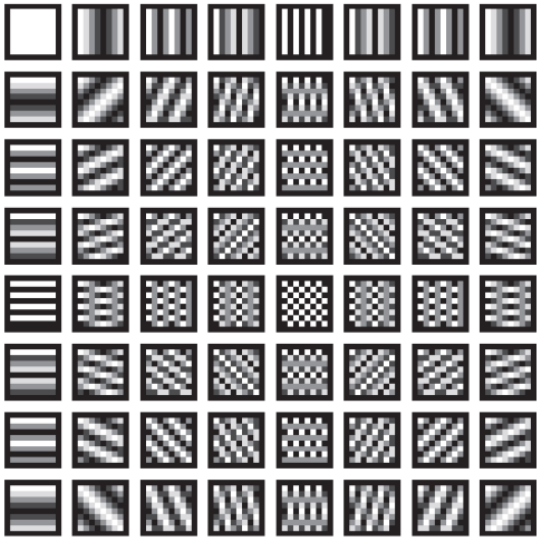
```
get_gradient_tetha(im)
```

```
array([[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
       [90.   , 90.   , 90.   , ...,  0.   ,  0.   ,  0.   ],
       [90.   , 90.   , 90.   , ..., 37.7  , 24.16 , 90.   ],
       ...,
       [ 0.   ,  0.   , 90.   , ...,  0.   ,  0.   ,  0.   ],
       [90.   , 90.   , 90.   , ...,  8.695,  4.656, 90.   ],
       [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]],
      dtype=float16)
```

سوال 2

هر تبدیل بازنمایی ای جدید از یک عکس دارد . گاهی عکس از در کنار هم گذاشتن اعداد ساخته می شوند (gray scale) و گاهی از کنار هم گذاشتن سه تایی ای از اعداد (عکس های RGB) .
تبدیل فوریه میگفت که به جای اینکه به هر پیکسل یک عدد اختصاص دهیم ، بگوییم عکس (کل پیکسل ها) چه قدر الگوی تغییر افقی ، عمودی یا هر دو را دارد .

آن الگو های بدین شکل هستند :



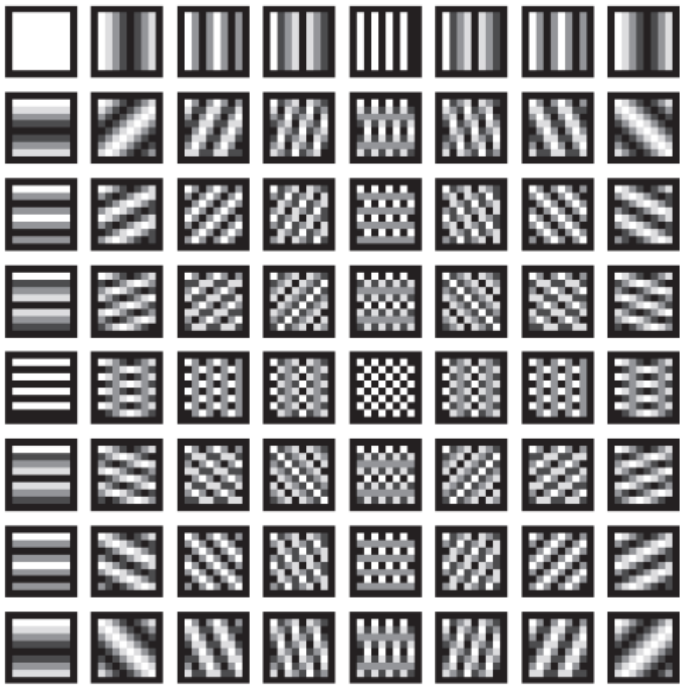
هر کدام از این خانه ها عکس هایی هستند به اندازه تصویر اصلی ما . برای کل این عکس ها به جای x و y ، پارامتر های u و v را داریم .
اگر عکسی تبدیل فوریه اش شبیه یکی از عکس های موجود در بالا باشد ، مثلا خانه ی 5 ام از بالا و 7 ام از چپ ، آن گاه مقدار تبدیل در u = 7 و v = 5 برابر 1 و در بقیه مکان ها برابر 0 است .

تنها در نقطه ی مثلا 31 و 4 (مختصات) تابع مقدار غیر 0 دارد . پس تبدیل برابر است با :

$$e^{-j2*\ pi\ *\ (\ 4\ *\ u\ +\ 31\ *\ v\ )\ /\ 32}$$

$$= e^{-j*\ pi\ *\ (\ 4\ *\ u\ )\ /16}\ *\ e^{-j*pi*\ (\ 31\ *\ v\ )\ /\ 16}$$

هر کدام از operand های ضرب توابعی سینوسی هستند ، در یکی x داریم و در دیگری y . تابع سینوسی که درونش x دارد ، در جهت x ها مقدارش به شکل متناوب ، کم و زیاد می شود و همینطور تابعی که y دارد . طبق این معادله در هر راستا تنها یک تابع سینوسی داریم در نتیجه تنها یک الگو متناوب در هر راستا قابل مشاهده است . در نتیجه اگر یک پیکسل در یک عکس روشن باشد ، تبدیل فوریه اش یکی اش خانه های موجود در این عکس است :



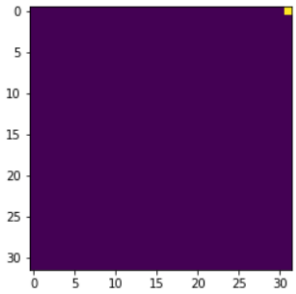
در ورودی یک تابع سینوسی عددی که در پی ضرب می شود بزرگتر باشد ، دوره تناوب کمتر می شود . نقطه روشن در عکس سوال ، x بسیار کم و y بیشینه دارد . در نتیجه در راستای x ، دوره تناوب زیاد و در راستای y دوره تناوب کم است . در نتیجه خط های موازی میبینیم که در جهت x بسیار تکرار شده اند و در جهت y بسیار کم .

در یک نقطه راست تر که مختصات ۳۱ و ۵ است ، y تغییر نکرده ولی x تغییر کرده . در نتیجه الگوی متناوب در راستای y ها همچنان تغییر نکرده و در راستای x اندکی دوره تناوب کاهش یافته در نتیجه بالا پایین های مقادیر روشنایی بیشتر می شوند .

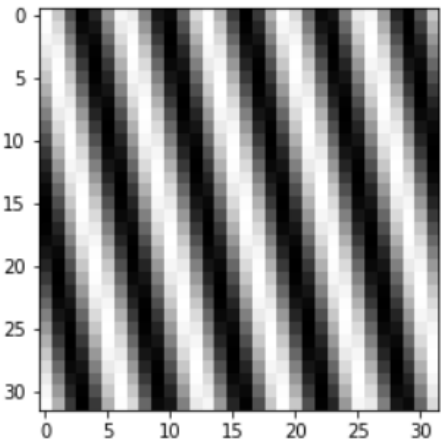
صحت سنجی : در google colab تصویر را ساختم و تبدیل فوریه اش را گرفتم :

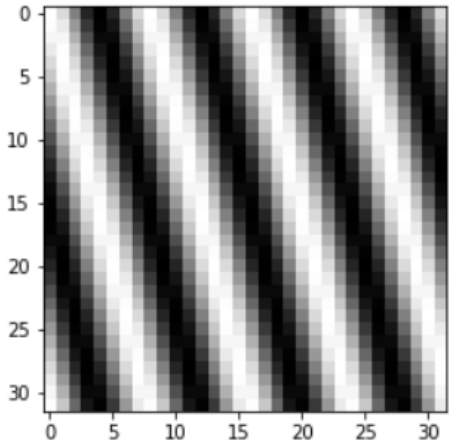
```
# constructing image
picture = np.zeros((32,32))
picture[31][4] = 1
plt.imshow(picture)
```

```
<matplotlib.image.AxesImage at 0x7ff38c0a0590>
```



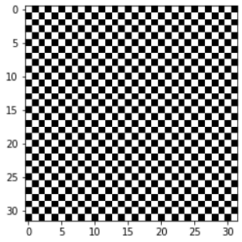
و تبدیل فوریه را در دو حالت حساب کردم (سمت راستی عکس اول ، سمت چپی عکس دوم) :





در سمت راستی وقتی از چپ به راست میرویم به 4 خط سیاه بر میخوریم و در دیگری به 5 تا . پس دوره تناوب بیشتر شده است .

هر چی آن تک نقطه روشن عکس را به وسط تر می بریم ، بیشتر به این شکل نزدیک می شویم :



که در مرکز الگوهای افقی و عمودی بود .

سوال 3

یک ماتریس میسازیم :

0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

یک لبه عمودی دارد . برای اعمال sobel ، ابتدا فیلتر های مشتق افقی و عمودی را می سازیم :

فیلتر مشتق افقی :

-1	0	1
-2	0	2
-1	0	1

فیلتر مشتق عمودی :

-1	-2	-1
0	0	0
1	2	1

حال باید این ها را در هر پیکسل کانوالو کنیم . در ابتدا padding اضافه میکنیم . از آنجایی که اندازه فیلتر 3 در 3 است تنها به اندازه یک پیکسل padding اضافه میکنیم . مقدار روشنایی این پیکسل ها را فعلا با شیوه reflect مقدار دهی میکنیم . در نتیجه بعد از اضافه کردن padding ، عکس به شکل زیر میشود :

0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0

نتیجه کانوالو فیلتر مشتق افقی :

0	4	0	-4	0
0	4	0	-4	0
0	4	0	-4	0
0	4	0	-4	0
0	4	0	-4	0

نتیجه کانوالو فیلتر مشتق عمودی :

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

دلیل صفر شدن : در راستای عمودی تغییری نداریم ، مشتق کل 0 است .

بعد از اعمال فیلتر ها ، اندازه برآیند این ها را حساب میکنیم . از آنجایی که مشتق در راستای عمودی صفر شد اندازه نتیجه sobel برابر است با مشتق در راستای افقی که برابر است با :

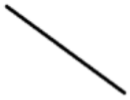
0	4	0	-4	0
0	4	0	-4	0
0	4	0	-4	0
0	4	0	-4	0
0	4	0	-4	0

سوال 4

در ابتدا عکس را لود میکنیم :

```
im = cv2.imread('/content/img_02.jpg', cv2.IMREAD_GRAYSCALE)
plt.imshow(im , cmap='gray')
plt.axis('off')
```

```
(-0.5, 268.5, 411.5, -0.5)
```



ابتدا باید پیکسل هایی که مقادیر زیر 255 دارند (سفید نیستند) را استخراج کنیم . از numpy.where استفاده می کنیم .

```
xs , ys = np.where(im < 255)
```

این تابع به ما دو آرایه باز میگرداند که هر کدام مختصات پیکسل منطبق با شرایط را در یک محور نشان می دهد.

سپس به کمک np.average میانگین های مورد نیاز را حساب میکنیم :

```
x_avg = np.average(xs)
y_avg = np.average(ys)
xy_avg = np.average(xs*ys)
xx_avg = np.average(xs**2)
```

و در نهایت طبق معادله زیر به کمک آن ها m و c را حساب می کنیم :

$$m = \frac{\bar{x}\bar{y} - \overline{xy}}{\bar{x}^2 - \overline{x^2}}$$

$$c = \bar{y} - m\bar{x}$$

```
# compute m
m = (x_avg*y_avg - xy_avg) / (x_avg**2 - xx_avg)
m
```

```
1.3709995907272559
```

```
[26] # compute c
c = y_avg - m * x_avg
c
```

```
-202.96710625458064
```

همانطور که مشاهده میشود مقادیر به صورت زیر هستند :

m= 1.37

c= -202.96

این مقادیر معادله خط را شکل می دهند .