

Gradient of Image

import library

```
In [ ]: import numpy as np  
import matplotlib.image as img  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import matplotlib.colors as colors
```

load input image ('test.jpeg')

```
In [ ]: I0 = img.imread('test.jpeg')
```

check the size of the input image

```
In [ ]: # ++++++  
# complete the blanks  
#  
num_row      = I0.shape[0]  
num_column   = I0.shape[1]  
num_channel  = I0.shape[2]  
#  
# ++++++  
  
print('number of rows of I0 = ', num_row)  
print('number of columns of I0 = ', num_column)  
print('number of channels of I0 = ', num_channel)
```

```
number of rows of I0 = 510  
number of columns of I0 = 512  
number of channels of I0 = 3
```

convert the color image into a grey image

- take the average of the input image with 3 channels with respect to the channels into an image with 1 channel

```
In [ ]: # ++++++  
# complete the blanks  
#  
  
I = I0.mean(axis= 2)  
  
# ++++++  
  
num_row      = I.shape[0]  
num_column   = I.shape[1]  
#  
# ++++++  
  
print('number of rows of I = ', num_row)  
print('number of columns of I = ', num_column)
```

```
number of rows of I = 510
number of columns of I = 512
```

normalize the converted image

- normalize the converted grey scale image so that its maximum value is 1 and its minimum value is 0

```
In [ ]: # ++++++
# complete the blanks
#
I = ( (I - np.min(I)) / (np.max(I) - np.min(I)) )
#
# ++++++
print('maximum value of I = ', np.max(I))
print('minimum value of I = ', np.min(I))

maximum value of I = 1.0
minimum value of I = 0.0
```

define a function to compute the derivative of input matrix in x(row)-direction

- forward difference : $I[x+1, y] - I[x, y]$

```
In [ ]: def compute_derivative_x_forward(I):
    D = np.zeros(I.shape)

    # ++++++
    # complete the blanks
    #

    xForward = I[1:, :]
    xForward = np.insert(xForward, -1, xForward[-1, :], axis=0)
    D = xForward - I

    #
    # ++++++
    #

    return D
```

- backward difference : $I[x, y] - I[x-1, y]$

```
In [ ]: def compute_derivative_x_backward(I):
    D = np.zeros(I.shape)

    # ++++++
    # complete the blanks
    #

    xBackward = I[:-1, :]
    xBackward = np.insert(xBackward, 0, xBackward[0, :], axis=0)
```

```

D = I - xForward

#
# ++++++
# complete the blanks
# 

return D

```

- central difference : $\frac{1}{2}(I[x+1,y] - I[x-1,y])$

```
In [ ]: def compute_derivative_x_central(I):
    D = np.zeros(I.shape)

    #
    # ++++++
    # complete the blanks
    #

    xForward = I[1:, :]
    xForward = np.insert(xForward, -1, xForward[-1,:], axis=0)

    xBackward = I[:-1, :]
    xBackward = np.insert(xBackward, 0, xBackward[0,:], axis=0)

    D = (xForward - xBackward) / 2

    #
    # ++++++
    #

    return D
```

define a function to compute the derivative of input matrix in y(column)-direction

- forward difference : $I[x,y+1] - I[x,y]$

```
In [ ]: def compute_derivative_y_forward(I):
    D = np.zeros(I.shape)

    #
    # ++++++
    # complete the blanks
    #

    yForward = I[:, 1:]
    yForward = np.insert(yForward, -1, yForward[:, -1], axis=1)
    D = yForward - I

    #
    # ++++++
    #

    return D
```

- backward difference : $I[x,y] - I[x,y-1]$

```
In [ ]: def compute_derivative_y_backward(I):
    D = np.zeros(I.shape)
```

```

# ++++++
# complete the blanks
#
yBackward = I[:, :-1]
yBackward = np.insert(yBackward, 0, yBackward[:, 0], axis=1)
D = I - yBackward

#
# ++++++
return D

```

- central difference : $\frac{1}{2}(I[x, y+1] - I[x, y-1])$

```

In [ ]: def compute_derivative_y_central(I):
    D = np.zeros(I.shape)

    # ++++++
    # complete the blanks
    #

    yForward = I[:, 1:]
    yForward = np.insert(yForward, -1, yForward[:, -1], axis=1)

    yBackward = I[:, :-1]
    yBackward = np.insert(yBackward, 0, yBackward[:, 0], axis=1)

    D = (yForward - yBackward) / 2

    #
    # ++++++
    return D

```

compute the norm of the gradient of the input image

- L_2^2 -norm of the gradient $\left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$ is defined by $\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2$

```

In [ ]: def compute_norm_gradient_central(I):
    norm_gradient = np.zeros(I.shape)

    # ++++++
    # complete the blanks
    #

    gradientX = compute_derivative_x_central(I)
    gradientY = compute_derivative_y_central(I)

    norm_gradient = gradientX**2 + gradientY**2

    #
    # ++++++

```

```
    return norm_gradient
```

functions for presenting the results

```
In [ ]: def function_result_01():

    plt.figure(figsize=(8,6))
    plt.imshow(l0)
    plt.show()
```

```
In [ ]: def function_result_02():

    plt.figure(figsize=(8,6))
    plt.imshow(l, cmap='gray', vmin=0, vmax=1, interpolation='none')
    plt.show()
```

```
In [ ]: def function_result_03():

    D = compute_derivative_x_forward(l)

    plt.figure(figsize=(8,6))
    plt.imshow(D, cmap='gray')
    plt.show()
```

```
In [ ]: def function_result_04():

    D = compute_derivative_x_backward(l)

    plt.figure(figsize=(8,6))
    plt.imshow(D, cmap='gray')
    plt.show()
```

```
In [ ]: def function_result_05():

    D = compute_derivative_x_central(l)

    plt.figure(figsize=(8,6))
    plt.imshow(D, cmap='gray')
    plt.show()
```

```
In [ ]: def function_result_06():

    D = compute_derivative_y_forward(l)

    plt.figure(figsize=(8,6))
    plt.imshow(D, cmap='gray')
    plt.show()
```

```
In [ ]: def function_result_07():

    D = compute_derivative_y_backward(l)
```

```
plt.figure(figsize=(8,6))
plt.imshow(D, cmap='gray')
plt.show()
```

```
In [ ]: def function_result_08():

    D = compute_derivative_y_central(l)

    plt.figure(figsize=(8,6))
    plt.imshow(D, cmap='gray')
    plt.show()
```

```
In [ ]: def function_result_09():

    D = compute_norm_gradient_central(l)

    plt.figure(figsize=(8,6))
    plt.imshow(D, cmap='gray')
    plt.show()
```

```
In [ ]: def function_result_10():

    D = compute_norm_gradient_central(l)

    plt.figure(figsize=(8,6))
    im = plt.imshow(D, cmap=cm.jet, norm=colors.LogNorm())
    plt.colorbar(im)
    plt.show()
```

```
In [ ]: def function_result_11():

    D = compute_derivative_x_forward(l)

    value1 = D[0, 0]
    value2 = D[-1, -1]
    value3 = D[100, 100]
    value4 = D[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

```
In [ ]: def function_result_12():

    D = compute_derivative_x_backward(l)

    value1 = D[0, 0]
    value2 = D[-1, -1]
    value3 = D[100, 100]
    value4 = D[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

```
In [ ]: def function_result_13():

    D = compute_derivative_x_central(l)
```

```
value1 = D[0, 0]
value2 = D[-1, -1]
value3 = D[100, 100]
value4 = D[200, 200]

print('value1 = ', value1)
print('value2 = ', value2)
print('value3 = ', value3)
print('value4 = ', value4)
```

```
In [ ]: def function_result_14():

    D = compute_derivative_y_forward(l)

    value1 = D[0, 0]
    value2 = D[-1, -1]
    value3 = D[100, 100]
    value4 = D[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

```
In [ ]: def function_result_15():

    D = compute_derivative_y_backward(l)

    value1 = D[0, 0]
    value2 = D[-1, -1]
    value3 = D[100, 100]
    value4 = D[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

```
In [ ]: def function_result_16():

    D = compute_derivative_y_central(l)

    value1 = D[0, 0]
    value2 = D[-1, -1]
    value3 = D[100, 100]
    value4 = D[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

```
In [ ]: def function_result_17():

    D = compute_norm_gradient_central(l)

    value1 = D[0, 0]
    value2 = D[-1, -1]
    value3 = D[100, 100]
    value4 = D[200, 200]

    print('value1 = ', value1)
```

```
print('value2 = ', value2)
print('value3 = ', value3)
print('value4 = ', value4)
```

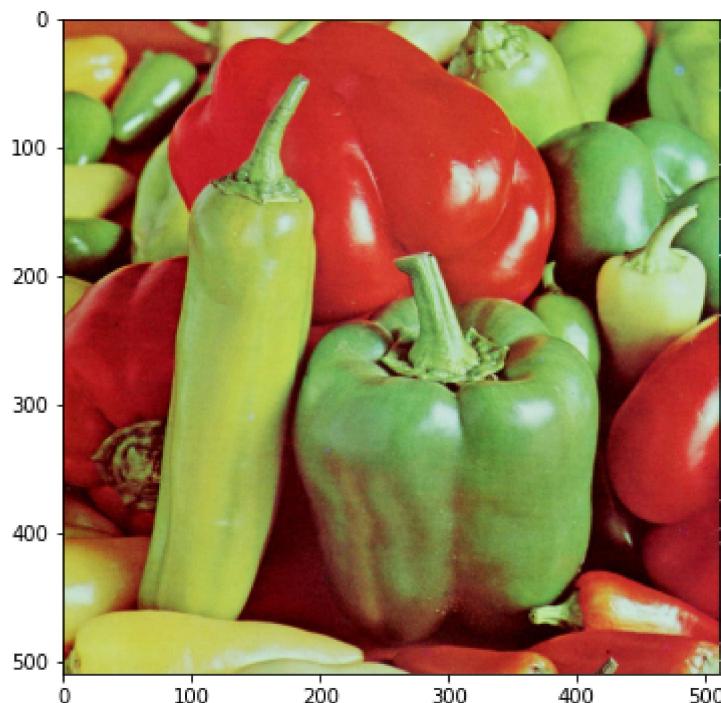
results

```
In [ ]: number_result = 17

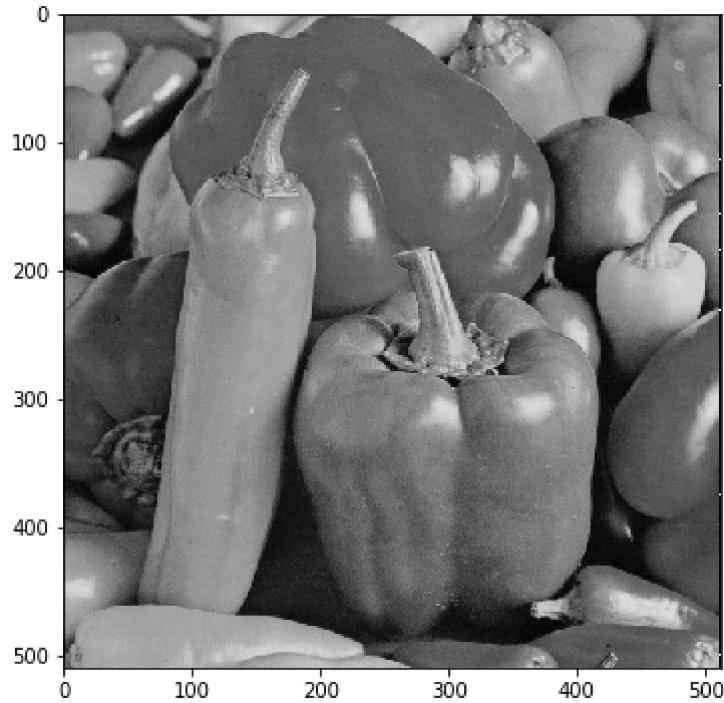
for i in range(number_result):
    title = '## [RESULT {:02d}]'.format(i+1)
    name_function = 'function_result_{:02d}{}'.format(i+1)

    print('*'*50)
    print(title)
    print('*'*50)
    eval(name_function)
```

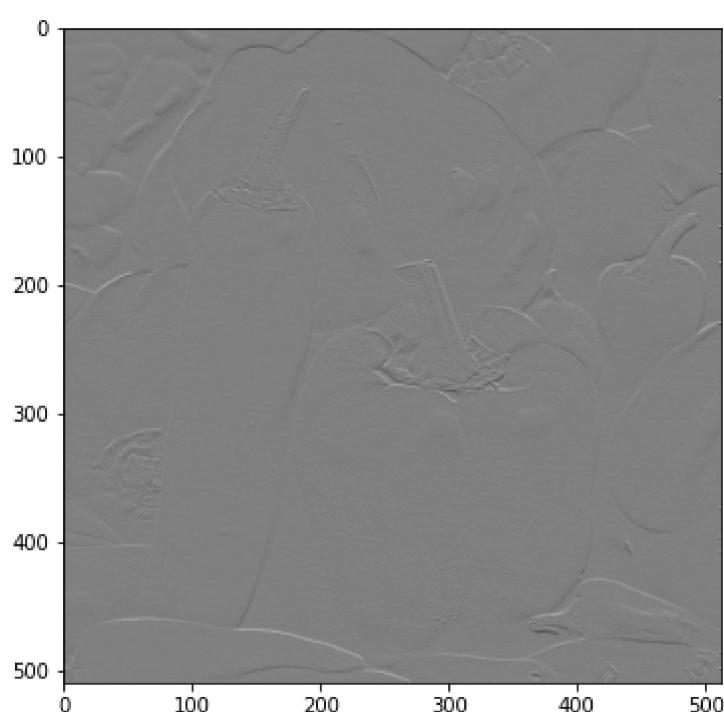
```
*****
## [RESULT 01]
*****
```



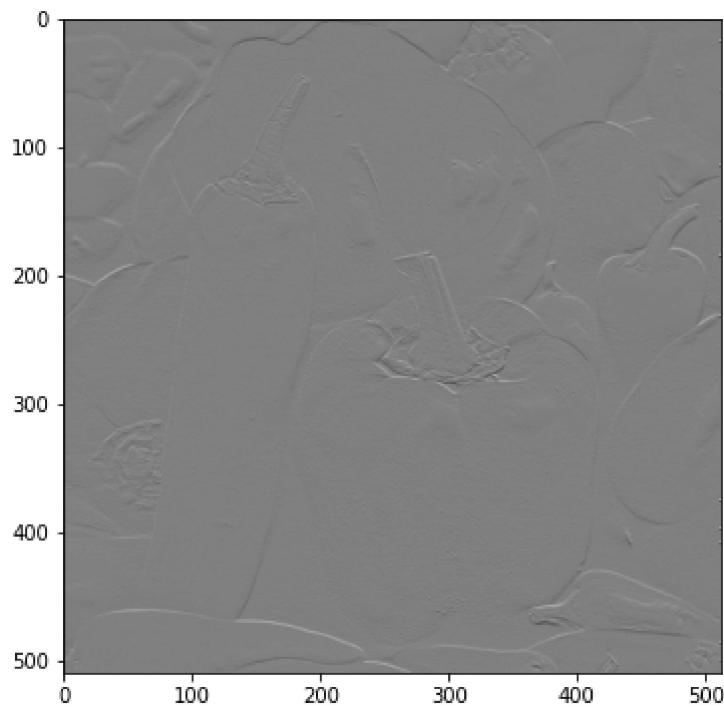
```
*****
## [RESULT 02]
*****
```



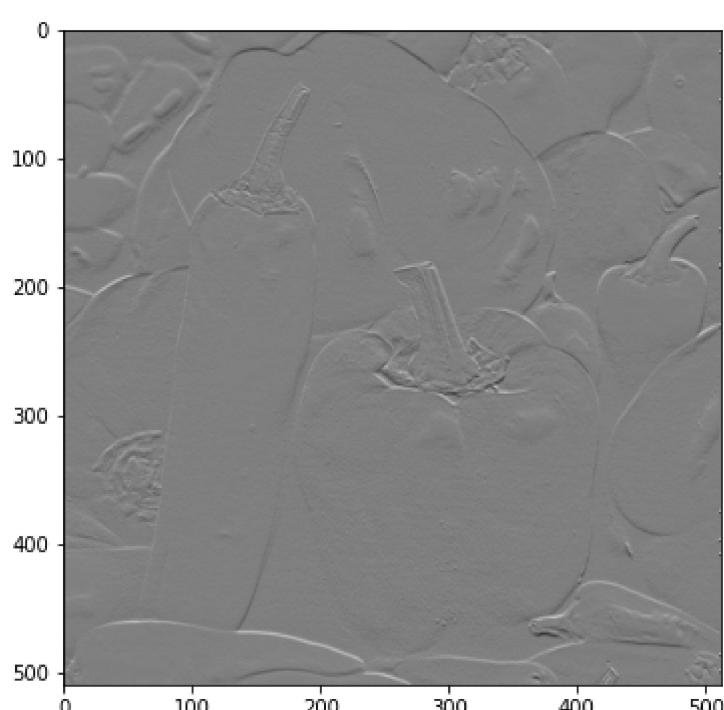
```
*****  
## [RESULT 03]  
*****
```



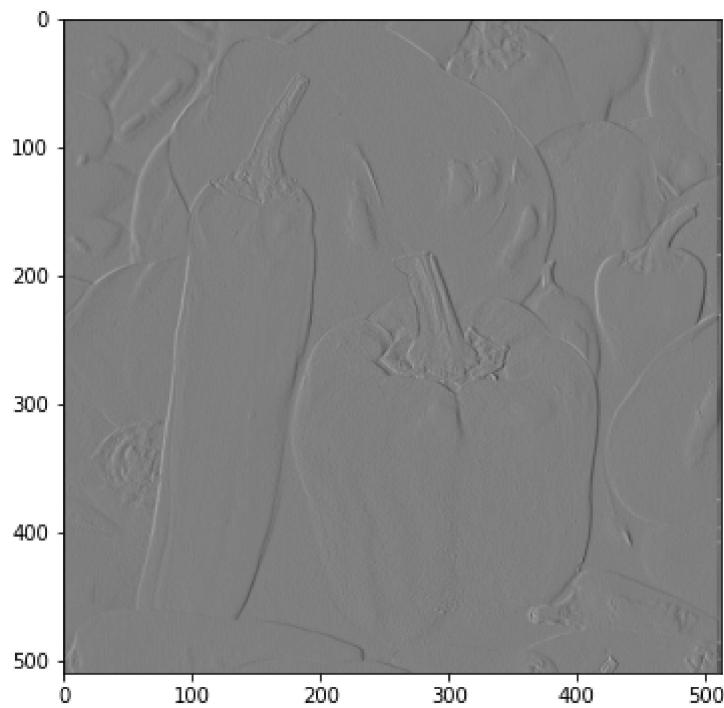
```
*****  
## [RESULT 04]  
*****
```



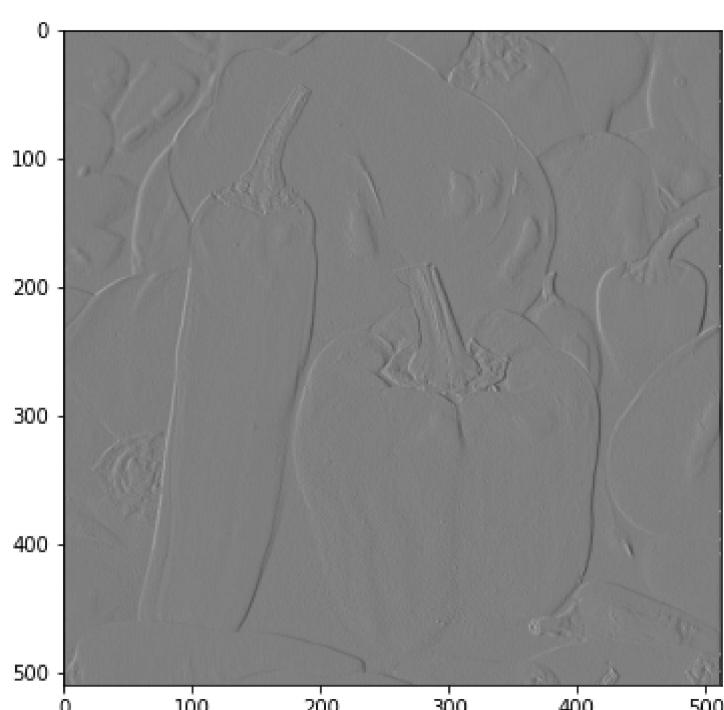
```
*****  
## [RESULT 05]  
*****
```



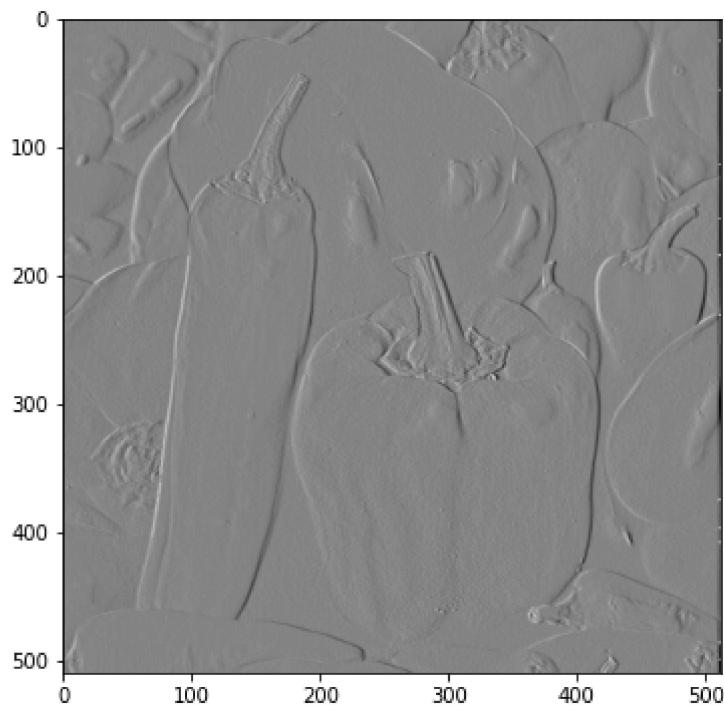
```
*****  
## [RESULT 06]  
*****
```



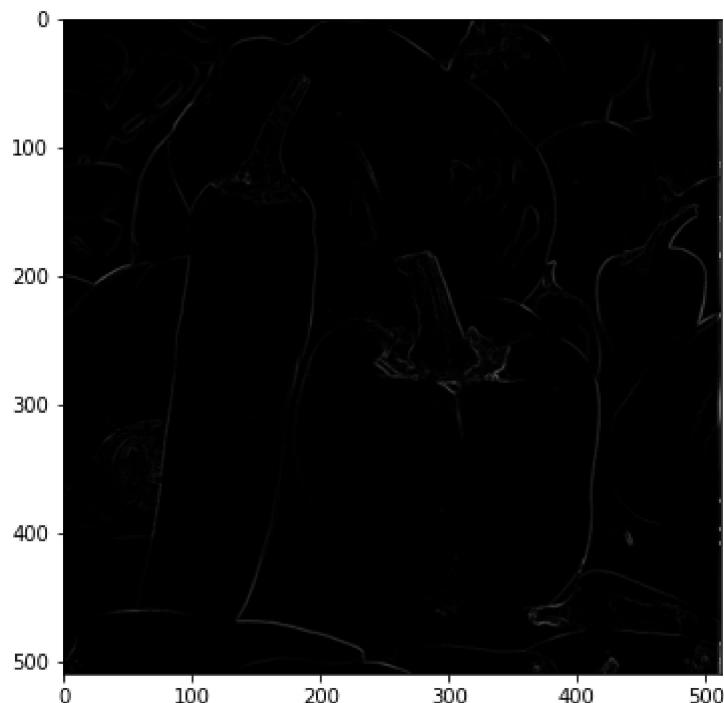
```
*****  
## [RESULT 07]  
*****
```



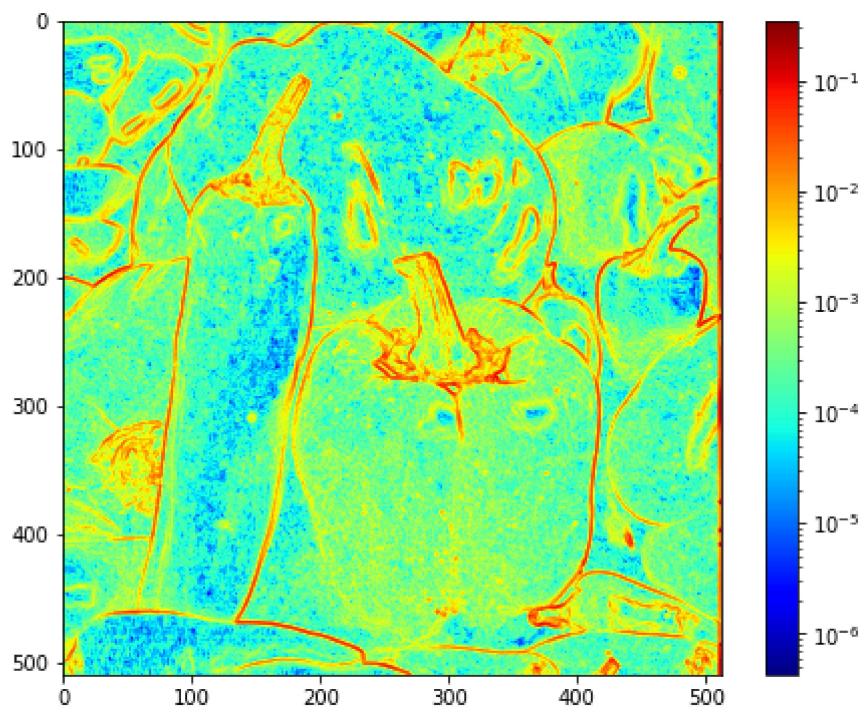
```
*****  
## [RESULT 08]  
*****
```



```
*****  
## [RESULT 09]  
*****
```



```
*****  
## [RESULT 10]  
*****
```



```
*****
## [RESULT 11]
*****
value1 = -0.007853403141361237
value2 = 0.0
value3 = -0.005235602094240843
value4 = 0.011780104712041883
*****
## [RESULT 12]
*****
value1 = 0.0
value2 = 0.0026178010471204186
value3 = 0.01570680628272253
value4 = -0.013089005235602025
*****
## [RESULT 13]
*****
value1 = -0.0039267015706806185
value2 = 0.0013089005235602093
value3 = 0.005235602094240843
value4 = -0.0006544502617800707
*****
## [RESULT 14]
*****
value1 = -0.03534031413612565
value2 = 0.0
value3 = -0.017015706806282727
value4 = 0.0
*****
## [RESULT 15]
*****
value1 = 0.0
value2 = -0.6426701570680627
value3 = 0.00916230366492149
value4 = 0.007853403141361293
*****
## [RESULT 16]
*****
value1 = -0.017670157068062825
value2 = -0.32133507853403137
value3 = -0.0039267015706806185
value4 = 0.003926701570680646
*****
## [RESULT 17]
*****
value1 = 0.00032765343603519625
value2 = 0.10325794591705269
value3 = 4.2830514514404736e-05
value4 = 1.5847290370329858e-05
```

In []: