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
```

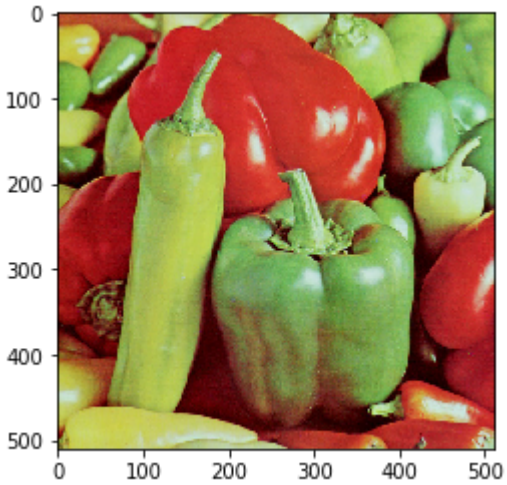
load input image ('test.jpeg')

In [ ]:

```
I0 = img.imread('test.jpeg')
```

In [ ]:

```
plt.imshow(I0)
plt.show()
```



In [ ]:

```
I0
```

Out[ ]:

```
array([[[192,  97,  53],
        [183,  87,  45],
        [189,  94,  48],
        ...,
        [142, 194,  84],
        [111, 195,  84],
        [  1,   0,   8]],

       [[190,  94,  52],
        [190,  95,  51],
        [180,  84,  42],
        ...,
        [142, 195,  81],
        [116, 199,  85],
        [  0,   0,   4]],

       [[179,  86,  43],
        [191,  98,  55],
        [179,  87,  40],
        ...,
        [152, 202,  87],
        [115, 195,  80],
        [  3,   1,   4]],

       ...,

       [[134, 139,  55],
        [129, 141,  55],
        [136, 145,  66],
        ...,
        [168, 198, 164],
        [202, 212, 178],
        [  3,   5,  17]],

       [[133, 131,  46],
        [137, 140,  59],
        [123, 133,  47],
        ...,
        [183, 214, 172],
        [194, 187, 141],
        [  0,   1,   0]],

       [[135, 125,  38],
        [131, 124,  54],
        [122, 134,  32],
        ...,
        [173, 203, 165],
        [193, 172, 129],
        [  2,   1,   0]]], dtype=uint8)
```

In [ ]:

```
np.shape(I0)
```

Out[ ]:

```
(510, 512, 3)
```

check the size of the input image

In [ ]:

```
# ++++++++++++++++++++++++++++++++++++++++++++++
# complete the blanks
#
num_row     = np.shape(I0)[0]
num_column  = np.shape(I0)[1]
num_channel = np.shape(I0)[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 [ ]:

```
I0[:,:,0]
```

Out[ ]:

```
array([[192, 183, 189, ..., 142, 111,   1],
       [190, 190, 180, ..., 142, 116,   0],
       [179, 191, 179, ..., 152, 115,   3],
       ...,
       [134, 129, 136, ..., 168, 202,   3],
       [133, 137, 123, ..., 183, 194,   0],
       [135, 131, 122, ..., 173, 193,   2]], dtype=uint8)
```

In [ ]:

```
np.shape(I0[:,:,0])
```

Out[ ]:

```
(510, 512)
```

In [ ]:

```
(IO[:, :, 0] + IO[:, :, 1] + IO[:, :, 2]) / 3
```

Out[ ]:

```
array([[28.66666667, 19.66666667, 25.        , ..., 54.66666667,
        44.66666667,  3.        ],
       [26.66666667, 26.66666667, 16.66666667, ..., 54.        ,
        48.        ,  1.33333333],
       [17.33333333, 29.33333333, 16.66666667, ..., 61.66666667,
        44.66666667,  2.66666667],
       ...,
       [24.        , 23.        , 30.33333333, ...,  6.        ,
        26.66666667,  8.33333333],
       [18.        , 26.66666667, 15.66666667, ..., 19.        ,
         3.33333333,  0.33333333],
       [14.        , 17.66666667, 10.66666667, ...,  9.66666667,
        79.33333333,  1.        ]])
```

In [ ]:

```
np.shape((IO[:, :, 0] + IO[:, :, 1] + IO[:, :, 2]) / 3)
```

Out[ ]:

```
(510, 512)
```

In [ ]:

```
print(IO[0,0,0])
print(IO[0,0,1])
print(IO[0,0,2])
print(IO[0, 0, 0] + IO[0, 0, 1] + IO[0, 0, 2])
```

```
192
97
53
86

/home/nextgen/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: Runti
meWarning: overflow encountered in ubyte_scalars
  after removing the cwd from sys.path.
```

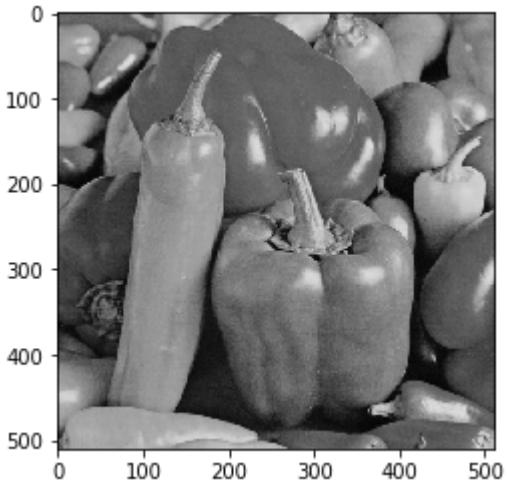
In [ ]:

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

I = np.mean(I0, axis=2)


num_row     = np.shape(I)[0]
num_column  = np.shape(I)[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
```

In [ ]:

```
plt.imshow(I, cmap='gray')
plt.show()
```



normalize the converted image

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

In [ ]:

```
I
```

Out[ ]:

```
array([[114.        , 105.        , 110.33333333, ..., 140.        ,
        130.        ,   3.        ],
       [112.        , 112.        , 102.        , ..., 139.33333333,
        133.33333333,   1.33333333],
       [102.66666667, 114.66666667, 102.        , ..., 147.        ,
        130.        ,   2.66666667],
       ...,
       [109.33333333, 108.33333333, 115.66666667, ..., 176.66666667,
        197.33333333,   8.33333333],
       [103.33333333, 112.        , 101.        , ..., 189.66666667,
        174.        ,   0.33333333],
       [ 99.33333333, 103.        ,  96.        , ..., 180.33333333,
        164.66666667,   1.        ]])
```

In [ ]:

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

I = (I - I.min()) / (I.max() - I.min())

#
# ++++++++++++++++++++++++++++++++++++++++++++++++

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
```

In [ ]:

```
I
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
np.shape(I)
```

Out[ ]:

```
(510, 512)
```

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

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

In [ ]:

```
I
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
I[num_row-1]
```

Out[ ]:

```
array([0.39005236, 0.40445026, 0.37696335, 0.38089005, 0.34947644,
       0.31544503, 0.37958115, 0.44240838, 0.54450262, 0.60602094,
       0.57329843, 0.56806283, 0.54973822, 0.54973822, 0.59424084,
       0.59293194, 0.62303665, 0.63089005, 0.63874346, 0.65314136,
       0.64528796, 0.62958115, 0.64528796, 0.66492147, 0.63481675,
       0.65706806, 0.65052356, 0.62696335, 0.62696335, 0.63350785,
       0.64267016, 0.64790576, 0.63481675, 0.64790576, 0.64528796,
       0.65314136, 0.64921466, 0.64397906, 0.67277487, 0.68586387,
       0.68193717, 0.71204188, 0.66884817, 0.68455497, 0.66099476,
       0.64528796, 0.66884817, 0.65052356, 0.67146597, 0.67277487,
       0.66099476, 0.68586387, 0.65968586, 0.67539267, 0.67539267,
       0.64659686, 0.66884817, 0.65837696, 0.64136126, 0.66492147,
       0.66361257, 0.66099476, 0.66492147, 0.66492147, 0.67277487,
       0.68193717, 0.64921466, 0.64659686, 0.64397906, 0.64136126,
       0.66753927, 0.63089005, 0.64397906, 0.64528796, 0.63481675,
       0.64790576, 0.64267016, 0.67801047, 0.68062827, 0.65968586,
       0.69371728, 0.69764398, 0.71858639, 0.7578534 , 0.72905759,
       0.71989529, 0.69371728, 0.69371728, 0.70549738, 0.70418848,
       0.70680628, 0.72382199, 0.70680628, 0.71727749, 0.71596859,
       0.70680628, 0.72251309, 0.73167539, 0.72643979, 0.72120419,
       0.69502618, 0.71989529, 0.70811518, 0.67801047, 0.69109948,
       0.67408377, 0.68324607, 0.67801047, 0.67408377, 0.67931937,
       0.66492147, 0.65183246, 0.66492147, 0.64528796, 0.65052356,
       0.64136126, 0.62696335, 0.62434555, 0.61649215, 0.61910995,
       0.61910995, 0.65052356, 0.65968586, 0.64790576, 0.61910995,
       0.63219895, 0.62565445, 0.59947644, 0.65183246, 0.64267016,
       0.62434555, 0.63219895, 0.60340314, 0.61649215, 0.57984293,
       0.59031414, 0.61387435, 0.57198953, 0.56544503, 0.58900524,
       0.58507853, 0.64528796, 0.63350785, 0.64136126, 0.61256545,
       0.58638743, 0.65052356, 0.59947644, 0.65314136, 0.62565445,
       0.64267016, 0.64136126, 0.64136126, 0.65706806, 0.63350785,
       0.67408377, 0.62827225, 0.68324607, 0.64790576, 0.68062827,
       0.65183246, 0.62303665, 0.70680628, 0.65968586, 0.68586387,
       0.64921466, 0.67670157, 0.66099476, 0.66623037, 0.67801047,
       0.69371728, 0.68979058, 0.69895288, 0.70680628, 0.69633508,
       0.71204188, 0.70811518, 0.69895288, 0.71335079, 0.71204188,
       0.71596859, 0.72513089, 0.71858639, 0.73429319, 0.72120419,
       0.72120419, 0.73429319, 0.72774869, 0.73429319, 0.7552356 ,
       0.7460733 , 0.7460733 , 0.7473822 , 0.7552356 , 0.77356021,
       0.7486911 , 0.7565445 , 0.77356021, 0.75      , 0.76308901,
       0.7578534 , 0.7460733 , 0.76308901, 0.76439791, 0.7617801 ,
       0.7565445 , 0.7447644 , 0.7604712 , 0.73036649, 0.73036649,
       0.7382199 , 0.73560209, 0.7421466 , 0.73298429, 0.71989529,
       0.7486911 , 0.73036649, 0.7617801 , 0.70942408, 0.75      ,
       0.73429319, 0.7604712 , 0.72905759, 0.7473822 , 0.73036649,
       0.72513089, 0.76963351, 0.70811518, 0.76439791, 0.70418848,
       0.7408377 , 0.73167539, 0.71335079, 0.7591623 , 0.72774869,
       0.7460733 , 0.71596859, 0.7447644 , 0.70811518, 0.73298429,
       0.7434555 , 0.73036649, 0.72513089, 0.72774869, 0.7395288 ,
       0.7408377 , 0.73560209, 0.72774869, 0.7513089 , 0.76570681,
       0.75      , 0.7382199 , 0.73167539, 0.72643979, 0.73167539,
       0.70680628, 0.71465969, 0.72513089, 0.7382199 , 0.69895288,
       0.75      , 0.68848168, 0.78141361, 0.72251309, 0.71596859,
       0.67801047, 0.68848168, 0.70287958, 0.61518325, 0.63089005,
       0.39659686, 0.32329843, 0.2526178 , 0.28272251, 0.27225131,
       0.26439791, 0.30366492, 0.28664921, 0.29973822, 0.28795812,
       0.30366492, 0.33246073, 0.33376963, 0.37041885, 0.32853403,
       0.34685864, 0.33246073, 0.36780105, 0.39136126, 0.38743455,
       0.37696335, 0.39921466, 0.39921466, 0.41492147, 0.40183246,
```

```
0.40968586, 0.39528796, 0.43324607, 0.36518325, 0.41623037,
0.34947644, 0.38350785, 0.39136126, 0.35602094, 0.35863874,
0.36387435, 0.40706806, 0.37434555, 0.38350785, 0.36387435,
0.36649215, 0.36780105, 0.38219895, 0.38612565, 0.37696335,
0.37303665, 0.35994764, 0.37827225, 0.38612565, 0.37958115,
0.37827225, 0.36125654, 0.34816754, 0.35471204, 0.36518325,
0.35863874, 0.37696335, 0.34947644, 0.34947644, 0.36256545,
0.34685864, 0.36125654, 0.34554974, 0.35994764, 0.36387435,
0.36387435, 0.38874346, 0.39659686, 0.40314136, 0.38089005,
0.37696335, 0.38350785, 0.39005236, 0.42670157, 0.44502618,
0.45157068, 0.43062827, 0.43062827, 0.45157068, 0.45811518,
0.46596859, 0.46596859, 0.46335079, 0.51308901, 0.53010471,
0.56151832, 0.54450262, 0.57984293, 0.55628272, 0.64659686,
0.67146597, 0.68717277, 0.77748691, 0.7526178 , 0.7604712 ,
0.78534031, 0.65314136, 0.59685864, 0.56282723, 0.67931937,
0.70157068, 0.59293194, 0.53272251, 0.58769634, 0.60863874,
0.64397906, 0.64136126, 0.63743455, 0.66099476, 0.59293194,
0.62958115, 0.57853403, 0.55104712, 0.45549738, 0.47905759,
0.40052356, 0.28141361, 0.30235602, 0.27617801, 0.29057592,
0.31413613, 0.29712042, 0.30497382, 0.28795812, 0.32068063,
0.28534031, 0.31937173, 0.34293194, 0.30104712, 0.31413613,
0.32329843, 0.34424084, 0.29581152, 0.34554974, 0.34685864,
0.31151832, 0.33638743, 0.34816754, 0.35994764, 0.28534031,
0.34947644, 0.32591623, 0.32984293, 0.30366492, 0.19502618,
0.22251309, 0.27094241, 0.15968586, 0.15837696, 0.31282723,
0.2578534 , 0.38089005, 0.70287958, 0.65575916, 0.57853403,
0.4908377 , 0.60078534, 0.57329843, 0.39528796, 0.33246073,
0.32984293, 0.36387435, 0.32853403, 0.33900524, 0.34685864,
0.34947644, 0.36518325, 0.33638743, 0.34554974, 0.33900524,
0.33115183, 0.33376963, 0.33900524, 0.34293194, 0.33900524,
0.33376963, 0.33115183, 0.34554974, 0.33507853, 0.32722513,
0.32591623, 0.36910995, 0.33900524, 0.36780105, 0.34293194,
0.37958115, 0.37041885, 0.35994764, 0.38743455, 0.36518325,
0.35732984, 0.37172775, 0.37958115, 0.36387435, 0.35863874,
0.36649215, 0.38743455, 0.37565445, 0.38743455, 0.35602094,
0.38219895, 0.38612565, 0.41099476, 0.43586387, 0.45026178,
0.41884817, 0.40968586, 0.43455497, 0.44109948, 0.40314136,
0.45549738, 0.42801047, 0.5117801 , 0.51570681, 0.54842932,
0.57853403, 0.56413613, 0.58769634, 0.61125654, 0.68324607,
0.83246073, 0.87041885, 0.81151832, 0.7604712 , 0.61518325,
0.56806283, 0.72382199, 0.66230366, 0.69109948, 0.7382199 ,
0.77094241, 0.78010471, 0.7552356 , 0.7591623 , 0.76570681,
0.7617801 , 0.76439791, 0.69502618, 0.68848168, 0.70811518,
0.64659686, 0.0039267 ])
```

In [ ]:

```
temp = I.copy()
temp[0] = temp[num_row-1]
```

In [ ]:

```
temp
```

Out[ ]:

```
array([[0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
np.roll(temp, -1, axis =0 )
```

Out[ ]:

```
array([[0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       [0.42670157, 0.47643979, 0.42539267, ..., 0.55497382, 0.53795812,
        0.0039267 ],
       ...,
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
I
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
def compute_derivative_x_forward(I):

    D = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    forward_x = I.copy()
    forward_x[0, :] = forward_x[num_row-1, :]
    forward_x = np.roll(forward_x, -1, axis =0 )
    D = forward_x - I

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++

    return D
```

In [ ]:

```
compute_derivative_x_forward(I)
```

Out[ ]:

```
array([[-0.0078534 ,  0.02748691, -0.03272251, ..., -0.0026178 ,
         0.01308901, -0.0065445 ],
       [-0.03664921,  0.0104712 ,  0.        , ...,  0.03010471,
        -0.01308901,  0.0052356 ],
       [ 0.02356021,  0.02617801,  0.02486911, ..., -0.02225131,
         0.02748691, -0.0065445 ],
       ...,
       [-0.02356021,  0.01439791, -0.05759162, ...,  0.05104712,
        -0.09162304, -0.03141361],
       [-0.01570681, -0.03534031, -0.01963351, ..., -0.03664921,
        -0.03664921,  0.0026178 ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ]])
```

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

In [ ]:

```
temp = I.copy()
temp[num_row-1] = temp[0]
temp
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ]])
```

In [ ]:

```
np.roll(temp, 1, axis =0 )
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       ...,
       [0.41230366, 0.42146597, 0.43717277, ..., 0.72774869, 0.71858639,
        0.0091623 ],
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ]])
```

In [ ]:

```
I
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
def compute_derivative_x_backward(I):

    D = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    backward_x = I.copy()
    backward_x[num_row-1, :] = backward_x[0, :]
    backward_x = np.roll(backward_x, 1, axis =0 )
    D = I - backward_x


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++

    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
    #
    forward_x = I.copy()
    forward_x[0, :] = forward_x[num_row-1, :]
    forward_x = np.roll(forward_x, -1, axis =0 )
    backward_x = I.copy()
    backward_x[num_row-1, :] = backward_x[0, :]
    backward_x = np.roll(backward_x, 1, axis =0 )
    D = (forward_x - backward_x) / 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 [ ]:

```
np.roll(I, -1, axis =1 )
```

Out[ ]:

```
array([[0.41230366, 0.43324607, 0.43979058, ..., 0.5104712 , 0.0117801 ,
        0.44764398],
       [0.43979058, 0.40052356, 0.42146597, ..., 0.52356021, 0.0052356 ,
        0.43979058],
       [0.45026178, 0.40052356, 0.43062827, ..., 0.5104712 , 0.0104712 ,
        0.40314136],
       ...,
       [0.42539267, 0.45418848, 0.35340314, ..., 0.77486911, 0.03272251,
        0.42931937],
       [0.43979058, 0.39659686, 0.39005236, ..., 0.68324607, 0.0013089 ,
        0.40575916],
       [0.40445026, 0.37696335, 0.38089005, ..., 0.64659686, 0.0039267 ,
        0.39005236]])
```

In [ ]:

```
I
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
temp = I.copy()
temp[:,0] = temp[:, num_column-1]
np.roll(temp, -1, axis =1 )
```

Out[ ]:

```
array([[0.41230366, 0.43324607, 0.43979058, ..., 0.5104712 , 0.0117801 ,
        0.0117801 ],
       [0.43979058, 0.40052356, 0.42146597, ..., 0.52356021, 0.0052356 ,
        0.0052356 ],
       [0.45026178, 0.40052356, 0.43062827, ..., 0.5104712 , 0.0104712 ,
        0.0104712 ],
       ...,
       [0.42539267, 0.45418848, 0.35340314, ..., 0.77486911, 0.03272251,
        0.03272251],
       [0.43979058, 0.39659686, 0.39005236, ..., 0.68324607, 0.0013089 ,
        0.0013089 ],
       [0.40445026, 0.37696335, 0.38089005, ..., 0.64659686, 0.0039267 ,
        0.0039267 ]])
```

In [ ]:

```
I
```

Out[ ]:

```
array([[0.44764398, 0.41230366, 0.43324607, ..., 0.54973822, 0.5104712 ,
        0.0117801 ],
       [0.43979058, 0.43979058, 0.40052356, ..., 0.54712042, 0.52356021,
        0.0052356 ],
       [0.40314136, 0.45026178, 0.40052356, ..., 0.57722513, 0.5104712 ,
        0.0104712 ],
       ...,
       [0.42931937, 0.42539267, 0.45418848, ..., 0.69371728, 0.77486911,
        0.03272251],
       [0.40575916, 0.43979058, 0.39659686, ..., 0.7447644 , 0.68324607,
        0.0013089 ],
       [0.39005236, 0.40445026, 0.37696335, ..., 0.70811518, 0.64659686,
        0.0039267 ]])
```

In [ ]:

```
def compute_derivative_y_forward(I):

    D = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
    forward_y = I.copy()
    forward_y[:,0] = forward_y[:, num_column-1]
    forward_y = np.roll(forward_y, -1, axis =1 )
    D = forward_y - 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
    #
    backward_y = I.copy()
    backward_y[:, num_column-1] = backward_y[:, 0]
    backward_y = np.roll(backward_y, 1, axis =1 )
    D = I - backward_y


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++

    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
    #
    forward_y = I.copy()
    forward_y[:,0] = forward_y[:, num_column-1]
    forward_y = np.roll(forward_y, -1, axis =1 )
    backward_y = I.copy()
    backward_y[:, num_column-1] = backward_y[:, 0]
    backward_y = np.roll(backward_y, 1, axis =1 )
    D = (forward_y - backward_y) / 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
    #
    central_x = compute_derivative_x_central(I)
    central_y = compute_derivative_y_central(I)
    norm_gradient = central_x**2 + central_y**2


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++

    return norm_gradient
```

functions for presenting the results

In [ ]:

```
def function_result_01():

    plt.figure(figsize=(8,6))
    plt.imshow(I0)
    plt.show()
```

In [ ]:

```
def function_result_02():

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

In [ ]:

```
def function_result_03():

    D = compute_derivative_x_forward(I)

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

In [ ]:

```
def function_result_04():

    D = compute_derivative_x_backward(I)

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

In [ ]:

```
def function_result_05():

    D = compute_derivative_x_central(I)

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

In [ ]:

```
def function_result_06():

    D = compute_derivative_y_forward(I)

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

In [ ]:

```
def function_result_07():

    D = compute_derivative_y_backward(I)

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

In [ ]:

```
def function_result_08():

    D = compute_derivative_y_central(I)

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

In [ ]:

```
def function_result_09():

    D = compute_norm_gradient_central(I)

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

In [ ]:

```
def function_result_10():

    D = compute_norm_gradient_central(I)

    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(I)

    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(I)

    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(I)

    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(I)

    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(I)

    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(I)

    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 [ ]:

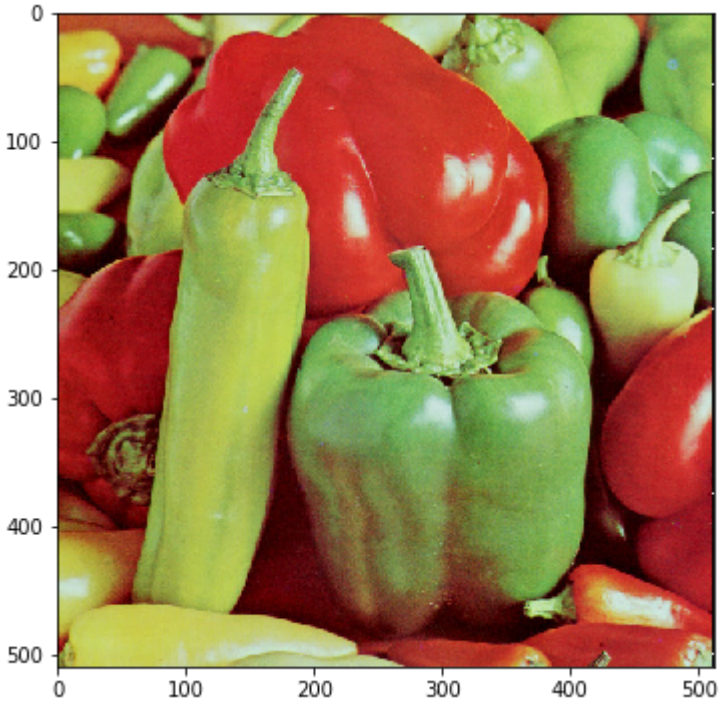
```
def function_result_17():

    D = compute_norm_gradient_central(I)

    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

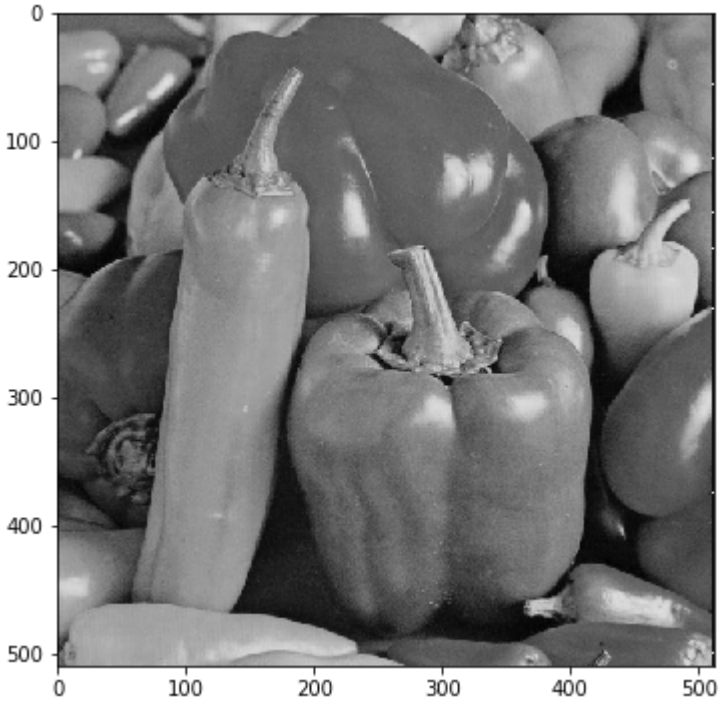
In [ ]:

```
number_result = 17

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

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

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



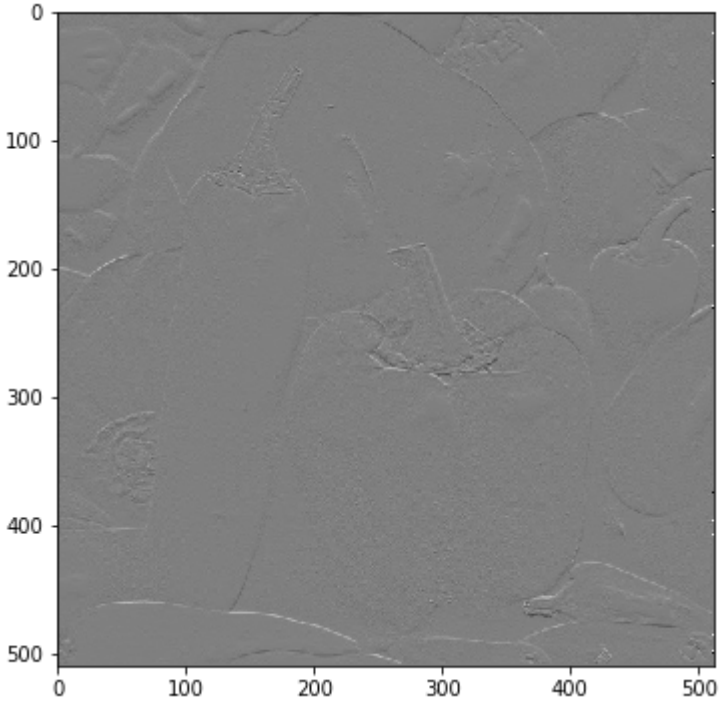

```
**************************************************
## [RESULT 02]
**************************************************
```


0
100
200
300
400
500
0
100
200
300
400
500

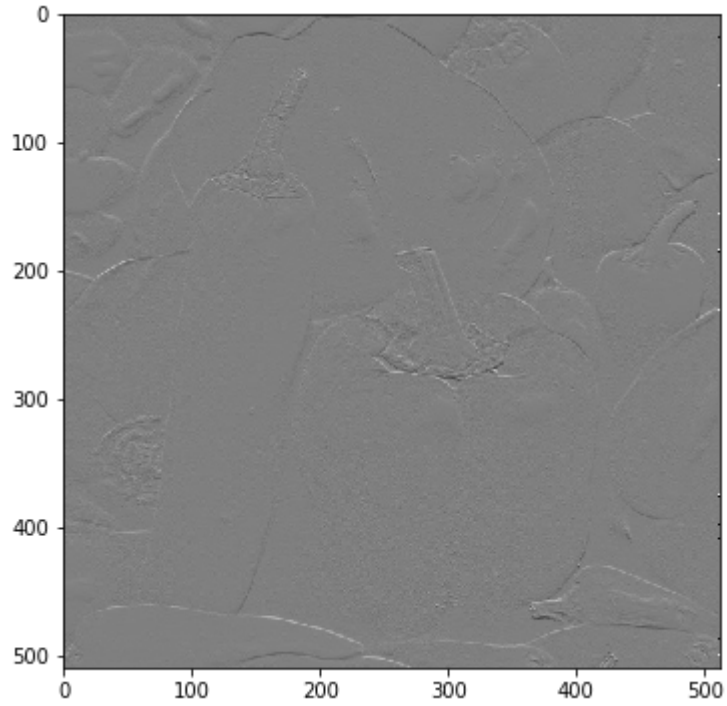
****

[RESULT 03]

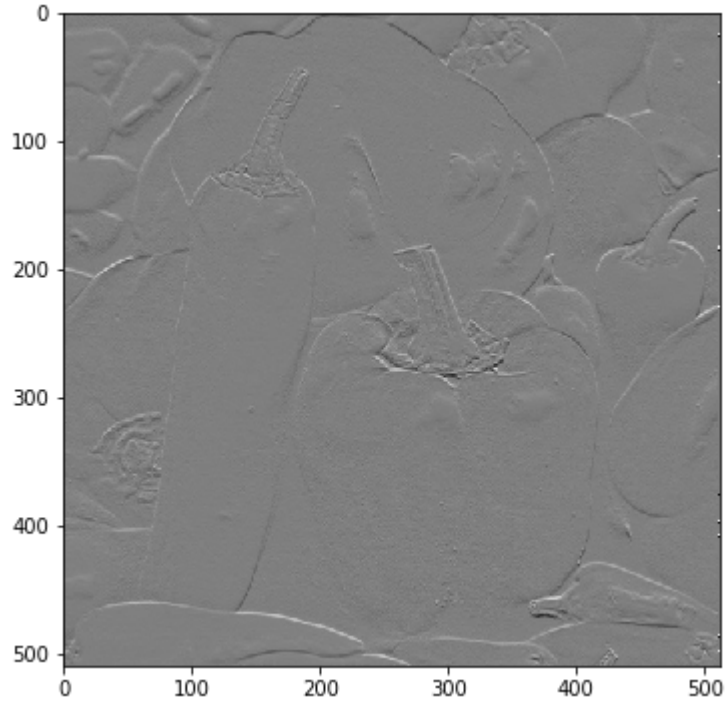
****


0
100
200
300
400
500
0
100
200
300
400
500

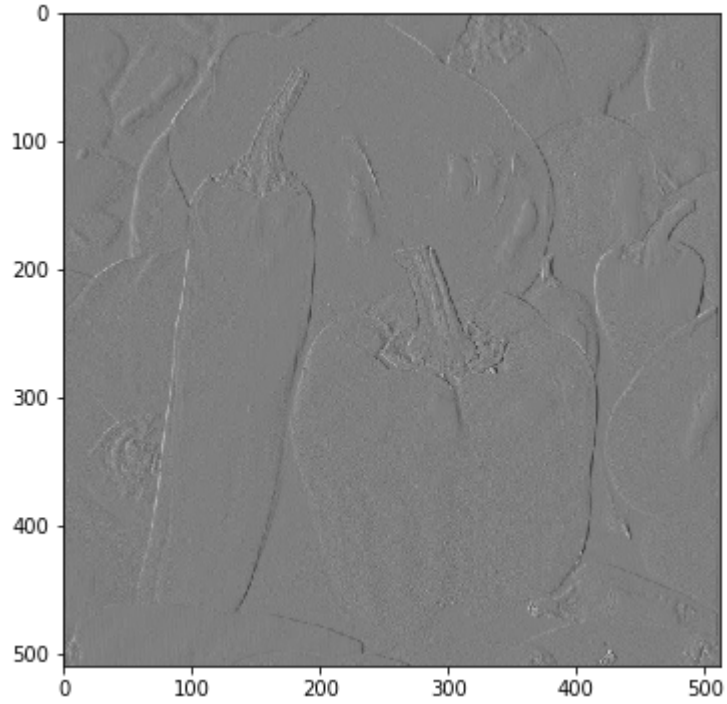
```
**************************************************
## [RESULT 04]
**************************************************
```



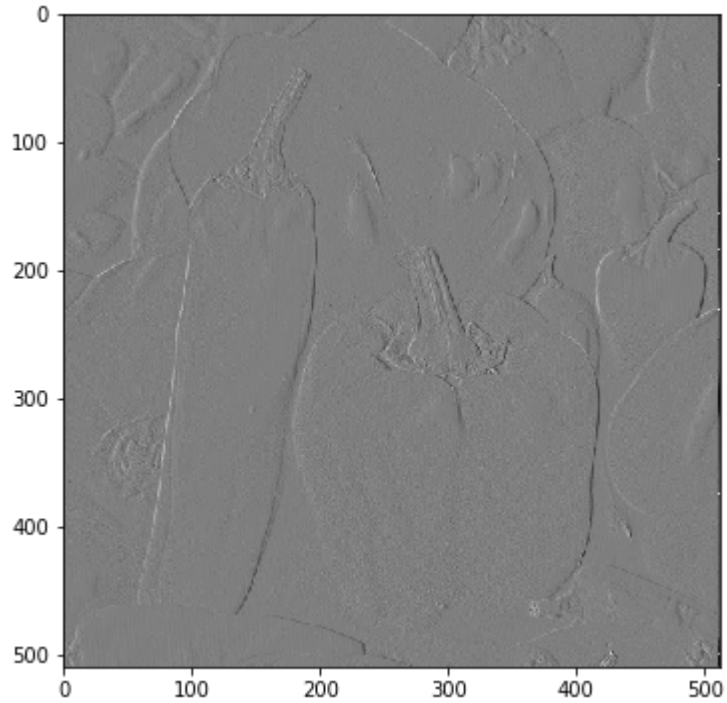

```
**************************************************
## [RESULT 05]
**************************************************
```



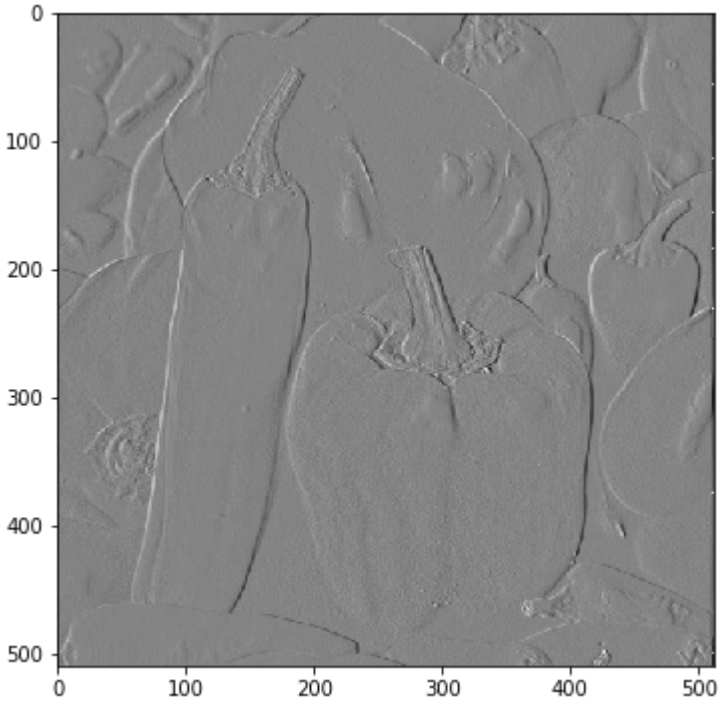

```
**************************************************
## [RESULT 06]
**************************************************
```



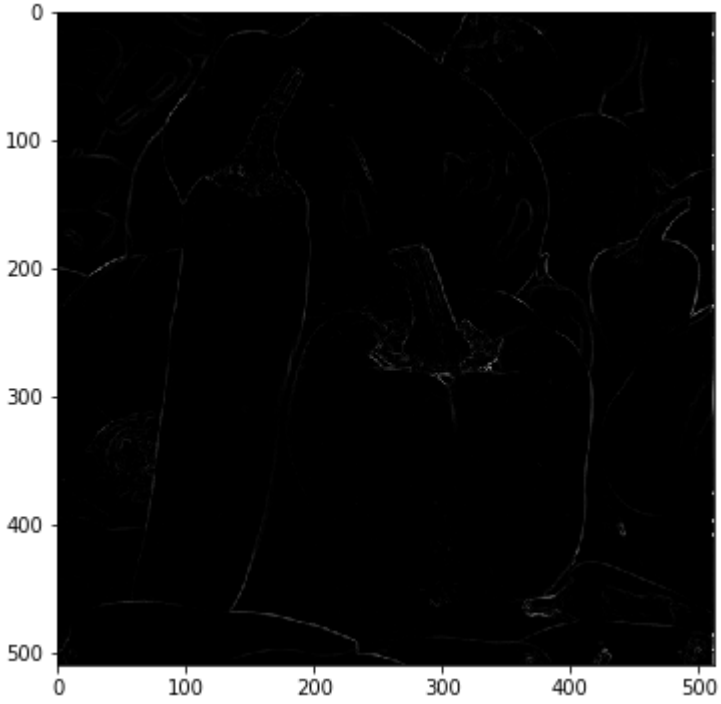
****
[RESULT 07]
****



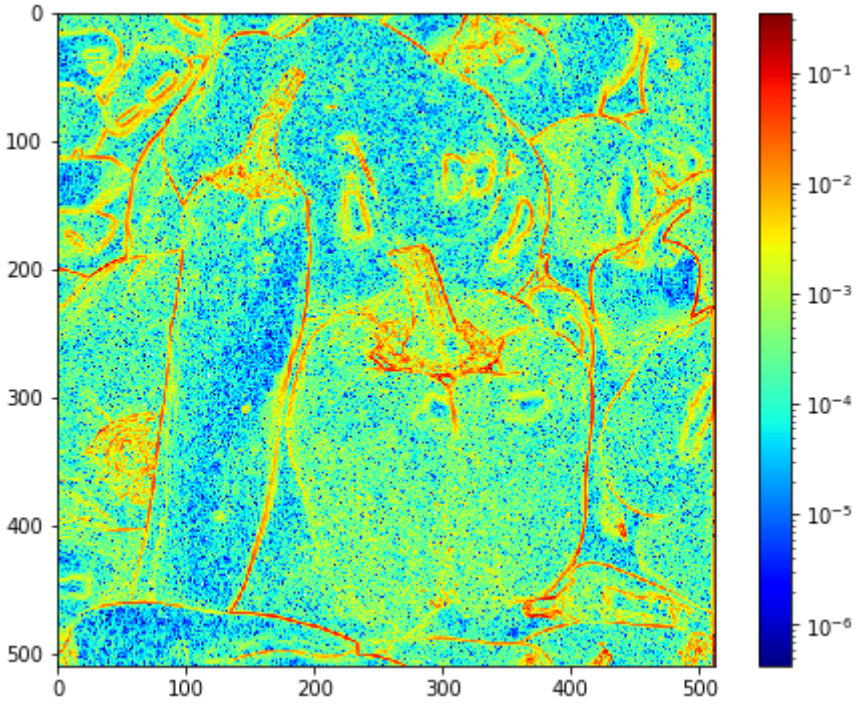
****
[RESULT 08]
****



****
[RESULT 09]
****



****
[RESULT 10]
****


0
100
200
300
400
500
0
100
200
300
400
500
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}

```
**************************************************
## [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 =  0.0003276534360351962 5
value2 =  0.10325794591705269
value3 =  4.2830514514404736e-05
value4 =  1.5847290370329858e-05
```

In [ ]: