

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 [ ]: 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)
```

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 = np.mean(I0,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
```

```
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.        ]])
```

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
```

```
In [ ]: I.shape
```

```
Out[ ]: (510, 512)
```

```
In [ ]: for i in range(I.shape[0]):
        print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110

111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179

180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248

249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317

318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386

387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455

456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509

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
```

```

#

for i in range(I.shape[0]-1):
    for j in range(I.shape[1]):
        D[i,j] = I[i+1,j] - I[i,j]

#
# ++++++

return D

```

```
In [ ]: compute_derivative_x_forward(I).shape
```

```
Out[ ]: (510, 512)
```

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

```

In [ ]: def compute_derivative_x_backward(I):

    D = np.zeros(I.shape)

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

    for i in range(1, I.shape[0]):
        for j in range(I.shape[1]):
            D[i,j] = I[i,j] - I[i-1,j]

    #
    # ++++++

    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
    #

    for i in range(1, I.shape[0]-1):
        for j in range(I.shape[1]):
            D[i,j] = 0.5*(I[i+1,j] - I[i-1,j])

    #
    # ++++++

    return D

```

```
In [ ]: compute_derivative_x_central(I)
```

```

Out[ ]: array([[ 0.          ,  0.          ,  0.          , ...,  0.          ,
                 0.          ,  0.          ],
               [-0.02225131,  0.01897906, -0.01636126, ...,  0.01374346,
                 0.          , -0.00065445],

```

```

[-0.0065445, 0.01832461, 0.01243455, ..., 0.0039267,
 0.00719895, -0.00065445],
...,
[-0.00327225, 0.0091623, -0.02028796, ..., 0.00850785,
 -0.01767016, -0.0039267 ],
[-0.01963351, -0.0104712, -0.03861257, ..., 0.00719895,
 -0.06413613, -0.01439791],
[ 0., 0., 0., ..., 0.,
 0., 0. ]])

```

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(l):

    D = np.zeros(l.shape)

    # ++++++
    # complete the blanks
    #
    for i in range(l.shape[0]):
        for j in range(l.shape[1]-1):
            D[i,j] = 0.5*(l[i,j+1] - l[i,j])

    #
    # ++++++

    return D

```

```

In [ ]: compute_derivative_y_forward(l)

```

```

Out[ ]: array([[ -0.01767016,  0.0104712,  0.00327225, ..., -0.01963351,
                -0.24934555,  0.          ],
               [ 0.          , -0.01963351,  0.0104712, ..., -0.0117801,
                -0.2591623,  0.          ],
               [ 0.02356021, -0.02486911,  0.01505236, ..., -0.03337696,
                -0.25          , 0.          ],
               ...,
               [-0.00196335,  0.01439791, -0.05039267, ...,  0.04057592,
                -0.3710733,  0.          ],
               [ 0.01701571, -0.02159686, -0.00327225, ..., -0.03075916,
                -0.34096859,  0.          ],
               [ 0.00719895, -0.01374346,  0.00196335, ..., -0.03075916,
                -0.32133508,  0.          ]])

```

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

```

In [ ]: def compute_derivative_y_backward(l):

    D = np.zeros(l.shape)

    # ++++++
    # complete the blanks
    #
    for i in range(l.shape[0]):
        for j in range(1, l.shape[1]):
            D[i,j] = 0.5*(l[i,j] - l[i,j-1])

```

```
#
# ++++++

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
    #
    for i in range(I.shape[0]):
        for j in range(1, I.shape[1]-1):
            D[i,j] = 0.5*(I[i,j+1] - I[i,j-1])

    #
    # ++++++

    return D
```

```
In [ ]: compute_derivative_y_central(I).shape
```

Out[]: (510, 512)

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
    #

    norm_gradient = compute_derivative_x_central(I)**2 + compute_derivative_y_central(I)**2

    #
    # ++++++

    return norm_gradient
```

```
In [ ]: compute_norm_gradient_central(I).shape
```

Out[]: (510, 512)

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(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]

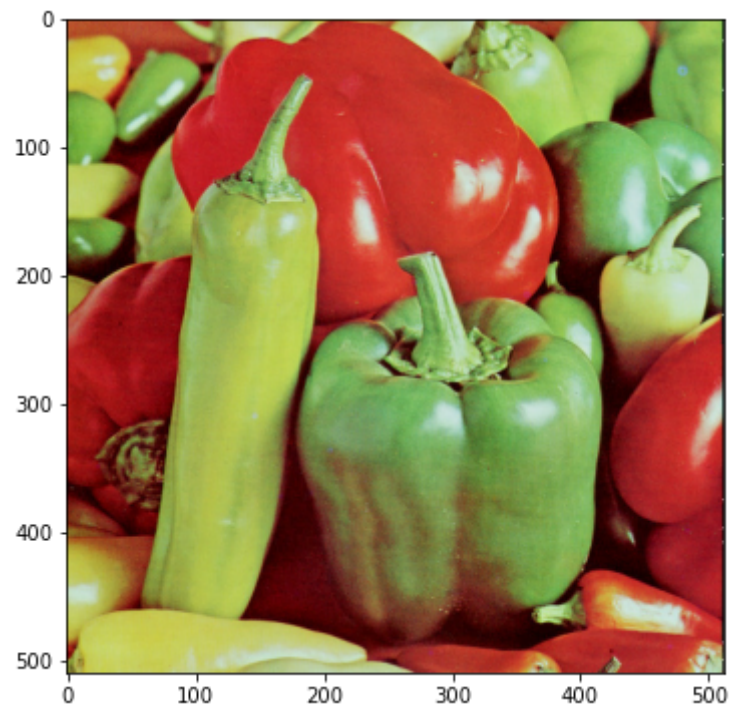
        print('value1 = ', value1)
        print('value2 = ', value2)
        print('value3 = ', value3)
        print('value4 = ', value4)
```

results

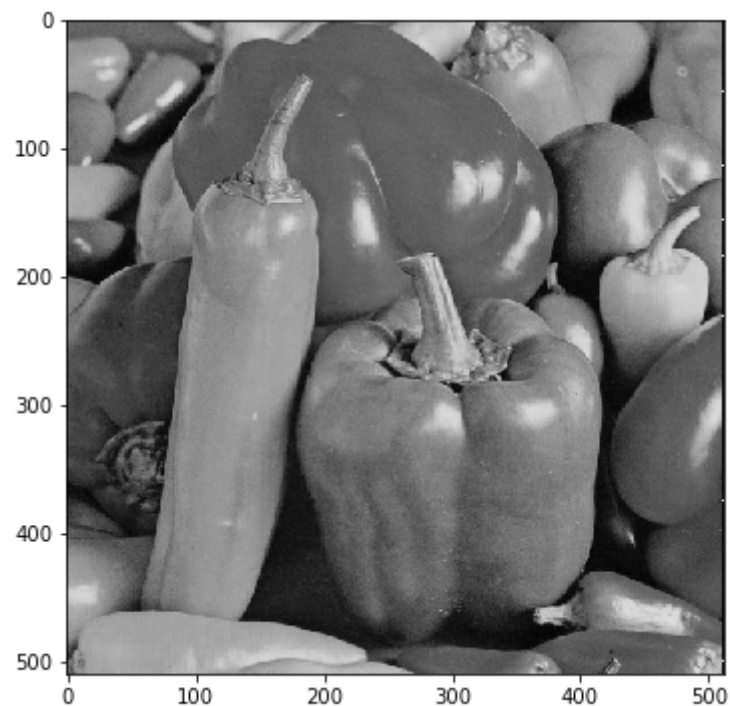
```
In [ ]: number_result = 17
```

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

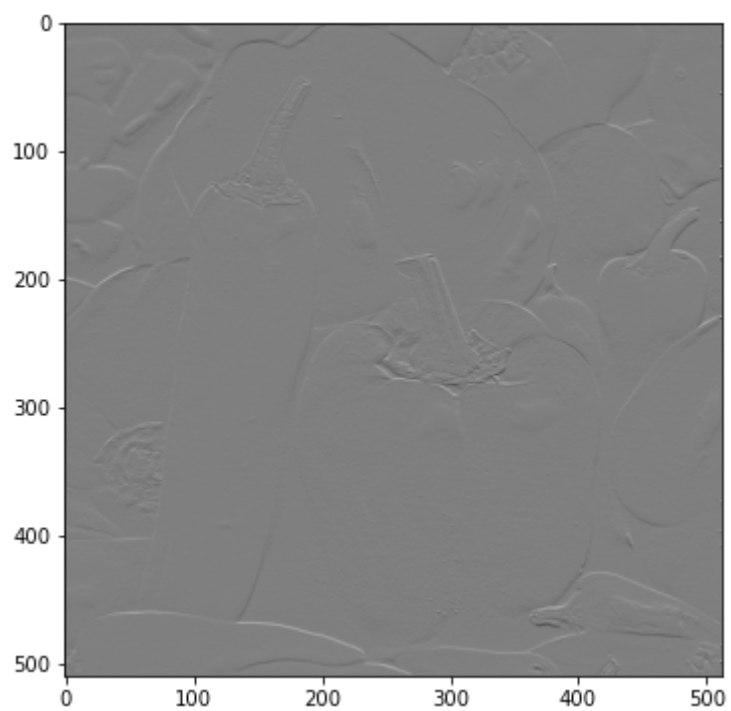
```
*****  
## [RESULT 01]  
*****
```



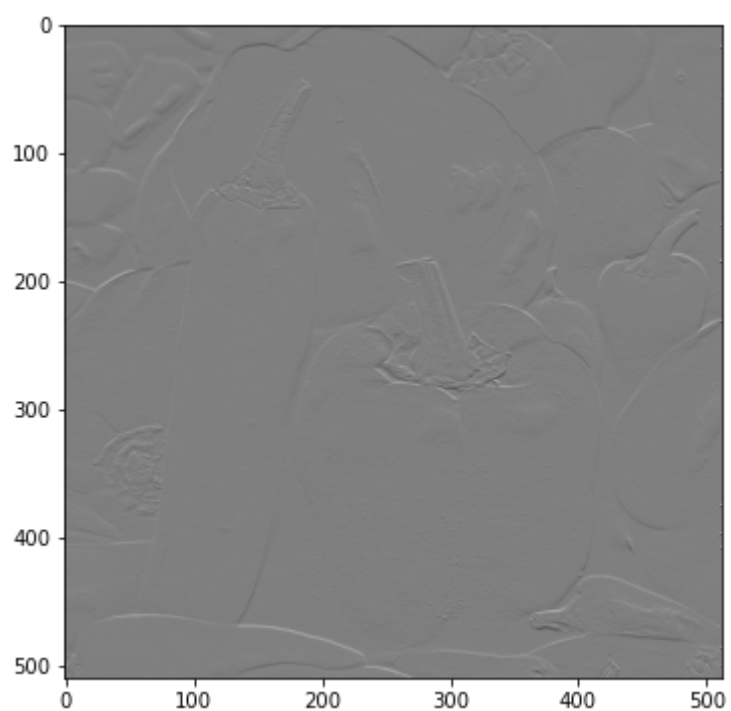
```
*****  
## [RESULT 02]  
*****
```



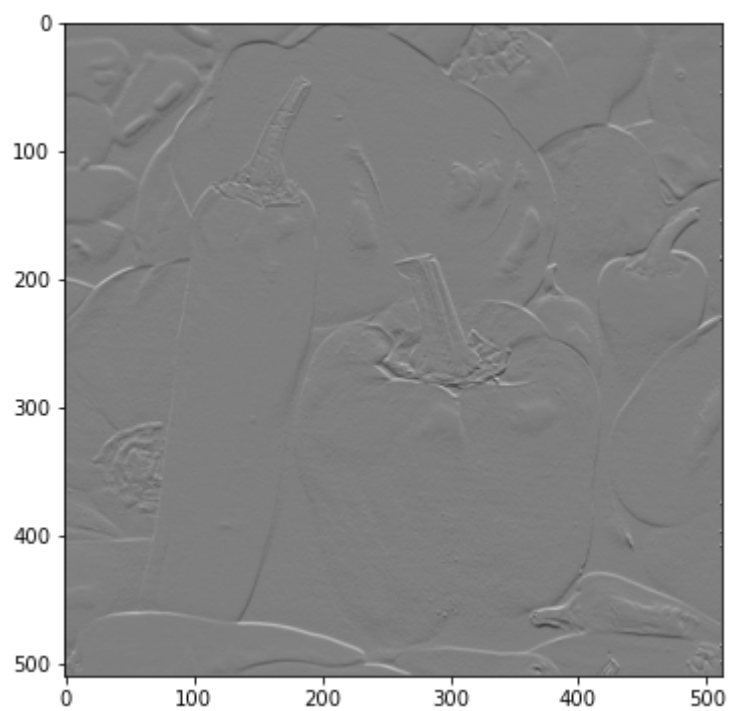
```
*****  
## [RESULT 03]  
*****
```



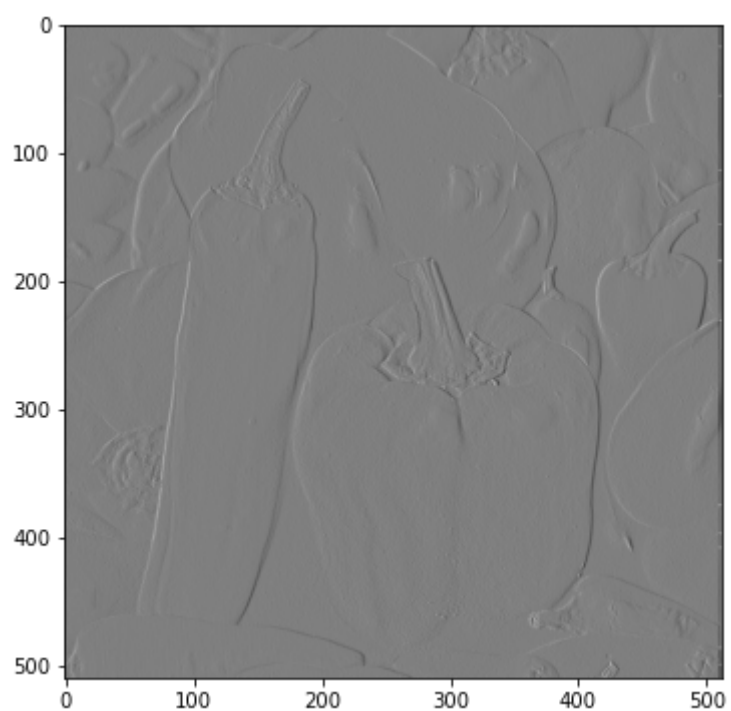
```
*****  
## [RESULT 04]  
*****
```



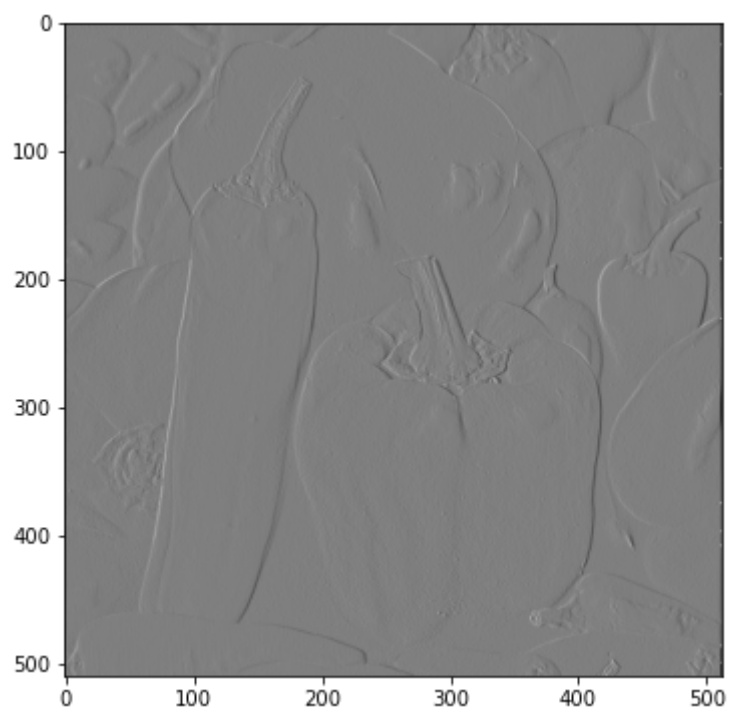
```
*****  
## [RESULT 05]  
*****
```



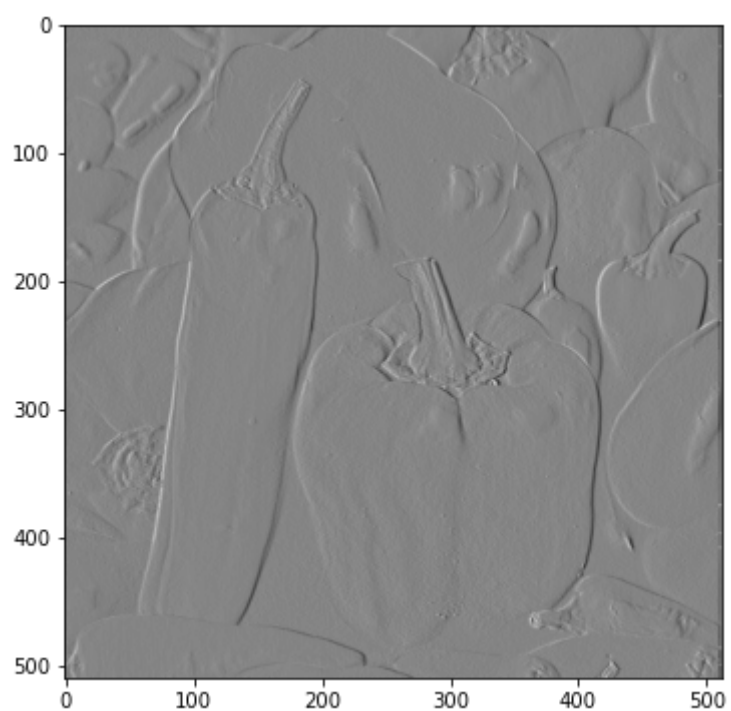
```
*****  
## [RESULT 06]  
*****
```



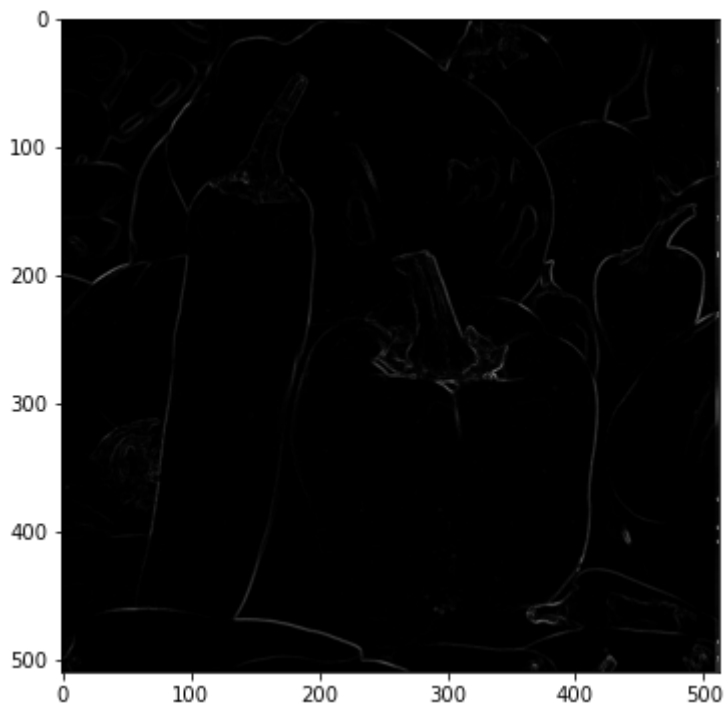
```
*****  
## [RESULT 07]  
*****
```



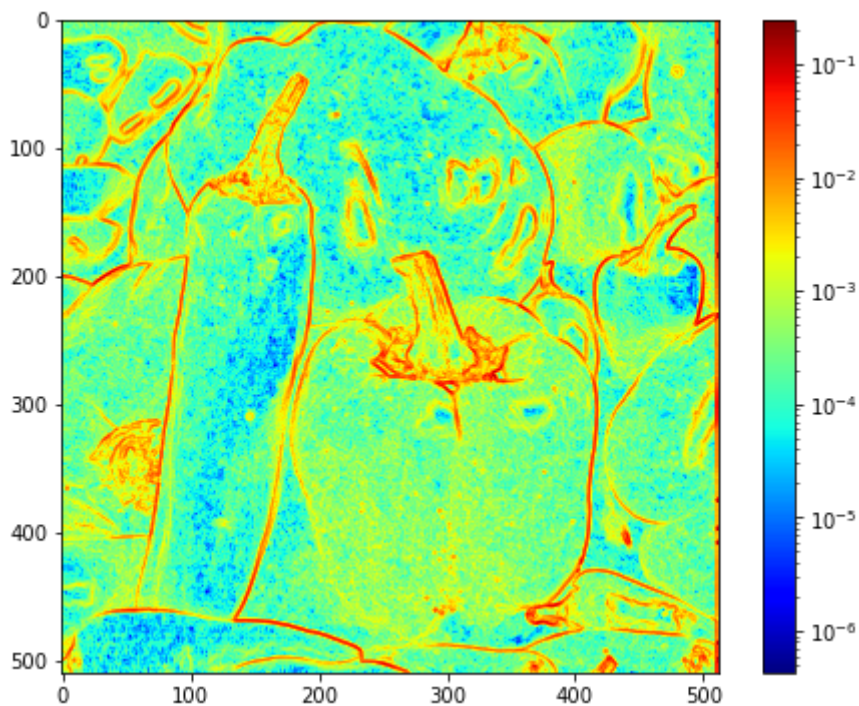
[RESULT 08]



[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 = 0.0
```

```
value2 = 0.0
value3 = 0.005235602094240843
value4 = -0.0006544502617800707
*****
## [RESULT 14]
*****
value1 = -0.017670157068062825
value2 = 0.0
value3 = -0.008507853403141363
value4 = 0.0
*****
## [RESULT 15]
*****
value1 = 0.0
value2 = -0.32133507853403137
value3 = 0.004581151832460745
value4 = 0.003926701570680646
*****
## [RESULT 16]
*****
value1 = 0.0
value2 = 0.0
value3 = -0.0039267015706806185
value4 = 0.003926701570680646
*****
## [RESULT 17]
*****
value1 = 0.0
value2 = 0.0
value3 = 4.2830514514404736e-05
value4 = 1.5847290370329858e-05
```

In []: