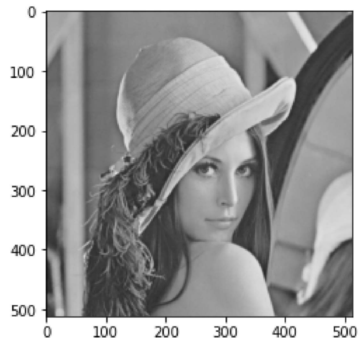
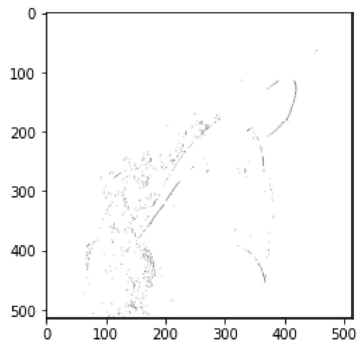
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

from google.colab import drive
drive.mount('/content/drive')


path = '/content/drive/MyDrive/Img-Lab/lab01/lena.png'
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()

kernel = np.array(([0,-1,0],
                   [-1,5,-1],
                   [0,-1,0]), np.float32)

img_copy = cv2.copyMakeBorder(img,1,1,1,1,cv2.BORDER_REPLICATE)
img_h , img_w , k_h, k_w = img_copy.shape[0], img_copy.shape[1], kernel.shape[0], kernel.shape[1]

# img_h , img_w , k_h, k_w = img.shape[0], img.shape[1], kernel.shape[0], kernel.shape[1]
print( 'height :' ,img_h , '\nwidth  :', img_w )


result = np.zeros(( img_copy.shape[0], img_copy.shape[1] ), dtype="float32")
# result = np.zeros(( img.shape[0], img.shape[1] ), dtype="float32")
```



```
height : 514
width  : 514
```

```
v = (k_h)//2

max_val = img_copy.max()
# print(img_copy.max())

# convolution
for row in range(v, img_h-v):
  for column in range(v, img_w-v):
    sum = 0
    for x in range(k_h):
       for y in range(k_w):
          sum = sum + kernel[x][y] * img_copy[ row+x-v][column+y-v]  #
    result[row-v][column-v] = sum
plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



```
'''
x = np.array([[1, 2], [4, 5]])
```

```
m = np.array([[1, 2, 3, 4, 5],
              [1, 2, 3, 4, 5],
              [1, 2, 3, 4, 5],
              [1, 2, 3, 4, 5],
              [1, 2, 3, 4, 5],
             ])

print(m)
'''

# # get the horizontal and vertical size of X and H
#    imageColumns = X.shape[1]
#    imageRows = X.shape[0]
#    kernelColumns = H.shape[1]
#    kernelRows = H.shape[0]

#    # calculate the horizontal and vertical size of Y (assume "full" convolution)
#    newRows = imageRows + kernelRows - 1
#    newColumns = imageColumns + kernelColumns - 1

#    # create an empty output array
#    Y = np.zeros((newRows,newColumns))


#    # go over output locations
#    for m in range(newRows):
#        for n in range(newColumns):

#     # go over input locations
#         for i in range(kernelRows):
#             for j in range(kernelColumns):
#                  if (m-i >= 0) and (m-i < imageRows ) and (n-j >= 0) and (n-j < imageColumns):
#                          Y[m,n] = Y[m,n] + H[i,j]*X[m-i,n-j]
#         # make sure kernel is within bounds

#         # calculate the convolution sum




'''
# converting an img to numpy array
np_img = np.asarray(kernel)

# flattening a 2D numpy array
np_img.flatten() # row-wise
np_img.flatten(order='F') # column-wise
'''
```

```
'\n# converting an img to numpy array\nnp_img = np.asarray(kernel) \n\n# flattening a 2D numpy array\nnp_img.fla
tten() # row-wise\nnp_img.flatten(order='F') # column-wise\n'
```

✓ 0s completed at 1:07 PM