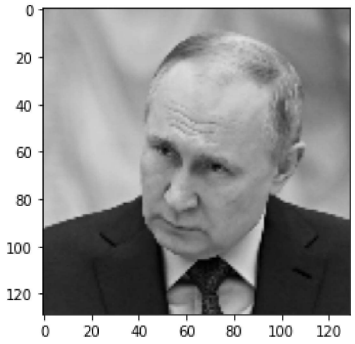
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

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


# path = '/content/drive/MyDrive/Img-Lab/lab01/obama.jpg'
path = '/content/drive/MyDrive/Img-Lab/lab01/putin.jpg'
# p = 0.205

# input image converted to grayscale format
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

# resizing the image
p = 0.21
w = int(img.shape[1] * p)
h = int(img.shape[0] * p)
img = cv2.resize(img, (w, h))

print( img.shape[0], img.shape[1] )
print('source img after grayscaling : ')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

```
129 129
source img after grayscaling :
```




```
# kernel
kernel = np.array(([0,-1,0],
                   [-1,5,-1],
                   [0,1,0]),
                  np.float32)
kernel_H = kernel.shape[0]
H = (kernel_H-1)//2
kernel_W = kernel.shape[1]
W = (kernel_W-1)//2


# padding
padImage = cv2.copyMakeBorder(img,H,H,W,W,cv2.BORDER_REPLICATE)
# plt.imshow(cv2.cvtColor(padImage, cv2.COLOR_BGR2RGB))
# plt.show()


# source image row columns
I_row_num, I_col_num = img.shape
# kernel row columns
F_row_num, F_col_num = kernel.shape

# output dimensions
output_row_num = I_row_num + F_row_num - 1
output_col_num = I_col_num + F_col_num - 1
output_shape = output_row_num, output_col_num
print(output_row_num, output_col_num)
# zero padding the kernel
F_zero_padded = np.pad(kernel, ((output_row_num-F_row_num, 0),
                                (0,output_col_num-F_col_num)),
                                'constant', constant_values = 0)
# print(F_zero_padded)
```

```
131 131
```

```
# convolution using toeplitz matrix
import scipy.linalg as sl
toeplitz_list = []
for i in range(F_zero_padded.shape[0]-1, -1,-1):      # iterating from last row to first because img will be flipped 180 degrees
  c = F_zero_padded[i,:];                             # taking the ith row in each iteration
  r = np.r_[c[0], np.zeros(I_col_num-1)]              # concatenating two arrays
  toeplitz_m = sl.toeplitz(c,r)                       # creating toeplitz matrices for each row
  toeplitz_list.append(toeplitz_m)


c = range(1, F_zero_padded.shape[0]+1)
# number of column in this symbolic doubly blocked toeplitz should be equal to the number of row in the input img
r = np.r_[c[0], np.zeros(img.shape[0]-1, dtype=int)]
doubly_indices = sl.toeplitz(c,r)
print(doubly_indices)
```

```
[[  1   0   0 ...   0   0   0]
 [  2   1   0 ...   0   0   0]
 [  3   2   1 ...   0   0   0]
 ...
 [129 128 127 ...   3   2   1]
 [130 129 128 ...   4   3   2]
 [131 130 129 ...   5   4   3]]
```

```
# shape of one toeplitz matrix of all
h = toeplitz_list[0].shape[0]*doubly_indices.shape[0]
w = toeplitz_list[0].shape[1]*doubly_indices.shape[1]
doubly_blocked_shape = [h, w]
doubly_blocked = np.zeros(doubly_blocked_shape)


b_h, b_w = toeplitz_list[0].shape
for i in range(doubly_indices.shape[0]):
  for j in range(doubly_indices.shape[1]):
    start_i = i*b_h
    start_j = j*b_w
    end_i = start_i+b_h
    end_j = start_j + b_w
    doubly_blocked[start_i: end_i, start_j: end_j]=toeplitz_list[doubly_indices[i,j]-1]
print(doubly_blocked)
print(doubly_blocked.shape)
```

```
[[ 0.  0.  0. ...  0.  0.  0.]
 [ 1.  0.  0. ...  0.  0.  0.]
 [ 0.  1.  0. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  0. -1.  0.]
 [ 0.  0.  0. ...  0.  0. -1.]
 [ 0.  0.  0. ...  0.  0.  0.]]
(17161, 16641)
```

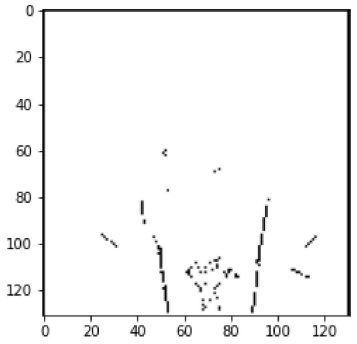
```
# input img matrix to vector
def matrix_to_vector(input):
  input_h, input_w = input.shape
  output_vector = np.zeros(input_h*input_w, dtype=np.float32)
  #flip the input matrix upside down and start from the last row
  input = np.flipud(input)
  for i, row in enumerate(input):
    st = i*input_w
    nd = st+input_w
    output_vector[st:nd]=row
  return output_vector


# converting the result matrix to vector
def matrix_to_vector(input):
  input_h , input_w = input.shape
  output_vector = np.zeros( input_h *input_w , dtype=input.dtype)
  # flipping the input matrix up down
  input = np.flipud(input)
  for i,row in enumerate(input):
    st = i* input_w
    nd = st + input_w
    output_vector [st:nd] = row
  return output_vector


vectorized_input = matrix_to_vector(img)
print(doubly_blocked.shape)
```

```
print(vectorized_input.shape)
print(vectorized_input)
result_vector = np.matmul(doubly_blocked, vectorized_input)
print("result: ", result_vector)
max_val=255
```

```
(17161, 16641)
(16641,)
[ 46  46  47 ... 116 117 117]
result:  [   0.   46.   46. ... -117. -117.    0.]
```

```
def vector_to_matrix(input, output_shape):
  output_h, output_w = output_shape
  output = np.zeros(output_shape, dtype=np.float32)
  for i in range(output_h):
    st = i*output_w
    nd = st+output_w
    output[i,:] = input[st:nd]
  # flipping again down -> up
  output = np.flipud(output)
  return output
out = vector_to_matrix(result_vector,  output_shape)/max_val

# output in matrix form
print('output in matrix form:')
from scipy import signal
result = signal. convolve2d (img, kernel, "full")
print(result, end='\n\n')

# output in img form
print('output in img form:')
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).
output in matrix form:
[[   0. -162. -163. ... -117. -117.    0.]
 [-162.  483.  325. ...  237.  351. -117.]
 [-164.  653.  488. ...  348.  472. -117.]
 ...
 [ -44.  172.  129. ...  113.  159.  -40.]
 [ -46.  228.  181. ...  155.  210.  -42.]
 [   0.   46.   46. ...   40.   42.    0.]]

output in img form:
```




```
# print("Output Image: \n")
# plt.imshow(cv2.cvtColor(out, cv2.COLOR_BGR2RGB))
# plt.show()


# from scipy import signal
# result = signal.convolve2d(img, kernel, "full")/255
# print("Using convolve2D: \n")
# plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
# plt.show()
```

✓ 0s completed at 12:54 PM