-Sai Krishna WDS
Unity ID: swupadr

DIGITAL IMAGING SYSTEMS (ECE 558)- PROJECT-2 REPORT

Question 1A:

Convolving the images using kernels and obtaining the outputs, with padding of 4 varieties.

For this question, My code implements the process of convolution by using the function I have developed called 'Convolve', and it slides the different kernels along the image (which can be given in grayscale or in RGB) and then computes the convolution of the image and the kernel in gray scale or in each of the R,G and B channels and finally merges these outputs to produce the RGB convolved image.

The kernels have all been taken in the form of a 3x3 matrix.

In case the kernels aren't of 3x3 dimension, I've put in zeros in the other indices of the matrix and converted them to 3x3 size and obtained the outputs. The same outputs are obtained because in the end convolution is a point by point multiplication and summing process. The outputs observed by my inbuilt function come out to be the same as the outputs obtained using the inbuilt convolution function *'cv2.filter2D'* , which proves its correctness.

I have attached two source code .py files for this problem- one with the padding and one without the padding functions so as to be able to observe the difference.

I have observed all the outputs with the source code file without padding first.

I have implemented one layer of pixels as the boundary around the image for each type of padding.

I have implemented 4 types of padding:

1. Clip/Zero padding
2. Copy edge padding
3. Wrap around padding
4. Reflect across edge padding

Note: In the case of padding of one layer of pixels, the wrap-around and the copy edge padding are the same.

-Sai Krishna WDS
Unity ID: swupadr

CODE 1: CONVOLUTION WITHOUT MY PADDING FUNCTION

```
from skimage.exposure import rescale_intensity

import numpy as np

import cv2


def convolve(image, kernel):

   # taking the spatial dimensions of the image, along with the spatial dimensions of the kernel

   (iH, iW) = image.shape[:2]

   (kH, kW) = kernel.shape[:2]


   # allocating memory for the output image, taking care to "pad" the borders of the input image so the spatial size is not reduced

   pad = (kW - 1) // 2

   image = cv2.copyMakeBorder(image, pad, pad, pad, pad,

      cv2.BORDER_REPLICATE)

   #Creating a zero pixel image , and then updating the  output of each convolution into this zero pixel image

   output = np.zeros((iH, iW), dtype="float32")


   # looping over the input image, "sliding" the kernel across each (x, y)-coordinate from left-to-right and top to bottom

   for y in np.arange(pad, iH + pad):

         for x in np.arange(pad, iW + pad):

               # extracting the ROI of the image by extracting the *center* region of the current (x, y)-coordinates dimensions

               roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]


               # performing the actual convolution by taking the element-wise multiplicate between the ROI and the kernel, then summing the matrix
```

```
            k = (roi * kernel).sum()


            # storing the convolved value in the output (x,y)- coordinate of the output 
image
            output[y - pad, x - pad] = k


  # rescaling the output image intensity to be in the range [0, 255]
  output = rescale_intensity(output, in_range=(0, 255))
  output = (output * 255).astype("uint8")


  # return the output image
  return output




# defining the different kernels to be used


boxfilter = np.array((
        [1/9, 1/9, 1/9],
        [1/9, 1/9, 1/9],
        [1/9, 1/9, 1/9]))
sobelX = np.array((
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]))
sobelY = np.array((
        [1, 2, 1],
        [0, 0, 0],
        [-1, -2, -1]))
prewittX = np.array((
        [-1, 0, 1],
        [-1, 0, 1],
        [-1, 0, 1]])
prewittY = np.array([
        [1, 1, 1],
        [0, 0, 0],
        [-1, -1, -1]])
rowder = np.array([
        [0, 0 ,0],
    [-1 ,1 ,0],
    [0 ,0 ,0]])
colder = np.array([
        [-1 ,0 ,0],
        [1 ,0 ,0],
    [0 ,0 ,0]])
robertX = np.array([
        [0, 1 ,0],
    [-1 ,0 ,0],
        [0, 0 ,0]])
robertY = np.array([
        [1, 0 ,0],
        [0, -1 ,0],
    [0 ,0 ,0]])



# constructing the kernel bank, a list of kernels that is being applied to the 'convolve'
function and the inbuilt opencv function
kernelBank = (
  ("boxfilter",boxfilter),
   ("sobelX", sobelX),
   ("sobelY", sobelY),
   ("prewittX", prewittX),
   ("prewittY", prewittY),
   ("rowder",rowder),
   ("colder", colder),
   ("robertX", robertX),
   ("robertY", robertY)

)



image = cv2.imread("wolves.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)              # here
b_img=image[:,:,0]
g_img=image[:,:,1]
r_img=image[:,:,2]

# looping over the kernels
for (kernelName, kernel) in kernelBank:
    # applying the kernel to theimage using both the  `convolve` function and OpenCV's
`filter2D` function
   print("[INFO] applying {} kernel".format(kernelName))
   convoleOutputg = convolve(gray, kernel)
   convoleOutput1 = convolve(b_img, kernel)
   convoleOutput2 = convolve(g_img, kernel)
   convoleOutput3 = convolve(r_img, kernel)
   opencvOutputg = cv2.filter2D(gray, -1, kernel)
   opencvOutput1 = cv2.filter2D(b_img, -1, kernel)
   opencvOutput2 = cv2.filter2D(g_img, -1, kernel)
```

```
    opencvOutput3 = cv2.filter2D(r_img, -1, kernel)


    """
    convoleOutput = convolve(gray, kernel)                # here
    opencvOutput = cv2.filter2D(gray, -1, kernel)         # here
    """


    # show the output images
    #cv2.imshow("original", image)                        #here
    cv2.imshow("{}-Gray img conv".format(kernelName), convoleOutputg)
    cv2.imshow("{}-Blue img conv".format(kernelName), convoleOutput1)
    cv2.imshow("{}-Green img conv".format(kernelName), convoleOutput2)
    cv2.imshow("{}-Red img conv".format(kernelName), convoleOutput3)
    #cv2.imshow("{} - opencv".format(kernelName), opencvOutput)
    rgbimg2=cv2.merge((convoleOutput1,convoleOutput2,convoleOutput3)) # Merging
the R,G and B channels and displaying the combined RGB image
    cv2.imshow("{}-Final merge".format(kernelName),rgbimg2)
```

CODE2: WITH MY PADDING FUNCTION

```
from skimage.exposure import rescale_intensity
import numpy as np
import cv2
import matplotlib.pyplot as plt


def convolve(image, kernel,pad):
    # taking the spatial dimensions of the image, along with the spatial dimensions of the
kernel
    (iH, iW) = image.shape[:2]
    (kH, kW) = kernel.shape[:2]
```

```
 # Implementing the padding using the user input


   if pad==1:
      image=zeropad(image)
   elif pad==2:
      image=borderreplicate(image)
   elif pad==3:
      image=borderreplicate(image)  # wrap around edge and border replicate are the
same for 1 layer of padding
   else:
      image=reflectacross(image)



 #Creating a zero pixel image , and then updating the  output of each convolution into
this zero pixel image
 output = np.zeros((iH, iW), dtype="float32")


 # looping over the input image, "sliding" the kernel across each (x, y)-coordinate from
left-to-right and top to bottom


 for y in np.arange(pad, iH + pad):
      for x in np.arange(pad, iW + pad):
            # extracting the ROI of the image by extracting the *center* region of the
current (x, y)-coordinates dimensions


            roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]


            # perform the actual convolution by taking the element-wise multiplicate
between the ROI and the kernel, then summing the matrix
```

```
            k = (roi * kernel).sum()

            # storing the convolved value in the output (x,y)- coordinate of the output
image

            output[y - pad, x - pad] = k

  # rescale the output image to be in the range [0, 255]
  output = rescale_intensity(output, in_range=(0, 255))
  output = (output * 255).astype("uint8")
    # Deleting the padded boundary after convolution:

  # we can just make the boundary pixels blank or white, as shown in the project
problem.

  imgarp=output # where imgarp is the final output image after removing padding

  for k in range(0,3):
    for j in range(1,c+3):
      imgarp.itemset((1,j,k),255)
      imgarp.itemset((r+2,j,k),255)    # deleting the padded rows

  for k in range(0,3):
    for i in range(1,r+3):
      imgarp.itemset((i,1,k),255)
      imgarp.itemset((i,1,k),255)     # deleting the padded columns

  cv2.imshow("Final image after padding removed",imgarp)

  # return the output image
```

```
    return imgarp




# defining the different kernels to be used


boxfilter = np.array((
            [1/9, 1/9, 1/9],
            [1/9, 1/9, 1/9],
            [1/9, 1/9, 1/9]))
sobelX = np.array((
            [-1, 0, 1],
            [-2, 0, 2],
            [-1, 0, 1]))
sobelY = np.array((
            [1, 2, 1],
            [0, 0, 0],
            [-1, -2, -1]))
prewittX = np.array((
            [-1, 0, 1],
            [-1, 0, 1],
            [-1, 0, 1]))
prewittY = np.array((
            [1, 1, 1],
            [0, 0, 0],
            [-1, -1, -1]))
rowder = np.array((
            [0, 0 ,0],
        [-1 ,1 ,0],
        [0 ,0 ,0]))
```

```
colder = np.array((
        [-1 ,0 ,0],
        [1 ,0 ,0],
    [0 ,0 ,0]))
robertX = np.array((
        [0, 1 ,0],
    [-1 ,0 ,0],
        [0, 0 ,0]))
robertY = np.array((
        [1, 0 ,0],
        [0, -1 ,0],
    [0 ,0 ,0]))



# constructing the kernel bank, a list of kernels that is being applied to the 'convolve'
function and the inbuilt opencv function


kernelBank = (
  ("boxfilter",boxfilter),
  ("sobelX", sobelX),
  ("sobelY", sobelY),
  ("prewittX", prewittX),
  ("prewittY", prewittY),
  ("rowder",rowder),
  ("colder", colder),
  ("robertX", robertX),
  ("robertY", robertY)
)
```

```
image = cv2.imread("wolves.png")


print(img.shape)

r=539

c=1500

"""

img=cv2.imread('lena.png')    # Uncomment this region and change the values of the r
and c if the input image is different

print(img.shape)

r=440

c=440

"""




#Zero padding

def zeropad(img):

   X1 = np.random.random((r+2, c+2,3))         # creating a black image of size (r+2) x
(c+2), where r, c are the number of rows and columns in the given image.

   for k in range(0,3):

      for i in range(0,r+2):

         for j in range(0,c+2):

            X1.itemset((i,j,k),0)




   for k in range(0,3):

      for i in range(2,r+1):

         for j in range(2,c+1):

            X1[i,j,k]=img[i-1,j-1,k]

   cv2.imshow('zero-padded image',img)         # X1 is the zero padded image, and we
take that as the model first and proceed with the other types of padding.
```

```
    return img


#Border replicate padding and wrap around padding

def borderreplicate(img):

    X1 = np.random.random((r+2, c+2,3))         # creating a black image of size (r+2) x (c+2), where r, c are the number of rows and columns in the given image.

    for k in range(0,3):

        for i in range(0,r+2):

            for j in range(0,c+2):

                X1.itemset((i,j,k),0)



    for k in range(0,3):

        for i in range(2,r+1):

            for j in range(2,c+1):

                X1[i,j,k]=img[i-1,j-1,k]

    cv2.imshow('zero-padded image',img)         # X1 is the zero padded image, and we take that as the model first and proceed with the other types of padding.

    X2=X1

    for k in range(0,3):

        for j in range(2,c+2):

            X2[1,j,k]=img[1,j-1,k]

            X2[r+2,j,k]=img[r,j-1,k]   # padding the lower and upper row with copy edge method

    for k in range(0,3):               # this is the same as wrap around padding when we consider only one layer of padding of rows and columns

        for i in range(1,r+3):

            X2[i,c+2,k]=X2[i,c+1,k]

            X2[i,1,k]=X2[i,2,k]         # X2 is the copy across padded image, and we can reflect the first and last columns of this image and

    cv2.imshow("Copy edge padding",X2)   # we can obtain the reflect across the edge padding.
    return X2


#Reflect across padding

def reflectacross(img):

    X1 = np.random.random((r+2, c+2,3))        # creating a black image of size (r+2) x
(c+2), where r, c are the number of rows and columns in the given image.

    for k in range(0,3):

        for i in range(0,r+2):

            for j in range(0,c+2):

                X1.itemset((i,j,k),0)




    for k in range(0,3):

        for i in range(2,r+1):

            for j in range(2,c+1):

                X1[i,j,k]=img[i-1,j-1,k]

    cv2.imshow('zero-padded image',img)        # X1 is the zero padded image, and we
take that as the model first and proceed with the other types of padding.

    X2=X1

    for k in range(0,3):

        for j in range(2,c+2):

            X2[1,j,k]=img[1,j-1,k]

            X2[r+2,j,k]=img[r,j-1,k]   # padding the lower and upper row with copy edge
method

    for k in range(0,3):               # this is the same as wrap around padding when we
consider only one layer of padding of rows and columns

        for i in range(1,r+3):

            X2[i,c+2,k]=X2[i,c+1,k]

            X2[i,1,k]=X2[i,2,k]        # X2 is the copy across padded image, and we can reflect
the first and last columns of this image and
```

```
   cv2.imshow("Copy edge padding",X2)   # we can obtain the reflect across the edge padding.

   X3=X2

   randrow=np.random.random((1,c+2,3))  # create a random one row image to store the values of the pixels of the first row and then exchange the values with the last row


   for k in range(0,3):

      for j in range(1,c+3):

         randrow[1,j,k]=X3[1,j,k]

         X3[1,j,k]=X3[r+2,j,k]

         X3[r+2,j,k]=randrow[1,j,k]




   randcol=np.random.random((r+2,1,3))  # create a random one column image to store the values of the pixels of the first column  and then exchange the values with the last row


   for k in range(0,3):

      for i in range(1,r+3):

         randcol[i,1,k]=X3[i,1,k]

         X3[i,1,k]=X3[i,c+2,k]

         X3[i,c+2,k]=randcol[i,1,k]

   cv2.imshow("reflect across padding",X3)

   return X3


###############################################################################


gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)             # here

b_img=image[:,:,0]
```

```
g_img=image[:,:,1]

r_img=image[:,:,2]


# loop over the kernels

for (kernelName, kernel) in kernelBank:

        # applying the kernel to theimage using both the  `convolve` function and OpenCV's
`filter2D` function


    p=int(input("Enter 1 for zero padding, 2 for border replicate padding, 3 for wrap
around padding, and 4 for reflect across padding"))

    print(" applying {} kernel".format(kernelName))

    convoleOutputg = convolve(gray, kernel,p)

    convoleOutput1 = convolve(b_img, kernel,p)

    convoleOutput2 = convolve(g_img, kernel,p)

    convoleOutput3 = convolve(r_img, kernel,p)

##    opencvOutputg = cv2.filter2D(gray, -1, kernel)

##    opencvOutput1 = cv2.filter2D(b_img, -1, kernel)

##    opencvOutput2 = cv2.filter2D(g_img, -1, kernel)

##    opencvOutput3 = cv2.filter2D(r_img, -1, kernel)

##

    """

    convoleOutput = convolve(gray, kernel)                # here

    opencvOutput = cv2.filter2D(gray, -1, kernel)          # here

    """


    # show the output images

    #cv2.imshow("original", image)                          #here

    cv2.imshow("{}-Gray img conv".format(kernelName), convoleOutputg)

    cv2.imshow("{}-Blue img conv".format(kernelName), convoleOutput1)

    cv2.imshow("{}-Green img conv".format(kernelName), convoleOutput2)

    cv2.imshow("{}-Red img conv".format(kernelName), convoleOutput3)
    #cv2.imshow("{} - opencv".format(kernelName), opencvOutput)


    rgbimg2=cv2.merge((convoleOutput1,convoleOutput2,convoleOutput3))
    cv2.imshow("{}-Final merge".format(kernelName),rgbimg2)
```

OUTPUT IMAGES:

The output convoluted images-labelled for each filter have been attached in the zipped folder- for both the 'Lena.png' and the 'wolves.png' sample images.

Question 1B:

Convolving a 1024x1024 dark image with a unit impulse at the centre (512,512) alone.

We pass the generated image of 1024x1024 through the 'convolve' function used for the first part of the question. I have used the box filter kernel to study what happens to this image, after it is convolved with it.

CODE:

```
from skimage.exposure import rescale_intensity
import numpy as np
import cv2
import matplotlib.pyplot as plt


def convolve(image, boxfilter):


    (iH, iW) = image.shape[:2]
    (kH, kW) = boxfilter.shape[:2]



    pad = (kW - 1) // 2
    image = cv2.copyMakeBorder(image, pad, pad, pad, pad,
        cv2.BORDER_REPLICATE)
    output = np.zeros((iH, iW), dtype="float32")
    for y in np.arange(pad, iH + pad):
        for x in np.arange(pad, iW + pad):

            roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]



            k = (roi * boxfilter).sum()



            output[y - pad, x - pad] = k



    output = rescale_intensity(output, in_range=(0, 255))
    output = (output * 255).astype("uint8")



    return output
```

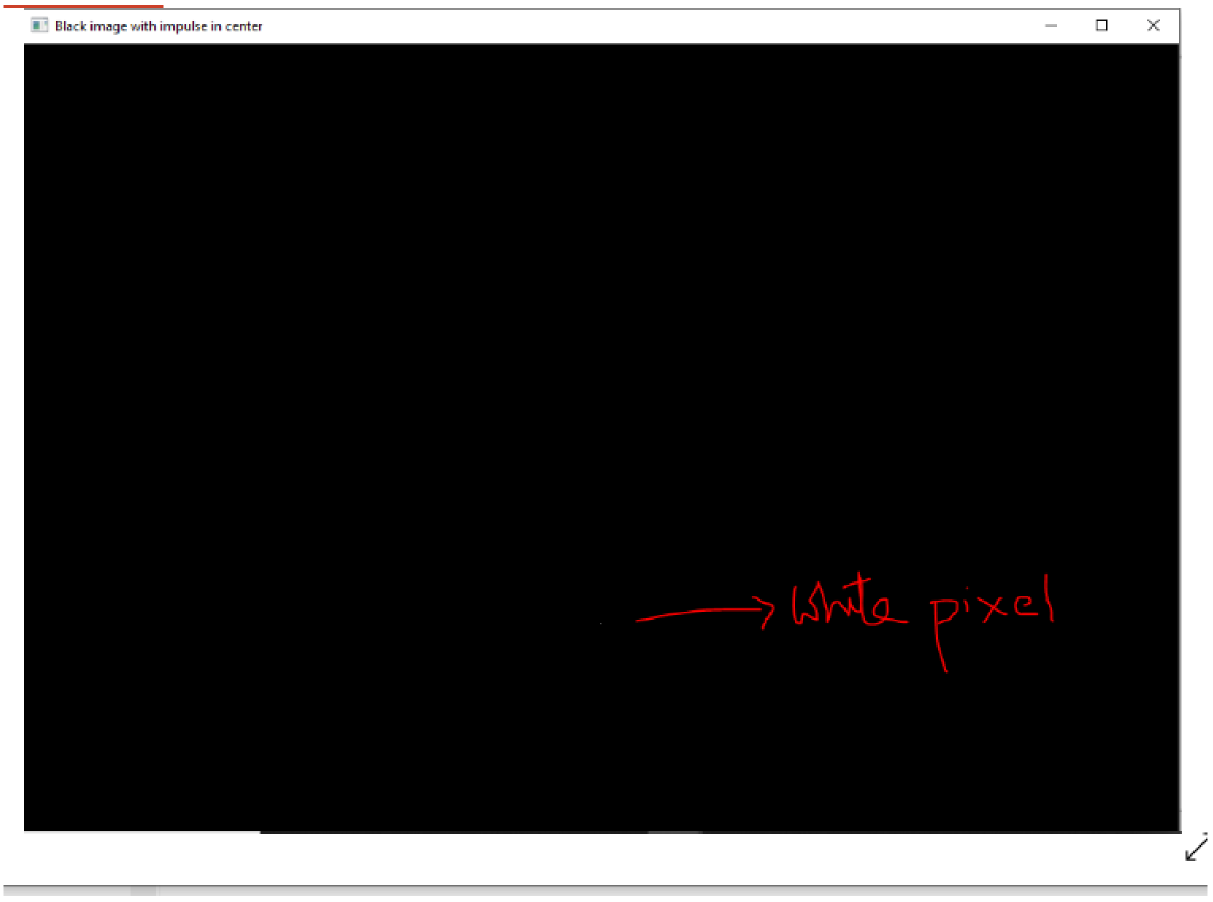
```
# using the box filter to see effects of convolution of a 1024x1024 image with impulse at the center.
boxfilter = np.array((
        [1/9, 1/9, 1/9],
        [1/9, 1/9, 1/9],
        [1/9, 1/9, 1/9]))
```

-Sai Krishna WDS
Unity ID: swupadr

```
X = np.random.random((1024, 1024)) # sample B&W image of size 1024 x 1024
plt.imshow(X, cmap="gray")



print(X.shape)


for i in range(0,1024):
   for j in range(0,1024):
      if i==512 and j==512:
         X.itemset(i,j,255)
      else:
         X.itemset(i,j,0)



cv2.imshow("Black image with impulse in center",X)
print(X.shape)




convoleOutputX = convolve(X, boxfilter)


cv2.imshow("X img conv", convoleOutputX)
```

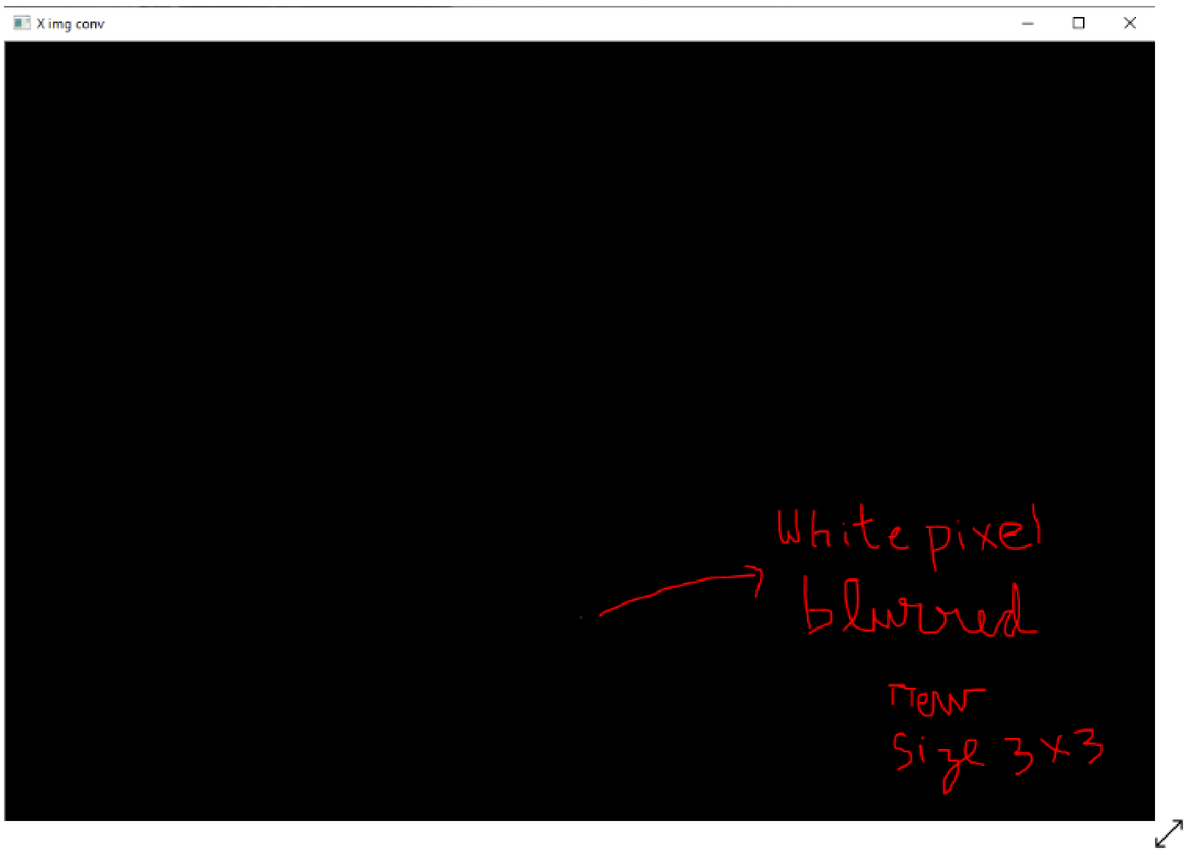
-Sai Krishna WDS
Unity ID: swupadr

OUTPUT IMAGES:



INPUT IMAGE

-Sai Krishna WDS
Unity ID: swupadr



OUTPUT IMAGE

The input image as visualized in the output of the code, is a completely dark with a white pixel intensity (255) in the centre.

The box filter is actually a blurring filter that averages the values of all the pixels present in the 3x3 kernel's vicinity and spreads it to all the pixels in the 3x3 area.

Now, after the image has been convolved with a box filter, we notice that the 1 pixel that was white got spread out and we see a 3x3 pixels box in the center sharing the intensity of the 1 pixel after convolution.

-Sai Krishna WDS
Unity ID: swupadr

Question 2A:

Visualising the magnitude and phase spectrum of an image after constructing a 2D DFT function using 1D DFT methods.

I have implemented this code in Matlab.

CODE:

```
 clc
I=imread('wolves.png');

%I=imread('lena.png');

%imshow(I)
whos I % we see that the image is of the size 440 x 440 x 3
%title('original Image')

f=rgb2gray(I);
figure(1)
imshow(f)
title('Black and white Image')
% The derived transformation we got was: Y=[(L-1)/x1]
% NOw, let us consider the example of L1=1 as we need all pixels scaled from 0
to 1; and x1=255 as always
L=1
x1=255
J=[(L)/x1]*(f);
figure(2)
imshow(J)
title('Transformed image for L between 0 and 1 ')
figure(3)
imhist(J)
title('cross check for intensity bet 0 and 1')
%
%
%  b=size(f)
c=440
%
r=440
%
% for i=(1:columns)
%     I2(:,i)=fft(I2(:,i));
% end


% for i=(1:rows)
%     I2(i,:)=fft(I2(i,:));
% end
```

```
Y = fft2(J);
% ok let's first take the DFT along the rows
I2_afterrow_dft = fft(J,[],2);
% now take it along the columns
I2_aftercol_dft = fft(I2_afterrow_dft,[],1);
figure(4)
imshow(I2_aftercol_dft)
title('My 2Dfft logic using 1D fft ')
max(max(abs(I2_aftercol_dft-Y)))
F=I2_aftercol_dft;
fshift=fftshift(F);
s=(1/20)*log(abs(fshift));
figure(5)
imshow(s)
title('magnitude spectrum')
figure(6)
imshow(angle(fshift))
title('Phase spectrum')
figure(7)
imshow(fft2(J))
title(' 2D fft using inbuilt function ')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   PART- a) Code
Completed%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

OUTPUT IMAGES:

They have been enclosed in the zipped folder.

Question 2B:

Implementing the 2D IDFT using the forward method of the 1D DFT again.

I have implemented this code in matlab as well.

CODE:

```
clc
%I=imread('lena.png');
I=imread('wolves.png');


%imshow(I)
whos I                                    % we see that the image is of the size 440
x 440 x 3
%title('original Image')


f=im2double(rgb2gray(I));
```

-Sai Krishna WDS
Unity ID: swupadr

```
figure(1)
imshow(f)
title('input image')
c=440
r=440




% ok let's first take the DFT along the rows
f_afterrow_dft = fft(f,[],2);
% now take it along the columns
f_aftercol_dft = fft(f_afterrow_dft,[],1);
F=f_aftercol_dft;
Fdash=conj(F);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


f_afterrow_dft = fft(Fdash,[],2);
% now take it along the columns
f_aftercol_dft = fft(f_afterrow_dft,[],1);
Fnew=f_aftercol_dft;
Fnewdash=conj(Fnew);
fin=Fnewdash/(r*c);
figure(2)
imshow(fin)
title(' IDFT image obtained using the 1D DFT method ')
figure(3)
imshow(f-fin)
title("Dark image '0' obtained on doing f-g as expected")
```

OUTPUT IMAGES:

Have been enclosed in the zipped folder.