Project 1 : CSE 473/573

Disha Mehra

Person No. :50288911

Instructor: Prof. Junsong Yuan

Task 1

Code for task 1:

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 29 04:03:49 2018

@author: 12144
"""

import cv2
import numpy as np
from matplotlib import pyplot as plt

def edge_detection(img_path):
    image = cv2.imread(img_path,0)
    cv2.namedWindow('Original image', cv2.WINDOW_NORMAL)
    cv2.imshow('Original image', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    sizey, sizex = image.shape

    h=3//2 #As kernel is 3*3
    w=3//2

    sobelimagey = image.copy()
    img_1 = image.copy()

    kernel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype = np.int8) #Sobel kernel along y-axis


    for i in range(h,sizey-h):
        for j in range(w,sizex-w):
            val=0

            for m in range(3):
                for n in range(3):
                    val=val+ kernel_y[m][n]*img_1[i-h+m][j-w+n]


            if val > 255:
                val = 255
            elif val < 0:
                val = 0

            sobelimagey[i][j]=val

    cv2.namedWindow('New image y', cv2.WINDOW_NORMAL)
    cv2.imshow('New image y',sobelimagey)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


    sobelimagex = image.copy()
```

```
img_2=image.copy()
img_final=image.copy()

kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype = np.int8)#Sobel kernel along x axis

for i in range(h,sizey-h):
   for j in range(w,sizex-w):
      val=0

      for m in range(3):
         for n in range(3):
            val=val+ kernel_x[m][n]*img_2[i-h+m][j-w+n]


      if val > 255:
         val = 255
      elif val < 0:
         val = 0

      sobelimagex[i][j]=val

cv2.namedWindow('New image x', cv2.WINDOW_NORMAL)
cv2.imshow('New image x',sobelimagex)
cv2.waitKey(0)
cv2.destroyAllWindows()


for i in range(sizey):
   for j in range(sizex):
      q=np.sqrt(sobelimagex[i][j] ** 2 + sobelimagey[i][j] ** 2)
      if(q>255):
         img_final[i][j]=255
      elif(q<0):
         img_final[i][j]=0

      img_final[i][j]=q


cv2.namedWindow('Final image', cv2.WINDOW_NORMAL)
cv2.imshow('Final image',img_final)
cv2.waitKey(0)
cv2.destroyAllWindows()

plt.subplot(2,2,1),plt.imshow(image,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(sobelimagex,cmap = 'gray')
plt.title('SobelX'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelimagey,cmap = 'gray')
plt.title('SobelY'), plt.xticks([]), plt.yticks([])

replicate = cv2.copyMakeBorder(img_final,10,10,10,10,cv2.BORDER_REPLICATE)
plt.subplot(2,2,4),plt.imshow(replicate,cmap = 'gray')
plt.title('Final Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

Figure 1:

Depicting the gray scale image of original image:

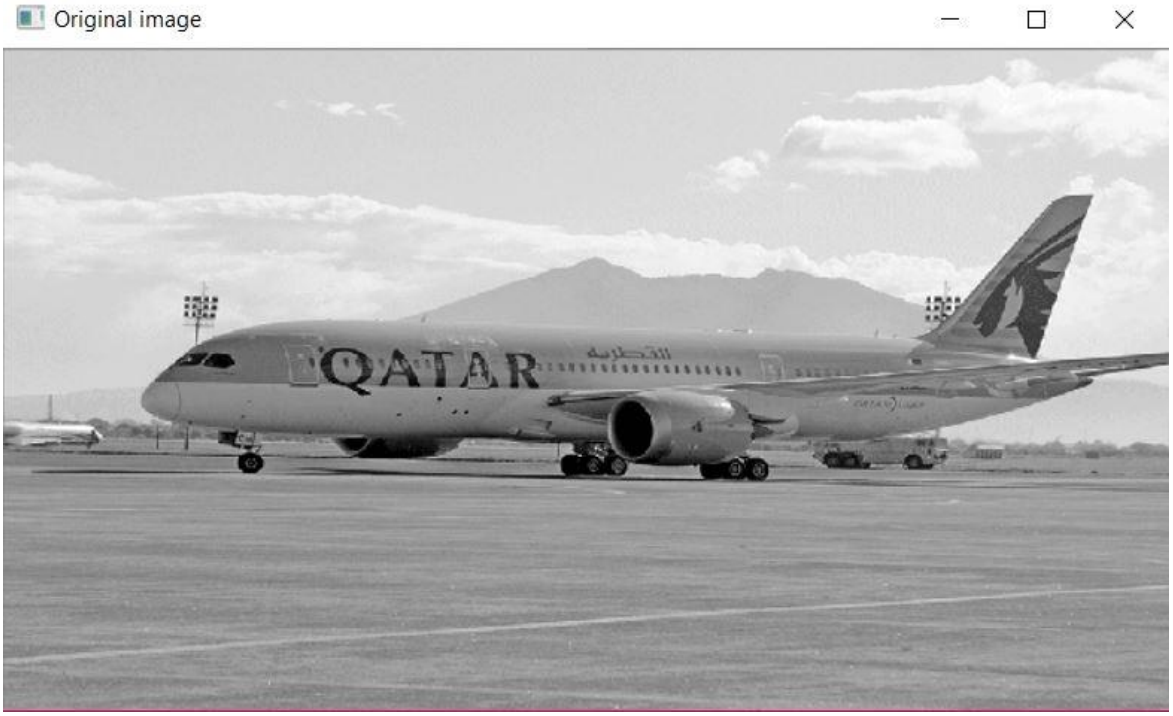


Figure 2:

Depicting the y-axis image after applying Sobel kernel

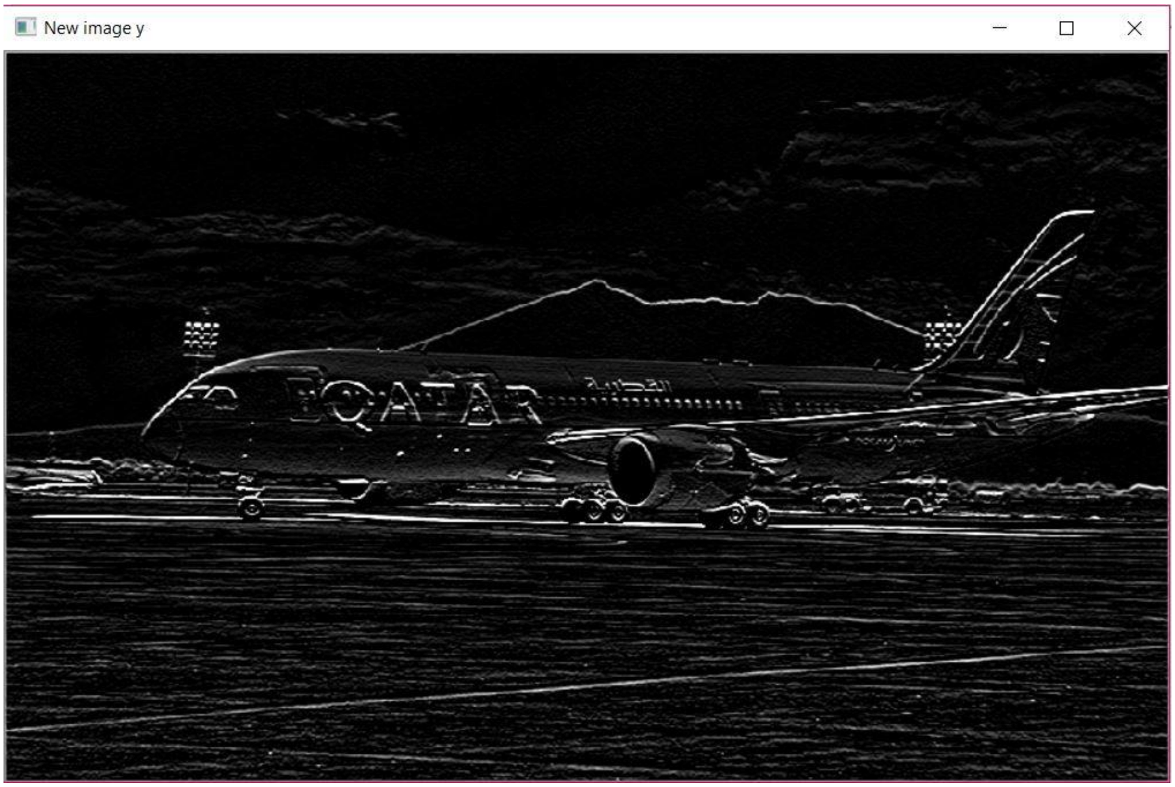


Figure 3:

Depicting the x-axis image after applying Sobel kernel

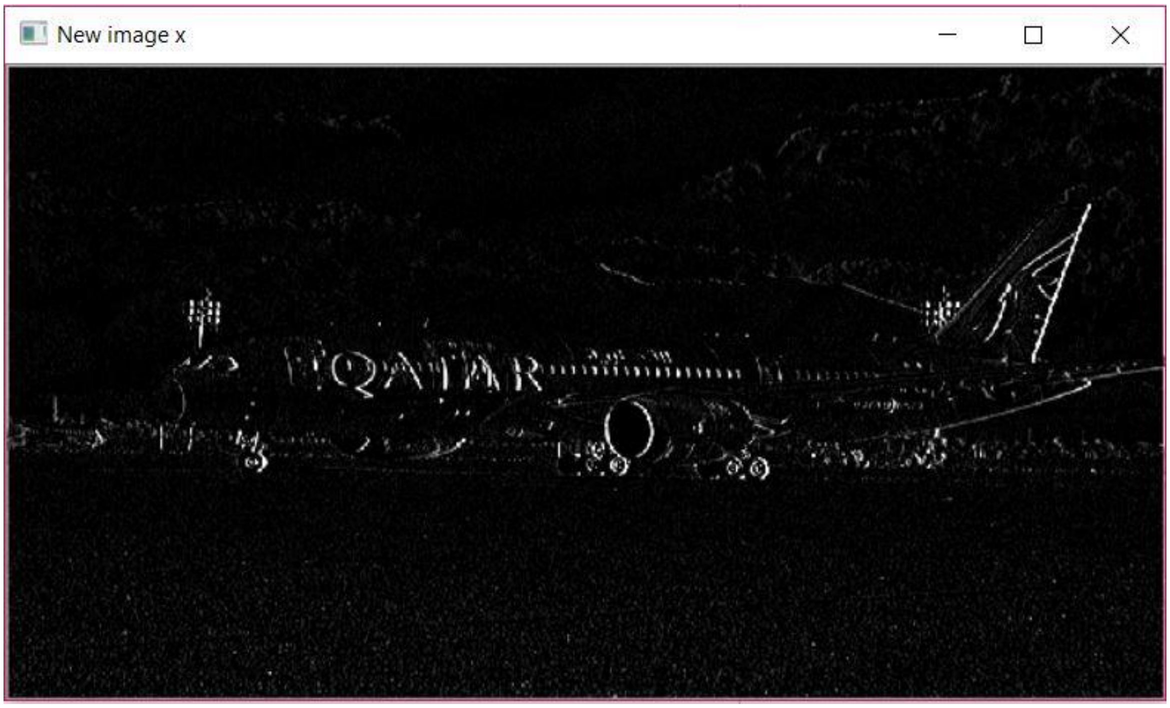
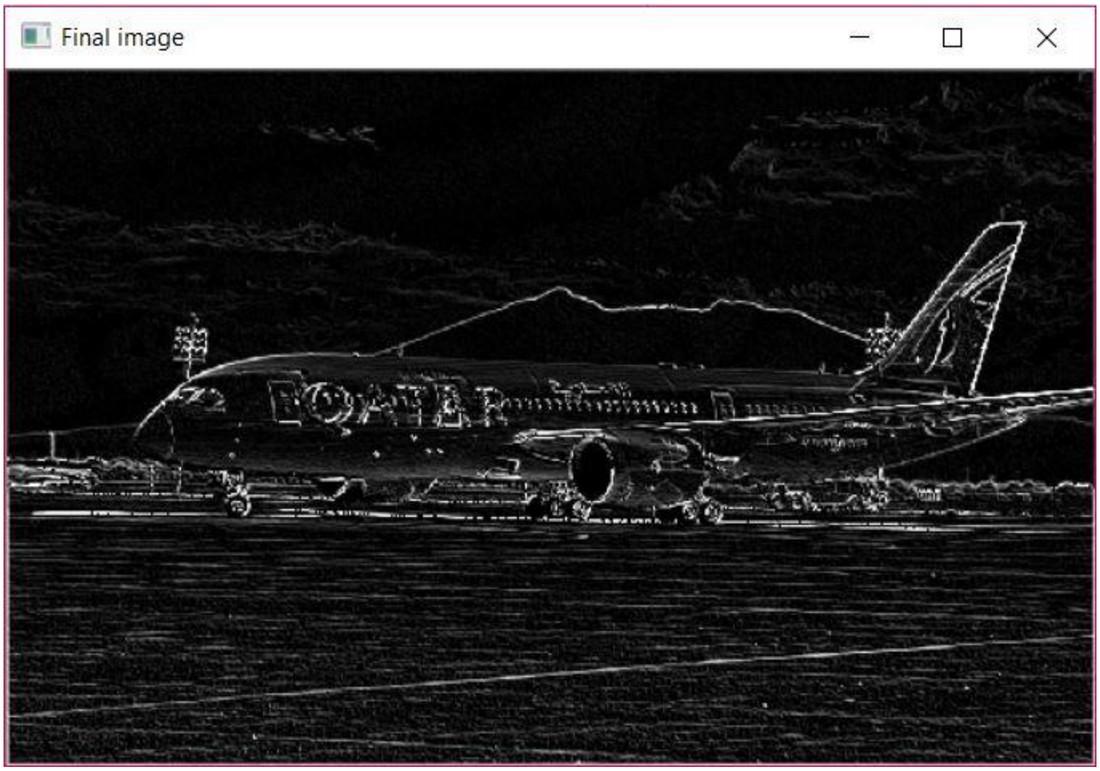


Figure 4:

Depicting the image after combining both x-axis and y-axis edge detection



Task2:

Code:

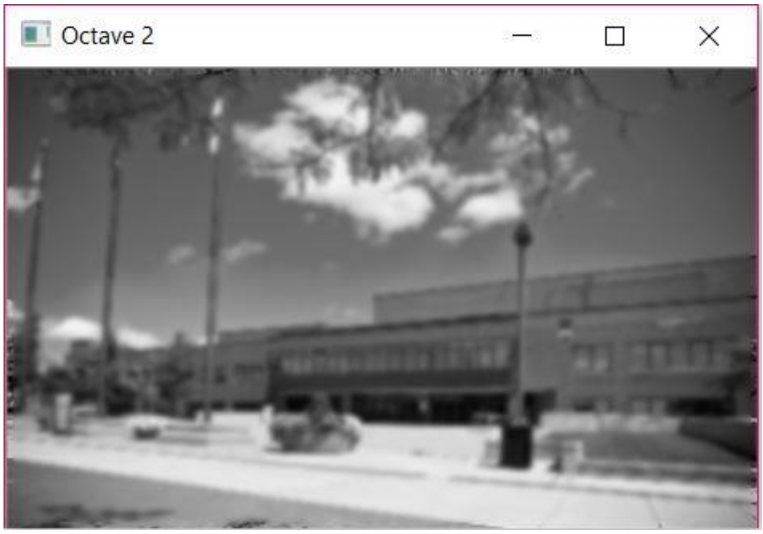
```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  6 16:15:35 2018

@author: disha
"""

import cv2
import numpy as np

def octave(img_path):
    img= cv2.imread(img_path,cv2.IMREAD_GRAYSCALE)

    def compute(image,kernel):
        img_h=image.shape[0]
        img_w=image.shape[1]
        img_new=image.copy()
        for i in range(3,img_h-3):
            for j in range(3,img_w-3):
                val1=0
                for m in range(-3,4):
                    for n in range(-3,4):
                        val1=val1+image[i+m][j+n]*kernel[3+m][3+n]
                img_new[i,j]=val1
        return img_new

    def gaussian_kernel(size,sigma_val):
        ax = np.arange(-size // 2 + 1., size // 2 + 1.)
        xx, yy = np.meshgrid(ax, ax)
        kernel = np.exp(-(xx**2 + yy**2) / (2. * sigma_val**2))
        return kernel / np.sum(kernel)

    first_oct_sigma_values=[(1/(np.sqrt(2))),1,np.sqrt(2),2,2*(np.sqrt(2))]
    sec_oct_sigma_values=[np.sqrt(2),2,2*(np.sqrt(2)),4,4*(np.sqrt(2))]
    third_oct_sigma_values=[2*(np.sqrt(2)),4,4*(np.sqrt(2)),8,8*(np.sqrt(2))]
    fourth_oct_sigma_values=[4,4*(np.sqrt(2)),8,8*(np.sqrt(2)),16,16*(np.sqrt(2))]

    Octave_final=[]
    #For first octave:

    kerns_1=[]
    gauss_conv_1=[]
    DoG_1_oct=[]

    for sigma_1 in first_oct_sigma_values:
        print(sigma_1)
        kerns_1.append(gaussian_kernel(7,sigma_1))

    for i in range(len(kerns_1)):
        img_out_1=img.copy()
        gauss_conv_1.append(compute(img_out_1,kerns_1[i]))
```

```
for i in range(len(gauss_conv_1)-1):
    DoG_1_oct.append(gauss_conv_1[i]-gauss_conv_1[i+1])


#For second octave:

kerns_2=[]
gauss_conv_2=[]
DoG_2_oct=[]
res_gauss_2=[]
res_dog_2=[]

for sigma_2 in sec_oct_sigma_values:
    print(sigma_2)
    kerns_2.append(gaussian_kernel(7,sigma_2))

for i in range(len(kerns_2)):
    img_out_2=img_out_1[::2,::2]
    gauss_conv_2.append(compute(img_out_2,kerns_2[i]))
    res_gauss_2.append(gauss_conv_2[i].shape)
print('Resolution of Octave 2')
print(res_gauss_2[1])

for i in range(len(gauss_conv_2)-1):
    DoG_2_oct.append(gauss_conv_2[i]-gauss_conv_2[i+1])



#For third octave:

kerns_3=[]
gauss_conv_3=[]
DoG_3_oct=[]
res_gauss_3=[]
res_dog_3=[]


for sigma_3 in third_oct_sigma_values:
  #  print(sigma_3)
    kerns_3.append(gaussian_kernel(7,sigma_3))

for i in range(len(kerns_3)):
    img_out_3=img_out_2[::2,::2]
    gauss_conv_3.append(compute(img_out_3,kerns_3[i]))
    res_gauss_3.append(gauss_conv_3[i].shape)
print('Resolution of Octave 3')
print(res_gauss_3[1])

for i in range(len(gauss_conv_3)-1):
    DoG_3_oct.append(gauss_conv_3[i]-gauss_conv_3[i+1])

#For fourth octave:

kerns_4=[]
gauss_conv_4=[]
DoG_4_oct=[]
```

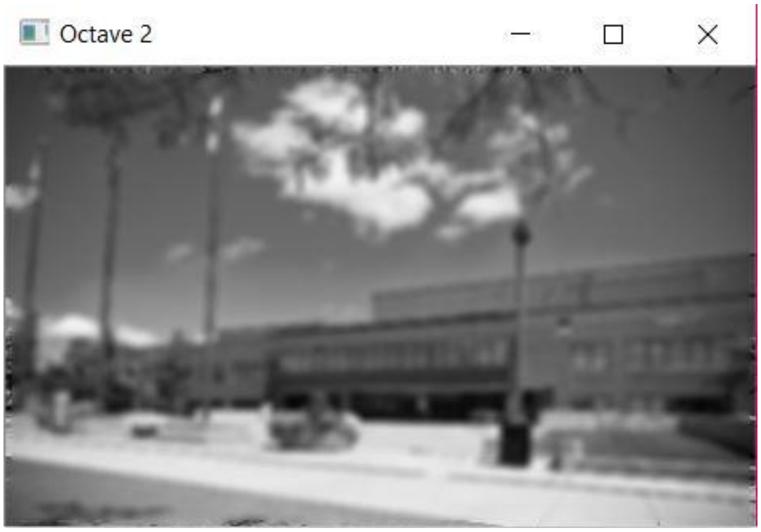
```
for sigma_4 in fourth_oct_sigma_values:
  #  print(sigma_4)
    kerns_4.append(gaussian_kernel(7,sigma_4))

for i in range(len(kerns_4)):
    img_out_4=img_out_3[::2,::2]
    gauss_conv_4.append(compute(img_out_4,kerns_4[i]))


for i in range(len(gauss_conv_4)-1):
    DoG_4_oct.append(gauss_conv_4[i]-gauss_conv_4[i+1])


for i in range(5):
    cv2.imshow('Octave 2',gauss_conv_2[i])
    cv2.waitKey(0)
    cv2.destroyAllWindows()

for i in range(5):
    cv2.imshow('Octave 3',gauss_conv_3[i])
    cv2.waitKey(0)
    cv2.destroyAllWindows()

for i in range(4):
    cv2.imshow('DoG_2',DoG_2_oct[i])
    cv2.waitKey(0)
    cv2.destroyAllWindows()

for i in range(4):
    cv2.imshow('DoG_3',DoG_3_oct[i])
    cv2.waitKey(0)
```

Octave 2:

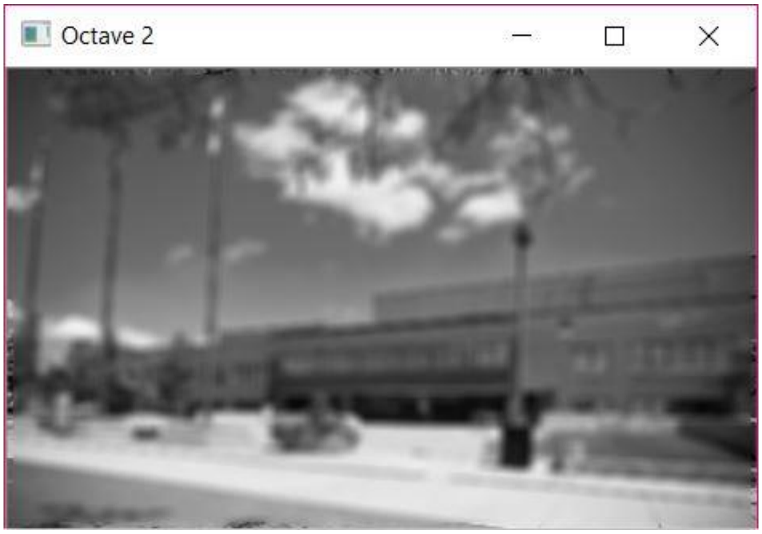
Blurred image when sigma is sqrt(2):



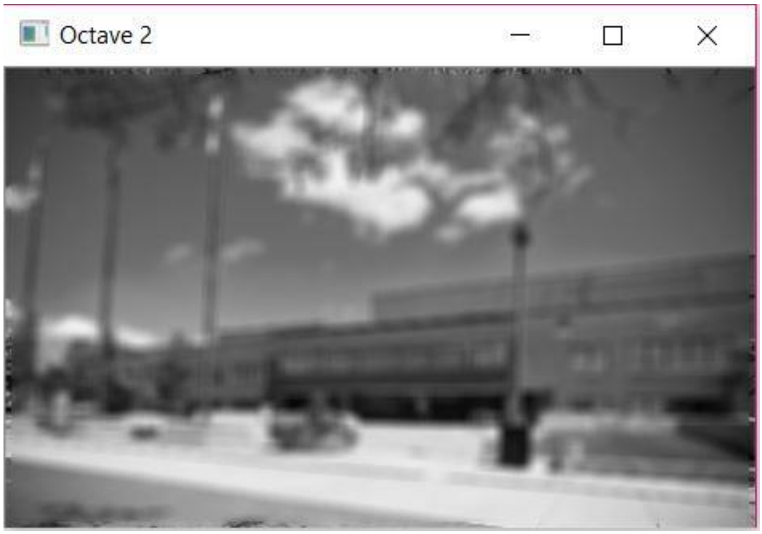
Blurred image when sigma is 2:



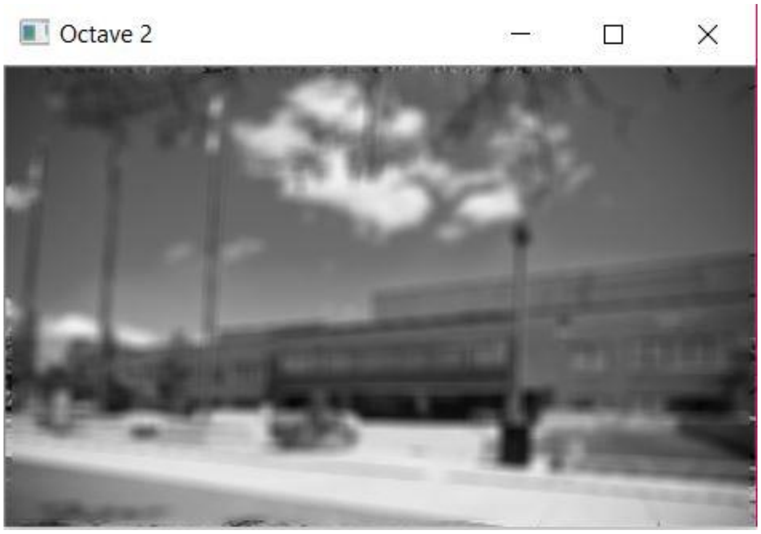
Blurred image when sigma is 2*sqrt(2):



Blurred image when sigma is 4:

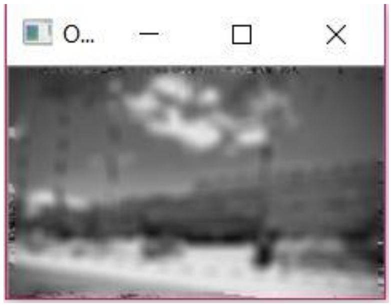


Blurred image when sigma is 4*sqrt(2):

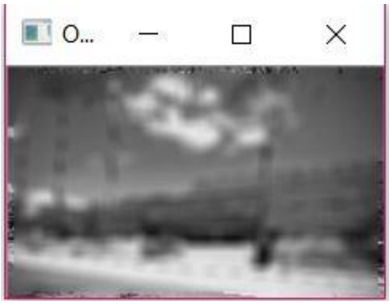


Octave 3:

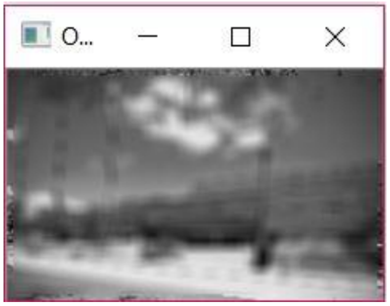
Blurred image when sigma is 2*sqrt(2):



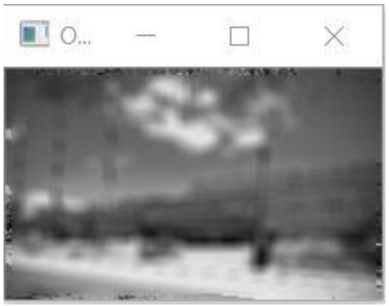
Blurred image when sigma is 4:



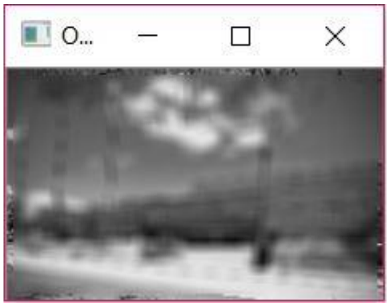
Blurred image when sigma is 4*sqrt(2):



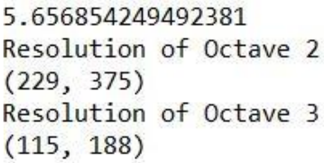
Blurred image when sigma is 8:



Blurred image when sigma is 8*sqrt(2):



Resolution of the Octave 2 and Octave 3:



```
5.656854249492381
Resolution of Octave 2
(229, 375)
Resolution of Octave 3
(115, 188)
```

Task 3:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct  3 19:26:28 2018

@author: disha
"""

import numpy as np
import cv2

def cursor_detection(img_path,template_path):
    img = cv2.imread(img_path)

    blur_img = cv2.GaussianBlur(img,(3,3),0)

    cv2.namedWindow('Temp image', cv2.WINDOW_NORMAL)
    cv2.imshow('Temp image',blur_img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    img_gray= cv2.cvtColor(blur_img, cv2.COLOR_BGR2GRAY)

    img_input=cv2.Laplacian(img_gray,cv2.CV_32F)

    template = cv2.imread(template_path)

    cv2.namedWindow('Temp image', cv2.WINDOW_NORMAL)
    cv2.imshow('Temp image',template)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    w= template.shape[0]
    h= template.shape[1]

    blur_templ= cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
    img_temp=cv2.Laplacian(blur_templ,cv2.CV_32F)

    cv2.namedWindow('Temp image', cv2.WINDOW_NORMAL)
    cv2.imshow('Temp image',img_input)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    cv2.namedWindow('Temp blur', cv2.WINDOW_NORMAL)
    cv2.imshow('Temp blur',img_temp)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    res=cv2.matchTemplate(img_input,img_temp,cv2.TM_CCORR_NORMED)
    print(res)
    cv2.namedWindow('Result image', cv2.WINDOW_NORMAL)
    cv2.imshow('Result image',res)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

    print(max_val)
    if max_val>0.35:
        top_left = max_loc
        bottom_right = (top_left[0] + w, top_left[1] + h)
        cv2.rectangle(img,top_left, bottom_right,(255,0,0),4)

    cv2.namedWindow('Detected', cv2.WINDOW_NORMAL)
    cv2.imshow('Detected',img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

There are some obvious flaws in template matching as a tool for object recognition:

1.If you don't have any matching object in image, you will still get a match, corresponding to max of templating matching method.

2.This matching is affine variant: a change in shape/size/shear etc. of object in image with respect to template will give a false match or no match at all.

3.The calculation is highly inefficient computationally.

Here I have tried and improvised the code of template matching by applying a Gaussian Blur to the image and then detect the edges in the blurred image. Further finding the edge in the template and then applying the template matching.

Though the efficiency is better but still it can’t detect all the images properly.

References:

1. https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html
2. https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html
3. https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html
4. http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/
5. https://stackoverflow.com/questions/43373521/how-to-do-convolution-matrix-operation-in-numpy