Shraddha Dwivedi

UB Person Number: 50290266

Computer Science and Engineering

University at Buffalo

[sdwivedi@buffalo.edu](mailto:sdwivedi@buffalo.edu)

**Edge Detection**

**Source code for Edge Detection**

import cv2

import numpy as np

import math as math

imgPath = 'C:/Users/Shraddha/Desktop/CVIP/Project\_1/task1.png'

SxKernel = (1/8) \* ( np.array( [ [1,0,-1], [2,0,-2], [1,0,-1] ] ))

SyKernel = (1/8) \* ( np.array( [ [-1,-2,-1], [0,0,0], [1,2,1] ] ))

def detectImage(img, outImageName):

#imwrite() save a image to a specified file

cv2.imwrite('C:/Users/Shraddha/Desktop/CVIP/Project\_1/OutputImage/Task1'+outImageName+'.jpg', img)

return 1

def convolve\_image(img, kernel):

sizeOfKernel = (int)((kernel.shape[0])/2)

img\_conv = np.zeros((img.shape[0], img.shape[1]))

for i in np.arange( sizeOfKernel, ((img.shape[0])-sizeOfKernel) ):

for j in np.arange( sizeOfKernel, ((img.shape[1])-sizeOfKernel) ):

temp = 0

for x in np.arange(-1, 2):

for y in np.arange(-1, 2):

coordinateOfKernel = kernel[x+sizeOfKernel, y+sizeOfKernel]

coordinateOfImg = img[i-x, j-y]

temp = temp + (coordinateOfImg\*coordinateOfKernel)

img\_conv[i][j] = temp

return img\_conv

#method 1

def zeros\_elimination\_m1(img\_conv):

outputImage = np.zeros((img\_conv.shape[0], img\_conv.shape[1]))

min\_Value = (img\_conv[0,0])

max\_Value = (img\_conv[0,0])

for i in np.arange( 0, img\_conv.shape[0] ):

for j in np.arange( 0, img\_conv.shape[1] ):

if (min\_Value>(img\_conv[i, j])):

min\_Value=(img\_conv[i, j])

if (max\_Value<(img\_conv[i, j])):

max\_Value=(img\_conv[i, j])

for i in np.arange( 0, img\_conv.shape[0] ):

for j in np.arange( 0, img\_conv.shape[1] ):

outputImage[i, j] = 255.0 \* ( (img\_conv[i, j] - min\_Value) / (max\_Value - min\_Value) )

return outputImage

#method 2

def zeros\_elimination\_m2(img\_conv):

outputImage = np.zeros((img\_conv.shape[0], img\_conv.shape[1]))

max\_Value = (img\_conv[0,0])

for i in np.arange( 0, img\_conv.shape[0] ):

for j in np.arange( 0, img\_conv.shape[1] ):

if (max\_Value<img\_conv[i, j]):

max\_Value=img\_conv[i, j]

for i in np.arange( 0, img\_conv.shape[0] ):

for j in np.arange( 0, img\_conv.shape[1] ):

outputImage[i, j] = 255.0 \* ( img\_conv[i, j] / max\_Value )

return outputImage

def SxSyKernelMerge(Sx\_conv\_img, Sy\_conv\_img):

convoluted\_img = np.zeros((Sx\_conv\_img.shape[0], Sx\_conv\_img.shape[1]))

for i in np.arange( convoluted\_img.shape[0] ):

for j in np.arange( convoluted\_img.shape[1] ):

convoluted\_img[i, j] = math.sqrt( ((Sx\_conv\_img[i, j])\*\*2) + ((Sy\_conv\_img[i, j])\*\*2) )

return convoluted\_img

def main():

# imread loads an image from the specified file and returns it

img = cv2.imread(imgPath,0)

#Edge Detection using Sobel kernel X

Sx\_conv\_img = convolve\_image(img, SxKernel)

Sx\_conv\_img\_abs = np.zeros((Sx\_conv\_img.shape[0], Sx\_conv\_img.shape[1]))

for i in range(0, Sx\_conv\_img.shape[0]):

for j in range(0, Sx\_conv\_img.shape[1]):

Sx\_conv\_img\_abs[i][j]=abs(Sx\_conv\_img[i][j])

# Edge Detection using Sobel kernel Y

Sy\_conv\_img = convolve\_image(img, SyKernel)

Sy\_conv\_img\_abs = np.zeros((Sy\_conv\_img.shape[0], Sy\_conv\_img.shape[1]))

for i in range(0, Sy\_conv\_img.shape[0]):

for j in range(0, Sy\_conv\_img.shape[1]):

Sy\_conv\_img\_abs[i][j]=abs(Sy\_conv\_img[i][j])

detectImage(Sx\_conv\_img\_abs, 'Sx\_conv\_img\_abs')

Sx\_conv\_img\_M1 = zeros\_elimination\_m1(Sx\_conv\_img\_abs)

detectImage(Sx\_conv\_img\_M1, 'Sx\_conv\_img\_M1')

Sx\_conv\_img\_M2 = zeros\_elimination\_m2(Sx\_conv\_img\_abs)

detectImage(Sx\_conv\_img\_M2, 'Sx\_conv\_img\_M2')

detectImage(Sy\_conv\_img\_abs, 'Sy\_conv\_img\_abs')

Sy\_conv\_img\_M1 = zeros\_elimination\_m1(Sy\_conv\_img\_abs)

detectImage(Sy\_conv\_img\_M1, 'Sy\_conv\_img\_M1')

Sy\_conv\_img\_M2 = zeros\_elimination\_m2(Sy\_conv\_img\_abs)

detectImage(Sy\_conv\_img\_M2, 'Sy\_conv\_img\_M2')

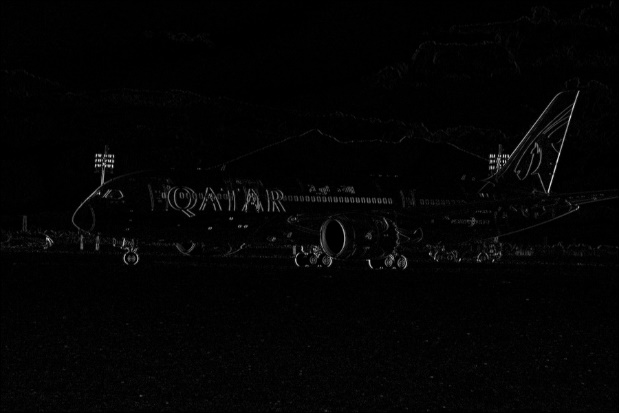
#merging the Sobel kernel X and Sobel kernel Y's output images to get the final image

finalImage = SxSyKernelMerge(Sx\_conv\_img\_abs, Sy\_conv\_img\_abs)

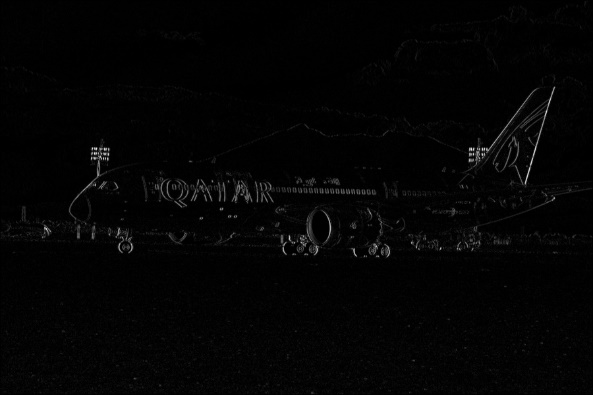
detectImage(finalImage, 'finalImage')

main()

**Image Results for Edge Detection**



**Fig 1 :** Edge detection along x- direction using method 1



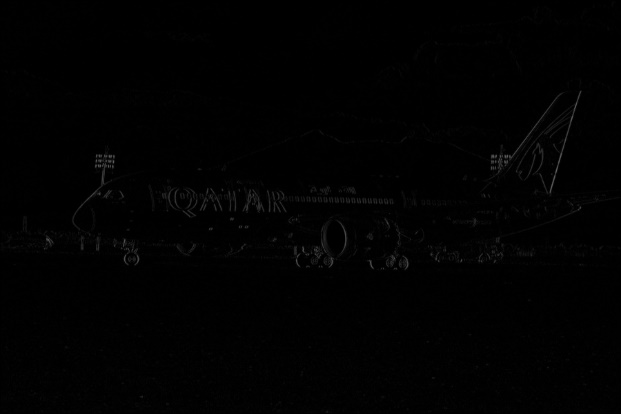
**Fig 2 :** Edge detection along x- direction using method 2



**Fig 3 :** Edge detection along y- direction using method 1



**Fig 4 :** Edge detection along y- direction using method 2



**Fig 5 :** Edge detection along x- direction by calculating the absolute value of x from method 1 and method 2



**Fig 6 :** Edge detection along x- direction by calculating the absolute value of y from method 1 and method 2



**Fig 7 :** Final image after merging the Sobel Kernel X’ output image and Sobel Kernel Y’ output image

**Keypoint Detection**

**Source code for Keypoint Detection**

import cv2

import numpy as np

import math

from math import pi as CONSTANT\_PI, e as CONSTANT\_E

img = 'C:/Users/Shraddha/Desktop/CVIP/Project\_1/task2.jpg'

noOfOct = 4

imgsPerOct = 5

size = 7

acr\_octaveRatio = 2

sigmaVal = (1/(math.sqrt(2)))

octaveRatio = (math.sqrt(2))

def detectImage(img, imageName):

cv2.imwrite('C:/Users/Shraddha/Desktop/CVIP/Project\_1/OutputImage/Task2'+imageName+'.jpg', img)

return 1

cv2.imshow(imgTitle, img)

cv2.waitKey(0)

cv2.destroyAllWindows()

def convolve\_image(img, kernel):

sizeOfKernel = (int)((kernel.shape[0])/2)

img\_conv = np.zeros((img.shape[0], img.shape[1]))

for i in np.arange( sizeOfKernel, ((img.shape[0])-sizeOfKernel) ):

for j in np.arange( sizeOfKernel, ((img.shape[1])-sizeOfKernel) ):

temp = 0

for x in np.arange(-1, 2):

for y in np.arange(-1, 2):

coordinateOfKernel = kernel[x+sizeOfKernel, y+sizeOfKernel]

coordinateOfImg = img[i-x, j-y]

temp = temp + (coordinateOfImg\*coordinateOfKernel)

img\_conv[i][j] = temp

return img\_conv

def getGaussianVal(k, sigma):

if (k%2==0):

k=k+1

kernelSizeHalf = (int)(k/2)

l=-1\*kernelSizeHalf

h=kernelSizeHalf

kernel = np.zeros((k, k))

sum=0

for i in np.arange( l, h+1 ):

for j in np.arange( l, h+1 ):

kernel[i+kernelSizeHalf][j+kernelSizeHalf] = (1/( 2\*(CONSTANT\_PI)\*(sigma\*\*2)\*( (CONSTANT\_E)\*\*( ( ((i)\*\*2)+((j)\*\*2) )/( 2\*(sigma\*\*2) ) ) ) ))

sum = sum + kernel[i+kernelSizeHalf][j+kernelSizeHalf]

for i in np.arange( l, h+1 ):

for j in np.arange( l, h+1 ):

kernel[i+kernelSizeHalf][j+kernelSizeHalf] = (kernel[i+kernelSizeHalf][j+kernelSizeHalf]) / sum

return kernel

def resize(img, factor=0.5):

resized\_img = np.zeros(( (int)((img.shape[0])\*factor) , (int)((img.shape[1])\*factor) ))

for resized\_img\_i in range(0, resized\_img.shape[0]):

for resized\_img\_j in range(0, resized\_img.shape[1]):

a = (int)((1/factor)\*resized\_img\_i)

b = (int)((1/factor)\*resized\_img\_j)

resized\_img[resized\_img\_i][resized\_img\_j] = (int)(img[a][b])

return resized\_img

def main():

original\_img = cv2.imread(img,0)

# Step 1 - Creating Scale Space images

scaleSpace = []

for i in range(0, noOfOct):

temp=[]

for j in range(0, imgsPerOct):

temp.append(None)

scaleSpace.append(temp)

octaveSigma\_startingValue = sigmaVal / acr\_octaveRatio

for octaveIndex in range(0, noOfOct):

octaveSigma\_startingValue = octaveSigma\_startingValue \* acr\_octaveRatio

sigma = octaveSigma\_startingValue / octaveRatio

for imageIndex in range(0, imgsPerOct):

if (imageIndex==0):

if (octaveIndex==0):

scaleSpace[0][0]=original\_img

else:

scaleSpace[octaveIndex][0] = resize(scaleSpace[octaveIndex-1][0], factor=0.5)

else:

sigma = sigma \* octaveRatio

gaussianKernel = getGaussianVal(size, sigma)

convoluted\_img, convoluted\_img\_abs = convolve(scaleSpace[octaveIndex][imageIndex-1], gaussianKernel)

scaleSpace[octaveIndex][imageIndex] = convoluted\_img

for octaveIndex in range(0, numberOfOctaves):

for imageIndex in range(0, imagesPerOctave):

detectImage(scaleSpace[octaveIndex][imageIndex], "ScaleSpace\_"+str(octaveIndex)+"\_"+str(imageIndex))

dogSpace = []

for i in range(0, numberOfOctaves):

temp=[]

for j in range(0, imagesPerOctave-1):

temp.append(None)

dogSpace.append(temp)

for octaveIndex in range(0, len(dogSpace)):

for imageIndex in range(0, len(dogSpace[0])):

currentDogSpace\_l = (scaleSpace[octaveIndex][imageIndex]).shape[0]

currentDogSpace\_b = (scaleSpace[octaveIndex][imageIndex]).shape[1]

dogSpace[octaveIndex][imageIndex] = np.zeros(( currentDogSpace\_l , currentDogSpace\_b ))

for i in range(0, currentDogSpace\_l):

for j in range(0, currentDogSpace\_b ):

dogSpace[octaveIndex][imageIndex][i][j] = abs(scaleSpace[octaveIndex][imageIndex+1][i][j] - scaleSpace[octaveIndex][imageIndex][i][j])

maxVal = dogSpace[octaveIndex][imageIndex][0,0]

for i in range(0, currentDogSpace\_l):

for j in range(0, currentDogSpace\_b):

if (maxVal<dogSpace[octaveIndex][imageIndex][i][j]):

maxVal=dogSpace[octaveIndex][imageIndex][i][j]

for i in range(0, currentDogSpace\_l):

for j in range(0, currentDogSpace\_b):

dogSpace[octaveIndex][imageIndex][i, j] = 255.0 \* ( dogSpace[octaveIndex][imageIndex][i][j] / maxVal )

for octaveIndex in range(0, len(dogSpace)):

for imageIndex in range(0, len(dogSpace[0])):

detectImage(dogSpace[octaveIndex][imageIndex], "DOGSpace\_"+str(octaveIndex)+"\_"+str(imageIndex))

maxMinDogSpace = []

for i in range(0, numberOfOctaves):

temp=[]

for j in range(0, imagesPerOctave-3):

temp.append(None)

maxMinDogSpace.append(temp)

for octaveIndex in range(0, len(maxMinDogSpace)):

for imageIndex in range(0, len(maxMinDogSpace[0])):

maxMinDogSpace[octaveIndex][imageIndex] = np.zeros(( (dogSpace[octaveIndex][0]).shape[0] , (dogSpace[octaveIndex][0]).shape[1] ))

prevImage = dogSpace[octaveIndex][imageIndex]

currentImage = dogSpace[octaveIndex][imageIndex+1]

nextImage = dogSpace[octaveIndex][imageIndex+2]

for i in range(1, (currentImage.shape[0])-1):

for j in range(1, (currentImage.shape[1])-1):

currentPixelVal = currentImage[i][j]

compArray = []

k = -1

l = -1

while(k<2):

while(l<2):

compArray.append(prevImage[i+k][j+l])

compArray.append(currentImage[i+k][j+l])

compArray.append(nextImage[i+k][j+l])

l=l+1

k=k+1

minComp = min(compArray)

maxComp = max(compArray)

if (minComp!=0 and maxComp!=0 and (minComp==currentPixelVal or maxComp==currentPixelVal)):

maxMinDogSpace[octaveIndex][imageIndex][i][j] = 255

for octaveIndex in range(0, len(maxMinDogSpace)):

for imageIndex in range(0, len(maxMinDogSpace[0])):

detectImage(maxMinDogSpace[octaveIndex][imageIndex], "maxMinDogSpace\_"+str(octaveIndex)+"\_"+str(imageIndex))

main()

**Image Results for Keypoint Detection**



**Cursor Detection**

In this problem, I have applied Laplacian, Gaussian Convolution and then Image sharpening. So first I read the template and then I have applied Laplacian and after that I applied Gaussian to smoothen the image. Cursor is surrounded by rectangle in the template.

**Source code for Cursor Detection**

import numpy as np

import argparse, glob

import cv2

from cv2 import CV\_8U as CONST\_CV\_8U, TM\_CCOEFF as CONST\_TM\_CCOEFF, TM\_CCORR as CONST\_TM\_CCORR

from scipy.ndimage.filters import gaussian\_filter

negative\_img = 10

positive\_img = 15

templateLoc = 'C:/Users/Shraddha/Desktop/CVIP/Project\_1/task3/template.png'

posImgLoc\_prefix = 'C:/Users/Shraddha/Desktop/CVIP/Project\_1/task3/pos\_'

posImgLoc\_suffix = '.jpg'

negaImgLoc\_prefix = 'C:/Users/Shraddha/Desktop/CVIP/Project\_1/task3/neg\_'

negaImgLoc\_suffix = '.jpg'

def detectImage(img, outImageName):

#imwrite() save a image to a specified file

cv2.imwrite('C:/Users/Shraddha/Desktop/CVIP/Project\_1/OutputImage/Task3\_'+outImageName+'.jpg', img)

return 1

cv2.imshow(imgTitle, img)

cv2.waitKey(0)

cv2.destroyAllWindows()

def main():

print("Process starts")

temp\_img = cv2.imread(templateLoc,0)

temp\_img = cv2.resize(temp\_img , (13,18))

temp\_img = cv2.Laplacian(temp\_img, CONST\_CV\_8U)

temp\_img = cv2.GaussianBlur(temp\_img, (3,3),0)

sharpeningKernel = sharpeningKernel = np.array([ [-1,-1,-1], [-1, 9,-1], [-1,-1,-1] ])

for i in range(1, positive\_img+1):

imgLoc = posImgLoc\_prefix+str(i)+posImgLoc\_suffix

print("Image at '"+imgLoc+"'")

o\_test\_img= cv2.imread(imgLoc,0) #loading the original test image from the specified location

test\_img = o\_test\_img.copy()

test\_img = cv2.Laplacian(test\_img, CONST\_CV\_8U) #it calculates the Laplacian of an image

test\_img = cv2.GaussianBlur(test\_img, (3,3), 0) #Blurs an image using GaussianFilter

test\_img = cv2.filter2D(test\_img, -1, sharpeningKernel)

result = cv2.matchTemplate(test\_img, temp\_img, CONST\_TM\_CCOEFF)

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

if (max\_val>400000):

print(" Template Found")

o\_test\_img = np.dstack([o\_test\_img, o\_test\_img, o\_test\_img])

cv2.rectangle(o\_test\_img, (max\_loc[0], max\_loc[1]), (max\_loc[0]+temp\_img.shape[1], max\_loc[1]+temp\_img.shape[0]), (0, 255, 255), 2)

else:

print(" Template Not Found")

detectImage (o\_test\_img, "Pos"+str(i))

print("Positive images processed")

for i in range(1, negative\_img+1):

if (i==7):

continue

imgLoc = negaImgLoc\_prefix+str(i)+negaImgLoc\_suffix

print("Image at '"+imgLoc+"'")

o\_test\_img = cv2.imread(imgLoc,0)

test\_img = o\_test\_img.copy()

test\_img = cv2.Laplacian(test\_img, CONST\_CV\_8U)

test\_img = cv2.GaussianBlur(test\_img, (3,3), 0)

test\_img = cv2.filter2D(test\_img, -1, sharpeningKernel)

result = cv2.matchTemplate(test\_img, temp\_img, CONST\_TM\_CCOEFF)

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

if (max\_val>400000):

print(" Template Found")

o\_test\_img = np.dstack([o\_test\_img, o\_test\_img, o\_test\_img])

cv2.rectangle(o\_test\_img, (max\_loc[0], max\_loc[1]), (max\_loc[0]+temp\_img.shape[1], max\_loc[1]+temp\_img.shape[0]), (0, 255, 255), 2)

else:

print(" Template Not Found")

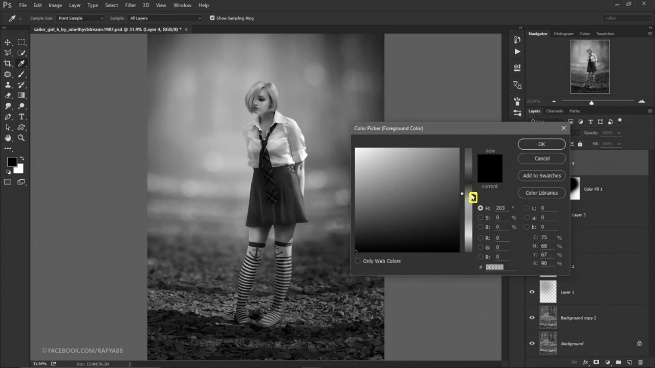
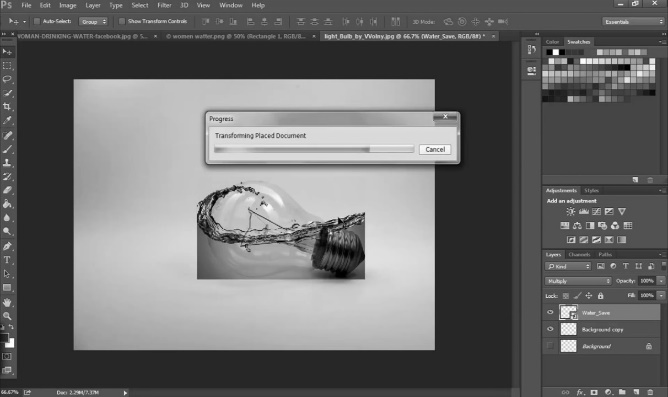
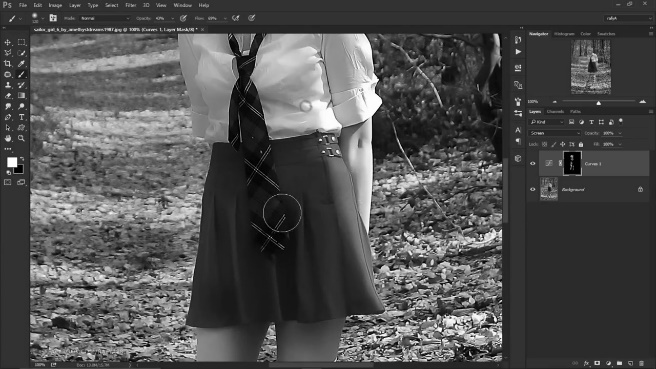
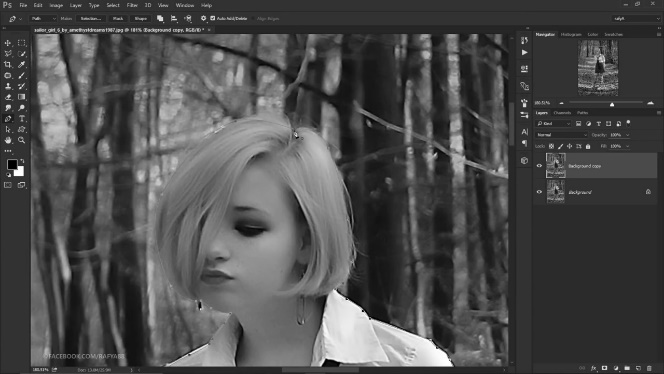
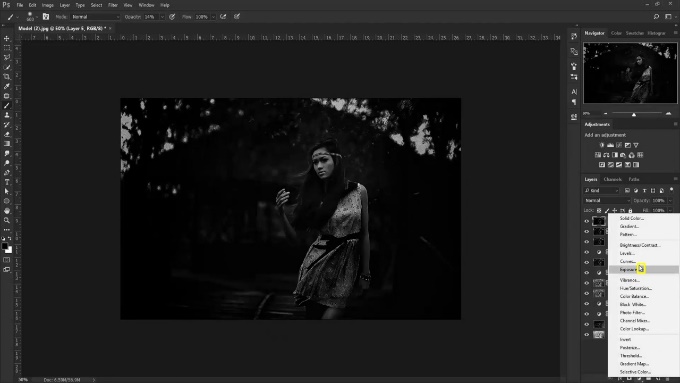
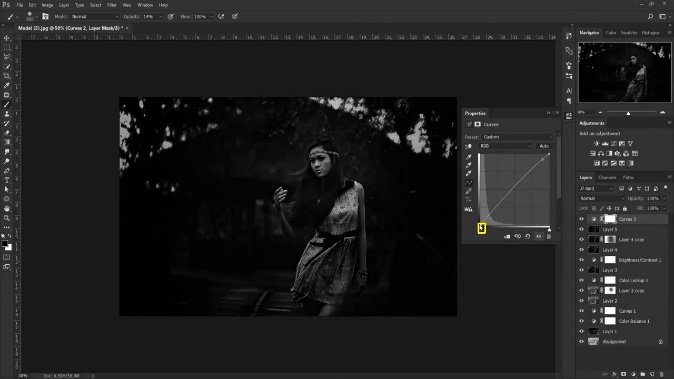
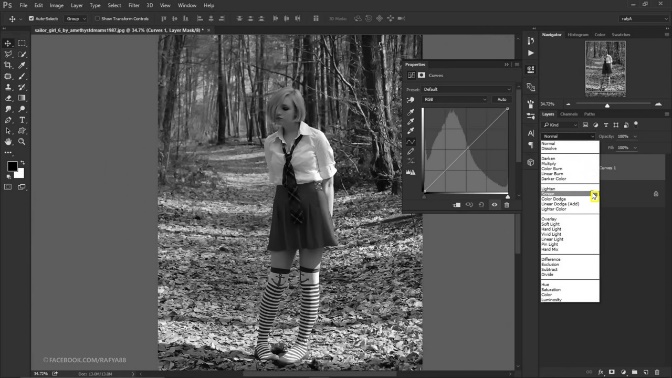
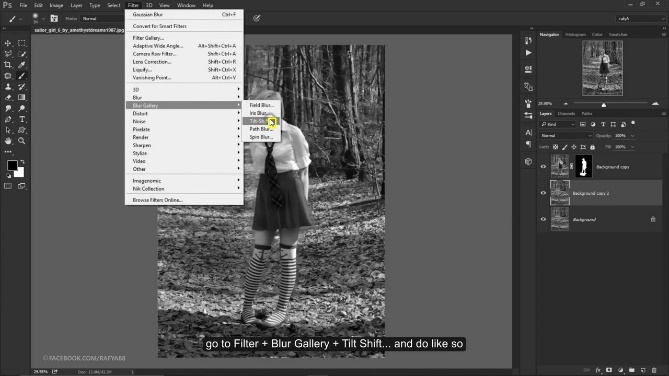
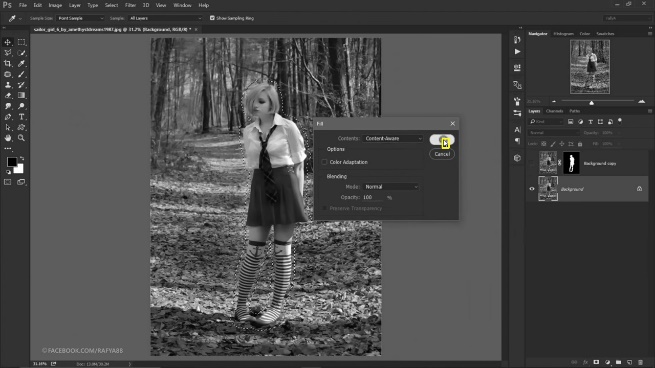
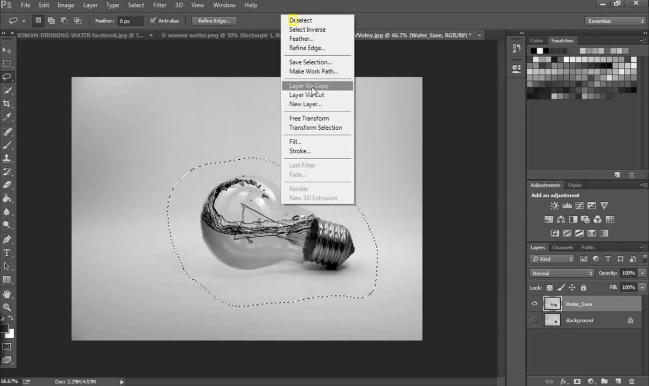
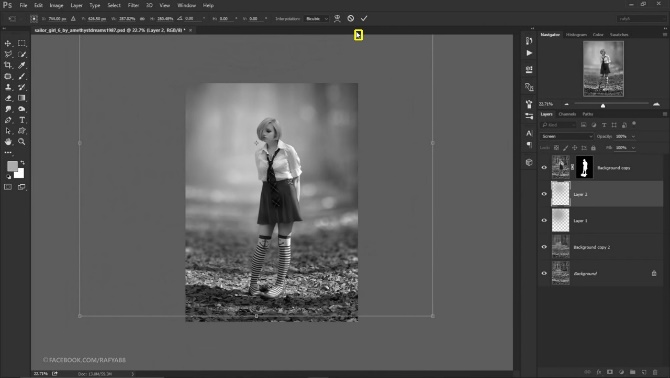
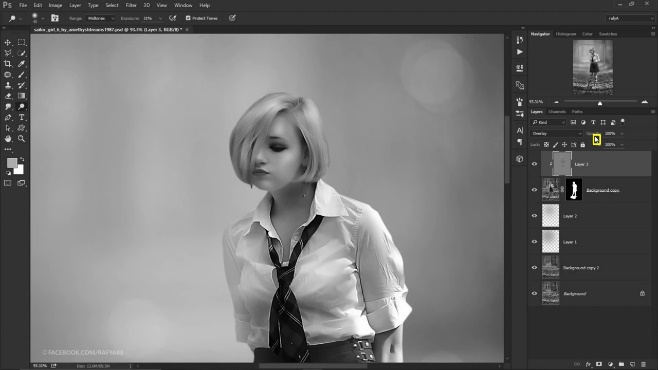
detectImage (o\_test\_img, "Neg"+str(i))

print("Negative images processed")

print("Process ends")

main()

**Image Results for Cursor Detection**



Expected output for all positive images : Template found

Expected output for all negative images : Template not found

|  |  |
| --- | --- |
| **Image name** | **Actual Output** |
| Pos\_1.jpg | Template found |
| Pos\_1.jpg | Template found |
| Pos\_2.jpg | Template found |
| Pos\_3.jpg | Template found |
| Pos\_4.jpg | Template found |
| Pos\_5.jpg | Template found |
| Pos\_6.jpg | Template found |
| Pos\_7.jpg | Template found |
| Pos\_8.jpg | Template not found |
| Pos\_9.jpg | Template found |
| Pos\_10.jpg | Template found |
| Pos\_11.jpg | Template found |
| Pos\_12.jpg | Template found |
| Pos\_13.jpg | Template found |
| Pos\_14.jpg | Template found |
| Pos\_15.jpg | Template found |
| neg\_1.jpg | Template not found |
| neg\_2.jpg | Template not found |
| neg\_3.jpg | Template not found |
| neg\_4.jpg | Template found |
| neg\_5.jpg | Template not found |
| neg\_6.jpg | Template not found |
| neg\_7.jpg | Template not found |
| neg\_8.jpg | Template not found |
| neg\_9.jpg | Template not found |
| neg\_10.jpg | Template not found |