CSC420 Assignment 1

Mengning Yang
Student Number: 1002437552

September 29, 2019

1. (a) Compute convolution of the 2D (grayscale) image and a 2D filter.

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

def calculate(padded_img, filter, i, j):
        f_h, f_w = filter.shape
        summed = 0
        for k in range(f_h):
                n_j = j
                for l in range(f_w):
                        summed =
                        summed + (filter[k][l] * padded_img[i][n_j])
                        n_j = n_j + 1
                i = i + 1
        return summed


def convolution(image, filter):
        # get size of the filter
        filter_size = filter.shape[1]

        #the image we passed in is already grey
        #get dimensions of the ORIGINAL image
        img_height, img_width = image.shape

        #we assume we will have a filter of odd size
        #and its a square matrix
        size_pad = filter_size - 1

        # create a empty matrix filled with zero
        ni_h = img_height + size_pad
        ni_w = img_width + size_pad
        new_img = numpy.zeros((ni_h, ni_w), 'uint8')

        # copy image into matrix
        new_img[(size_pad//2):(ni_h-(size_pad//2)),
        (size_pad//2):(ni_w-(size_pad//2))] = image

        # flip the filter
        flipped_filter = filter[::-1,::-1]

        #a new image container
        img_copy = image.copy()

        # get the convoved value of each pixel
        #remember here we are using the size of the ORIGINAL image
        #because we want the size to be the same
        for i in range(img_height):
                for j in range(img_width):
                        img_copy[i][j] =
                        calculate(new_img, flipped_filter, i, j)
```

```
        return img_copy


if __name__ == '__main__':
    f = numpy.array([[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]])
    #Load an color image in grayscale
    img=cv2.imread('waldo.png',cv2.IMREAD_GRAYSCALE)
    result = convolution(img, f)
    cv2.imshow('result_image',result)
    cv2.waitKey(0)
```

(b) All the other functions remain the same.

```
if __name__ == '__main__':
    f = numpy.array([[0,0,0],[0,1,0],[0,0,0]])[..., None]

    #convert a 2D filter to a 3D filter => (3,3,3)
    img_filter_3d = numpy.repeat(f, 3, axis=2)

    #Load an color image in RGB => (3,3,3)
    img=cv2.imread('waldo.png',cv2.IMREAD_COLOR)
    R = img[:,:,0].copy()
    G = img[:,:,1].copy()
    B = img[:,:,2].copy()

    R = convolution(R, img_filter_3d[:,:,0])
    G = convolution(G, img_filter_3d[:,:,1])
    B = convolution(B, img_filter_3d[:,:,2])

#    result = numpy.zeros((img.shape[0], img.shape[1], 3))
    result = cv2.merge((B,G,R))

    cv2.imshow('result_image',result)
    cv2.waitKey(0)
```

2. (a) Yes, it is possible to get the same final result by just performing one convolution. As we know, convolution is associative, we can first perform convolution between the two filters and then, convolve it with the image. One example would be convolving twice with Gaussian kernel of width is the same as convolving once with kernel of width $\sigma\sqrt{2}$.

(b) Write your own function that creates an isotropic Gaussian filter with as an input parameter.

```
def create_gauss_filter(sigma):
    #length of the kernel is approx. 6*sigma round to odd
    if (6*sigma)%2 != 0:
        l = 6*sigma
    else:
        l = 6*sigma + 1

    #create the matrix
    ax = np.linspace(-(l-1)/2., (l-1)/2.,l)
```

```
    x, y = np.meshgrid(ax, ax)

    kernel = np.exp(-0.5*(np.square(x)+np.square(y))/np.square(sigma))

    return kernel / np.sum(kernel)
```

(c) Convolve the attached waldo.png with a (2D) Gaussian filter with $\sigma = 1$ and visualize the result (display the result of the convolution). You can use built-in functions for convolution. Include the visualized result in the assignment's document.

```
if __name__ == '__main__':
    img=cv2.imread('waldo.png',cv2.IMREAD_GRAYSCALE)
    gaussian_filter = create_gauss_filter(1)
    result =
    ndimage.convolve(img,gaussian_filter, mode='constant', cval=0.0)

    cv2.imshow('result_image',result)
    cv2.waitKey(0)
```

The result is as following:

Figure 1: Convolved waldo.png with a 2D Gaussian filter and $\sigma = 1$

(d) Gaussian filters are separable, i.e. they can be written as a product of 2 1D-filters (a function of x and a function of y). So, the vertical derivative, Gy, of a Gaussian filter G is separable in both isotropic and anisotropic case, as explained in the photo attached.

$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{\sigma^2}} = \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right) \cdot \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right)$

The isotropic case:

$$\frac{\partial G(x,y)}{\partial y} = \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right) \cdot \frac{\partial \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right)}{\partial x}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right) \cdot \left(\frac{2y}{\sigma^2} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right)$$

As we can see, $\frac{\partial G(x,y)}{\partial y}$ is still the product of function of y and a function of x. So, in isotropic case, G_y is separable.

The anisotropic case: (when σ in diff. dimensions are not equal)

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right)} = \frac{1}{2\pi\sigma_x\sigma_y} e^{\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2}\right)}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma_x} e^{\frac{-x^2}{2\sigma_x^2}}\right) \cdot \left(\frac{1}{\sqrt{2\pi}\sigma_y} e^{\frac{-y^2}{2\sigma_y^2}}\right)$$

$$\frac{\partial G(x,y)}{\partial y} = \left(\frac{1}{\sqrt{2\pi}\sigma_x} e^{\frac{-x^2}{2\sigma_x^2}}\right) \frac{\partial \left(\frac{1}{\sqrt{2\pi}\sigma_y} e^{\frac{-y^2}{2\sigma_y^2}}\right)}{\partial y}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma_x} e^{\frac{-x^2}{2\sigma_x^2}}\right)\left(-\frac{y}{\sigma_y^2} \frac{1}{\sqrt{2\pi}\sigma_y} e^{\frac{-y^2}{2\sigma_y^2}}\right)$$

as we can see, $\frac{\partial G(x,y)}{\partial y}$ is still separable, it can be separated into a function of y and a function of x.

(e) There are n^2 pixels and one pixel has m^2 operations performed. Therefore, the computational cost of computing a 2D convolution is O(n^2 m^2).
If h is separable, then one filter becomes n x 1 and the other, 1 x n. Each filter's cost is mn^2. The computational cost if h is separable is O(2mn^2), simplified to O(mn^2).

3. (a) This function also returns the orientation of the gradient
because later this function will be used in the canny detection

```
import numpy as np
import cv2
from scipy import ndimage
import matplotlib.pyplot as plt
import math

def magnitude_of_gradients(img):

    img=cv2.imread(img,cv2.IMREAD_GRAYSCALE)
```

```
    # Define kernel for x differences
    kx = np.array([[-1,0,1],[-2,0,2],[-1,0,1]],np.float32)

    # Define kernel for y differences
    ky = np.array([[1,2,1],[0,0,0],[-1,-2,-1]],np.float32)

    #x convolution
    x = ndimage.convolve(img,ky, mode='constant', cval=0.0)

    #y convolution
    y = ndimage.convolve(img,kx, mode='constant', cval=0.0)

    #result = math.sqrt(x*x + y*y)
    result = np.hypot(x, y)
    result = result / result.max() * 255

    #convert type int32 to type unit8
    result2 = np.array(result,dtype=np.uint8)

    #get the orientation of the gradient
    theta = np.arctan2(y, x)

    #cv2.imshow('result',result2)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    return (result2,theta)
```

(b) This function is written in MATLAB because most part of it is given on the course website

```
%get the result
output = q3b(im, filter);


function out = q3b(im, template)

% my function magnitude_of_gradients(img) already takes in an image and
    convert it to grey scale, then return the gradient of the image

%Get the gradient of each image returned by the function
   magnitude_of_gradients(img) in Q3(a)
G_im, D_im = magnitude_of_gradients(im);
G_temp, D_temp = magnitude_of_gradients(template);
% normalized cross-correlation
out = normxcorr2(G_temp, G_im);

% plot the cross-correlation results
figure('position', [100,100,size(out,2),size(out,1)]);
subplot('position',[0,0,1,1]);
imagesc(out)
axis off;
```

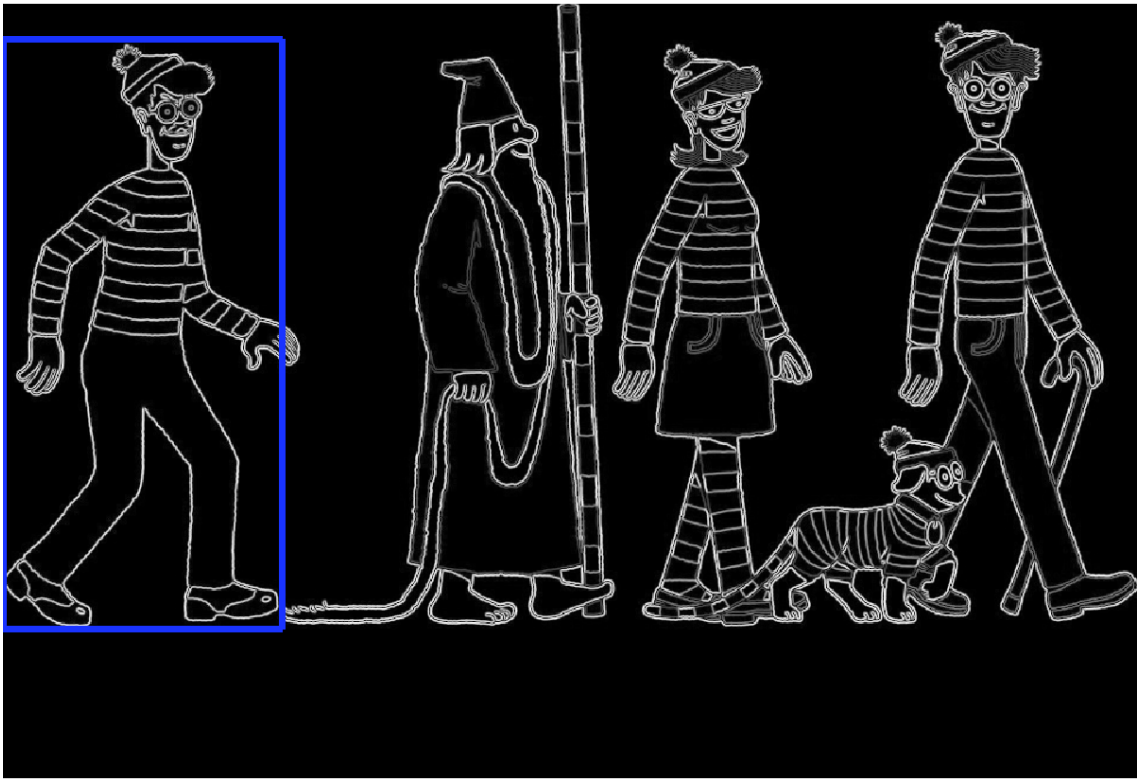
```
axis equal;

% find the peak in response
[y,x] = find(out == max(out(:)));
y = y(1) - size(template, 1) + 1;
x = x(1) - size(template, 2) + 1;

% plot the detection's bounding box
figure('position', [300,100,size(im,2),size(im,1)]);
subplot('position',[0,0,1,1]);
imshow(G_im, []);
imshow(im_input);
axis off;
axis equal;
rectangle('position', [x,y,size(template,2),size(template,1)], '
    edgecolor', [0.1,0.2,1], 'linewidth', 3.5);

end
```

Figure 2: Localised template.png on waldo.png



4. Canny Edge Detection

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

#Filter image with derivative of Gaussian (horizontal and vertical directions)
#Find magnitude and orientation of gradient
#Non-maximum suppression
#Linking and thresholding/hysteresis(NOT required)

def magnitude_of_gradients(img):

    img=cv2.imread(img,cv2.IMREAD_GRAYSCALE)

    # Define kernel for x differences
    kx = np.array([[-1,0,1],[-2,0,2],[-1,0,1]],np.float32)

    # Define kernel for y differences
    ky = np.array([[1,2,1],[0,0,0],[-1,-2,-1]],np.float32)

    #x convolution
    x = ndimage.convolve(img,ky, mode='constant', cval=0.0)

    #y convolution
    y = ndimage.convolve(img,kx, mode='constant', cval=0.0)

    #result = math.sqrt(x*x + y*y)
    result = np.hypot(x, y)
    result = result / result.max() * 255

    #convert type int32 to type unit8
    result2 = np.array(result,dtype=np.uint8)

    #get the orientation of the gradient
    theta = np.arctan2(y, x)

    #cv2.imshow('result',result2)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    return (result2,theta)


def non_max_suppression(img, D):
        M, N = img.shape
        Z = np.zeros((M,N), dtype=np.int32)
        angle = D * 180. / np.pi
        angle[angle < 0] += 180

        for i in range(1,M-1):
            for j in range(1,N-1):
```

```
                try:
                    q = 255
                    r = 255

                   #angle 0
                    if (0<=angle[i,j]<22.5)or(157.5<=angle[i,j] <= 180):
                        q = img[i, j+1]
                        r = img[i, j-1]
                    #angle 45
                    elif (22.5 <= angle[i,j] < 67.5):
                        q = img[i+1, j-1]
                        r = img[i-1, j+1]
                    #angle 90
                    elif (67.5 <= angle[i,j] < 112.5):
                        q = img[i+1, j]
                        r = img[i-1, j]
                    #angle 135
                    elif (112.5 <= angle[i,j] < 157.5):
                        q = img[i-1, j-1]
                        r = img[i+1, j+1]

                    if (img[i,j] >= q) and (img[i,j] >= r):
                        Z[i,j] = img[i,j]
                    else:
                        Z[i,j] = 0

                except IndexError as e:
                    pass

        return Z

def canny_detect(img):
    img=cv2.imread('waldo.png',cv2.IMREAD_GRAYSCALE)

    #first smooth the original image
    smoothed = scipy.ndimage.filters.gaussian_filter(img,3)

    #using the function written in Q3(a) to get the magnitude
    #and orientation of the image
    gradient, theta = magnitude_of_gradients(smoothed)
    img2 = non_max_suppression(gradient, theta)

    #convert type int32 to type unit8
    result = np.array(img2,dtype=np.uint8)

    cv2.imshow('image', result)
    cv2.imwrite("testresult.png",result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return result
```

Figure 3: canny edge detection on waldo.png