CSC420 Assignment1

Yilin Qu

Student number: 1002995734

Utorid: quyilin

Question 1

```
def correlation(I, f, mode):


    Image = I
    Filter = f
    Image_W, Image_H = Image.shape[0], Image.shape[1]
    Filter_W, Filter_H = Filter.shape[0], Filter.shape[1]
    if mode == 'valid':
        # no padding
        result = np.zeros(((Image_W - Filter_W + 1), (Image_H -
Filter_H + 1), Image.shape[2]))
        for i in range(result.shape[0]):
            for j in range(result.shape[1]):
                for c in range(result.shape[2]):
                    result[i, j, c] = (Filter * Image[i: i + Filter_W,
j: j + Filter_H, c]).sum() / Filter.sum()
        return result
    if mode == 'same':
        padded = np.zeros((Image_W + Filter_W - 1, Image_H + Filter_H
- 1, Image.shape[2]))
        padded_W, padded_H = padded.shape[0], padded.shape[1]
        for c in range(padded.shape[2]):
            padded[(Filter_W - 1) // 2: padded_W - (Filter_W - 1) //
2, (Filter_H - 1) // 2: padded_H - (Filter_H - 1) // 2, c] = Image[0:
Image_W, 0: Image_H, c]
        return correlation(padded, Filter, 'valid')
    if mode == 'full':
        padded = np.zeros((Image_W + 2 * Filter_W - 2, Image_H + 2 *
```

```
Filter_H - 2, Image.shape[2]))
        padded_W, padded_H = padded.shape[0], padded.shape[1]
        for c in range(padded.shape[2]):
            padded[(Filter_W - 1): padded_W - (Filter_W - 1), (Filter_H
- 1): padded_H - (Filter_H - 1), c] = Image[0: Image_W, 0: Image_H,
c]
        return correlation(padded, Filter, 'valid')
```

Question 2

```
#  ------- Q2 -------
xx = cv.getGaussianKernel(5, 3)
yy = cv.getGaussianKernel(5, 5)
Q2 = ((xx * yy).transpose())
Q2_1 = np.flip(Q2, axis=0)
Q2_2 = np.flip(Q2_1, axis=1)
# convolution is the filter with its top to bottom, left to right,
so just apply a flip operation.
cv.imwrite("q2.jpg", correlation(A, Q2_2, "same"))
#  ------- Q2 -------
```

To perform convolution, we can flip the filter top to bottom and left to right, and the perform the correlation operation on the image.

Output as follows:

It has been **blurred** a bit.

q2.jpg

Question 3

Q3:

- Convolution is Commutative. i.e.:

$$f * g = g * f.$$

Proof:

$$f(i,j) * g(i,j) = \sum_{k=-\infty}^{\infty} \sum_{t=-\infty}^{\infty} f(k,t) \cdot g(i-k, j-t)$$

Let $K = i-k$, $T = j-t$

then $k = i-K$, $t = j-T$

which also means: $k \to \infty \Leftrightarrow K \Leftrightarrow -\infty$; $t \to \infty \Leftrightarrow T \Leftrightarrow -\infty$

So we can write $f * g = \sum_{k=-\infty}^{\infty} \sum_{t=-\infty}^{\infty} f(k,t) \cdot g(i-k, j-t)$

$$= \sum_{K=-\infty}^{\infty} \sum_{T=-\infty}^{\infty} f(i-K, j-T) \cdot g(K,T)$$

$$= \sum_{K=-\infty}^{\infty} \sum_{T=-\infty}^{\infty} g(K,T) f(i-K, j-T)$$

$$= g * f.$$

- Correlation is not Commutative:

Suppose we have $f = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $G = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

If we compute $f \otimes g$ with "same":

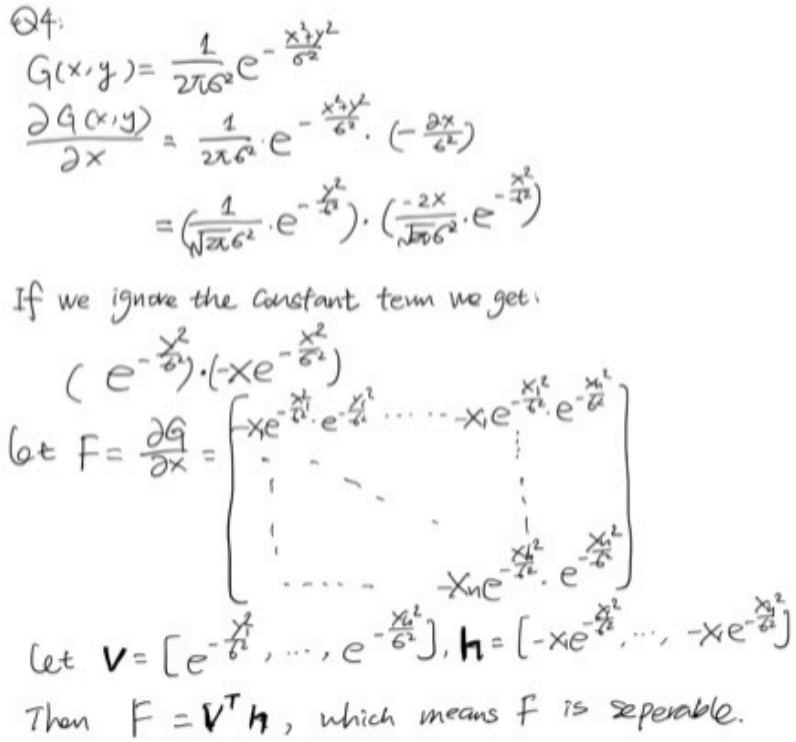
We have: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

If we compute $g \otimes f$ with "same":

We have $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Therefore $f \otimes g \neq g \otimes f$

Question 4



Question 5

If it is not separable, then it costs $O(m^2n^2)$,

If it is separable, then it costs $O(2mn^2) = O(mn^2)$

Question 6

```
A = np.array([[9, 0, 9],
              [0, 0, 0],
              [9, 0, 9],
              ])

B = np.array([[16, 0, 16],
              [0, 0, 0],
              [16, 0, 16],
              ])
```

Let A and B be above matrix, let **a** = [3, 0, 3]**, b =** [4, 0, 4]**,** so A and B are separable since A = $\boldsymbol{a^T a}$, B = $\boldsymbol{b^T b}$**,** then A + B = C , where:

```
C = np.array([[25, 0, 25],
              [0, 0, 0],
              [25, 0, 25],
              ])
```

Let **c =** [5, 0, 5], then C = $\boldsymbol{c^T c}$**,** so C is separable as well.

Question 7

```
import cv2 as cv
import numpy as np
img = cv.imread('portrait.jpg', 0)

#  ------- Q7 -------
blur = cv.GaussianBlur(img, (3, 3), 0)
sobelx = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=3)
sobely = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=3)

cv.imwrite("q7-dev-x.jpg", sobelx)
cv.imwrite("q7-dev-y.jpg", sobely)
img = cv.Laplacian(img, cv.CV_64F, ksize=3)
cv.imwrite("q7-lab.jpg", img)
#  ------- Q7 -------
```

Outputs as follow:

(Left to right: Gaussian with sobelx, Gaussian with sobely, Laplacian)

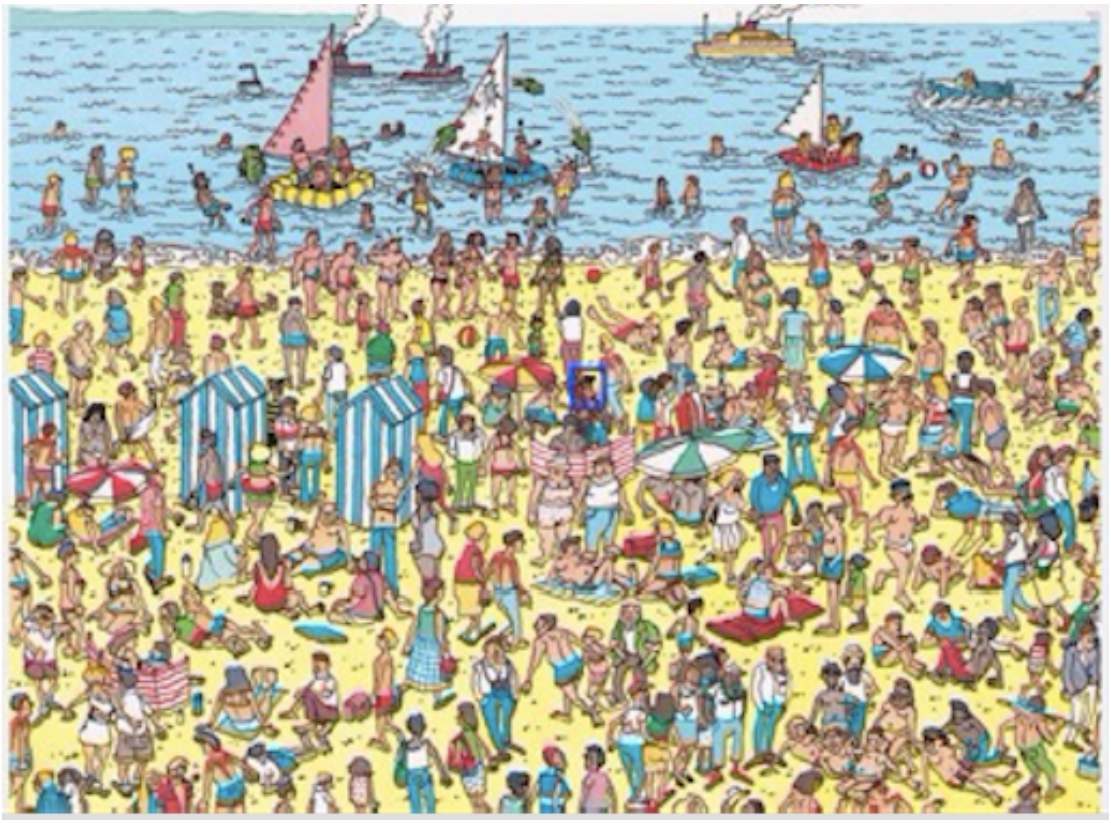
Question 8

(Using Matlab for this question)

```
function out = findWaldo(im, filter)

% convert image (and filter) to grayscale
im_input = im;
im = rgb2gray(im);
im = double(im);
filter = rgb2gray(filter);
filter = double(filter);
filter = filter/sqrt(sum(sum(filter.^2)));

% normalized cross-correlation
out = normxcorr2(filter, im);

% find the peak in response
[y,x] = find(out == max(out(:)));
y = y(1) - size(filter, 1) + 1;
x = x(1) - size(filter, 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(im_input)
axis off;
axis equal;
rectangle('position', [x,y,size(filter,2),size(filter,1)],
'edgecolor', [0.1,0.2,1], 'linewidth', 3.5);

end
# Script
im = imread('whereswaldo.png');
filter = imread('waldo.png');
```

```
output = findWaldo(im, filter);
```

Output as follow:



Question 9

Canny edge detection has the following four main steps:

1. Smooth, this is to reduce the noise in our image by applying a Gaussian filter on the image, we can reduce the influence by the noise in our original image.

2. Finding Intensity Gradient of the Image, in this step we will need to compute the intensity gradient and the orientation on each pixel, so that we can determine the strength of each pixel and its direction.

3. Non-max suppression, this is used to check if a pixel is a local maximal, and this step will thining and sharpen the edge by discarding those neighbor pixels with lower intensities, left the local-max pixel.

4. Hysteresis Thresholding, this final step decides which "edges" are real edges. We set up two thresholds, the **max** and the **min,** we know every edges with intensity gradient greater than **max** must be real edges, and those whose below the **min** are not, for those pixels whose intensity gradient are between these two cutoffs, we check if they are connected to some real edges, we classify them to be edges, otherwise they will be classified as non-edge.

Question 10

Zero crossings are points where the sign of the function changed. Zero crossing of Laplacian is where the intensities **second derivative changes sign**, which is somewhere its **first derivative encountered a local max or local min**, which also means the **intensity of that pixel changes sharply**, implies there must be an edge, so it is useful for edge detection.

Question 11

```
# ------ Q11 ------
import cv2 as cv
img = cv.imread('portrait.jpg')
edges = cv.Canny(img, 180, 485)


cv.imwrite("q11.jpg", edges)
# ------ Q11 ------
```

Output as follow: