CSC420 Assignment 1

Yubo Wang, 1002138377

1. Code

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

def correlation(image, patch, mode):
    img = cv2.imread(image)
    mid = int((patch.shape[0] - 1) / 2)
    i_shape, p_shape = img.shape, patch.shape     # get image amd patch size
    img_h, img_w = i_shape[0], i_shape[1]
    patch_h, patch_w = p_shape[0], p_shape[1]
    p_size = 2 * mid

    if mode == 'same':                            # based on input type create corresponding pad
        output = np.zeros((img_h, img_w, 3))
        pseudo = np.zeros((img_h + p_size, img_w + p_size, 3))
        pseudo[mid:mid+img_h, mid:mid+img_w] = img
    elif mode == 'valid':
        output = np.zeros((img_h - p_size, img_w - p_size, 3))
        pseudo = np.zeros((img_h, img_w, 3))
        pseudo = img
    elif mode == 'full':
        output = np.zeros((img_h + p_size, img_w + p_size, 3))
        pseudo = np.zeros((img_h + 2*p_size, img_w + 2*p_size, 3))
        pseudo[p_size:p_size+img_h, p_size:p_size+img_w] = img

    for row in range(output.shape[0]):            # multiply the filter with the image
        for col in range(output.shape[1]):
            square = np.array(pseudo[row:row+patch_h, col:col+patch_w])
            r_sq = np.reshape(square, (-1, 3))
            r_patch = np.reshape(patch, -1)
            res = np.zeros(3)
            for i in range(r_sq.shape[0]):
                res += r_sq[i] * r_patch[i]
            output[row, col] = res

    return output
```

2. Code:

Function correlation from question 1

```python
def gaussian_kernel(sigmax, sigmay):            # use cv2.getGaussianKernel to get required filter
    gausx = cv2.getGaussianKernel(5, sigmax)
    gausy = cv2.getGaussianKernel(5, sigmay)
    gausy = np.reshape(gausy, (1, -1))
    kernel = np.multiply(gausx, gausy)
    return kernel

def gaussian_filter(image, mode):
    filter = gaussian_kernel(3, 5)
    filter = np.flipud(filter)                  # flip the filter to perform convolution
    filter = np.fliplr(filter)
    output = correlation(image, filter, mode)
    return output


output = gaussian_filter('iris.jpg', 'same')
cv2.imwrite('result.jpg', output)
cv2.imshow("output", output)
cv2.waitKey(0)
```

Result:

3. Convolution is commutative.
   Proof:
   $$(x \circledast y)_i = \sum_{u=0}^{k} x(u)y(i-u)$$
   Assume $v = i - u$
   $$\sum_{u=0}^{k} x(u)y(i-u) = \sum_{v=i}^{i-k} x(i-v)y(v) = \sum_{v=0}^{k} x(i-v)y(v) = \sum_{v=0}^{k} y(v)x(i-v) = (y \circledast x)_i$$
   Therefore, convolution is commutative

   Correlation is not commutative
   As the slides in lecture, convolution is equivalent to flipping the filter in both dimensions and apply correlation. If we apply that to formula, we need to negate the parameter in function to change correlation to convolution.
   $$(x(i) \otimes y(i))_i = (x^*(-i) \circledast y(i))_i = (y(i) \circledast x^*(-i))_i$$
   While $(y(i) \otimes x(i))_i = (y^*(-i) \circledast x(i))_i$
   Therefore, correlation is not commutative
4. The horizontal derivative of a Gaussian Filter G is a separable filter.
   $$\mathrm{G(x,y)} = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{\sigma^2}} = \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right) * \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right)$$
   $$\frac{\partial \mathrm{G(x,y)}}{\partial \mathrm{x}} = \frac{\partial\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right)}{\partial \mathrm{x}} * \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right) + \frac{\partial\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right)}{\partial \mathrm{x}} * \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right)$$
   $$= \frac{\partial\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right)}{\partial \mathrm{x}} * \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right) = -\frac{x^2}{\sigma^2} * \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right)$$
   We have separated the filter into product of 2 1D filters. Therefore, the horizontal derivative of a Gaussian Filter G is a separable filter.

5. Since the image size is n x n and filter size is m x m, there will be n^2 pixels need to be convolved with the filter. The convolution need m^2 operations per pixel. Therefore, the time complexity should be $O(n^2m^2)$
   If the filter is separable, the filter can be separated into two filter with size 1 x m and m x 1. Then convolution just need 2m operations per pixel. Therefore, the time complexity will be $O(mn^2)$
6. Sobel x filter $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ is separable. By flipping the matrix, we get $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ which is also a separable filter. Add these two matrixes we get $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, which is of course a separable filter.
7. Apply the derivative of Gaussian filter and Laplacian of Gaussian filter to portrait.jpg

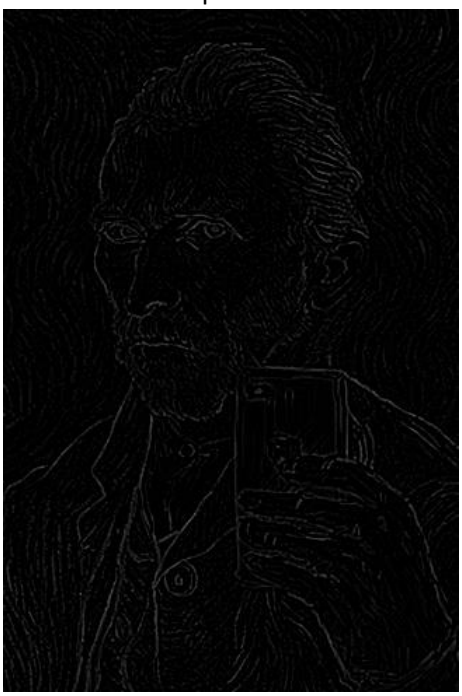
```
window_name = "Laplace Demo"
window_name1 = "Horizontal derivative Demo"
window_name2 = "Vertical derivative Demo"
img = cv2.imread('portrait.jpg')
img = cv2.GaussianBlur(img, (3, 3), 0)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
derx = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0, 3)
dery = cv2.Sobel(img_gray, cv2.CV_64F, 0, 1, 3)
derxy = cv2.Sobel(derx, cv2.CV_64F, 0, 1, 3)
lap = cv2.Laplacian(img_gray, cv2.CV_64F, 3)
cv2.imwrite('derx.jpg', derx)
cv2.imwrite('dery.jpg', dery)
cv2.imwrite('derxy.jpg', derxy)
cv2.imwrite('lap.jpg', lap)
```

Sobel X

Sobel Y

Sobel XY

Laplacian



8. Detect waldo.jpg in whereswaldo.jpg using correlation

```
waldo = cv2.imread('waldo.jpg')
wshape = waldo.shape
scene = cv2.imread('whereswaldo.jpg')
res = cv2.matchTemplate(scene, waldo, cv2.TM_CCOEFF)
minmax = cv2.minMaxLoc(res)
min_val, max_val, min_loc, max_loc = minmax
corner1 = max_loc
corner2 = (max_loc[0] + wshape[1], max_loc[1] + wshape[0] )
cv2.rectangle(scene, corner1, corner2, (255, 0, 0), 3)
cv2.imshow("output", scene)
cv2.waitKey(0)
```

9. Steps of Canny edge detection
    1. Apply Gaussian filter

       By applying the Gaussian filter, we smooth the image in order to remove the noise, which will increase our accuracy of detecting edges since noise might be considered as edges.

    2. Find the intensity gradients of the image

       By getting the magnitude and direction of the gradient of the image, we can count the pixel with high gradient as pixels lie on the edge. Combine all such pixels we can get edges in the image.

    3. Apply non-maximum suppression

       NMS set all gradient values to 0 except the local maxima to get the location of pixel with biggest change of magnitude of intensity. After finishing this step, we can get thinner edges, but we might lose some edges and gain some gaps.
    4. Apply double threshold

       We apply a low and a high threshold to the result get from previous step to fill the gap mentioned above and eliminate spurious edges caused by color variation and noise.

       Pixels with gradient value larger than high threshold will be marked as strong edge pixel. Pixels with gradient value smaller than high threshold but smaller than low threshold will be marked as weak edge pixel. Pixels with gradient value smaller than low threshold will be completely suppressed.

    5. Track edge by hysteresis

       Finally suppress all the weak edges that not connected to any strong edges to eliminate all spurious edges.

10. Since Laplacian can be considered as second derivative, zero crossing of Laplacian of Gaussian indicated a huge change of intensity at the point, which means an edge in the original picture.

11. Code of Canny edge detection

```
img = cv2.imread('portrait.jpg',0)
edges = cv2.Canny(img,120,470)
cv2.imshow("output", edges)
cv2.waitKey(0)
```