CSC420H1, Fall 2018
Assignment 1

Name: JingXi, Hao
Student Number: 1000654188

Due Date: 26 March, 2018 10:59pm

1. Write your own function that implements the correlation (for grayscale or color images and 2D filters) between an input image and a given correlation filter. The function must take as input: an input image 'I', a filter 'f', and a string 'mode', that can either be 'valid', 'same', or 'full'. The output must match what is specified by 'mode'.

   Solution:

   For this question, the screenshots for code are shown below. Also, the code snippets for this question are able to be found in the file, *a1.py*.

```
5  # question 1
6  # this is a helper function for question 1, which adds zero padding for image
7  # for full and same mode
8  def doZeroPaddingForImage(img, imgRows, imgColumns, rowPadding, columnPadding,
   isColorImage):
9      if not isColorImage:
10         # add zero padding for grayscale image
11         paddedImg = np.pad(img, rowPadding, "constant")
12     else:
13         # add zero padding for color image
14         paddedImgRows = imgRows + (rowPadding * 2)
15         paddedImgColumns = imgColumns + (columnPadding * 2)
16         paddedImg = np.zeros((paddedImgRows, paddedImgColumns, 3))
17
18         # reassign the values in the paddedImage
19         for t in range(0, imgRows):
20             for s in range(0, imgColumns):
21                 newT = t + rowPadding
22                 newS = s + columnPadding
23                 paddedImg[newT][newS] = img[t][s]
24
25     return paddedImg
26
27 # this is a helper function for question 1, which computes sum for single pixel
28 def computeSum(filter, partialImg):
29     sum = 0
30     for p in range(0, filter.shape[0]):
31         for q in range(0, filter.shape[1]):
32             sum = sum + filter[p][q] * partialImg[p][q]
33     return sum
34
35 # this is a helper function for question 1, which generate the result image
36 def generateResultImage(rows, columns, filterRows, filterColumns, paddedImg,
   filter):
37     resultImg = []
38
39     for k in range(0, rows):
40         newRow = []
41         for l in range(0, columns):
42             sum = computeSum(filter, paddedImg[k : k+filterRows, l :
   l+filterColumns])
43             newRow.append(sum)
44         resultImg.append(newRow)
45
46     result = np.array(resultImg)
47     return result
48
```

```python
# this function is for question 1, which computes the correlation between an
# input image and a given correlation filter
def computeCorrelation(I, f, mode):
    # drop the redundant information gotten from imread
    img = np.array(I.tolist())

    # get the shape of input grayscale or color img and 2D filter, assume filter
    # is a square, so rowK equals to columnK
    filterRows, filterColumns = f.shape
    rowK = filterRows // 2
    columnK = filterColumns // 2

    if len(img.shape) == 3:
        # img is a color image
        isColorImage = True
        imgRows = img.shape[0]
        imgColumns = img.shape[1]
    else:
        # img is a grayscale image
        isColorImage = False
        imgRows, imgColumns = img.shape

    if mode == "same":
        # do zero padding for the img to generate a padded img
        # compute dimension for padded img
        paddedImg = doZeroPaddingForImage(img, imgRows, imgColumns, rowK, columnK,
isColorImage)

        # compute the correlation
        result = generateResultImage(imgRows, imgColumns, filterRows,
filterColumns, paddedImg, f)

    elif mode == "full":
        # do zero padding for the img to generate a padded img
        # compute dimension for padded img
        rowPadding = filterRows - 1
        columnPadding = filterColumns - 1
        paddedImg = doZeroPaddingForImage(img, imgRows, imgColumns, rowPadding,
columnPadding, isColorImage)

        # compute the correlation
        rows = imgRows + (rowPadding * 2) - (rowK * 2)
        columns = imgColumns + (columnPadding * 2) - (columnK * 2)
        result = generateResultImage(rows, columns, filterRows, filterColumns,
paddedImg, f)

    elif mode == "valid":
        # compute the correlation
        rows = imgRows - (rowK * 2)
        columns = imgColumns - (columnK * 2)
        result = generateResultImage(rows, columns, filterRows, filterColumns, img,
f)

    return result
```

```python
# main function is defined here
if __name__ == "__main__":
    # question 1
    iris = cv.imread("./iris.jpg")
    filter = np.array([
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
    ])
    result1 = computeCorrelation(iris, filter, "full")
    cv.imwrite("./result1.jpg", result1)

```

As you can see above, the function *computeCorrelation(I, f, mode)* takes an image, a filter, and a mode as parameters and returns a result image. In main function, I try with an example which use

iris.jpg as input image, a 5 by 5 filter filling with zeros except a one in the center, and a string "$full$" as mode. Then, save the output as *result1.jpg* by using *imwrite* from OpenCV.

2. How would you use your function from part A to calculate the convolution between a filter and an image? Use your function from question 1 to convolve iris.jpg with a Gaussian filter $\sigma_x = 3$, $\sigma_y = 5$, use 'mode' = 'same'.

   Solution:

   For this question, we first build a 2D Gaussian filter by applying dot product on two 1D Gaussian filters each with $\sigma_x = 3$ and $\sigma_y = 5$ since Gaussian filter is separable. Then, we flip the 2D Gaussian filter mentioned above twice (flip bottom to top first, then flip right to left) to get a flipped filter. Then, we pass this flipped filter as the input filter into the function from part A to compute the convolution. The code snippets are able to be found in *a1.py*. The output by taking *iris.jpg* as input image, flipped filter with dimension 29 by 29 as input filter, and 'same' as mode is shown below.

3. Is convolution a commutative operation ($f * g \; ? = g * f$)? Is correlation a commutative operation? Briefly Explain.

Solution:

Firstly, convolution is a commutative operation, which means that $f * g = g * f$. Let f be the filter and g be the image. We give the proof below.

$$\begin{aligned} G(i,j) &= (f*g)(i,j) \\ &= \sum_{u=-\infty}^{\infty}\sum_{v=-\infty}^{\infty} f(u,v)\cdot g(i-u,j-v) \\ &= \sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty} f(i-n,j-m)\cdot g(n,m) \quad \text{\# let n = i-u and m = j-v} \\ &= (g*f)(i,j) \end{aligned}$$

Thus, we have that $f * g = g * f$, which indicates that convolution is a commutative operation.

Secondly, correlation is not a commutative operation and we produce a counter-example here. Let A and B be two 3 by 3 matrices, where

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, B = \begin{bmatrix} 4 & 8 & 9 \\ 1 & 5 & 6 \\ 3 & 2 & 0 \end{bmatrix}$$

. Then, we treat A as filter and B as image and pass these as parameters with mode = 'same' to function from question 1, we obtain that

$$A \otimes B = \begin{bmatrix} 121 & 211 & 160 \\ 109 & 149 & 90 \\ 44 & 51 & 25 \end{bmatrix}$$

. Then, we treat A as image and B as filter and pass these as parameters with mode = 'same' to function from question 1, we obtain that

$$B \otimes A = \begin{bmatrix} 25 & 51 & 44 \\ 90 & 149 & 109 \\ 160 & 211 & 121 \end{bmatrix}$$

. Hence, as shown above, $A \otimes B \neq B \otimes A$, which indicates that correlation is not a commutative operation.

4. Is the horizontal derivative, $\partial G(x,y)/\partial x$, of a Gaussian filter G a separable filter? Explain.

Solution:

First, we compute the horizontal derivative, $\partial G(x,y)/\partial x$, of a Gaussian filter G. Since $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{\sigma^2}}$, therefore we have that

$$\begin{aligned} \frac{\partial G(x,y)}{\partial x} &= (\frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{\sigma^2}})(\frac{-2x}{\sigma^2}) \\ &= \frac{-2x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{\sigma^2}} \\ &= \frac{-x}{\pi\sigma^4} e^{-\frac{x^2+y^2}{\sigma^2}} \end{aligned}$$

$$= (xe^{\frac{-x^2}{\sigma^2}})(\frac{-1}{\pi\sigma^4})(e^{\frac{-y^2}{\sigma^2}})$$

As shown above, the horizontal derivative, $\partial G(x,y)/\partial x$, can be expressed as a multiplication of three terms, which are $xe^{\frac{-x^2}{\sigma^2}}$, $\frac{-1}{\pi\sigma^4}$, and $e^{\frac{-y^2}{\sigma^2}}$. Therefore, the horizontal derivative, $\partial G(x,y)/\partial x$, follows the singular value decomposition, where $xe^{\frac{-x^2}{\sigma^2}}$ refers to each u_i, $e^{\frac{-y^2}{\sigma^2}}$ refers to each v_i, and $\frac{-1}{\pi\sigma^4}$ refers to σ_i which is the only one non-zero singular value in Σ. Therefore, the horizontal derivative, $\partial G(x,y)/\partial x$, of a Gaussian filter G is a separable filter.

5. Given a $n \times n$ image, I, and $m \times m$ filter, h, what is the computational cost of computing $h \cdot I$ if h is not separable? What is the computational cost if h is separable?

   Solution:

   Firstly, if h is not separable, then it requires m^2 operations per pixel to perform a convolution. Since we have n^2 pixels in this case, therefore the computational cost of computing $h \cdot I$ when h is not separable is $O(n^2m^2)$.

   Secondly, if h is separable, then it requires $2m$ operations per pixel to perform a convolution since we can first perform a 1D horizontal convolution followed by a 1D vertical convolution. Since we have n^2 pixels in total, thus the computation cost of computing $h \cdot I$ when h is separable is $O((n^2)(2m)) = O(mn^2)$.

6. Construct two different separable filters, such that when added, the result is a separable filter.

   Solution:

   Let the two different separable filters be

$$f_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, f_2 = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

   , where f_1 is the outer product of $[1,1,1]$ and $[1,1,1]^T$ and f_2 is the outer product of $[1,1,1]$ and $[3,3,3]^T$. Then, we add this two separable filters together, we get that

$$f = f_1 + f_2 = \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix}$$

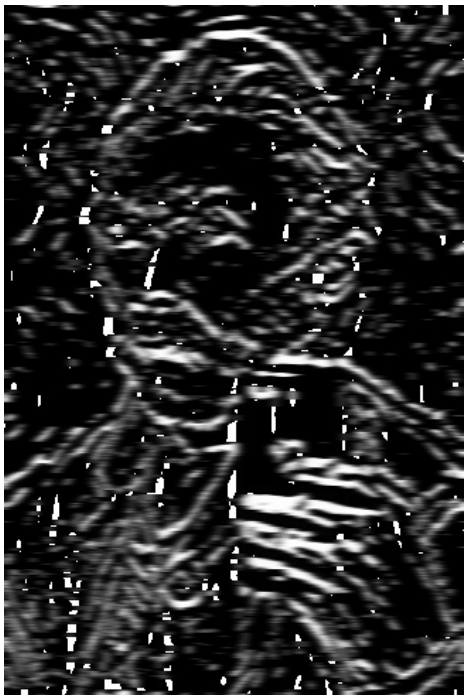
   . Hence, the output filter, f, obtained from $f_1 + f_2$ is a separable filter as well since it can be written as a outer product of $[2,2,2]$ and $[2,2,2]^T$.

7. Apply the derivative of Gaussian filter and Laplacian of Gaussian filter to portrait.jpg, show your results.

Solution:

The code snippets for this question can be found in $a1.py$. First, we show the output for applying the derivative of Gaussian filter with sigma equals to 3 to portrait.jpg below.

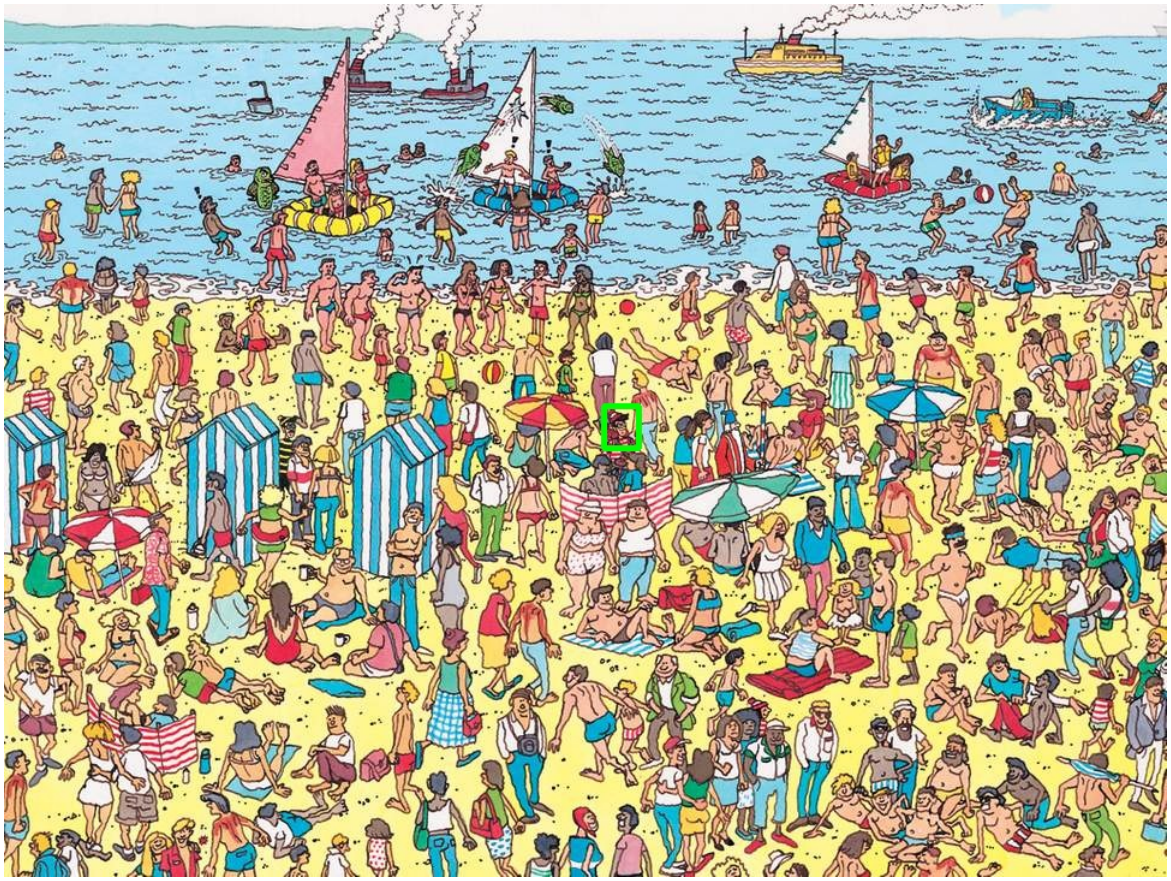
Then, we show the output for applying the Laplacian of Gaussian filter with sigma equals to 3 to portrait.jpg below.



8. Detect waldo.jpg in whereswaldo.jpg using correlation (use built-in methods). Your output should show whereswaldo.jpg with a rectangle around waldo.

Solution:

For this question, the code snippets are able to be found in $a1.py$. The output, whereswaldo.jpg, with a green rectangle around waldo is shown below.



9. How does Canny edge detection work? In your explanation, state the purpose of each step.

Solution:

- The first step is to filter image with the derivative of Gaussian in horizontal and vertical directions. This step is to make the image blur a little bit and get rid of the noise.
- The second step is to find magnitude and orientation of gradient. For this step, the magnitude of the gradient at each pixel is to determine whether this pixel lies on an edge and the large magnitude implies that there is a rapid change in color (implies occurrence of edges). The orientation of the gradient indicates how the edge is oriented. So, this step highlights the pixel that is potentially lie on the edge and produce thick edges.

- The third step is to make non-maximum suppression. This step is to determine whether the pixel is maximum, if it is not the maximum, then it is going to be suppressed, which is for thin out the edges.
- The last step is linking and thresholding and hysteresis. In this step, we define two thresholds, high threshold and low threshold. High threshold is to start edges curves and low threshold is to continue edges curves. By comparing pixel values with thresholds, we are able to find some weak edges and strong edges. Then, by using edge track by hysteresis, we are able to determine whether weak edges found are real edges. If yes, connect the edges together.

10. Briefly explain why the zero crossings of Laplacian of Gaussian can be used to detect edges (Hint: Laplacian is like second derivatives in 2D).

    Solution:

    The zero crossings are the intercepts of the zero-valued axis in a graph, representing the change in the sign of the mathematical function. Since the Laplacian is like second derivatives in 2D, then the zero crossings of Laplacian of Gaussian are the inflections points where the extremes of the gradient graph occur. Since the edge pixels occur at the extremes of the gradient graph, thus the zero crossings of Laplacian of Gaussian can be used to detect edge pixels. Also, inflection points are where the function changes concavity, which implies the discontinuities in the intensity that indicates a rapid change in intensity, implying the occurrence of edges. Therefore, the zero crossings of Laplacian of Gaussian can be used to detect edges.

11. Use Canny Edge detection on portrait.jpg, adjust the parameters to get rid of the details from the background.

    Solution:

    For this question, the code snippets are shown below, which is also can be found in $a1.py$.

```
154  # question 11
155  def doEdgeDetection():
156      # read image of portrait and convert it into grayscale one
157      portrait = cv.imread("./portrait.jpg")
158      portraitGray = cv.cvtColor(portrait, cv.COLOR_BGR2GRAY)
159
160      # do canny edge detection
161      edges = cv.Canny(portrait, 140, 500)
162      cv.imwrite("./result11.jpg", edges)
```

    The output for applying Canny Edge detection with parameters 140 and 500 on portrait.jpg is shown on the next page.