Xiang Chen
1004704992
October 7, 2020

CSC420 Report

Part I: Theory (60 marks)

Q1) LTI Systems and Convolution (20 marks)

In this question, I will first use $\delta(n)$ to represent $x(n)$.

$$
\begin{aligned}
x(n) &= 0 + x(n) + 0 \\
&= \sum_{i=-\infty}^{n-1} 0 + x(n) + \sum_{i=n+1}^{\infty} 0 \\
&= \sum_{i=-\infty}^{n-1} 0 * x(i) + x(n) + \sum_{i=n+1}^{\infty} 0 * x(i) \\
&= \sum_{i=-\infty}^{n-1} \delta(n-i)x(i) + \delta(n-n)x(n) + \sum_{i=n+1}^{\infty} \delta(n-i)x(i) \\
&= \sum_{i=-\infty}^{\infty} x(i)\delta(n-i) \quad (1.1)
\end{aligned}
$$

We can use (1.1) to extend $T[x(n)]$. Then, we can get the following.

$$T[x(n)] = T[\sum_{i=-\infty}^{\infty} x(i)\delta(n-i)] \quad (1.2)$$

Since T is a linear time-invariant system, we can move some constant out in (1.2).

$$\sum_{i=-\infty}^{\infty} x(i)T[\delta(n-i)] \quad (1.3)$$

We simplify (1.3) and get the following.

$$\sum_{i=-\infty}^{\infty} x(i)h(n-i) \quad (1.4)$$

By the definition of convolution, (1.4) is $h(n) * x(n)$.

QED

Q2) Polynomial Multiplication and Convolution (20 marks)

First, let we have some intuitions for this question. When we multiply two polynomials, u and v, we multiply begin the first element in u to the whole elements in v and do this until all elements in u has finished the process. For the convolution of two vectors. We first rotate one of them and then we do the horizon shift, which should give us the same answers. Then, I will prove it mathematically.

Let assume $u = < u_m, u_{m-1}, \ldots, u_1, u_o >$ and $v = < v_n, v_{n-1}, \ldots, v_1, v_0 >$.

Let assume $U(x) = u_m x^m + u_{m-1} x^{m-1} + \ldots + u_0 = \sum_{i=0}^{m} u_i x^i$

$$V(x) = v_n x^n + v_{n-1} x^{n-1} + \ldots + v_0 = \sum_{j=0}^{n} v_j x^j.$$

Then we can get the following.

$$U(x) \cdot V(x) = u_m x^m \cdot V(x) + u_{m-1} x^{m-1} \cdot V(x) + \ldots + u_0 \cdot V(x) = \sum_{i=0}^{m} \sum_{j=0}^{n} u_i x^i v_j x^j \quad (2.1)$$

Let assume $i + j = k$, and we can simplify (2.1).

$$U(x) \cdot V(x) = \sum_{i=0}^{m} \sum_{j=0}^{n} u_i x^i v_j x^j = \sum_{k=0}^{m+n} \sum_{j=max(0,k-m)}^{k} u_{k-j} v_j x^k = u * v \quad (2.2)$$

By the definition of convolution, (2,2) is the convolution of u and v. Therefore, we have the final answers.

QED

Q3) Laplacian Operator (20 marks)

Let assume θ is the rotation angle between axis x and new axis r. We can find relationship among r , r' , x, y and θ.

Let assume $x = \alpha cos(t)$, $y = \alpha sin(t)$, $r = \alpha cos(t + \theta)$, and $r' = \alpha sin(t + \theta)$.

We can use anger formula to get the following.

$$\begin{pmatrix} r \\ r' \end{pmatrix} = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

We can write it into equations.

$$r = cos(\theta)x - sin(\theta)y$$

$$r' = sin(\theta)x + cos(\theta)y$$

Here, what we what to prove is

$$I_{rr} + I_{r'r'} = I_{xx} + I_{yy}.$$

By chain rule, we can get the following.

$$I_x = \frac{\partial I}{\partial x} = \frac{\partial I}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial I}{\partial r'}\frac{\partial r'}{\partial x}$$

$$\rightarrow \quad I_r(cos(\theta)x - sin(\theta)y)_x + I_{r'}(sin(\theta)x + cos(\theta)y)_x$$

$$\rightarrow \quad cos(\theta)I_r + sin(\theta)I_{r'}$$

We need to differentiate again on I_x to get I_{xx}.

$$I_{xx} = (I_x)_x = \frac{\partial I_x}{\partial x} = \frac{\partial I_x}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial I_x}{\partial r'}\frac{\partial r'}{\partial x}$$

$$\rightarrow \quad \frac{\partial(cos(\theta)I_r + sin(\theta)I_{r'})}{\partial r}cos(\theta) + \frac{\partial(cos(\theta)I_r + sin(\theta)I_{r'})}{\partial r'}sin(\theta)$$

$$\rightarrow \quad (cos(\theta)I_{rr} + sin(\theta)I_{rr'})cos(\theta) + (cos(\theta)I_{rr'} + sin(\theta)I_{r'r'})sin(\theta)$$

$$\rightarrow \quad cos^2(\theta)I_{rr} + sin(\theta)cos(\theta)I_{rr'} + sin(\theta)cos(\theta)I_{rr'} + sin^2(\theta)I_{r'r'}$$

$$\rightarrow \quad cos^2(\theta)I_{rr} + 2sin(\theta)cos(\theta)I_{rr'} + sin^2(\theta)I_{r'r'} \quad (3.1)$$

By chain rule, we can get I_y.

$$I_y = \frac{\partial I}{\partial y} = \frac{\partial I}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial I}{\partial r'}\frac{\partial r'}{\partial y}$$

$$\rightarrow \quad I_r(cos(\theta)x - sin(\theta)y)_y + I_{r'}(sin(\theta)x + cos(\theta)y)_y$$

$$\rightarrow \quad -sin(\theta)I_r + cos(\theta)I_{r'}$$

Then, we do the same procedure for I_y to get I_{yy}.

$$I_{yy} = (I_y)_y = \frac{\partial I_y}{\partial y} = \frac{\partial I_y}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial I_y}{\partial r'}\frac{\partial r'}{\partial y}$$

$$\rightarrow \quad \frac{\partial(-sin(\theta)I_r + cos(\theta)I_{r'})}{\partial r}(-sin(\theta)) + \frac{\partial(-sin(\theta)I_r + cos(\theta)I_{r'})}{\partial r'}(cos(\theta))$$

$$\rightarrow \quad (-sin(\theta)I_{rr} + cos(\theta)I_{r'r})(-sin(\theta)) + (-sin(\theta)I_{rr'} + cos(\theta)I_{r'r'})(cos(\theta))$$

$$\rightarrow \quad sin^2(\theta)I_{rr} - 2cos(\theta)sin(\theta)I_{rr'} + cos^2(\theta)I_{r'r'} \quad (3.2)$$

We use (3.1) and (3.2) to get our final answer.

$$I_{xx} + I_{yy} = cos^2(\theta)I_{rr} + 2sin(\theta)cos(\theta)I_{rr'} + sin^2(\theta)I_{r'r'} + sin^2(\theta)I_{rr} - 2cos(\theta)sin(\theta)I_{rr'} + cos^2(\theta)I_{r'r'}$$

$$\rightarrow \quad cos^2(\theta)I_{rr} + sin^2(\theta)I_{r'r'} + sin^2(\theta)I_{rr} + cos^2(\theta)I_{r'r'}$$

$$\rightarrow \quad (cos^2(\theta) + sin^2(\theta))(I_{rr} + I_{r'r'})$$

$$\rightarrow \quad I_{rr} + I_{r'r'}$$

Then, we finish our proof.

QED

Part II: Application (90 marks)

Q4) Edge Detection (45 marks)

Step one:

When sigma is 1.5 and the kernel size is 3, we get the following matrix.

[0.04535423476987057, 0.05664058479678963, 0.04535423476987057]

[0.05664058479678963, 0.0707355302630646, 0.05664058479678963]

[0.04535423476987057, 0.05664058479678963, 0.04535423476987057]

When sigma is 0.5 and the kernel size is 5, we get the following matrix.

[7.164211731312074e-08, 2.890249295089729e-05, 0.00021356214181312774, 2.890249295089729e-05, 7.164211731312074e-08]

[2.890249295089729e-05, 0.011660097860112774, 0.08615711720739454, 0.011660097860112774, 2.890249295089729e-05]

[0.00021356214181312774, 0.08615711720739454, 0.6366197723675814, 0.08615711720739454, 0.00021356214181312774]

[2.890249295089729e-05, 0.011660097860112774, 0.08615711720739454, 0.011660097860112774, 2.890249295089729e-05]

[7.164211731312074e-08, 2.890249295089729e-05, 0.00021356214181312774, 2.890249295089729e-05, 7.164211731312074e-08]

Step two:

This is my convolution function. It takes four parameters: the filter matrix, x position of the pixel, y position of the pixel, and the array of image.

```
def convolution(matrix,x,y,src):
# This function is only for grey scale image. #
# Please use getGreyImge first, if input is a RGB image. #
# Can be optimized by the product of two vectors. #
    x_boundary, y_boundary = src.shape
    kernel_size = len(matrix)
# Help to track the row in Matrix. #
    index_i = 0
    res = 0
# Get the start and end of k. #
    start = int(-(kernel_size-1)/2)
    end = int((kernel_size-1)/2 + 1)
    for u in range(start,end):
# Help to track the coloum in Matrix. #
        index_j = 0
        for v in range(start,end):
# Boundary check. Smarter than padding the image. #
            if(x-u<0 or y-v<0 or x-u>=x_boundary or y-v>=y_boundary): res += 0
            else: res += src[x-u][y-v] * matrix[index_i][index_j]
            index_j += 1
        index_i += 1
    return res
```

This is my Sobel_Operation function. It takes one parameters, the array of the image.

```
49 def Sobel_Operation(src):
50 # This function is only for grey scale image. #
51 # Please use getGreyImge first, if input is a RGB image. #
52     sobel_x = [[-1,0,1],[-2,0,2],[-1,0,1]]
53     sobel_y = [[-1,-2,-1],[0,0,0],[1,2,1]]
54     x_length, y_length = src.shape
55     res = np.empty((x_length,y_length), dtype=float)
56     for i in range(x_length):
57         for j in range(y_length):
58             res[i][j] = math.sqrt(convolution(sobel_x,i,j,src)**2+convolution(sobel_y,i,j,src)**2)
59     return res
```

Step three:

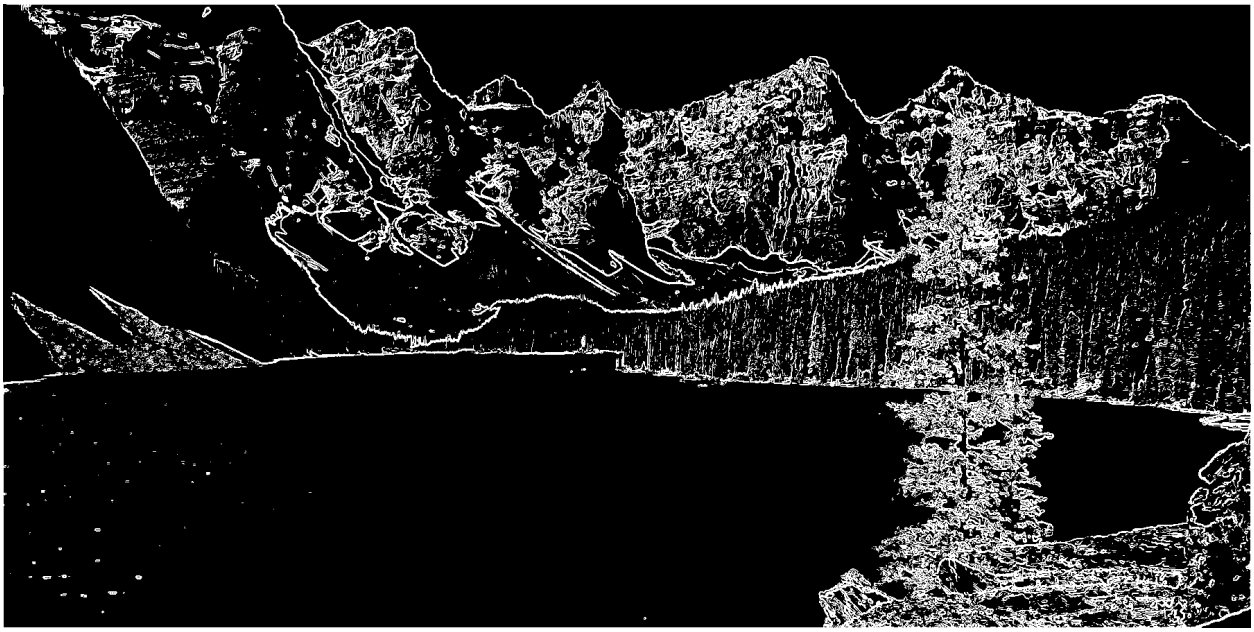
This is my Threshold_Algorithm function.

```
61 def Threshold_Algorithm(gradients):
62     tau = 0
63     h, w = gradients.shape
64     for i in range(h):
65         for j in range(w):
66             tau += gradients[i][j]
67     tau = tau / (h*w)
68     tau_0 = tau
69     tau_1 = -1
70     while 1:
71         lower_bound = []
72         upper_bound = []
73         for i in range(h):
74             for j in range(w):
75                 if gradients[i][j]<tau_0: lower_bound.append(gradients[i][j])
76                 else: upper_bound.append(gradients[i][j])
77         M_L = sum(lower_bound) / len(lower_bound)
78         M_H = sum(upper_bound) / len(upper_bound)
79         tau_1 = (M_L+M_H) / 2
80 # Set epsilon to 0.0000001 (enough small)#
81         if(abs(tau_0-tau_1)<=0.0000001): break
82         tau_0 = tau_1
83     edge_map = np.empty((h,w), dtype=float)
84     for i in range(h):
85         for j in range(w):
86             if(gradients[i][j]>=tau_1): edge_map[i][j] = 255
87             else: edge_map[i][j] = 0
88     return edge_map
```

Step four:

This is the binary picture for Q4_image_1.jpg.

This is the binary picture for Q4_image_2.jpg.



This is the binary picture for my own picture .

Threshold algorithm is very easy to implement and has a nice effect on the edge processing. We can see all binary images have clear and sharpen edges. We could see there are still some artifacts in the final image. Maybe we can find a more accruable algorithm to change it or simply choose a better filter. Threshold algorithm also need a lot of division operations. It may slow down the process when the input is extremely large since division operations need more atomic operations.

Q5) Connected-component labeling (30 marks):

This is the data structure of queue which has the property of FIFO.

```
class Queue:
# Class queue has FIFO properity. #
    def __init__(self):
        self.list = []

    def push(self,item):
        self.list.insert(0,item)

    def pop(self):
        return self.list.pop()

    def isEmpty(self):
        return len(self.list) == 0

```

This is the function to check 8 neighbours of one specific pixel.

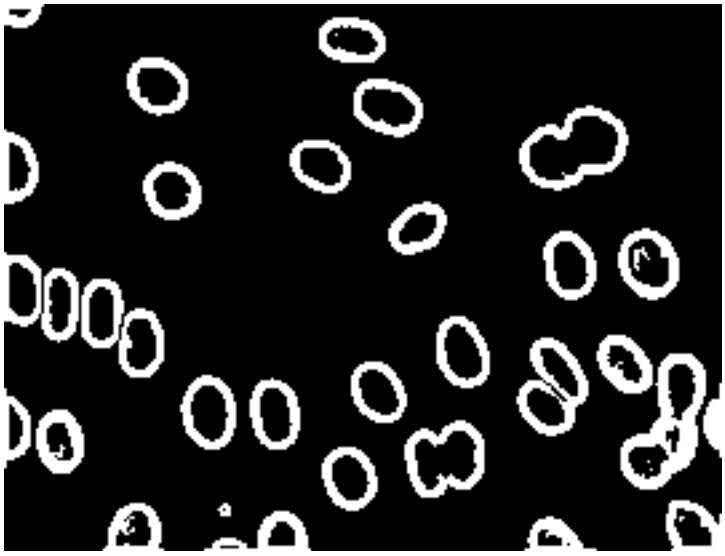
```
def neighbor_check(src,queue,labels):
# Check the eight neighbours. #
    x_boundary, y_boundary = src.shape
    while 1:
        if(queue.isEmpty()): break
        position ,counter =  queue.pop()
        x = position[0]
        y = position[1]
        for i in range(-1,2):
            for j in range(-1,2):
# Boundary situation #
                if (x+i<0 or x+i>=x_boundary or y+j<0 or y+j>=y_boundary): continue
                if src[x+i][y+j]==255 and labels[x+i][y+j]==-1:
                    labels[x+i][y+j] = counter
                    queue.push(((x+i,y+j),counter))
# Return the new counter value. #
    return counter + 1
```

This is the main function of Connected-component labeling.

```
def CC_label(src):
# This function is only for grey scale image. #
# Please use getGreyImge (q4_code) first, if input is a RGB image. #
# Construction a new queue. #
    queue = Queue()
    x, y = src.shape
    counter = 1
    labels = np.empty((x,y), dtype=int)
# Initilize the labels. #
    for i in range(x):
        for j in range(y):
            labels[i][j] = -1
    for i in range(x):
        for j in range(y):
# Foreground check #
            if(src[i][j]==255 and labels[i][j]==-1):
                queue.push(((i,j),counter))
                labels[i][j] = counter
                counter = neighbor_check(src,queue,labels)
    return (labels ,counter)
```

Q6) Count number of cells (15 marks)

I use Gaussian Matrix with sigma is 3 and kernel size is 3, and then I get the number of 36.



To improve the accuracy for the estimation, we can have a better filter to get rid more noise. We can see there are still many small artifacts in some cells above the picture. We can have a better Gaussian Matrix to eliminate these small artifacts. Maybe we can have a Gaussian Matrix with larger kernel size. We can also have a heuristic searching algorithm based on the cycle bound. Instead of label the pixels with eight neighbours, we will check whether these pixels come up with a cycle or not. We do not judge the pixel only with its neighbours. We will go over the picture and collect enough information to label one pixel. We can also apply convolution neural networks with deep learning model to train our program to recognize the cells and count them. However, what I have mentioned may take a large time complexity and space complexity.