Computer Vision Homework #9

Student-ID: r10944020; Name: 林顯倫; Department: GINM

[Results]

(a) Robert's Operator: 12

(b) Prewitt's Edge Detector: 24

(c) Sobel's Edge Detector: 38

(d) Frei and Chen's Gradient Operator: 30

(e) Kirsch's Compass Operator: 135

(f) Robinson's Compass Operator: 43

(g) Nevatia-Babu 5x5 Operator: 12500

[Code Fragment & Explanation]

Part1. Operators 以下各個運算子，都是依照課程投影片的說明

(a) Robert's Operator: 12

```
40 def roberts(img):
41     imageW, imageH = img.shape
42     r1 = np.zeros((imageW-2, imageH-2), dtype='int32')
43     r2 = np.zeros((imageW-2, imageH-2), dtype='int32')
44     kernel1 = np.array([[1, 0], [0, -1]])
45     kernel2 = np.array([[0, 1], [-1, 0]])
46     for x in range(1,imageW-1):
47         for y in range(1,imageH-1):
48             r1[x-1][y-1] = convolution(img[x:x+2, y:y+2], kernel1)
49             r2[x-1][y-1] = convolution(img[x:x+2, y:y+2], kernel2)
50     return np.sqrt(r1**2+r2**2)
```

(b) Prewitt's Edge Detector: 24

```
52 def prewittsEdge(img):
53     imageW, imageH = img.shape
54     p1 = np.zeros((imageW-2, imageH-2), dtype='int32')
55     p2 = np.zeros((imageW-2, imageH-2), dtype='int32')
56     kernel1 = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
57     kernel2 = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
58     for x in range(imageW-2):
59         for y in range(imageH-2):
60             p1[x][y] = convolution(img[x:x+3, y:y+3], kernel1)
61             p2[x][y] = convolution(img[x:x+3, y:y+3], kernel2)
62     return np.sqrt(p1**2+p2**2)
```

(c) Sobel's Edge Detector: 38

```
64 def sobelsEdge(img):
65     imageW, imageH = img.shape
66     s1 = np.zeros((imageW-2, imageH-2), dtype='int32')
67     s2 = np.zeros((imageW-2, imageH-2), dtype='int32')
68     kernel1 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
69     kernel2 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
70     for x in range(imageW-2):
71         for y in range(imageH-2):
72             s1[x][y] = convolution(img[x:x+3, y:y+3], kernel1)
73             s2[x][y] = convolution(img[x:x+3, y:y+3], kernel2)
74     return np.sqrt(s1**2+s2**2)
```

(d) Frei and Chen's Gradient Operator: 30

```
76 def freiAndChensGradient(img):
77     imageW, imageH = img.shape
78     f1 = np.zeros((imageW-2, imageH-2), dtype='int32')
79     f2 = np.zeros((imageW-2, imageH-2), dtype='int32')
80     kernel1 = np.array([[-1, -np.sqrt(2), -1], [0, 0, 0], [1, np.sqrt(2), 1]])
81     kernel2 = np.array([[-1, 0, 1], [-np.sqrt(2), 0, np.sqrt(2)], [-1, 0, 1]])
82     for x in range(imageW-2):
83         for y in range(imageH-2):
84             f1[x][y] = convolution(img[x:x+3, y:y+3], kernel1)
85             f2[x][y] = convolution(img[x:x+3, y:y+3], kernel2)
86     return np.sqrt(f1**2+f2**2)
```

(e) Kirsch's Compass Operator: 135

```
88 def kirschsCompass(img):
89     imageW, imageH = img.shape
90     res = np.zeros((imageW-2, imageH-2), dtype='int32')
91     kernel1 = np.array([[-3, -3, 5], [-3, 0, 5], [-3, -3, 5]])
92     kernel2 = np.array([[-3, 5, 5], [-3, 0, 5], [-3, -3, -3]])
93     kernel3 = np.array([[5, 5, 5], [-3, 0, -3], [-3, -3, -3]])
94     kernel4 = np.array([[5, 5, -3], [5, 0, -3], [-3, -3, -3]])
95     kernel5 = np.array([[5, -3, -3], [5, 0, -3], [5, -3, -3]])
96     kernel6 = np.array([[-3, -3, -3], [5, 0, -3], [5, 5, -3]])
97     kernel7 = np.array([[-3, -3, -3], [-3, 0, -3], [5, 5, 5]])
98     kernel8 = np.array([[-3, -3, -3], [-3, 0, 5], [-3, 5, 5]])
99     for x in range(imageW-2):
100        for y in range(imageH-2):
101            res[x][y] = max( convolution(img[x:x+3, y:y+3], kernel1),
102                convolution(img[x:x+3, y:y+3], kernel2),
103                convolution(img[x:x+3, y:y+3], kernel3),
104                convolution(img[x:x+3, y:y+3], kernel4),
105                convolution(img[x:x+3, y:y+3], kernel5),
106                convolution(img[x:x+3, y:y+3], kernel6),
107                convolution(img[x:x+3, y:y+3], kernel7),
108                convolution(img[x:x+3, y:y+3], kernel8))
109    return res
```

(f) Robinson's Compass Operator: 43

```
111 def robinsonsCompass(img):
112     imageW, imageH = img.shape
113     res = np.zeros((imageW-2, imageH-2), dtype='int32')
114     kernel1 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
115     kernel2 = np.array([[0, 1, 2], [-1, 0, 1], [-2, -1, 0]])
116     kernel3 = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
117     kernel4 = np.array([[2, 1, 0], [1, 0, -1], [0, -1, -2]])
118     kernel5 = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
119     kernel6 = np.array([[0, -1, -2], [1, 0, -1], [2, 1, 0]])
120     kernel7 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
121     kernel8 = np.array([[-2, -1, 0], [-1, 0, 1], [0, 1, 2]])
122     for x in range(imageW-2):
123         for y in range(imageH-2):
124             res[x][y] = max( convolution(img[x:x+3, y:y+3], kernel1),
125                 convolution(img[x:x+3, y:y+3], kernel2),
126                 convolution(img[x:x+3, y:y+3], kernel3),
127                 convolution(img[x:x+3, y:y+3], kernel4),
128                 convolution(img[x:x+3, y:y+3], kernel5),
129                 convolution(img[x:x+3, y:y+3], kernel6),
130                 convolution(img[x:x+3, y:y+3], kernel7),
131                 convolution(img[x:x+3, y:y+3], kernel8))
132     return res
```

(g) Nevatia-Babu 5x5 Operator: 12500

```
134 def nevatiaBabu(img):
135     imageW, imageH = img.shape
136     res = np.zeros((imageW-5, imageH-5), dtype='int32')
137     kernel1 = np.array([[100, 100, 100, 100, 100],
138                         [100, 100, 100, 100, 100],
139                         [0, 0, 0, 0, 0],
140                         [-100, -100, -100, -100, -100],
141                         [-100, -100, -100, -100, -100]])
142     kernel2 = np.array([[100, 100, 100, 100, 100],
143                         [100, 100, 100, 78, -32],
144                         [100, 92, 0, -92, -100],
145                         [32, -78, -100, -100, -100],
146                         [-100, -100, -100, -100, -100]])
147     kernel3 = np.array([[100, 100, 100, 32, -100],
148                         [100, 100, 92, -78, -100],
149                         [100, 100, 0, -100, -100],
150                         [100, 78, -92, -100, -100],
151                         [100, -32, -100, -100, -100]])
152     kernel4 = np.array([[-100, -100, 0, 100, 100],
153                         [-100, -100, 0, 100, 100],
154                         [-100, -100, 0, 100, 100],
155                         [-100, -100, 0, 100, 100],
156                         [-100, -100, 0, 100, 100]])
157     kernel5 = np.array([[-100, 32, 100, 100, 100],
158                         [-100, -78, 92, 100, 100],
159                         [-100, -100, 0, 100, 100],
160                         [-100, -100, -92, 78, 100],
161                         [-100, -100, -100, -32, 100]])
162     kernel6 = np.array([[100, 100, 100, 100, 100],
163                         [-32, 78, 100, 100, 100],
164                         [-100, -92, 0, 92, 100],
165                         [-100, -100, -100, -78, 32],
166                         [-100, -100, -100, -100, -100]])
167     for x in range(imageW-5):
168         for y in range(imageH-5):
169             res[x][y] = max( convolution(img[x:x+5, y:y+5], kernel1),
170                 convolution(img[x:x+5, y:y+5], kernel2),
171                 convolution(img[x:x+5, y:y+5], kernel3),
172                 convolution(img[x:x+5, y:y+5], kernel4),
173                 convolution(img[x:x+5, y:y+5], kernel5),
174                 convolution(img[x:x+5, y:y+5], kernel6))
175     return res
```

Part2. 以上運算子都有運用的 convolution 函式

```
31 def convolution(img, kernel):
32     imageW, imageH = img.shape
33     kernelW, kernelH = len(kernel), len(kernel[0])
34     res = 0
35     for x in range(imageW):
36         for y in range(imageH):
37             res += img[x][y] * kernel[x][y]
38     return res
```

Part3. 在影像傳入之前，還要先做padding的動作。

而Padding又分成兩種，向外補一圈以及向外補兩圈。

向外補一圈 = padding_img_3

向外補兩圈 = padding_img_5

```python
def padding_img_5(img):
    return padding_img_3(img)

def padding_img_3(img):
    imageW, imageH = img.shape
    # imageW, imageH = len(img), len(img[0])
    newImageW, newImageH = imageW+2, imageH+2
    res = np.zeros((newImageW, newImageH), dtype='int32')
    # corner
    res[0][0] = img[0][0]
    res[0][newImageH-1] = img[0][imageH-1]
    res[newImageW-1][0] = img[imageW-1][0]
    res[newImageW-1][newImageH-1] = img[imageW-1][imageH-1]
    # border
    for idx_x in range(1,newImageW-1):
        res[idx_x][0] = img[idx_x-1][0]
        res[idx_x][newImageH-1] = img[idx_x-1][imageH-1]
    for idx_y in range(1,newImageH-1):
        res[0][idx_y] = img[0][idx_y-1]
        res[newImageW-1][idx_y] = img[imageW-1][idx_y-1]
    # original case
    for x in range(1, newImageW-1):
        for idx_y in range(1,newImageH-1):
            res[x][idx_y] = img[x-1][idx_y-1]
    return res
```