Computer Vision HW9 楊閔喻 R09921012

Write a program to generate images and histograms:

(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

每個 Operator 的實作皆為透過函式對 pixel 與其臨域的 pixel 作取值運算，僅 Nevatia-Babu 因其需要操作的範圍較大，因此先在程式中意用 expand 函式將原圖擴充後，再進行操作。每個 operator 得到判斷值後，依據定義與 threshold 進行比較。每個函式的計算方法詳見附錄。

附錄

(a) Robert's Operator: 12

```
def Roberts(image, i, j, threshold):

    x0 = x1 = x2 = x3 = 0

    x0 = float(image[i, j])

    if j+1 < image.shape[1]:
        x1 = float(image[i, j+1])
        if i+1 < image.shape[0]:
            x3 = float(image[i+1, j+1])
        else:
            x3 = x1
    else:
        x1 = x0
        x3 = x0

    if i+1 < image.shape[0]:
        x2 = float(image[i+1, j])
    else:
        x2 = x0

    r1 = x3 - x0
    r2 = x2 - x1
    if math.sqrt(r1**2+r2**2) >= threshold:
        return 0
    else:
        return 255
```

(b) Prewitt's Edge Detector: 24

```
 x1 = x2 = x3 = x4 = x5 = x6 = x7 = x8 = 0.

    if i-1 >= 0:
        x2 = float(image[i-1, j])
        if j-1 >= 0:
```

```
            x7 = float(image[i-1, j-1])

    if j+1 < image.shape[1]:
        x1 = float(image[i, j+1])
        if i-1 >= 0:
            x6 = float(image[i-1, j+1])

    if j-1 >= 0:
        x3 = float(image[i, j-1])
        if i+1 < image.shape[0]:
            x8 = float(image[i+1, j-1])

    if i+1 < image.shape[0]:
        x4 = float(image[i+1, j])
        if j+1 < image.shape[1]:
            x5 = float(image[i+1, j+1])

    p1 = - x2 - x6 - x7 + x4 + x5 + x8
    p2 = x1 + x5 + x6 - x3 - x7 - x8

    if math.sqrt(p1**2+p2**2) >= threshold:
        return 0
    else:
        return 255
```

(c) Sobel's Edge Detector: 38

```
def Sobel(image, i, j, threshold):

    x1 = x2 = x3 = x4 = x5 = x6 = x7 = x8 = 0.

    if i-1 >= 0:
        x2 = float(image[i-1, j])
        if j-1 >= 0:
            x7 = float(image[i-1, j-1])

    if j+1 < image.shape[1]:
        x1 = float(image[i, j+1])
```

```
        if i-1 >= 0:
            x6 = float(image[i-1, j+1])

    if j-1 >= 0:
        x3 = float(image[i, j-1])
        if i+1 < image.shape[0]:
            x8 = float(image[i+1, j-1])

    if i+1 < image.shape[0]:
        x4 = float(image[i+1, j])
        if j+1 < image.shape[1]:
            x5 = float(image[i+1, j+1])

    s1 = - 2*x2 - x6 - x7 + 2*x4 + x5 + x8
    s2 = 2*x1 + x5 + x6 - 2*x3 - x7 - x8

    if math.sqrt(s1**2+s2**2) >= threshold:
        return 0
    else:
        return 255
```

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

```
def FreiandChen(image, i, j, threshold):

    x1 = x2 = x3 = x4 = x5 = x6 = x7 = x8 = 0.

    if i-1 >= 0:
        x2 = float(image[i-1, j])
        if j-1 >= 0:
            x7 = float(image[i-1, j-1])

    if j+1 < image.shape[1]:
        x1 = float(image[i, j+1])
        if i-1 >= 0:
            x6 = float(image[i-1, j+1])

    if j-1 >= 0:
```

```
        x3 = float(image[i, j-1])
        if i+1 < image.shape[0]:
            x8 = float(image[i+1, j-1])

    if i+1 < image.shape[0]:
        x4 = float(image[i+1, j])
        if j+1 < image.shape[1]:
            x5 = float(image[i+1, j+1])

    f1 = - math.sqrt(2)*x2 - x6 - x7 + math.sqrt(2)*x4 + x5 + x8
    f2 = math.sqrt(2)*x1 + x5 + x6 - math.sqrt(2)*x3 - x7 - x8

    if math.sqrt(f1**2+f2**2) >= threshold:
        return 0
    else:
        return 255
```

(e) Kirsch's Compass Operator: 135

```
def Kirsch(image, i, j, threshold):

    x1 = x2 = x3 = x4 = x5 = x6 = x7 = x8 = 0.

    if i-1 >= 0:
        x2 = float(image[i-1, j])
        if j-1 >= 0:
            x7 = float(image[i-1, j-1])

    if j+1 < image.shape[1]:
        x1 = float(image[i, j+1])
        if i-1 >= 0:
            x6 = float(image[i-1, j+1])

    if j-1 >= 0:
        x3 = float(image[i, j-1])
        if i+1 < image.shape[0]:
            x8 = float(image[i+1, j-1])
```

```
    if i+1 < image.shape[0]:
        x4 = float(image[i+1, j])
        if j+1 < image.shape[1]:
            x5 = float(image[i+1, j+1])

    k0 = 5*x1-3*x2-3*x3-3*x4+5*x5+5*x6-3*x7-3*x8
    k1 = 5*x1+5*x2-3*x3-3*x4-3*x5+5*x6-3*x7-3*x8
    k2 = -3*x1+5*x2-3*x3-3*x4-3*x5+5*x6+5*x7-3*x8
    k3 = -3*x1+5*x2+5*x3-3*x4-3*x5-3*x6+5*x7-3*x8
    k4 = -3*x1-3*x2+5*x3-3*x4-3*x5-3*x6+5*x7+5*x8
    k5 = -3*x1-3*x2+5*x3+5*x4-3*x5-3*x6-3*x7+5*x8
    k6 = -3*x1-3*x2-3*x3+5*x4+5*x5-3*x6-3*x7+5*x8
    k7 = 5*x1-3*x2-3*x3+5*x4+5*x5-3*x6-3*x7-3*x8

    if max(k0, k1, k2, k3, k4, k5, k6, k7) >= threshold:
        return 0
    else:
        return 255
```

(f) Robinson's Compass Operator: 43

```
def Robinson(image, i, j, threshold):
    x1 = x2 = x3 = x4 = x5 = x6 = x7 = x8 = 0.

    if i-1 >= 0:
        x2 = float(image[i-1, j])
        if j-1 >= 0:
            x7 = float(image[i-1, j-1])

    if j+1 < image.shape[1]:
        x1 = float(image[i, j+1])
        if i-1 >= 0:
            x6 = float(image[i-1, j+1])

    if j-1 >= 0:
        x3 = float(image[i, j-1])
        if i+1 < image.shape[0]:
            x8 = float(image[i+1, j-1])
```

```
    if i+1 < image.shape[0]:
        x4 = float(image[i+1, j])
        if j+1 < image.shape[1]:
            x5 = float(image[i+1, j+1])

    r0 = -x7+x6-2*x3+2*x1-x8+x5
    r1 = x2+2*x6+x1-x3-x4-2*x8
    r2 = 2*x2-2*x4-x5+x6+x7-x8
    r3 = -x1+x2+x3-x4-2*x5+2*x7
    r4 = -2*x1+2*x3-x5-x6+x7+x8
    r5 = -x1-x2+x3+x4-2*x6+2*x8
    r6 = -2*x2+2*x4+x5-x6-x7+x8
    r7 = x1-x2-x3+x4+2*x5-2*x7

    if max(r0, r1, r2, r3, r4, r5, r6, r7) >= threshold:
        return 0
    else:
        return 255
```

(g) Nevatia-Babu 5x5 Operator: 12500

```
def expand(image):
    temp = np.zeros((image.shape[0]+4, image.shape[1]+4), dtype=np.uint
8)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            temp[i+2, j+2] = image[i, j]

    for j in range(image.shape[1]):
        temp[0, j+2] = temp[2, j+2]
        temp[1, j+2] = temp[2, j+2]
        temp[temp.shape[0]-1, j+2] = temp[temp.shape[0]-3, j+2]
        temp[temp.shape[0]-2, j+2] = temp[temp.shape[0]-3, j+2]

    for i in range(image.shape[0]):
        temp[i+2, 0] = temp[i+2, 2]
        temp[i+2, 1] = temp[i+2, 2]
```

```
        temp[i+2, temp.shape[1]-1] = temp[i+2, temp.shape[1]-3]
        temp[i+2, temp.shape[1]-2] = temp[i+2, temp.shape[1]-3]

    for i in range(2):
        for j in range(2):
            temp[i, j] = temp[2, 2]
            temp[i+image.shape[0]+2, j] = temp[temp.shape[0]-3, 2]
            temp[i, j+image.shape[1]+2] = temp[2, temp.shape[1]-3]
            temp[temp.shape[0]-2+i, temp.shape[0]-2 +
                 j] = temp[temp.shape[0]-3, temp.shape[1]-3]

    '''
    for i in range(temp.shape[0]):
        for j in range(temp.shape[1]):
            print(temp[i,j],end=' ')
        print('')
    '''
    return temp


def Nevatia_Babu(image, i, j, threshold):
    x1 = float(image[i-2, j-2])
    x2 = float(image[i-2, j-1])
    x3 = float(image[i-2, j])
    x4 = float(image[i-2, j+1])
    x5 = float(image[i-2, j+2])
    x6 = float(image[i-1, j-2])
    x7 = float(image[i-1, j-1])
    x8 = float(image[i-1, j])
    x9 = float(image[i-1, j+1])
    x10 = float(image[i-1, j+2])
    x11 = float(image[i, j-2])
    x12 = float(image[i, j-1])
    x14 = float(image[i, j+1])
    x15 = float(image[i, j+2])
    x16 = float(image[i+1, j-2])
    x17 = float(image[i+1, j-1])
    x18 = float(image[i+1, j])
    x19 = float(image[i+1, j+1])
```

```
    x20 = float(image[i+1, j+2])
    x21 = float(image[i+2, j-2])
    x22 = float(image[i+2, j-1])
    x23 = float(image[i+2, j])
    x24 = float(image[i+2, j+1])
    x25 = float(image[i+2, j+2])
    N1 = 100*(x1+x2+x3+x4+x5+x6+x7+x8+x9+x10)-100 * \
        (x16+x17+x18+x19+x20+x21+x22+x23+x24+x25)
    N2 = 100*(x1+x2+x3+x4+x5+x6+x7+x8+x11-x15-x18-x19-x20-x21 -
              x22-x23-x24-x25)+78*(x9-x17)+32*(x16-x10)+92*(x12-x14)
    N3 = 100*(x1+x2+x3-x5+x6+x7-x10+x11+x12-x14-x15+x16-x19-x20 +
              x21-x23-x24-x25)+32*(x4-x22)+92*(x8-x18)+78*(x17-x9)
    N4 = 100*(-x1-x6-x11-x16-x21-x2-x7-x12-x17-x22 +
              x4+x9+x14+x19+x24+x5+x10+x15+x20+x25)
    N5 = 100*(-x1+x3+x4+x5-x6+x9+x10-x11-x12+x14+x15-x16-x17+x20 -
              x21-x22-x23+x25)+32*(x2-x24)+78*(x19-x7)+92*(x8-x18)
    N6 = 100*(x1+x2+x3+x4+x5+x8+x9+x10-x11+x15-x16-x17-x18-x21 -
              x22-x23-x24-x25)+32*(x20-x6)+78*(x7-x19)+92*(x14-x12)

    if max(N1, N2, N3, N4, N5, N6) < threshold:
        return 255
    else:
        return 0
```