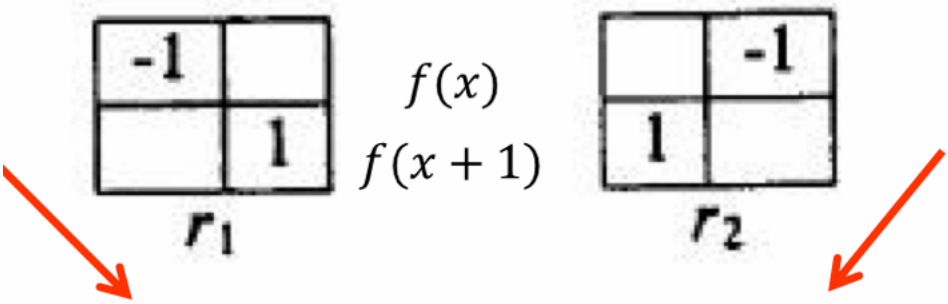
B06902042 劉愷爲

HW9 Report

Part 1: Robert's Operator (threshold = 12)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果 gradient大於threshold則將pixel設為0。gx和gy即為乘完矩陣的值。



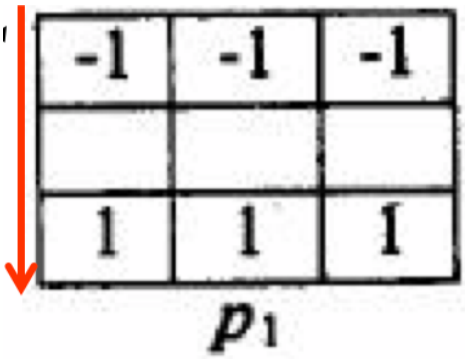

```
def robert(img, threshold):
    tmp = np.zeros((img.shape[0] + 1, img.shape[1] + 1), np.int)
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            tmp[x, y] = img[x, y]
    for x in range(img.shape[0]):
        tmp[x, img.shape[1]] = img[x, img.shape[1] - 1]
    for y in range(img.shape[1]):
        tmp[img.shape[0], y] = img[img.shape[0] - 1, y]

    ans = np.zeros(img.shape, np.int);

    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            ans[x, y] = 255
            gx = int(tmp[x + 1, y + 1]) - int(tmp[x,y])
            gy = int(tmp[x + 1, y]) - int(tmp[x, y + 1])
            g = (gx**2) + (gy**2)
            if g >= (threshold**2):
                ans[x,y] = 0
    return ans
```

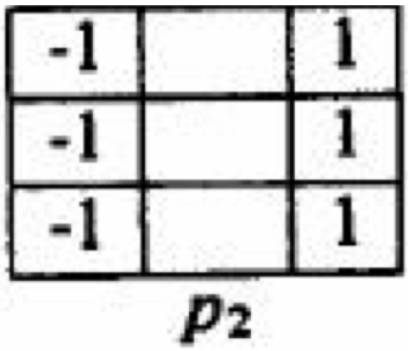
以下部分都會使用到extend，這是我用來擴增矩陣的函數

```
def extend(img):
    img_ext = np.zeros((img.shape[0] + 2, img.shape[1] + 2))
    img_ext[0, 1:-1] = img[0, :]
    img_ext[-1, 1:-1] = img[-1, :]
    img_ext[1:-1, 0] = img[:, 0]
    img_ext[1:-1, -1] = img[:, -1]
    img_ext[0, 0] = img[0, 0]
    img_ext[0, -1] = img[0, -1]
    img_ext[-1, 0] = img[-1, 0]
    img_ext[-1, -1] = img[-1, -1]
    img_ext[1:-1, 1:-1] = img[:, :]
    return img_ext
```

Part 2: Prewitt's edge detector (threshold = 24)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果 gradient大於threshold則將pixel設為0。gx和gy即為乘完矩陣的值。

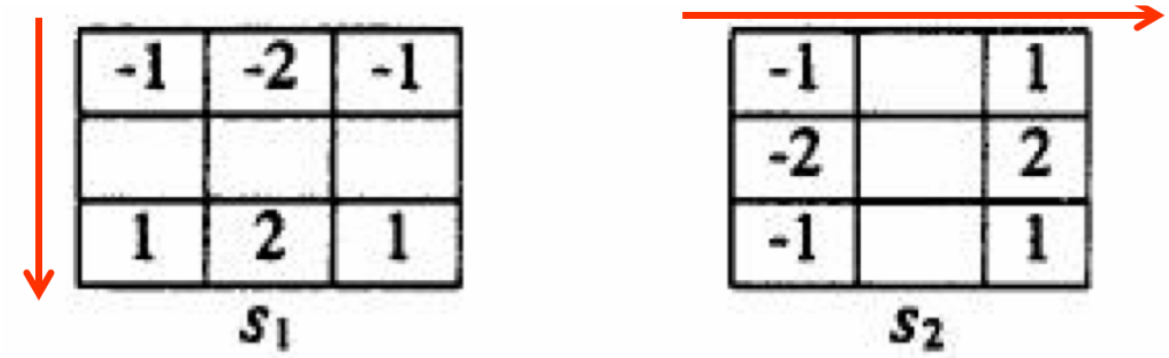





```
def prewitt(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            gx = int(tmp[x + 1, y - 1]) + int(tmp[x + 1, y]) + int(tmp[x + 1, y + 1])
                - int(tmp[x - 1, y - 1]) - int(tmp[x - 1, y]) - int(tmp[x - 1, y + 1])
            gy = int(tmp[x - 1, y + 1]) + int(tmp[x, y + 1]) + int(tmp[x + 1, y + 1])
                - int(tmp[x - 1, y - 1]) - int(tmp[x, y - 1]) - int(tmp[x + 1, y - 1])
            g = (gx**2) + (gy**2)
            if g >= (threshold**2):
                ans[x - 1, y - 1] = 0
    return ans
```

Part 3: Sobel's edge detector (threshold = 38)

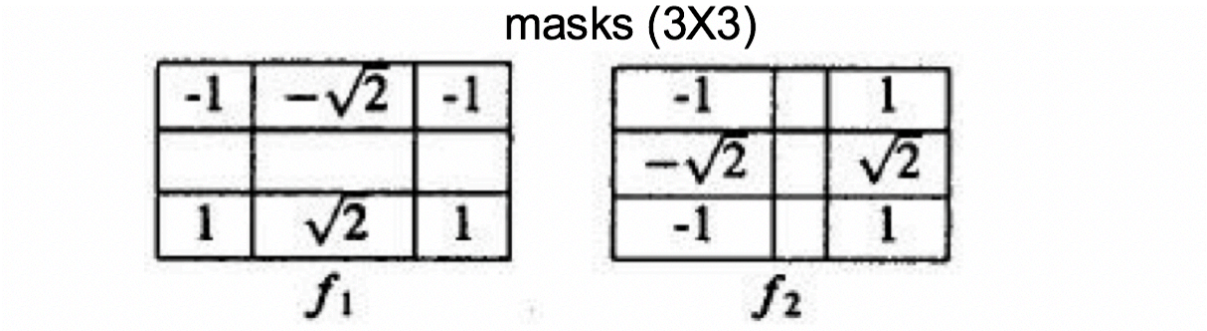
初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果 gradient大於threshold則將pixel設為0。gx和gy即為乘完矩陣的值。




```python
def sobel(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            gx = int(tmp[x + 1, y - 1]) + int(tmp[x + 1, y] * 2) + int(tmp[x + 1, y + 1])
                - int(tmp[x - 1, y - 1]) - int(tmp[x - 1, y] * 2) - int(tmp[x - 1, y + 1])
            gy = int(tmp[x - 1, y + 1]) + int(tmp[x, y + 1] * 2) + int(tmp[x + 1, y + 1])
               - int(tmp[x - 1, y - 1]) - int(tmp[x, y - 1] * 2) - int(tmp[x + 1, y - 1])
            g = (gx**2) + (gy**2)
            if g >= (threshold**2):
                ans[x - 1, y - 1] = 0
    return ans
```

Part 4: Frei and Chen's gradient operator (threshold = 30)

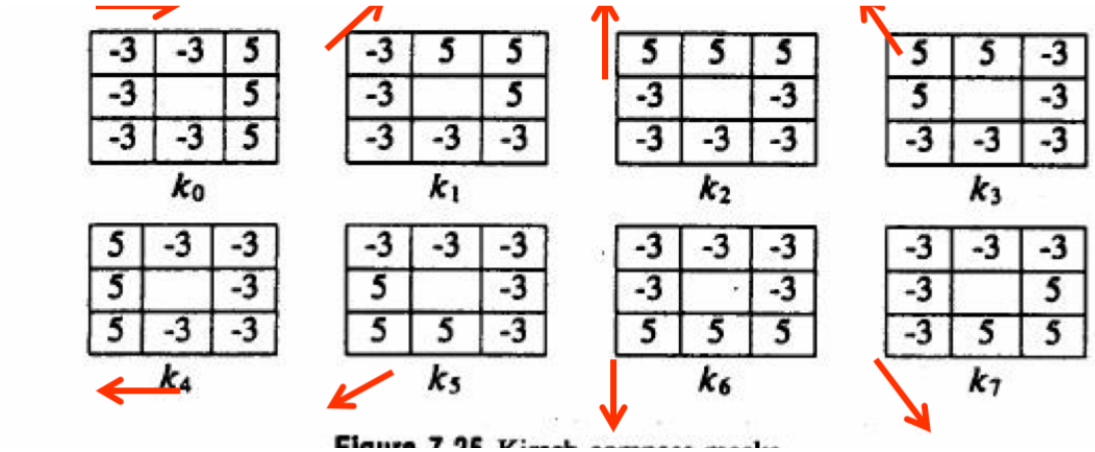
初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，如果 gradient大於threshold則將pixel設為0。gx和gy即為乘完矩陣的值。



```
def frei_chen(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            gx = int(tmp[x + 1, y - 1]) + int(tmp[x + 1, y] * math.sqrt(2)) + int(tmp[x + 1, y + 1])
                - int(tmp[x - 1, y - 1]) - int(tmp[x - 1, y] * math.sqrt(2)) - int(tmp[x - 1, y + 1])
            gy = int(tmp[x - 1, y + 1]) + int(tmp[x, y + 1] * math.sqrt(2)) + int(tmp[x + 1, y + 1])
               - int(tmp[x - 1, y - 1]) - int(tmp[x, y - 1] * math.sqrt(2)) - int(tmp[x + 1, y - 1])
            g = (gx**2) + (gy**2)
            if g >= (threshold**2):
                ans[x - 1, y - 1] = 0
    return ans
```

Part 5: Kirsch's compass operator (threshold = 135)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，取其中最大的gradient，如果 gradient大於threshold則將pixel設為0。gx即為乘完矩陣的值。




```
def kirsch(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    k = [-3, -3, 5, 5, 5, -3, -3, -3]
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            tmp_max = 0
            for z in range(8):
                gx = tmp[x - 1, y - 1] * k[z] + tmp[x - 1, y] * k[(z+1)%8] + tmp[x - 1, y + 1] * k[(z+2)%8]
                    + tmp[x, y + 1] * k[(z+3)%8] + tmp[x + 1, y + 1] * k[(z+4)%8] + tmp[x + 1, y] * k[(z+5)%8]
                    + tmp[x + 1, y - 1] * k[(z+6)%8] +tmp[x, y - 1] * k[(z+7)%8]
                if gx > tmp_max:
                    tmp_max = gx

            g = tmp_max
            if g >= threshold:
                ans[x - 1, y - 1] = 0
    return ans
```

Part 6: Robinson's compass operator (threshold = 43)

初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，取其中最大的gradient，如果 gradient大於threshold則將pixel設為0。gx即為乘完矩陣的值。

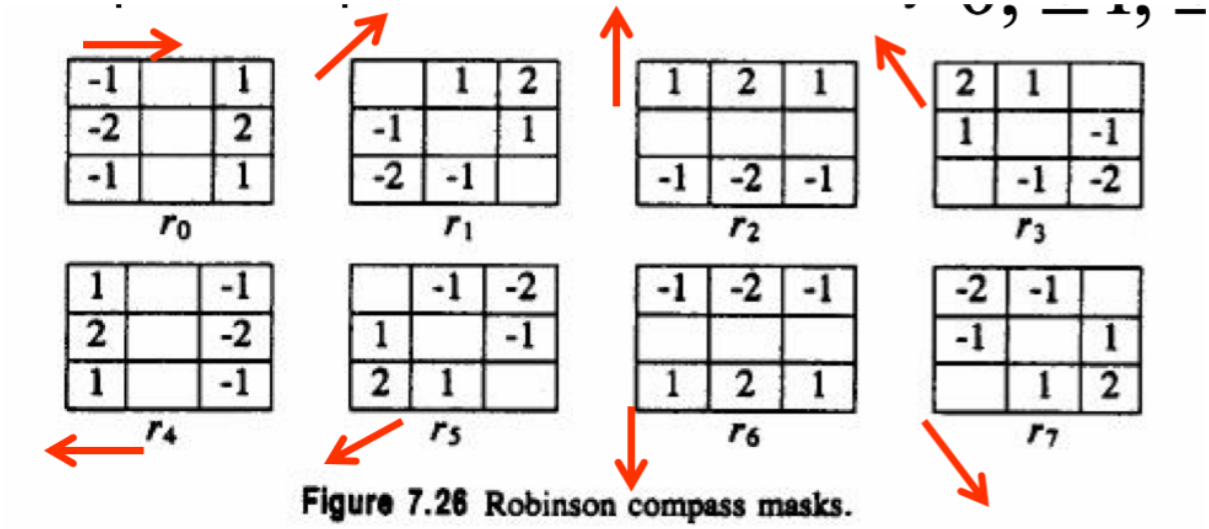
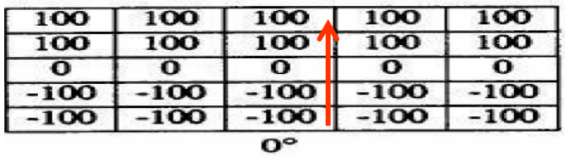


Figure 7.26 Robinson compass masks.

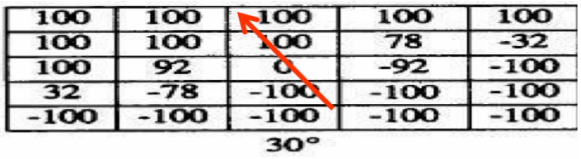
```python
def robinson(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(img)
    k = [-1, 0, 1, 2, 1, 0, -1, -2]
    for x in range(1, img.shape[0] + 1):
        for y in range(1, img.shape[1] + 1):
            ans[x - 1, y - 1] = 255
            tmp_max = 0
            for z in range(8):
                gx = tmp[x - 1, y - 1] * k[z] + tmp[x - 1, y] * k[(z+1)%8] + tmp[x - 1, y + 1] * k[(z+2)%8]
                    + tmp[x, y + 1] * k[(z+3)%8] + tmp[x + 1, y + 1] * k[(z+4)%8] + tmp[x + 1, y] * k[(z+5)%8]
                    + tmp[x + 1, y - 1] * k[(z+6)%8] +tmp[x, y - 1] * k[(z+7)%8]
                if gx > tmp_max:
                    tmp_max = gx

            g = tmp_max
            if g >= threshold:
                ans[x - 1, y - 1] = 0
    return ans
```

Part 7: Nevatia-Babu 5x5 operator (threshold = 12500)

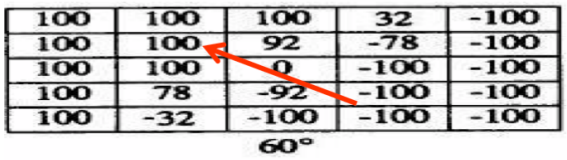
初始化先將所有的pixel設為255，再用下圖的陣列作為mask，用周遭乘以矩陣計算gradient，取其中最大的gradient，如果 gradient大於threshold則將pixel設為0。g0~g5即為暫時的gradient。



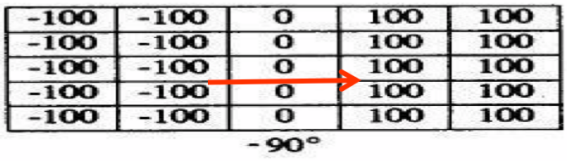
0°



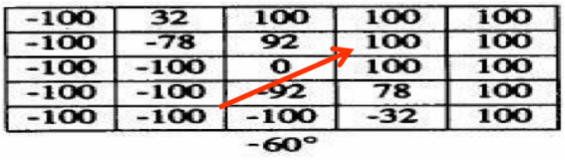
30°



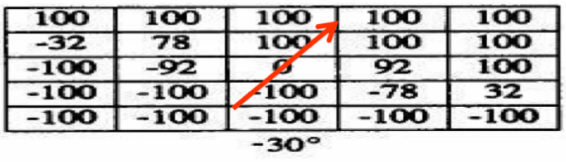
60°



-90°



-60°



-30°

```python
def nevatia(img, threshold):
    ans = np.zeros(img.shape, np.int)
    tmp = extend(extend(img))
    kernel = [ [-2, -2], [-1, -2], [0, -2], [1, -2], [2, -2],
               [-2, -1], [-1, -1], [0, -1], [1, -1], [2, -1],
               [-2, 0], [-1, 0], [0, 0], [1, 0], [2, 0],
               [-2, 1], [-1, 1], [0, 1], [1, 1], [2, 1],
               [-2, 2], [-1, 2], [0, 2], [1, 2], [2, 2] ]
    k0 = [ 100, 100, 0, -100, -100, 100, 100, 0, -100, -100, 100, 100, 0,
           -100, -100, 100, 100, 0, -100, -100, 100, 100, 0, -100, -100]
    k1 = [ 100, 100, 100, 100, 100, 100, 100, 100, 78, -32, 100, 92, 0, -92,
           -100, 32, -78, -100, -100, -100, -100, -100, -100, -100, -100]
    k2 = [-100, -100, -100, -100, -100, 32, -78, -100, -100, -100, 100, 92,
           0, -92, -100, 100, 100, 100, 78, -32, 100, 100, 100, 100, 100]
    k3 = [ 100, 100, 100, 32, -100, 100, 100, 92, -78, -100, 100, 100, 0,
           -100, -100, 100, 78, -92, -100, -100, 100, -32, -100, -100, -100]
    k4 = [ -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, 0,
           0, 0, 0, 0, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100]
    k5 = [ 100, -32, -100, -100, -100, 100, 78, -92, -100, -100, 100, 100,
           0, -100, -100, 100, 100, 92, -78, -100, 100, 100, 100, 32, -100]

    for x in range(2, img.shape[0] + 2):
        for y in range(2, img.shape[1] + 2):
            ans[x - 2, y - 2] = 255
            cnt = 0
            tmp_max = 0
            g0 = g1 = g2 = g3 = g4 = g5 = 0
            for z in kernel:
                g0 += tmp[x + z[0], y + z[1]] * k0[cnt]
                g1 += tmp[x + z[0], y + z[1]] * k1[cnt]
                g2 += tmp[x + z[0], y + z[1]] * k2[cnt]
                g3 += tmp[x + z[0], y + z[1]] * k3[cnt]
                g4 += tmp[x + z[0], y + z[1]] * k4[cnt]
                g5 += tmp[x + z[0], y + z[1]] * k5[cnt]
                cnt += 1

            g = max(g0, g1, g2, g3, g4, g5)
            if g >= threshold:
                ans[x - 2, y - 2] = 0
    return ans
```

robert 12

prewitt 24

sobel 38

frei and chen 30

krisch 135

robinson 43

nevatia 12500