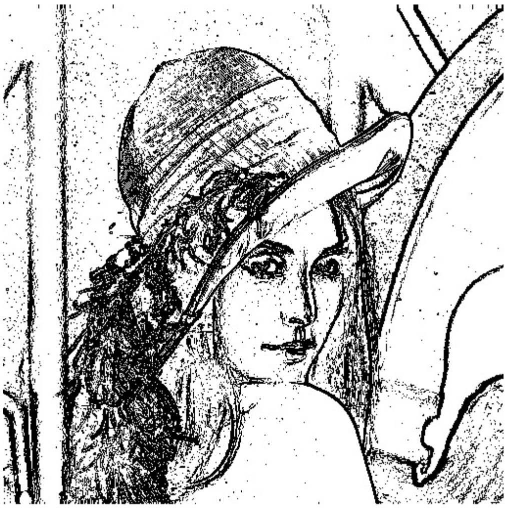
學生 蔡宇晴

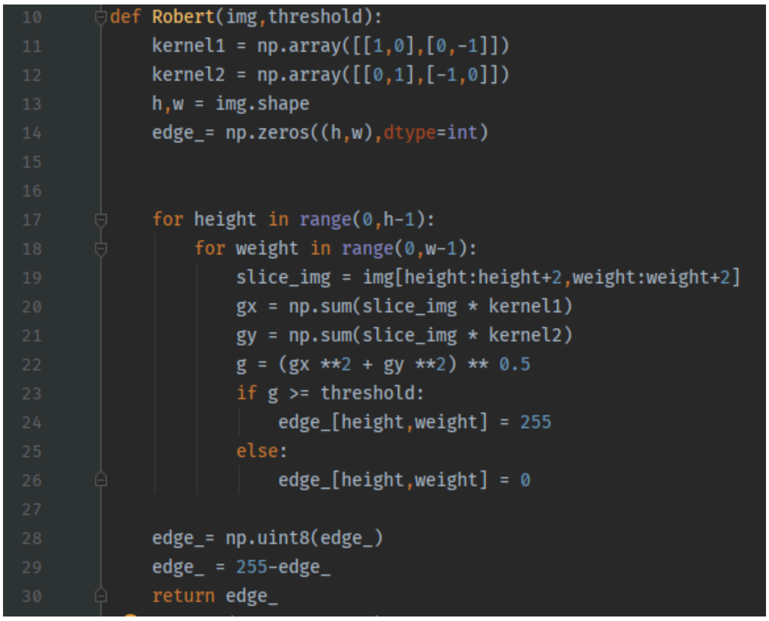
電腦視覺 作業九

指導老師 傅楸善

學生 蔡宇晴

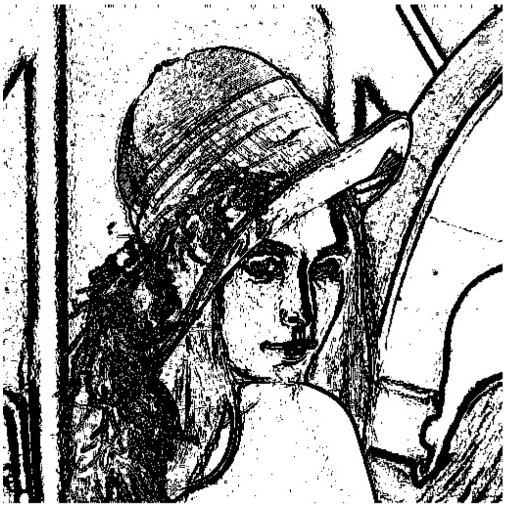
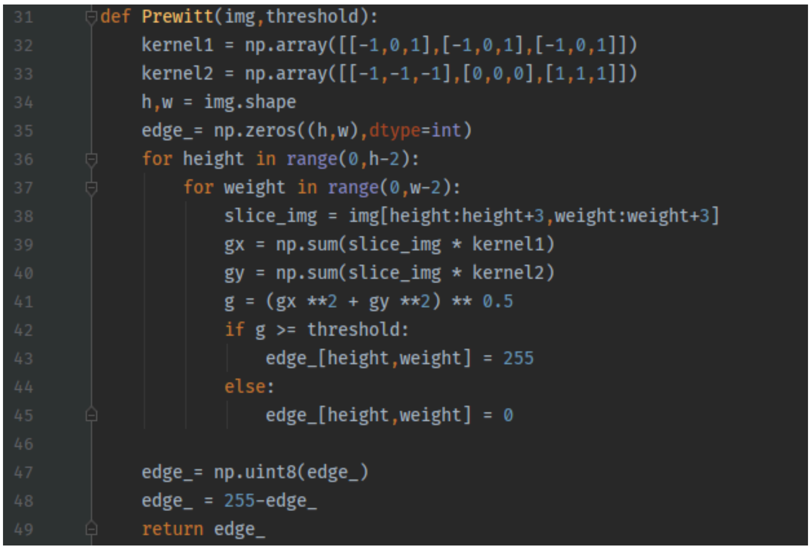
學號 R08945050

<table>
<tr><td></td><td>原圖</td><td>Robert</td></tr>
<tr><td>圖
片</td><td> </td><td></td></tr>
<tr><td>程
式
碼</td><td colspan="2">

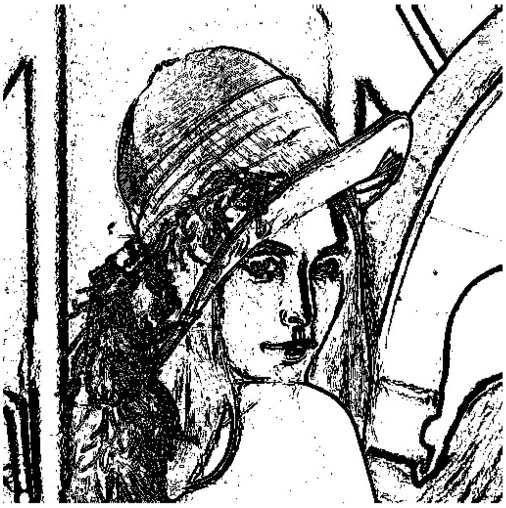
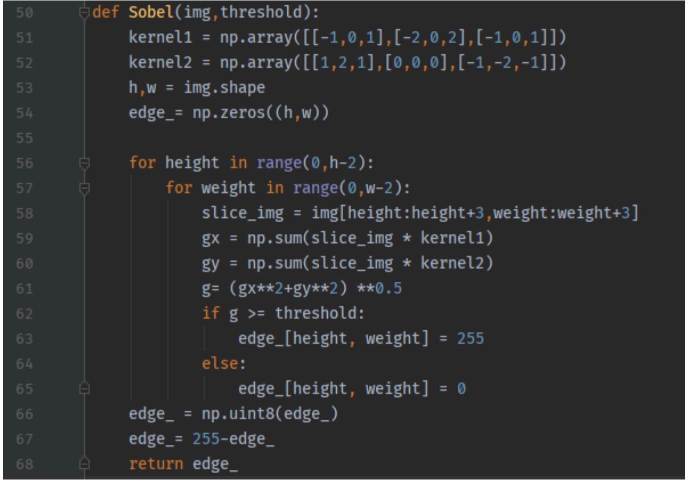


```python
def Robert(img,threshold):
    kernel1 = np.array([[1,0],[0,-1]])
    kernel2 = np.array([[0,1],[-1,0]])
    h,w = img.shape
    edge_= np.zeros((h,w),dtype=int)


    for height in range(0,h-1):
        for weight in range(0,w-1):
            slice_img = img[height:height+2,weight:weight+2]
            gx = np.sum(slice_img * kernel1)
            gy = np.sum(slice_img * kernel2)
            g = (gx **2 + gy **2) ** 0.5
            if g >= threshold:
                edge_[height,weight] = 255
            else:
                edge_[height,weight] = 0

    edge_= np.uint8(edge_)
    edge_ = 255-edge_
    return edge_
```

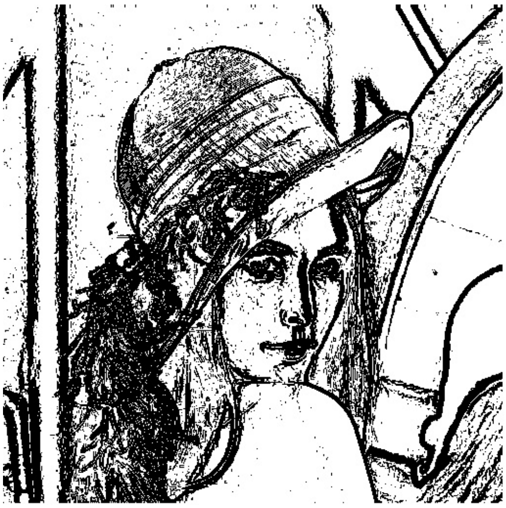
</td></tr>
<tr><td>描
述</td><td colspan="2">邊緣定位准，但是對噪聲敏感。適用於邊緣明顯且噪聲較少的圖像分割。影像的雜點看起來很多。
在邊緣線條上的粗度相較於其他 6 個檢測來說較細。</td></tr>
</table>

	原圖	Prewitt
圖片		
程式碼		
描述	對噪聲有抑制作用，抑制噪聲的原理是通過像素平均，但是像素平均相當於對圖像的低通濾波。邊緣檢測結果較 robert 粗。	

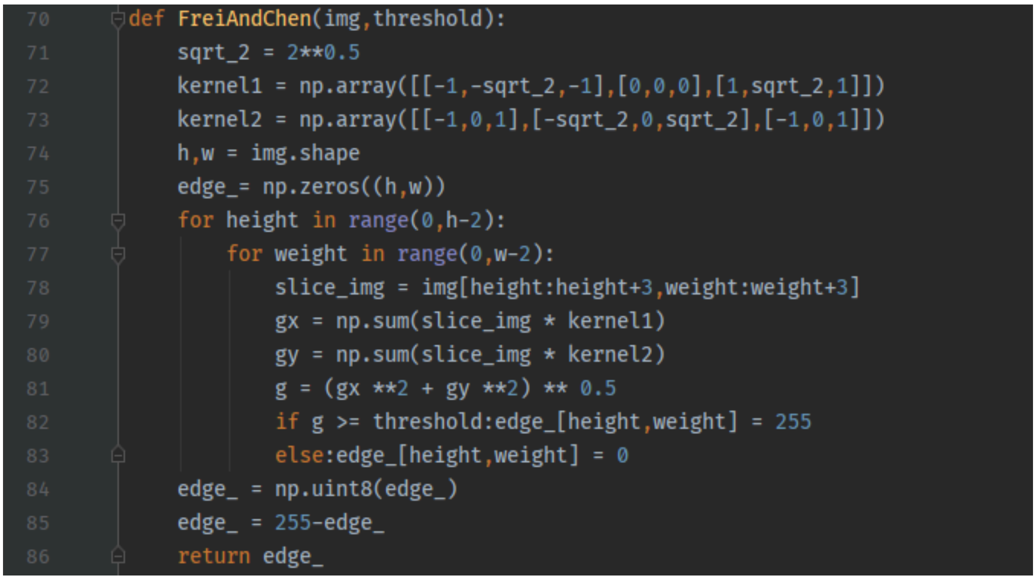
```python
def Prewitt(img,threshold):
    kernel1 = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
    kernel2 = np.array([[-1,-1,-1],[0,0,0],[1,1,1]])
    h,w = img.shape
    edge_= np.zeros((h,w),dtype=int)
    for height in range(0,h-2):
        for weight in range(0,w-2):
            slice_img = img[height:height+3,weight:weight+3]
            gx = np.sum(slice_img * kernel1)
            gy = np.sum(slice_img * kernel2)
            g = (gx **2 + gy **2) ** 0.5
            if g >= threshold:
                edge_[height,weight] = 255
            else:
                edge_[height,weight] = 0

    edge_= np.uint8(edge_)
    edge_ = 255-edge_
    return edge_
```

	原圖	Sobel
圖 片		
程 式 碼		
描 述	Sobel 和 Prewitt 都是加權平均，但是 sobel 對於距離不同的像素給予不同的權重。結果圖差不多，但噪聲少了許多。	

```python
def Sobel(img,threshold):
    kernel1 = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    kernel2 = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
    h,w = img.shape
    edge_= np.zeros((h,w))

    for height in range(0,h-2):
        for weight in range(0,w-2):
            slice_img = img[height:height+3,weight:weight+3]
            gx = np.sum(slice_img * kernel1)
            gy = np.sum(slice_img * kernel2)
            g= (gx**2+gy**2) **0.5
            if g >= threshold:
                edge_[height, weight] = 255
            else:
                edge_[height, weight] = 0
    edge_ = np.uint8(edge_)
    edge_= 255-edge_
    return edge_
```

	原圖	FreiAndChen
圖 片		

程
式
碼

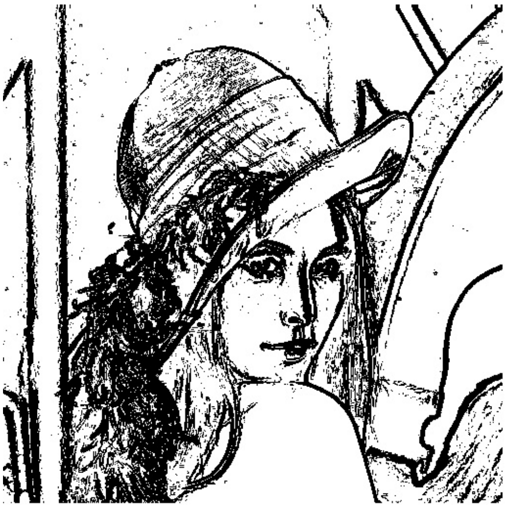


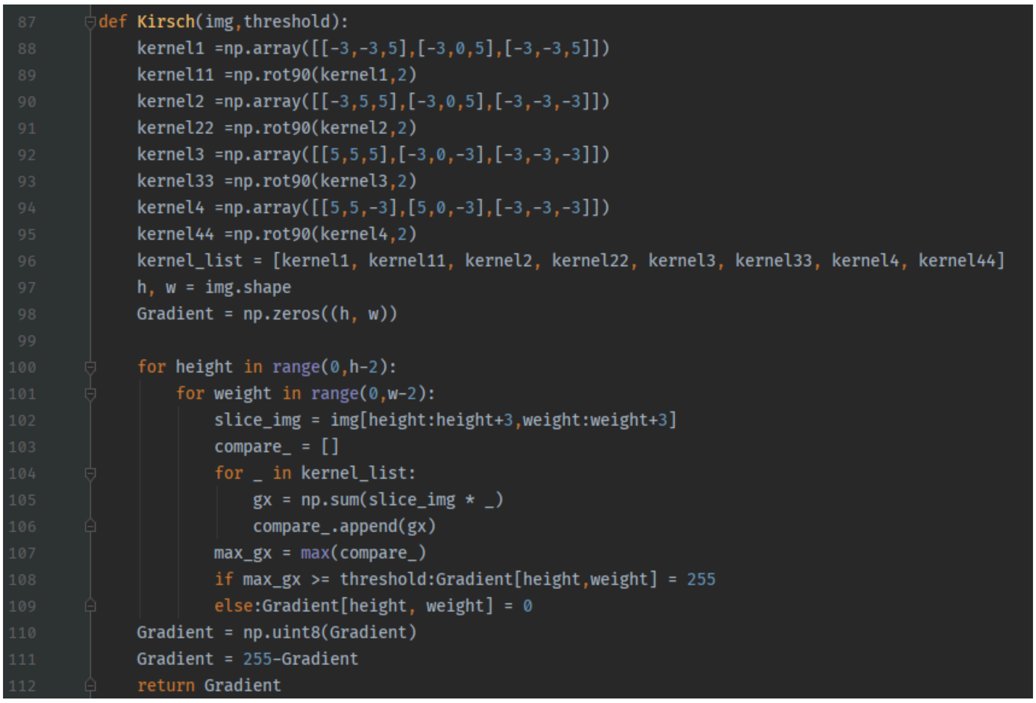
```python
def FreiAndChen(img,threshold):
    sqrt_2 = 2**0.5
    kernel1 = np.array([[-1,-sqrt_2,-1],[0,0,0],[1,sqrt_2,1]])
    kernel2 = np.array([[-1,0,1],[-sqrt_2,0,sqrt_2],[-1,0,1]])
    h,w = img.shape
    edge_= np.zeros((h,w))
    for height in range(0,h-2):
        for weight in range(0,w-2):
            slice_img = img[height:height+3,weight:weight+3]
            gx = np.sum(slice_img * kernel1)
            gy = np.sum(slice_img * kernel2)
            g = (gx **2 + gy **2) ** 0.5
            if g >= threshold:edge_[height,weight] = 255
            else:edge_[height,weight] = 0
    edge_ = np.uint8(edge_)
    edge_ = 255-edge_
    return edge_
```

描
述

Frei-Chen 比 Sobel 看起來更好，因為它對噪聲不太敏感。

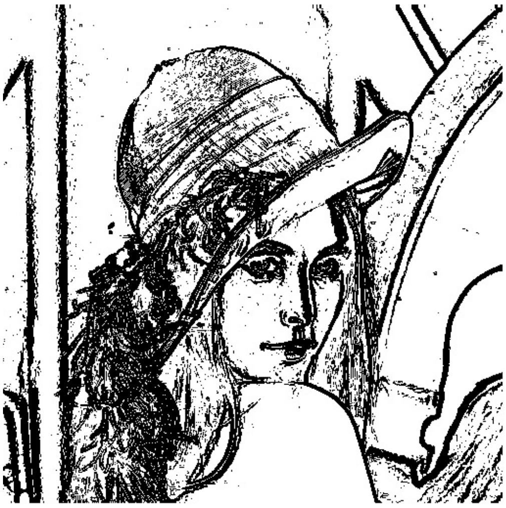
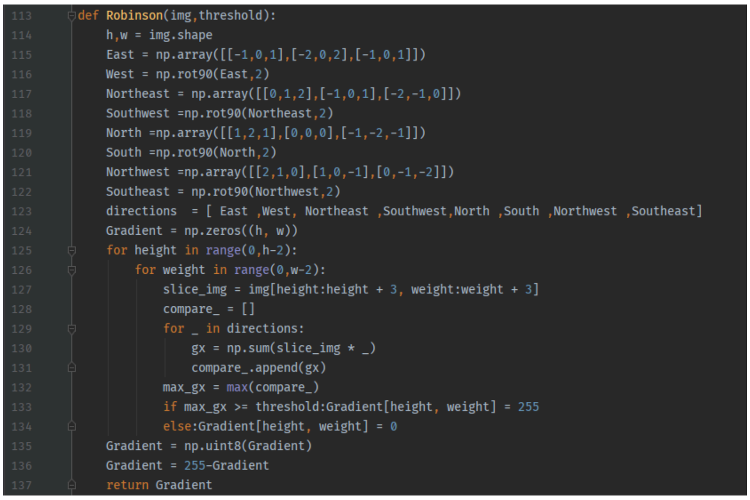
整體的結果差不多，但是噪聲少了一些。

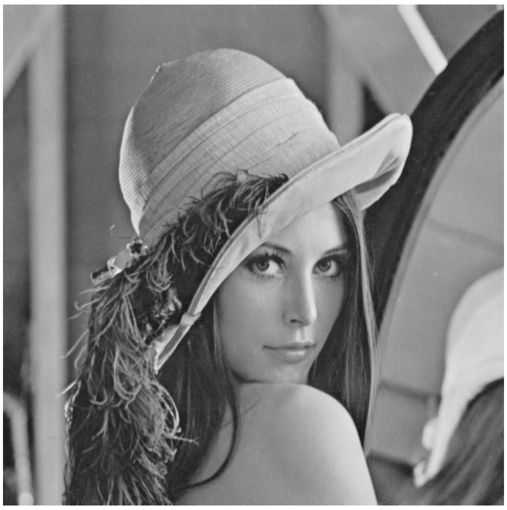
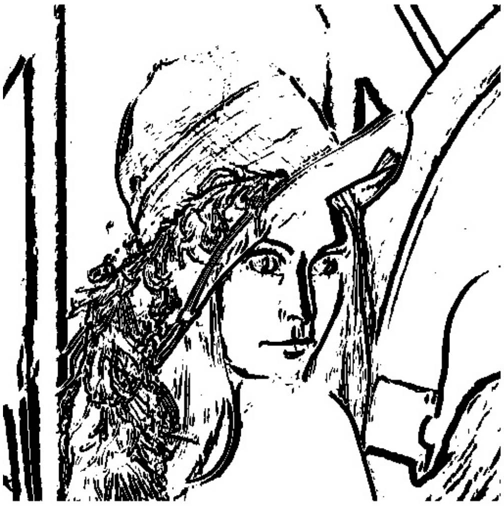
<table>
<tr><th></th><th>原圖</th><th>Kirsch</th></tr>
<tr><td>圖片</td><td> </td><td></td></tr>
<tr><td>程式碼</td><td colspan="2">



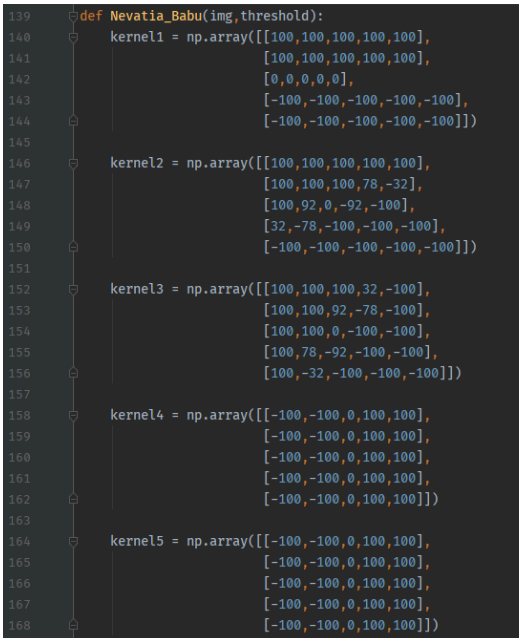
```python
def Kirsch(img,threshold):
    kernel1 =np.array([[-3,-3,5],[-3,0,5],[-3,-3,5]])
    kernel11 =np.rot90(kernel1,2)
    kernel2 =np.array([[-3,5,5],[-3,0,5],[-3,-3,-3]])
    kernel22 =np.rot90(kernel2,2)
    kernel3 =np.array([[5,5,5],[-3,0,-3],[-3,-3,-3]])
    kernel33 =np.rot90(kernel3,2)
    kernel4 =np.array([[5,5,-3],[5,0,-3],[-3,-3,-3]])
    kernel44 =np.rot90(kernel4,2)
    kernel_list = [kernel1, kernel11, kernel2, kernel22, kernel3, kernel33, kernel4, kernel44]
    h, w = img.shape
    Gradient = np.zeros((h, w))

    for height in range(0,h-2):
        for weight in range(0,w-2):
            slice_img = img[height:height+3,weight:weight+3]
            compare_ = []
            for _ in kernel_list:
                gx = np.sum(slice_img * _)
                compare_.append(gx)
            max_gx = max(compare_)
            if max_gx >= threshold:Gradient[height,weight] = 255
            else:Gradient[height, weight] = 0
    Gradient = np.uint8(Gradient)
    Gradient = 255-Gradient
    return Gradient
```

</td></tr>
<tr><td>描述</td><td colspan="2">相較於 FreiAndChen，噪點減少了許多。因為考慮的方向比
FreiAndChen 多了許多方向的檢測。</td></tr>
</table>

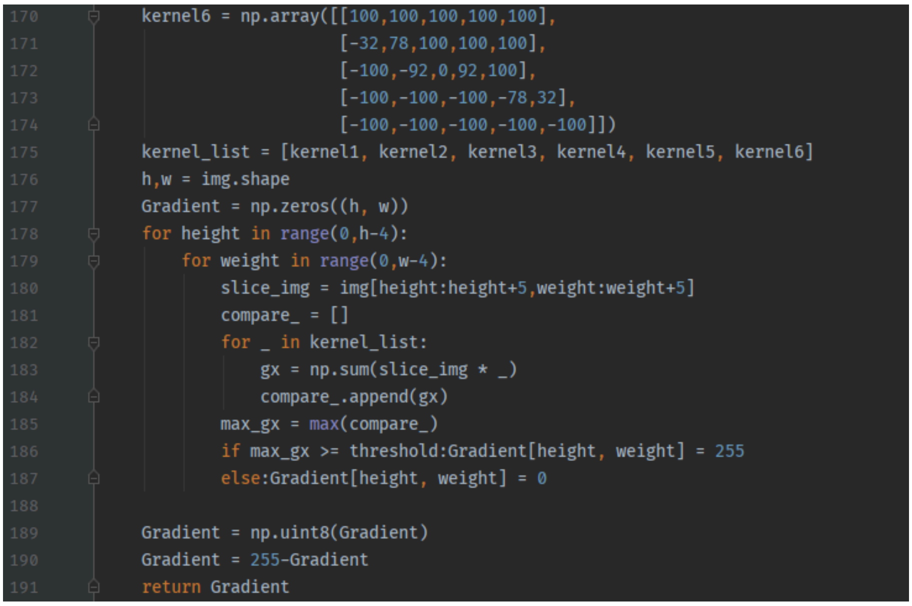
<table>
<tr><th></th><th>原圖</th><th>Robinson</th></tr>
<tr><td>圖片</td><td> </td><td></td></tr>
<tr><td>程式碼</td><td colspan="2">
<pre>
def Robinson(img,threshold):
    h,w = img.shape
    East = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    West = np.rot90(East,2)
    Northeast = np.array([[0,1,2],[-1,0,1],[-2,-1,0]])
    Southwest =np.rot90(Northeast,2)
    North =np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
    South =np.rot90(North,2)
    Northwest =np.array([[2,1,0],[1,0,-1],[0,-1,-2]])
    Southeast = np.rot90(Northwest,2)
    directions = [East ,West, Northeast ,Southwest,North ,South ,Northwest ,Southeast]
    Gradient = np.zeros((h, w))
    for height in range(0,h-2):
        for weight in range(0,w-2):
            slice_img = img[height:height + 3, weight:weight + 3]
            compare_ = []
            for _ in directions:
                gx = np.sum(slice_img * _)
                compare_.append(gx)
            max_gx = max(compare_)
            if max_gx >= threshold:Gradient[height, weight] = 255
            else:Gradient[height, weight] = 0
    Gradient = np.uint8(Gradient)
    Gradient = 255-Gradient
    return Gradient
</pre></td></tr>
<tr><td>描述</td><td colspan="2">和 Kirsch 結果類似，但產生的噪點又更少了，在髮尾部分的白色斑點比 Kirsch 還多。</td></tr>
</table>

	原圖	Nevatia_Bubu
圖 片		

程式碼



```python
def Nevatia_Babu(img,threshold):
    kernel1 = np.array([[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]])

    kernel2 = np.array([[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]])

    kernel3 = np.array([[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]])

    kernel4 = np.array([[-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]])

    kernel5 = np.array([[-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]])
```



```python
    kernel6 = np.array([[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]])
    kernel_list = [kernel1, kernel2, kernel3, kernel4, kernel5, kernel6]
    h,w = img.shape
    Gradient = np.zeros((h, w))
    for height in range(0,h-4):
        for weight in range(0,w-4):
            slice_img = img[height:height+5,weight:weight+5]
            compare_ = []
            for _ in kernel_list:
                gx = np.sum(slice_img * _)
                compare_.append(gx)
            max_gx = max(compare_)
            if max_gx >= threshold:Gradient[height, weight] = 255
            else:Gradient[height, weight] = 0

    Gradient = np.uint8(Gradient)
    Gradient = 255-Gradient
    return Gradient
```

描述

Nevatia_Bubu 的噪聲是所有邊緣偵測最少的，但在帽子邊緣的部分檢測的不太好，已經被閥值判定成非邊緣了。