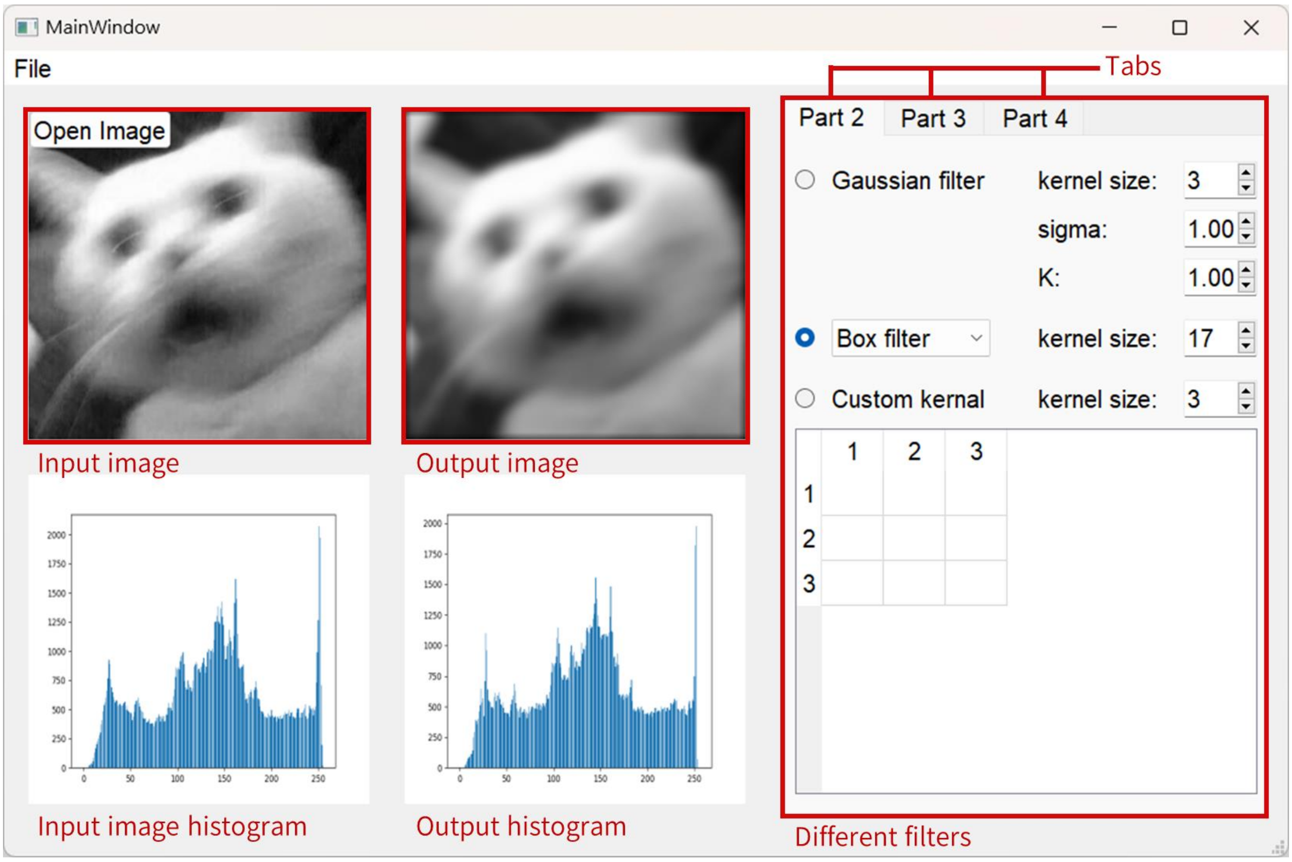
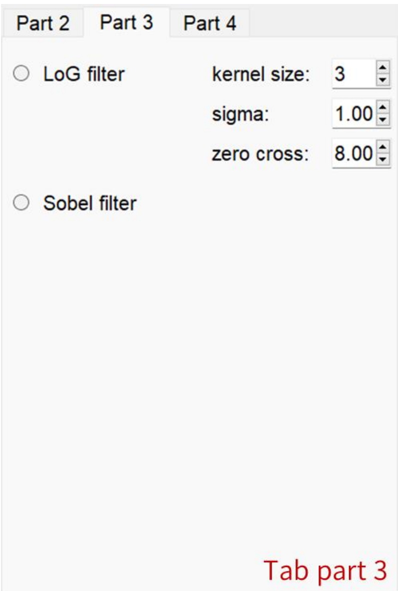
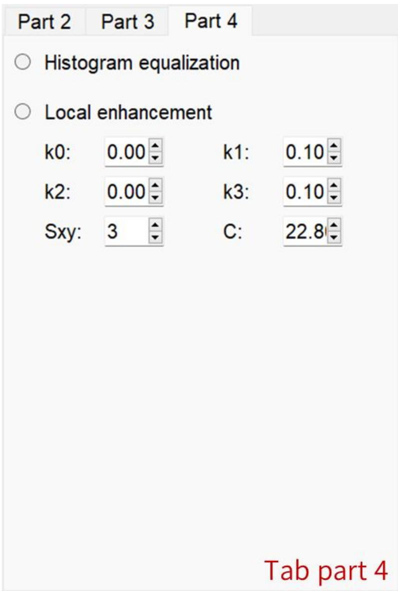
Principles and Applications of Digital Image Processing

Homework 3

GUI







圖一、介面使用說明。

Part 1: (25%)

3.22

(a). yes

(b). $W_{2\times3} = \begin{bmatrix} 1 & 3 & 1 \\ 2 & 6 & 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}_{2\times1} \begin{bmatrix} 1 & 3 & 1 \end{bmatrix}_{1\times3}$

3.28

$G(x,y) = k e^{-\frac{x^2+y^2}{2\sigma^2}}$, 1.5 2 4 ; 3×3 5×5 7×7

(a) yes, Gaussian kernels are separable.

1-D Gaussian functions.

f	g	$f \times g$	$f \star g$
m_f	m_g	$m_{f\times g} = \frac{m_f \sigma_g^2 + m_g \sigma_f^2}{\sigma_f^2 + \sigma_g^2}$	$m_{f\star g} = m_f + m_g$
σ_f	σ_g	$\sigma_{f\times g} = \sqrt{\frac{\sigma_f^2 \sigma_g^2}{\sigma_f^2 + \sigma_g^2}}$	$\sigma_{f\star g} = \sqrt{\sigma_f^2 + \sigma_g^2}$

(b).

$\sigma' = \sqrt{1.5^2 + 2^2 + 4^2} = 4.717$

(c)

$3 \times 5 \times 7 = 105$

105×105

3.44

a) If rank = 1, seperable

from numpy.linalg import matrix_rank as rank

① $\text{rank}\left(\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}\right) = 2 \Rightarrow$ not seperable

② $\text{rank}\left(\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}\right) = 2 \Rightarrow$ not seperable

b).

$\text{rank}\left(\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 2 \Rightarrow$ not seperable

$\text{rank}\left(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\right) = 2 \Rightarrow$ not seperable

c) $\text{rank}\left(\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}\right) = 1$

$w = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$

$\text{rank}\left(\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}\right) = 1$

$w = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$

4.3

a). $\delta(t) \star \delta(t-t_0) = \delta(t-t_0)$

b). $\delta(t-t_0) \star \delta(t+t_0) = \delta(t)$

4.7

$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ $g(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$

$f(x-x_0, y-y_0) \Leftrightarrow F(u,v)\, e^{-j2\pi(x_0 u/M + y_0 v/N)}$

$\Rightarrow G(u,v) = F(u,v)\left[e^{-j2\pi\frac{u}{M}} + e^{-j2\pi\frac{-u}{M}} + e^{-j2\pi\frac{v}{N}} + e^{-j2\pi\frac{-v}{N}} - 4\right]$

where $e^{-j\theta} + e^{j\theta} = 2\cos\theta$

$G(u,v) = F(u,v)\left[2\cos\left(\frac{2\pi u}{M}\right) + 2\cos\left(\frac{2\pi v}{N}\right) - 4\right]$

Part 2: (25%)

1. Discuss the effect of mask size on the processed images and the computation time.

圖二、經不同 kernel size 的 Box filter 或 Gaussian blur 之影像。

看圖二可發現越大的 kernel size 所需的計算時間越多，且影像周圍會有黑邊。Box filter 的大小越大影像越模糊。Gaussian 變化與 kernel size 並不大。若使用者想使用自定義的 kernel，可透過程式內的 table 達成（圖一）。Convolution 的實作如下：

```
def conv_2d(img_arr, kernel):
    time_start = time.time()
    kernel = np.flipud(kernel)
    kernel = np.fliplr(kernel)
    img_padding = padding(img_arr, kernel)
    img_conv = np.zeros((img_arr.shape[0], img_arr.shape[1]))
    for i in range(img_arr.shape[0]):
        for j in range(img_arr.shape[1]):
            window = kernel * img_padding[i:i+kernel.shape[0],
                                          j:j+kernel.shape[1]]
            img_conv[i, j] = window.sum()
    img_conv[img_conv > 255] = 255
    img_conv[img_conv < 0] = 0
    print(f'Convolution takes {time.time()-time_start} seconds.')
    return img_conv
```

Part 3: (25%)

1. The Marr-Hildreth Edge Detector 透過實作課本的 10-29 式達成，程式如下：

$$\nabla^2 G(x,y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

```python
def log_filter(img_arr, kernel_size, sigma, threshold, img_path):
    c = int((kernel_size-1)/2)
    distance = np.fromfunction(lambda i, j:
                               pow((i-c), 2)+pow((j-c), 2), (kernel_size, kernel_size), dtype=float)
    kernel = np.zeros((kernel_size, kernel_size))
    for i in range(kernel_size):
        for j in range(kernel_size):
            kernel[i][j] = ((distance[i][j]-2*pow(sigma, 2)) /
                            pow(sigma, 4))*np.exp(-distance[i][j]/(2*pow(sigma, 2)))
    kernel /= kernel.sum()
    img_log = conv_2d(img_arr, kernel)
    img_log_cross = zero_crossing(img_log, threshold)
    save_path = save_output(img_log_cross, img_path, 'LoG')
    return img_log_cross, save_path
```

```python
def zero_crossing(img, threshold):
    img_crossing = img.copy()
    for i in range(1, (img.shape[0]-1)):
        for j in range(1, (img.shape[1]-1)):
            if img[i, j]*img[i+1, j] > 0 and abs(img[i, j]-img[i+1, j]) > threshold:
                img_crossing[i, j] = 255
            elif img[i, j]*img[i-1, j] > 0 and abs(img[i, j]-img[i-1, j]) > threshold:
                img_crossing[i, j] = 255
            elif img[i, j]*img[i, j+1] > 0 and abs(img[i, j]-img[i, j+1]) > threshold:
                img_crossing[i, j] = 255
            elif img[i, j]*img[i, j-1] > 0 and abs(img[i, j]-img[i, j-1]) > threshold:
                img_crossing[i, j] = 255
            else:
                img_crossing[i, j] = 0
    return img_crossing
```

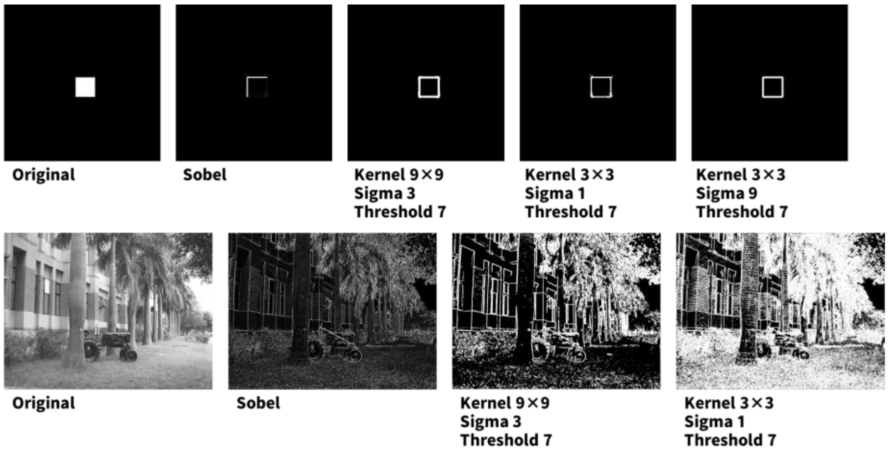
Sobel 實作如下：

```python
def sobel_filter(img_arr, img_path):
    hx = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    hy = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    img_gx = conv_2d(img_arr, hx)
    img_gy = conv_2d(img_arr, hy)
    img_sobel = np.zeros((img_arr.shape[0], img_arr.shape[1]))
    img_sobel = np.sqrt((np.power(img_gx, 2))+(np.power(img_gy, 2)))
    save_path = save_output(img_sobel, img_path, 'sobel')
    return img_sobel, save_path
```

2. Compare the Marr-Hildreth edge results with those processed with the Sobel operator.

Original | Sobel | Kernel 9×9 Sigma 3 Threshold 7 | Kernel 3×3 Sigma 1 Threshold 7 | Kernel 3×3 Sigma 9 Threshold 7

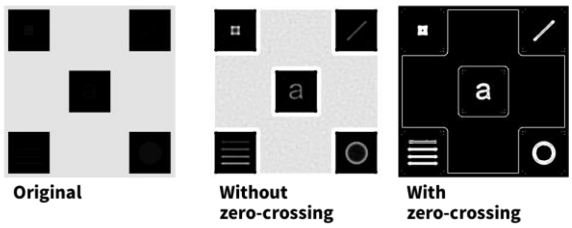
Original | Sobel | Kernel 9×9 Sigma 3 Threshold 7 | Kernel 3×3 Sigma 1 Threshold 7



圖四、用不同邊緣檢測方法之影像。

用 Sobel 可以取得相當不錯的成果，並且不用人工設定參數。然而若要提取更具代表性的輪廓，用 LoG 可以取得最好的效果。

3. Discuss the effect of zero-crossing threshold on the Marr-Hildreth edge detection method.

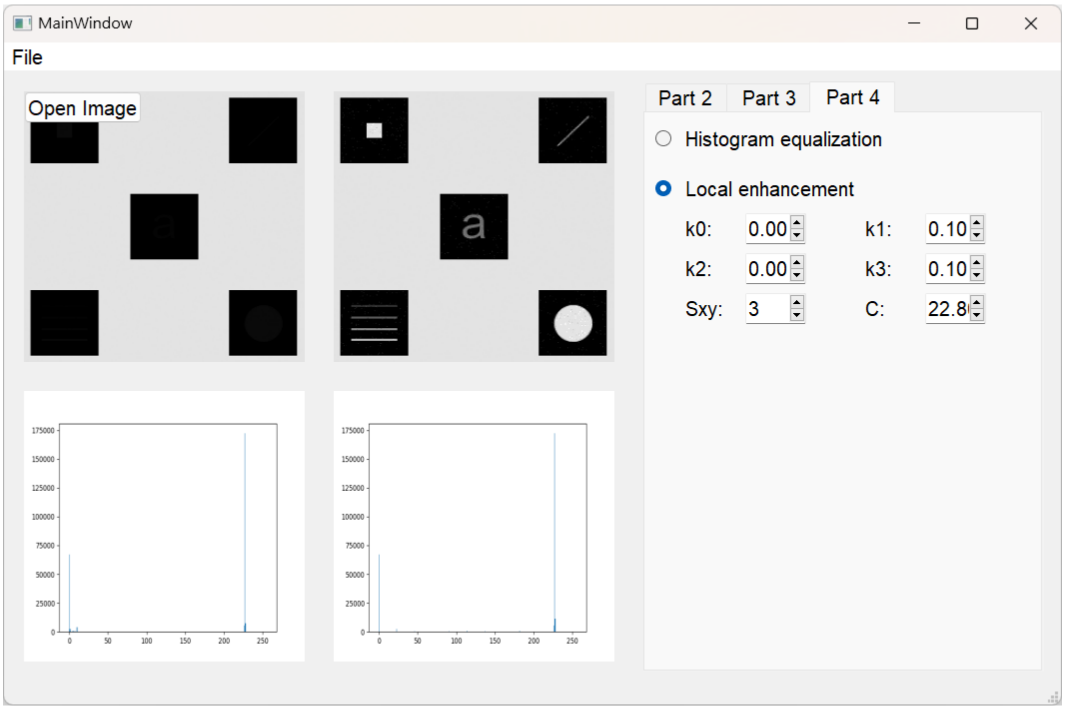


圖五、Marr-Hildreth with/without zero-crossing.

沒有經過 zero-crossing 的影像可以找出影像中的輪廓，但仍保留原始影像中不分顏色資訊。用 zero-crossing 取得乾淨的影像輪廓。

Part 4: (25%)

1. Reproduce 3.27b

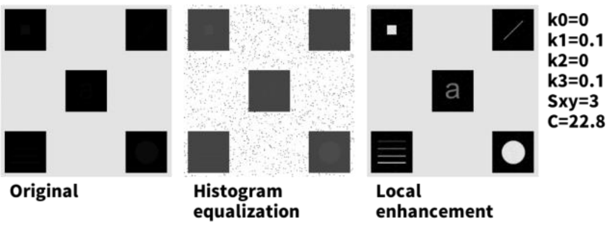


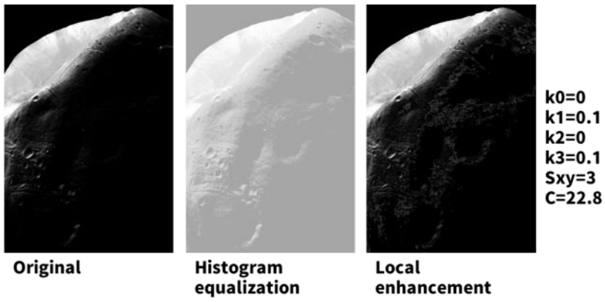
圖六、還原之 3.27b 影像。

實作過程如下：

```python
def local_enhancement(img_arr, kernel_size, k0, k1, k2, k3, cc, img_path):
    time_start = time.time()
    mean_g = img_arr.mean()
    var_g = pow(img_arr.var(), .5)
    kernel = np.zeros((kernel_size, kernel_size))
    img_padding = padding(img_arr, kernel)
    img_conv = np.zeros((img_arr.shape[0], img_arr.shape[1]))
    for i in range(img_arr.shape[0]):
        for j in range(img_arr.shape[1]):
            window = img_padding[i:i+kernel.shape[0], j:j+kernel.shape[1]]
            mean_kernel = window.mean()
            var_kernel = pow(window.var(), .5)
            if k0*mean_g <= mean_kernel and mean_kernel <= k1*mean_g and k2*var_g <= var_kernel and var_kernel <= k3*var_g:
                img_conv[i, j] = cc*img_arr[i, j]
            else:
                img_conv[i, j] = img_arr[i, j]
    img_conv[img_conv > 255] = 255
    img_conv[img_conv < 0] = 0
    print(f'Convolution takes {time.time()-time_start} seconds.')
    save_path = save_output(img_conv, img_path, 'Local')
    return img_conv, save_path
```

2. Compare local enhancement with histogram equalization.

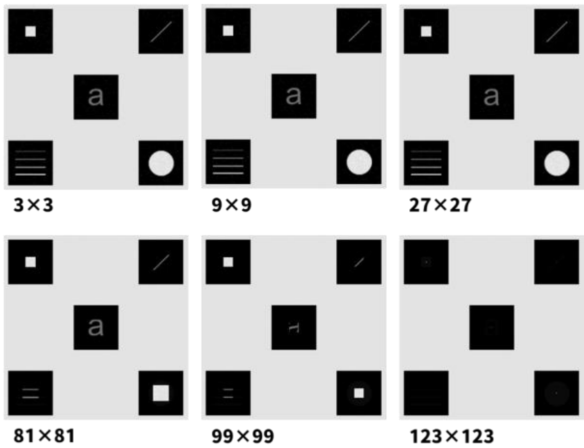




圖七、經 local enhancement 和 histogram equalization 後之影像。

可發現用 local enhancement 才可以將影像中的細節顯現出來。並不是所有影像皆適合用 histogram localization。

3. Effect of region size.



圖八、用不同 region size 做 local enhancement 之影像。

如圖八所示，隨 region size 越大，影像中的部分特徵會消失。若要顯示較細緻的細節，應該 region size 小的 local enhancement。