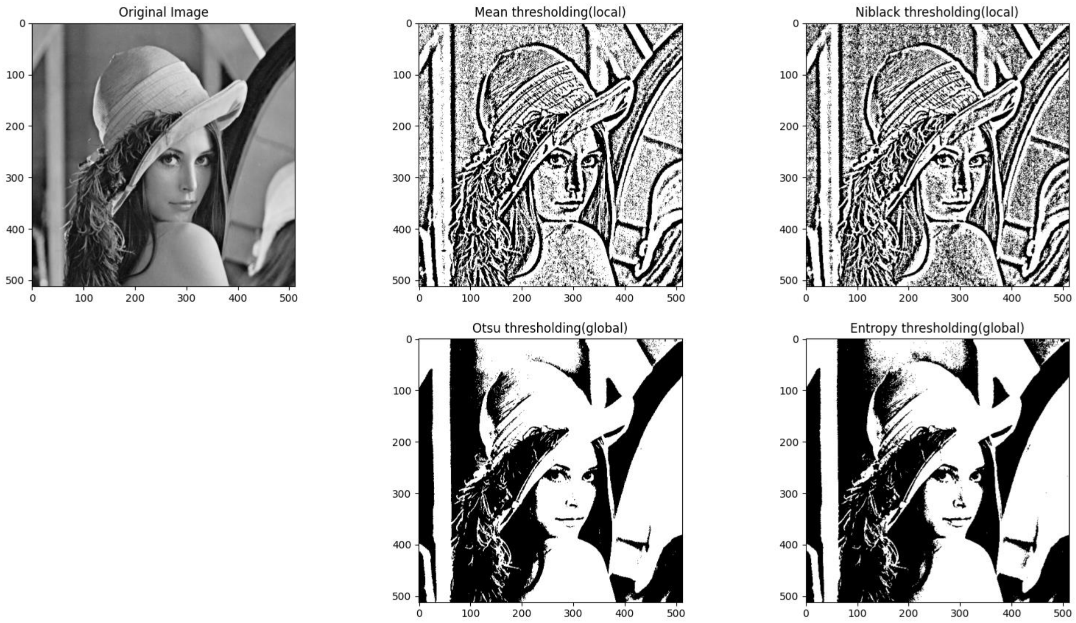
計算機視覺-HW1

S1154005-李育林

Thresholding:

成果:



使用方式:

- Mean-threshold method(local): 在 image 的每塊 pixel 都以其為中心，重新透過指定大小的 block 去取整塊的平均-定值 C 後當作 threshold 去做處理圖片。

```
def mean_thresholding(image, block_size=3, c=2):
    new_image = np.zeros(shape=(image.shape[0], image.shape[1]), dtype=np.uint8)
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            # calculate neighbor block
            x_min = max(x-block_size//2,0)
            x_max = min(x+block_size//2+1, image.shape[0]-1)
            y_min = max(y-block_size//2, 0)
            y_max = min(y+block_size//2+1, image.shape[1]-1)

            #calculate neighbor block mean as thershold for (x,y)
            local_mean_threshold = np.mean(image[x_min:x_max, y_min:y_max])

            #thresholding
            if image[x,y] > local_mean_threshold-c:
                new_image[x,y] = 255
            else:
                new_image[x,y] = 0
    return new_image
```

- Niblack-threshold method(local): 在 image 的每塊 pixel 都以其為中心，重新透過指定大小的 block 去取整塊的平均值+標準差*定值 k 後當作 threshold 去做處理圖片。

```python
def niblack_thresholding(image, block_size=3, k=0.2):
    new_image = np.zeros(shape=(image.shape[0], image.shape[1]), dtype=np.uint8)
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            # calculate neighbor block
            x_min = max(x-block_size//2,0)
            x_max = min(x+block_size//2+1, image.shape[0]-1)
            y_min = max(y-block_size//2, 0)
            y_max = min(y+block_size//2+1, image.shape[1]-1)

            #calculate neighbor block standard_deviation as thershold for (x,y)
            local_mean = np.mean(image[x_min:x_max, y_min:y_max])
            local_standard_deviation = np.std(image[x_min:x_max, y_min:y_max].flatten(), ddof=0)
            local_threshold = local_mean + k*local_standard_deviation

            #thresholding
            if image[x,y] > local_threshold:
                new_image[x,y] = 255
            else:
                new_image[x,y] = 0
    return new_image
```

- Otsu-threshold method(global): 透過尋找灰度 1~255 間能使前景和背景兩類形成最大的組間方差的 k 當作 threshold

```python
def otsu_thresholding(image, histogram):
    new_image = image.copy()
    max_variance_between, max_threshold = -999, 1
    for k in range(1, 256):
        #計算P(k)->前景, 背景的機率總和分別為w1,w2
        w1 = np.sum(histogram[:k])
        w2 = np.sum(histogram[k:])

        if w1 == 0 or w2 == 0:
            continue
        #計算mean intensity
        mean1 = np.sum(np.arange(0, k)*histogram[:k])/w1
        mean2 = np.sum(np.arange(k, 256)*histogram[k:])/w2
        meanG = np.sum(np.arange(0, 256)*histogram[:])/(w1+w2)

        variance_between = w1*(mean1-meanG)**2+w2*(mean2-meanG)**2

        #找k=1~255間最大的組間變數
        threshold_lst = []
        if variance_between > max_variance_between:
            threshold_lst.clear()
            threshold_lst.append(k)
            max_variance_between = variance_between
            max_threshold = np.mean(threshold_lst)
        elif variance_between == max_variance_between:
            threshold_lst.append(k)
            max_threshold = np.mean(threshold_lst)

    #thresholding
    for x in range(new_image.shape[0]):
        for y in range(new_image.shape[1]):
            if image[x, y] > max_threshold:
                new_image[x, y] = 255
            else:
                new_image[x, y] = 0
    return new_image
```

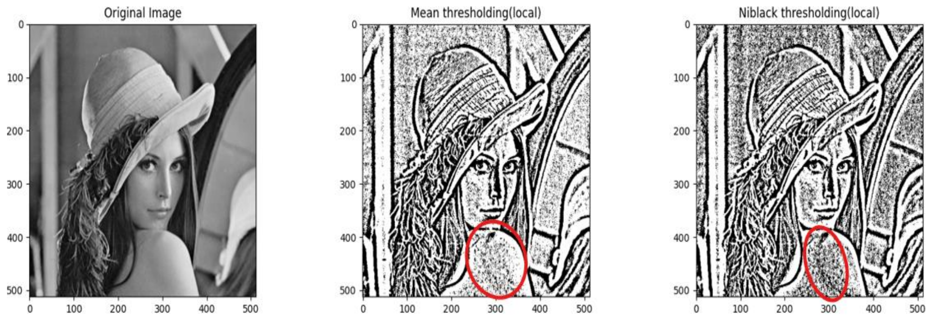
- Entropy-threshold method(global): 使用熵的概念去找最佳的 threshold，一樣是透過在 1~255 的灰度間找到前景和背景的 entropy 加起來最小的 k 當作 threshold。(熵越大代表不確定性越高)

```
def entropy_thresholding(image, histogram):
    H = np.zeros(256)
    for k in range(1,256):
        w1 = np.sum(histogram[:k])
        w2 = np.sum(histogram[k:])

        if w1==0 or w2==0:
            continue

        #計算前景,背景熵
        h1 = -np.sum( Entropy(histogram[:k]/w1))
        h2 = -np.sum( Entropy(histogram[k:]/w2))
        entropy = h1+h2
        H[k] = entropy

    #thresholding
    threshold = np.argmax(H)
    new_image = image.copy()
    return new_image > threshold
```

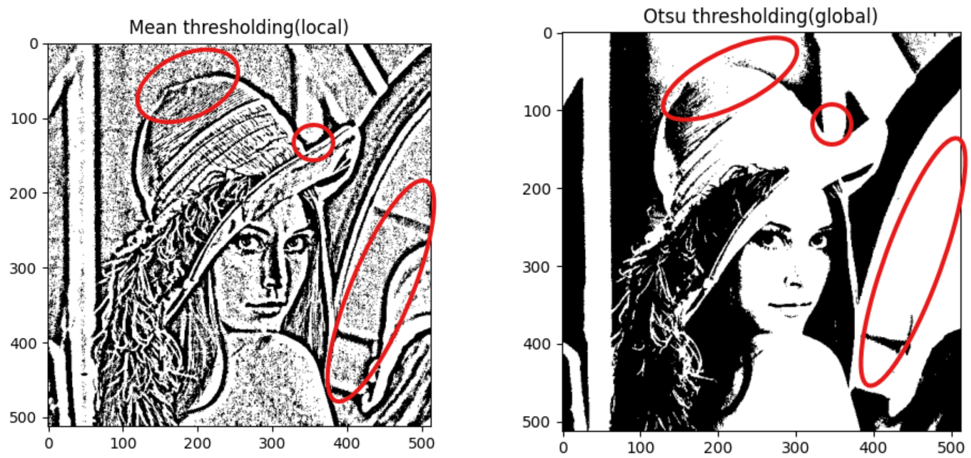
```
def Entropy(x):
    tmp = np.multiply(x, np.log(x+1e-5))
    tmp[np.isnan(tmp)] = 0
    return tmp
```

成果分析:

1. 兩種 local thresholding 的方式，niblack method 的灰階深淺的呈現差異較小，推測是因為平均+標準差代表大部分數值會落在此範圍，因此較能將和 kernel 平均較相近的區塊做較細的區分，而單純使用平均則可能因數值差異大導致處於中間值的數值被極端分到前景或背景。

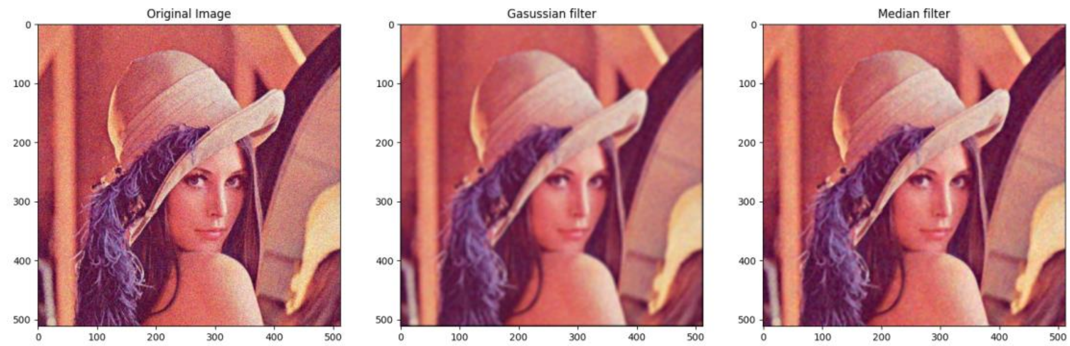


2. Local 和 Global 兩種主要不同方式的 thresholding 差別在於，local 的每一塊 pixel 依據 kernel 和計算方式套用的都是不同 threashold，而 global 方式則是將全圖套用同一個 threshold，優點是不會和 local 方式一樣有明顯噪點，但相較之下在一些部份的前背景或邊緣部分可能較不明顯。



Filtering:

成果:



使用方式:

- Gaussian-filter: 透過高斯函數形成的 kernel，將中心點的 pixel 和 kernel 內其他 pixel 的分布權重進行計算(平滑化)

```
def gaussian_filter(image, kernel_size=3, sigma=1):
    H, W, C = image.shape

    #padding image
    pad = kernel_size // 2
    padded_image = np.zeros((H+pad*2, W+pad*2, 3), dtype=np.float64)
    padded_image[pad: pad+H, pad:pad+W] = image.copy().astype(np.float64)

    #gen gaussian kernel
    kernel = np.zeros((kernel_size, kernel_size), dtype=np.float64)
    for x in range(-pad, -pad+kernel_size):
        for y in range(-pad, -pad+kernel_size):
            kernel[x+pad, y+pad] = np.exp( -(x**2+y**2)/(2*(sigma**2)))
    kernel /= (2*np.pi*sigma*sigma)
    kernel /= np.sum(kernel)

    #filter the image
    new_image = padded_image.copy()
    for x in range(H):
        for y in range(W):
            for c in range(C):
                new_image[pad+x, pad+y, c] = np.sum(kernel*padded_image[x: x+kernel_size, y: y+kernel_size, c])
                #pad+0~pad+511

    new_image = new_image[pad: pad+H, pad: pad+W].astype(np.uint8)
    return new_image
```

● Median-filter: 將 kernel 中心點的 pixel 取代成整個 kernel 內所有 pixel 的中位數

```python
def median_filter(image, kernel_size=3):
    H, W, C = image.shape[0], image.shape[1], image.shape[2]
    pad = kernel_size//2
    padded_image = np.zeros((H+pad*2, W+pad*2, 3), dtype=np.float64)
    padded_image[pad: pad+H, pad:pad+W] = image.copy().astype(np.float64)
    new_image = image.copy()

    #filtering
    for x in range(H):
        for y in range(W):
            for c in range(C):
                kernel = padded_image[x: x+kernel_size, y:y+kernel_size, c]
                new_image[x, y, c] = np.median(kernel.reshape(-1))

    new_image = new_image[pad:pad+H, pad:pad+W].astype(np.uint8)
    return new_image
```

成果分析:

1. 兩個 filter 在 kernel 越大的情況下，處理過後出來的圖片都會越模糊，然而相對的去噪效果也越好。
2. 在處理 kernel 邊界問題時使用 pad zero 的方式，但是當 kernel size 很大時仍然會有些微的黑邊問題