HW3_影像還原 Adaptive Median Filter

一、 影像還原-適應性中間值濾波器

Adaptive Median Filter 是一種用於去除雜訊的圖像處理演算法，通過在圖像上應用中值濾波器來實現去除雜訊的效果，與中值濾波器不同的是 Adaptive Median Filter 能夠根據圖像的不同區域適應性地調整濾波的大小，從而更加有效地去除雜訊。

Adaptive Median Filter 的基本原理是，對於每個像素，它會先設置一個 window，並計算出該區域內所有像素的中間值、最小值、最大值以及目前像素值，並透過第二點之演算法進行運算，最後得到去除雜訊後的圖片。

圖 一 影像還原過程

	原圖	胡椒鹽雜訊	解雜訊後
黑色(0)	0	9985	19
白色(255)	0	9985	0

圖 二 雜訊總量

二、 演算法

符號如下：

$$Zmax = maximun\ gray\ level\ value\ in\ Sxy$$
$$Zmin\ = minimun\ gray\ level\ value\ in\ Sxy$$
$$Zmed = median\ of\ gray\ levels\ in\ Sxy$$
$$Zxy\ \ = gray\ level\ at\ coordinates(x, y)$$
$$Smax = \ maximum\ allowed\ size\ of\ Sxy$$

演算法如下：

$$Level\ A: \quad A1 = Zmed - Zmin$$
$$A2 = Zmed - Zmax$$
$$if\ A1 > 0\ AND\ A2 < 0\ , Go\ to\ level\ B$$
$$Else\ increase\ the\ window\ size$$
$$if\ window\ size \leq Smax\ repeat\ level\ A$$
$$Else\ output\ Zxy$$
$$Level\ B: \quad B1 = Zxy - Zmin$$
$$B2 = Zxy - Zmax$$
$$if\ B1 > 0\ AND\ B2 < 0, output\ Zxy$$
$$Else\ output\ Zmed$$

依照上方的演算法持續運算，就能夠有效地去除雜訊。

三、程式實作

首先，我們先透過 cv2.imread 讀取圖片(img)，因為 Adaptive Median Filter 只能在圖片為灰階時執行，因此我們透過 trans_gray 將 img 轉為灰階色彩，並透過 calculate_noise 函數記錄此時灰階圖片的 0(黑色)與 255(白色)，從程式的輸出結果來看，可以發現此時沒有像素等於 0 與 255。

```
img = cv2.imread('Lenna.jpg')
img_gray = trans_gray(img.copy())
org_white , org_black = calculate_noise(img_gray.copy())
print(f'原本的black = {org_black}\n原本的white = {org_white}')
```

圖 三　　讀取圖片、轉灰階及計算 0 與 255 的個數

```python
def trans_gray(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

圖 四　　轉灰階函數

```python
def calculate_noise(img):
    white , black = 0 , 0
    for i in range(img.shape[0]):
        for j in  range(img.shape[1]):
            if img[i,j]==0:
                black+=1
            if img[i,j]==255:
                white+=1
    return white , black
```

圖 五　　計算 0 與 255 的函數內容

```
原本的black = 0
原本的white = 0
```

圖 六　　運算結果

得到此時的 0 與 255 後，可以透過 salt_pepper_noise 函數將灰階圖片撒滿雜訊。

Salt_pepper_noise 首先會先讀取圖片的 row 與 col，之後建立白雜訊與黑雜訊的數量為 $row * col * 0.2 * 0.5$，確保白雜訊與黑雜訊的數量為總圖片的 20%且各半，之後進入 while 迴圈，當白雜訊或黑雜訊還有數量時，就會一直執行，進入後先使用 random.random()建立一個機率值 prob，prob 會藉於 0~1 之間，再使用 random.randint(0,row-1)及 random.randint(0,col-1)讓程式幫我們生成要撒雜訊的位置，分別給 noise_row 變數及 noise_col 變數，若 prob>0.5，就需要再確保黑雜訊還有數量(black_noise>0)，通過後需要再確保此時的位置還未被撒過雜訊(使用 if 判斷該位置是否等於 0 或 255，若相等，就跳脫此次的迴圈)，都通過後，即可將該點(noise_row,noise_col)設置為 0，並且將黑雜訊數量-1(black_noise -1)，直至黑雜訊與白雜訊的數量都為 0 後才

結束迴圈，白雜訊邏輯與上述方式相同，不再贅述，這個函數可以確保我們只會撒 20%的雜訊，並且黑雜訊與白雜訊各一半，也確保雜訊不會重複撒在同一個像素上。

```python
def salt_pepper_noise(img):
    row , col = img.shape
    white_noise = black_noise = int(row * col * 0.2 * 0.5) #20%的雜訊
    while(white_noise > 0 or black_noise > 0):
        prob = random.random()
        noise_row = random.randint(0,row-1)
        noise_col = random.randint(0,col-1)
        if prob >0.5:
            if black_noise>0:
                if img[noise_row,noise_col] == 0 or img[noise_row,noise_col]==255 : continue
                img[noise_row,noise_col] = 0
                black_noise -=1
        else:
            if white_noise>0:
                if img[noise_row,noise_col] == 255 or img[noise_row,noise_col] == 0: continue
                img[noise_row,noise_col] = 255
                white_noise -=1
    return img
```

圖 七　　salt_pepper_noise 函數內容

```
增加雜訊後的black = 9985
增加雜訊後的white = 9985
```

圖 八　　撒完雜訊後雜訊數量

圖 九　　撒胡椒鹽雜訊後的結果

撒完雜訊後，我們就能來解雜訊了，首先我們將撒完雜訊後的圖片傳進 Adaptive_Median_Filter 函數內，函數內我們先定義 Windwo_size 為 3，並且使用 np.zeros_like 生成一張與原圖一樣大小的圖片(output)，為了避免我們的 window 超出圖片邊界，因此我們再使用 np.pad 將圖片做 padding 的動作，之後再使用 for 迴圈將整張圖片跑過一輪，每一個像素點我們都傳入 Adaptive_Median 內進行計算。

```
def Adaptive_Median_Filter(img):
    window_size = 3
    output = np.zeros_like(img.copy())
    padded_image = np.pad(img.copy() , window_size//2 , mode = 'constant')
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            output[row,col] = Adaptive_Median(padded_image , window_size , row , col)
    return output
```

圖 十　Adaptive_Median_Filter 函數內容

Adaptive_Median 函數會接收到該圖片、window_size 以及 row、col，此時我們先設定 window_size 最大只能到 7，之後設置一個變數為 window，內容為原圖片的$row：row + window$, $col：col + window$，即能夠計算在當前像素值加上 window_size 後所有像素值的最小值、最大值、中值，但計算前我們需要先將剛剛設置的 window 使用 flatten()展平再透過 np.sort 排序，才能進行計算。

取得該點的像素值、window 內的最大值、最小值及中值後，即可進入我們的演算法內計算，我們先判斷是否$最小值 < 中間值 < 最大值$，再判斷是否$最小值 < 當前像素值 < 最大值$，若都成立，即可將當前值回傳，並填入我們

剛剛所建立的 output 內，若*最小值* < *中間值* < *最大值* 成立但 *最小值* < *當前像素值* < *最大值* 不成立，則回傳中間值，若一開始的 *最小值* < *中間值* < *最大值* 就不成立了的話，就會將 window_size 增加 2，再判斷 window_size 是否小於我們初始建立的 7，是的話我們就能透過遞迴的方式將新的 window_size 傳給 Adaptive_Median，若超過 7 的話，就直接回傳中間值，至此完成我們的演算法，也一點一點地將值填入 output 中，當所有像素值跑完也等同於完成解雜訊了。

```
def Adaptive_Median(img , window_size , row , col):
    Smax = 7
    window = img[row:row+window_size , col:col+window_size]
    sorted_pixel = np.sort(window.flatten())
    zxy , zmin , zmed , zmax = img[row , col] , sorted_pixel[0] , sorted_pixel[len(sorted_pixel)//2] , sorted_pixel[-1]
    if zmin < zmed < zmax:
        if zmin < zxy < zmax:
            return zxy
        else:
            return zmed
    else:
        window_size += 2
        if window_size <= Smax :
            return Adaptive_Median(img, window_size, row, col)
        else:
            return zmed
```

圖 十一　Adaptive_Median 函數

```
影像還原後的black = 18
影像還原後的white = 0
```

圖 十二　解雜訊後的雜訊數量

圖 十三　解雜訊後的結果

之後我們嘗試使用 AI 的方式替我們解決胡椒鹽雜訊，我將撒完胡椒鹽雜訊後的結果上傳至[AI.IMAGE ENLARGER](https://imglarger.com/Denoiser)，此網站標榜 AI Denoiser,Clean Your Photo，但回傳的結果如圖十四，看起來差強人意了點。

圖 十四　AI 解雜訊

完整程式碼如下：

```python
import cv2
import numpy as np
import random
def trans_gray(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
def salt_pepper_noise(img):
    row , col = img.shape
    white_noise = black_noise = int(row * col * 0.2 * 0.5) #20%的雜訊
    while(white_noise > 0 or black_noise > 0):
        prob = random.random()
        noise_row = random.randint(0,row-1)
        noise_col = random.randint(0,col-1)
        if prob >0.5:
            if black_noise>0:
                if img[noise_row,noise_col] == 0 or img[noise_row,noise_col]==255 : continue
                img[noise_row,noise_col] = 0
                black_noise -=1
        else:
            if white_noise>0:
                if img[noise_row,noise_col] == 255 or img[noise_row,noise_col] == 0: continue
                img[noise_row,noise_col] = 255
                white_noise -=1
    return img

def Adaptive_Median(img , window_size , row , col):
    Smax = 7
    window = img[row:row+window_size , col:col+window_size]
    sorted_pixel = np.sort(window.flatten())
    zxy , zmin , zmed , zmax = img[row , col] , sorted_pixel[0] , sorted_pixel[len(sorted_pixel)//2] , sorted_pixel[-1]
    if zmin < zmed < zmax:
        if zmin < zxy < zmax:
            return zxy
        else:
            return zmed
    else:
        window_size += 2
        if window_size <= Smax :
            return Adaptive_Median(img, window_size, row, col)
        else:
            return zmed
def Adaptive_Median_Filter(img):
    window_size = 3
    output = np.zeros_like(img.copy())
    padded_image = np.pad(img.copy() , window_size//2 , mode = 'constant')
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            output[row,col] = Adaptive_Median(padded_image , window_size , row , col)
    return output

def calculate_noise(img):
    white , black = 0 , 0
    for i in range(img.shape[0]):
        for j in  range(img.shape[1]):
            if img[i,j]==0:
                black+=1
            if img[i,j]==255:
                white+=1
    return white , black

img = cv2.imread('Lenna.jpg')
img_gray = trans_gray(img.copy())
org_white , org_black = calculate_noise(img_gray.copy())
print(f'原本的black = {org_black}\n原本的white = {org_white}')
img_noise = salt_pepper_noise(img_gray.copy())
noise_white , noise_black = calculate_noise(img_noise.copy())
print(f'增加雜訊後的black = {noise_black} \n增加雜訊後的white = {noise_white}')
img_adaptive = Adaptive_Median_Filter(img_noise.copy())
denoise_white , denoise_black = calculate_noise(img_adaptive.copy())
print(f'影像還原後的black = {denoise_black}\n影像還原後的white = {denoise_white}')
cv2.imshow('gray',img_gray)
cv2.imshow('salt_img',img_noise)
cv2.imshow('img_adaptive',img_adaptive)
cv2.waitKey(0)
```

圖 十五　程式碼

四、心得

此次作業其實不難，只需要照著老師上課所說的演算法實際順過一輪，就可以寫出來，只是前面需要先思考如何撒雜訊才能有效地只撒 20%以及黑白雜訊要平均，幾經思考過後才寫出此版本，雖然使用 while 迴圈較複雜，但總體來說能公平有效地撒雜訊才是我們目前所需要的。

老師上課有提到希望我們將自己解的雜訊與 AI 解的雜訊做比較，在我所找到的 AI 解雜訊網站可能較不適合解胡椒鹽雜訊，所以才會有此結果，但我相信在其他雜訊的方面 AI 一定會做的更好。