Homework 1

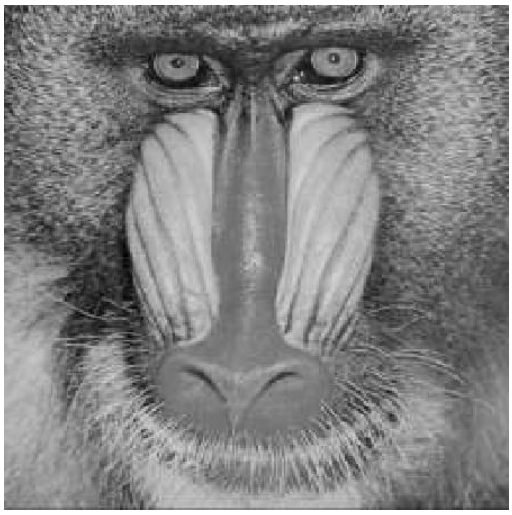
Digital Halftoning

Student： 台科電子碩一 M11302149 趙孟哲

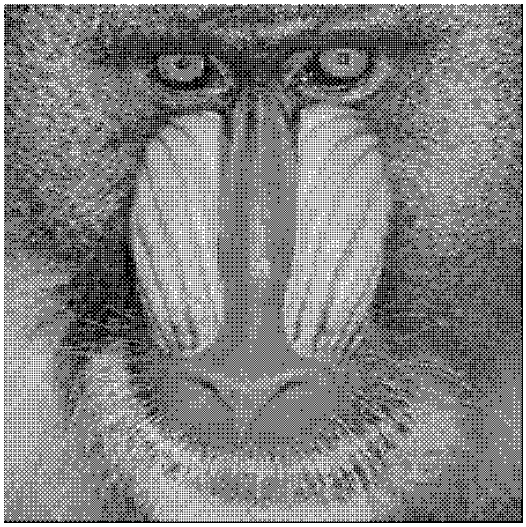
程式碼： github

1.Result

Original:

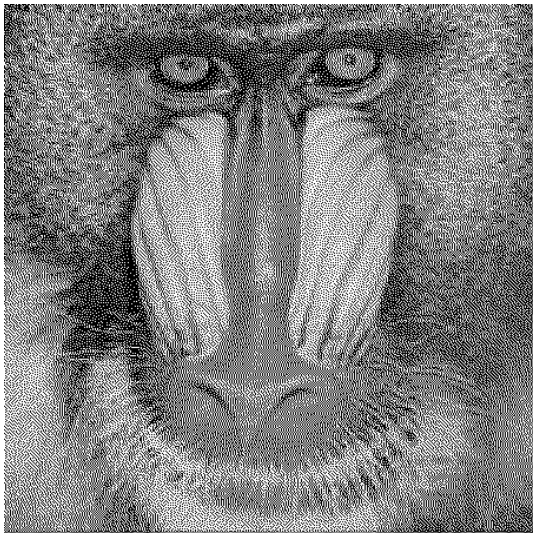


Ordered Dithering



PSNR: 27.45088678234982 28.07529491912581

Error Diffusion



PSNR: 27.456307352536275 28.085691152853034

```
(base) PS D:\OneDrive - 國立臺灣科技大學\桌面\image_processing_2024> &
_processing_2024/HW1Digital_Halftoning/sample.py"
PSNR_o1: 27.45088678234982

PSNR_o2: 28.07529491912581

PSNR_e1: 27.456307352536275

PSNR_e2: 28.085691152853034
```

2.Explain

Ordered Dithering

```python
def Ordered_Dithering(img, thresholds_matrix):
    # TODO:Implementing the ordered dithering algorithm
    N = thresholds_matrix.shape[0]
    Ordered_Dithering_img = np.zeros_like(img,np.uint8)
    height,width = img.shape

    for i in range(0,img.shape[0],N):
        for j in range(0,img.shape[1],N):
            for k in range(N):
                for l in range(N):
                    if (i + k < height and j + l < width):
                        if(img[i+k,j+l]>thresholds_matrix[k,l]):
                            Ordered_Dithering_img[i+k,j+l] = 255
                        else:
                            Ordered_Dithering_img[i+k,j+l] = 0

    return Ordered_Dithering_img
```

說明：使用 threshold matrix 將同樣區域內的像素與原圖做比較，如果大於 threshold 就設為 255，否則為 0。使用四層迴圈，i, j 為迭代圖片的每個 pixel，每次遞增 N (kernel 的大小)；k, l 為迭代每次比較的 kernel 區域。注意圖片的格式要使用 unsigned int 8bit。

```python
def generate_bayer_matrix(n):
    if n == 1:
        return np.array([[0, 2], [3, 1]])
    else:
        smaller_matrix = generate_bayer_matrix(n - 1)
        size = 2 ** n
        new_matrix = np.zeros((size, size), dtype=int)
        for i in range(2 ** (n - 1)):
            for j in range(2 ** (n - 1)):
                base_value = 4 * smaller_matrix[i, j]
                new_matrix[i, j] = base_value
                new_matrix[i, j + 2 ** (n - 1)] = base_value + 2
                new_matrix[i + 2 ** (n - 1), j] = base_value + 3
                new_matrix[i + 2 ** (n - 1), j + 2 ** (n - 1)] = base_value + 1
        return new_matrix
```

這是產生 Bayer matrix 的函式，會從 2*2 矩陣 ([[0, 2], [3, 1]]) 攤展到 2^n *2^n 大小的矩陣。

```python
def generate_thresholds_matrix(bayer_matrix):
    # TODO:Calculate each bayer matrix element threshold
    N = bayer_matrix.shape[0]
    thresholds_matrix = np.zeros_like(bayer_matrix, int)

    for i in range(N):
        for j in range(N):
            thresholds_matrix[i,j] = (255*(bayer_matrix[i,j]+0.5))/(N**2)

    return thresholds_matrix
```

在來使用 Bayer matrix 來產生 threshold matrix 的函式，按造公式設定 threshold。之後就能用 Ordered Dithering 演算法產生 halftoning 圖片。

Error Diffusion

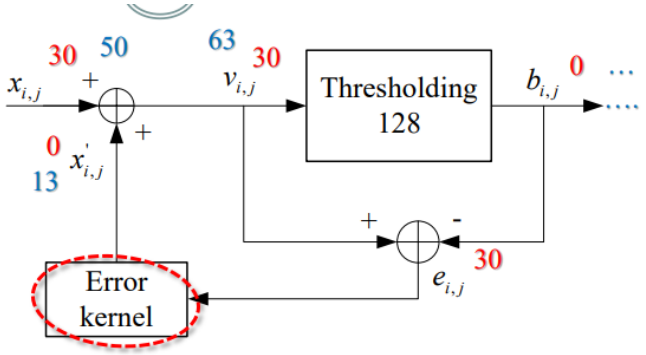
```python
def Error_Diffusion(img):
    # TODO:Implementing the error diffusion algorithm
    Error_Diffusion_img = img.astype(float).copy()
    height, width = img.shape
        You, now • Uncommitted changes
    '''
    kernel:
    1/16
    [   0  0  7
        3  5  1    ]
    '''

    for i in range(height - 1):
        for j in range(width - 1):
            old_pixel = Error_Diffusion_img[i, j]

            if( old_pixel > 128):
                new_pixel = 255
            else:
                new_pixel = 0

            Error_Diffusion_img[i, j] = new_pixel
            error = old_pixel - new_pixel

            # diffusion
            if(j+1<width):
                Error_Diffusion_img[i, j+1] += (error * 7) /16
            if(i+1<height and j-1>=0):
                Error_Diffusion_img[i + 1, j - 1] += (error * 3) /16
            if(i+1<height):
                Error_Diffusion_img[i+1, j] += (error * 5 )/16
            if(i+1<height and j+1<width ):
                Error_Diffusion_img[i+1, j+1] += (error * 1) /16

    return np.clip(Error_Diffusion_img, 0, 255).astype(np.uint8)
```

```python
kernel = np.array([
    [0,  0,  0,  7,  5],
    [3,  5,  7,  5,  3],
    [1,  3,  5,  3,  1]
    ]) / 48.0


for i in range(height - 2):
    for j in range(2, width - 2):
        old_pixel = Error_Diffusion_img[i, j]
        new_pixel = 255 if old_pixel > 128 else 0

        Error_Diffusion_img[i, j] = new_pixel
        error = old_pixel - new_pixel

        for ki in range(3):
            for kj in range(-2, 3):
                ni, nj = i + ki, j + kj
                if 0 <= ni < height and 0 <= nj < width:
                    Error_Diffusion_img[ni, nj] += error * kernel[ki, kj + 2]
```

上圖: N=3 的 error matrix。下圖 N=5 的 error matrix

說明：使用 Error Kernel 。設定 threshold 為 128，對於新的值：大於 128 會視為 255，小於則視為 0。然後用原本像素值去減掉新的值，乘上 error matrix 去將誤差擴散到各個像素。注意這便要使用浮點數以確保圖片的數值在誤差範圍。

演算法示意圖(來自老師講義)：



Calculate PSNR

```python
def calculate_PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if mse == 0:
        return np.infty
    max_pixel = 255.0
    psnr = 10 * np.log10((max_pixel ** 2) / mse)
    return psnr
```

PSNR 是評估影像處理後的品質的指標，以分貝為單位，如果越大代表圖片經

過處理後失真越少，也就是品質越好。

$$PSNR = 10\log_{10}\frac{\psi_{max}^{2}}{\sigma_e^2} \text{ (unit: dB)}$$

3.Discussion

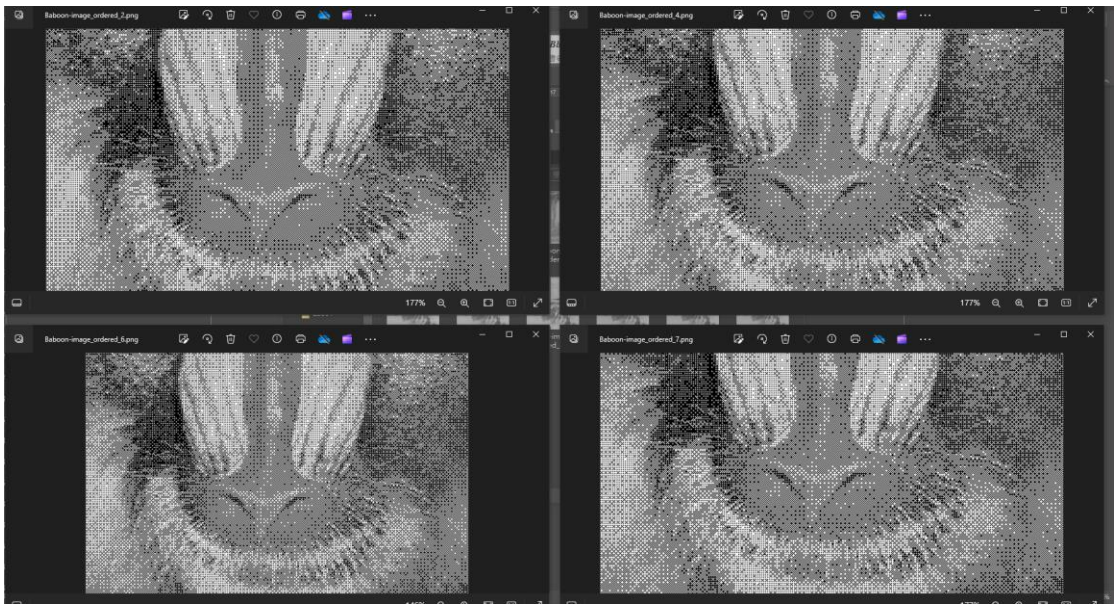
Ordered Dithering

```python
def process_image(n, img, img2):
    bayer_matrix = ht.generate_bayer_matrix(n)
    thresholds_matrix = ht.generate_thresholds_matrix(bayer_matrix)

    output1 = ht.Ordered_Dithering(img, thresholds_matrix)
    output2 = ht.Ordered_Dithering(img2, thresholds_matrix)

    cv.imwrite(f'./HW1Digital_Halftoning/evaluation/F-16-image_ordered_{n}.png', output1)
    print(f"{n} F_16 PSNR: {ht.calculate_PSNR(img,output1)}")
    cv.imwrite(f'./HW1Digital_Halftoning/evaluation/Baboon-image_ordered_{n}.png', output2)
    print(F"{n} Baboon PSNR: {ht.calculate_PSNR(img2,output2)}")

if __name__ == "__main__":
    img = cv.imread("./HW1Digital_Halftoning/images/F-16-image.png", cv.IMREAD_GRAYSCALE)
    img2 = cv.imread("./HW1Digital_Halftoning/images/Baboon-image.png", cv.IMREAD_GRAYSCALE)

    with ThreadPoolExecutor() as executor:
        futures = [executor.submit(process_image, n, img, img2) for n in range(2, 8)]
        for future in futures:
            future.result()

    print("done!")
```

實驗： 變化 N 大小，觀察圖片及 PSNR 變化。

Baboon:

N	N=2	N=4	N=6	N=7
PSNR	28.0748249065	28.08381223086	28.08316337632	28.08326373467



F16:

N	N=2	N=4	N=6	N=7
PSNR	27.45105661685	27.45633814043	27.456667158	27.4566947275

我改變 kernel 的大小，去觀察圖片，他們的 PSNR 沒有多大變化，但從 F-16 的背景(雲層)，可以看出 N 小時，會將背景的層次區塊變得更明顯；而 N 大時會有較平滑的效果，但會丟失細節。

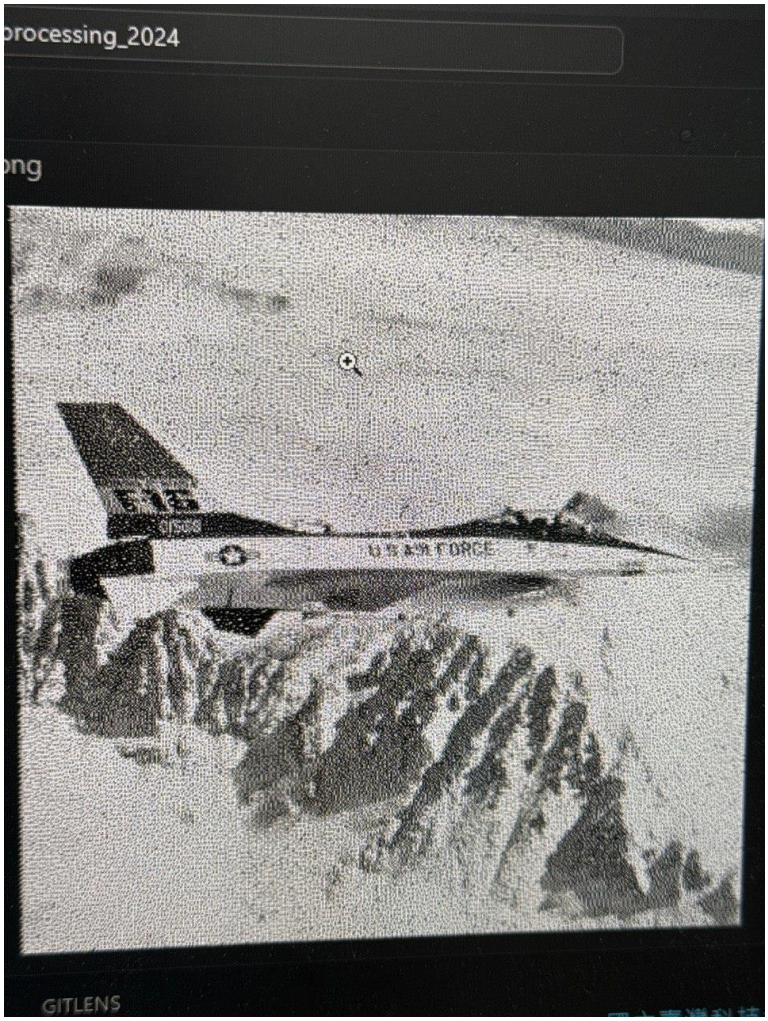
Error Diffusion

PSNR: 28.085691152853034 27.454146209203948

PSNR: 28.085691152853034 28.090867453211974

觀察：當 error kernel 使用較大的矩陣時，worm effect 反而比較明顯。

數據誤差：



我在實作 Error Diffusion 時，一開始使用 unsigned int 8bit 搭配 shift operation，想說可以加快運算速度和減少消耗資源，但結果因為精確度而產生雜訊 (上圖的背景)，所以後來改用 float v 減少計算誤差，就解決了這個問題。