HW 4 Report

Name: 陳翰雯

ID：B08902092

Problem 1: Digital Halftoning

(a) Dithering with I_2

Fig 1. sample1.png

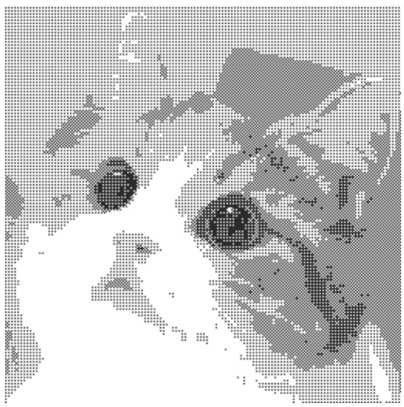

Fig 2. result1.png

(b) Dithering with I_{256}

Fig 1. sample1.png

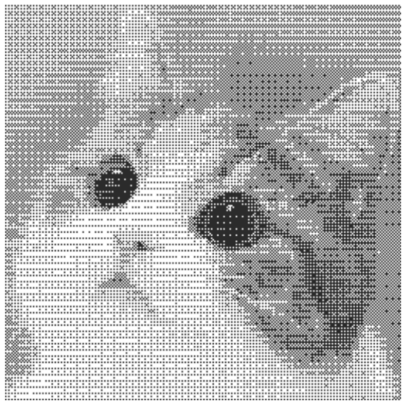

Fig 3. result2.png

From both images of result1.png and result2.png, there is a great difference in the details shown. Image that is dithered with (256x256) matrix results in a more detailed image in terms of its color intensity and the edges of the object portrayed.

Here to obtain the (256x256) matrix, we use the following code:

```
# Creating 256x256 dither matrix
matrix = dither_2
for i in range(int(np.log2(N))-1):
    A = 4 * matrix + 1
    B = 4 * matrix + 2
    C = 4 * matrix + 3
    D = 4 * matrix + 4
    matrix = np.block([[A,B],[C,D]])

dither_256 = matrix
threshold = 255 * ((dither_256+0.5) / (N**2))
```

(c) Error diffusion

Fig 1. sample1.png

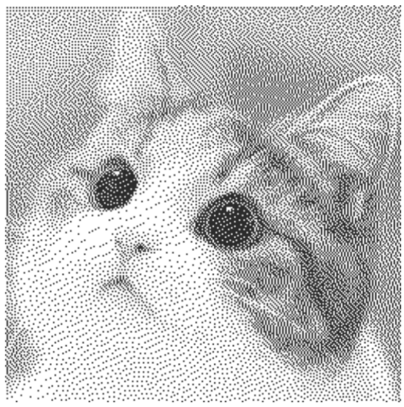

Fig 4. result3.png

The above image is a result of error diffusion with Floyd-Steinberg pattern as follows:

$1/16 \begin{bmatrix} & X & 7 \\ 3 & 5 & 1 \end{bmatrix}$, where X denotes the current pixel

Fig 1. sample1.png

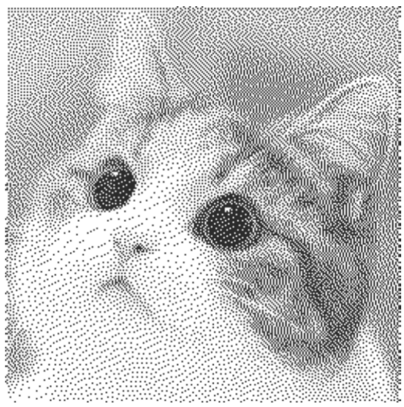

Fig 5. result4.png

The above image is a result of error diffusion with Jarvis' pattern as follows:

$$1/48 \begin{bmatrix} & & X & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$, where X denotes the current pixel

For further experiment, the image is also error diffused with Stucki dithering which was published by Peter Stucki. It was only a slight modification from Jarvis' pattern and it was designed this way mainly to improve processing time.

The pattern of Stucki is as follows:

$$1/42 \begin{bmatrix} & & X & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

As it was only a slight modification in its pattern, the resulting image of this pattern also did not show noticeable difference. Below are the comparison of the resultant image of the 3 patterns:

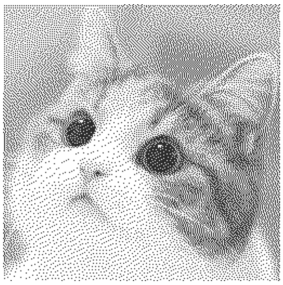

Fig 4. result3.png

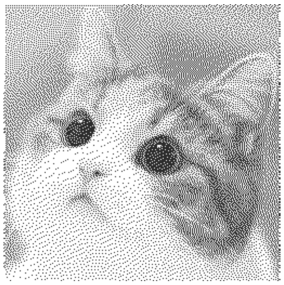

Fig 5. result4.png

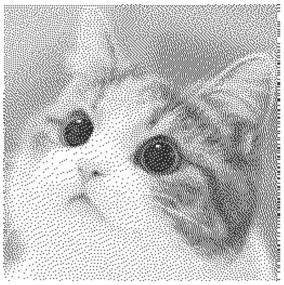

Fig 6. Stucki dithering method

Problem 2: Image Sampling

(a) "Inappropriate" image sampling

To avoid aliasing we have to sample with a spatial frequency that is higher than twice the highest spatial frequency of the original image.

To perform the “inappropriate” image sampling, I use the idea of down sampling which is basically to reduce the spatial resolution while keeping the same two-dimensional representation. Here, what I did was to down sample the image to 50% of its original size using the following code:

```
ds_img = cv2.resize(img, (0,0), fx=0.5, fy=0.5)
```

To show the aliasing effect, what I did was to reconstruct the down-sampled image back to its original size using the following implementation:

```
final = cv2.resize(ds_img, (h,w)) # h,w are the size of the original image (600x600)
```

Fig 7. sample2.png

Fig 8. result5.png

In the results, we could not see much difference between the 2 images. However, when you zoom in towards a part of the image, we could see great difference between the original image and the reconstructed, down-sampled image. Below are the visible differences:

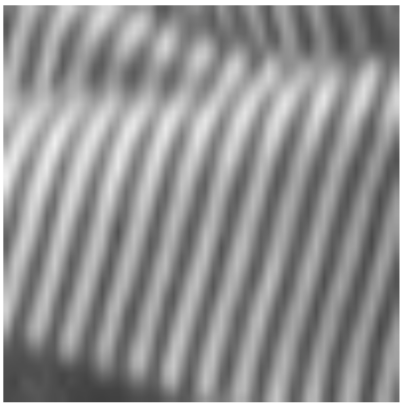

Fig 11. a portion from the original image

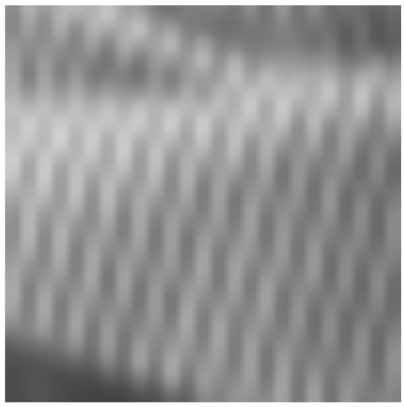

Fig 12. a portion from the down-sampled image

(b) Unsharp masking in the frequency domain

Fig 9. sample3.png

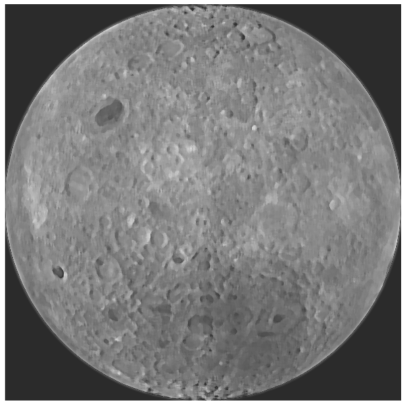

Fig 10. result6.png

The unsharp masking is done with the following formula:

$G(j,k) = F(j,k) + 2 \times (F(j,k) - F_L(j,k))$, where $F_L(j,k)$ is the low-pass of the Fourier Transform

To perform unsharp masking in the frequency domain, we first apply Fourier Transform to the image with the following code:

```
# Converting it into frequency domain by Fourier Transform
ft_img = np.fft.fft2(img)
ft_img = np.fft.fftshift(ft_img)
```

To obtain $F_L(j,k)$ we use Gaussian Low Pass filter which is the applied to out obtained frequency domain. This is done through the following implementation:

```
# Gaussian Low Pass
h,w = img.shape
D0 = 50
D = np.fromfunction(lambda u,v: np.sqrt((u-h//2)**2 + (v-w//2)**2), (h,w))
H = np.exp(-D**2/(2*D0**2))

# Applying filter or frequency domain
G = H * ft_img
```

We can then apply the unsharp masking formula through the following implementation:

```
# Unsharp masking
final = ft_img + 2 * (ft_img - G)
```

Lastly, we transform the result back to pixel domain by inverse Fourier transform through the following code:

```
# Inverse Fourier Transform
ift_img = np.fft.ifftshift(final)
ift_img = np.abs(np.fft.ifft2(ift_img))
```