HOMEWORK ASSIGNMENT #4
Digital Halftoning, Frequency Domain

Poy Lu 呂栢頤
D09944015
網媒所博一
ariapoy@gmail.com

May 26, 2021

1 Problem 1: DIGITAL HALFTONING

sample1.png is given in Figure 1.(a) Please apply several halftoning methods to the given image and provide discussions about the detail of the results.

Original image **sample1.jpg** for question Problem 1: DIGITAL HALFTONING.

1.1 (a)

Perform dithering using the dither matrix I_2 in Figure 1.(b) and output the result as **result1.png**

Motivation Render the illusion of a continuous-tone image based on half-tone (black and white only) display. **Dithering** methods could create an image with the same size of the original image. So we could start form dither matrix with I_2.

Approach Follow the steps in **Lec 6 page 11**.

1. Add noise $N(j,k)$ to get $H(j,k)$. I only implement **white noise** with `np.random.normal(0, 0.05`. Other noise type generator could reference Generate colors of noise in Python.
2. Use dither matrix I_2 to generate a threshold matrix $T_2^{(2)}(j,k)$.
3. Repeat threshold matrix to $T_{256}^{(2)}(j,k)$ with the same size with original image.
4. Dither by threshold matrix $T_{256}^{(2)}$ to get binary output $G(j,k)$.

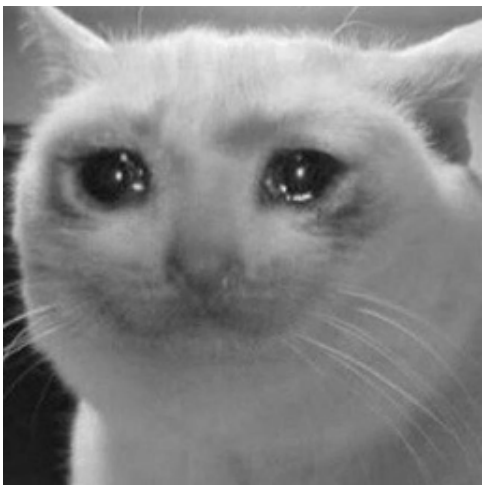

Figure 1: **sample1.jpg**

Figure 2: **result1.jpg** Dithering with I_2

Performance of results In the end, given dither matrix $I_2 = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$, and white noise $N(j,k) = \mathcal{N}(0, 0.05)$.

Result of problem 1(a): **result1.jpg** Dithering with I_2.

Discussion

1.2 (b)

Expand the dither matrix I_2 to I_{256} (256×256) and use it to perform dithering. Output the result as **result2.png**. Compare **result1.png** and **result2.png** along with some discussions.

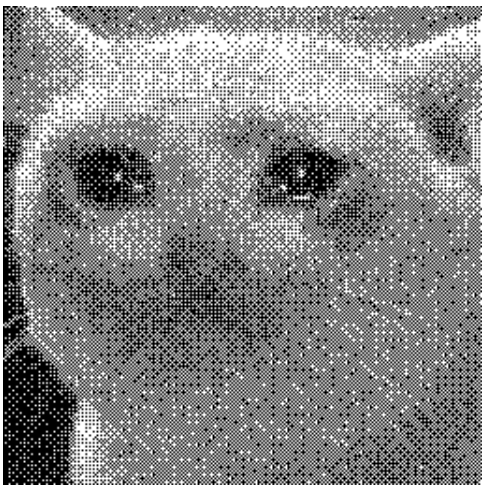

Figure 3: **result2.jpg** Dithering with I_{256}

Motivation Apart from 'Repeat threshold matrix' in step 3. We could create $N \times N$ dither matrix based on general form from $I_2 \to I_4 \to \ldots$.

Approach Follow the steps in **Lec 6 page 16**.

1. Design one-step expand with `expandonce_dither_mat`. Use `np.block` to generate block of sub-array.
   $I_{2n}(i,j) = \begin{bmatrix} 4I_n(i,j)+1 & 4I_n(i,j)+2 \\ 4I_n(i,j)+3 & 4I_n(i,j)+0 \end{bmatrix}$
2. Repeatedly I_2 to I_{256} as the same size as **sample1.jpg**.

Performance of results In the end, I choose the same settings as 1.1 to get I_{256} as well as $T_{256}^{(256)}$.

Result of problem 1(b): **result2.jpg** Dithering with I_{256}.

Discussion Compare **result1.png** and **result2.png**.

result1.jpg Dithering with I_2 generate half-tone with more $\times$ black patterns.

result2.jpg Dithering with I_{256} sketch fine and smooth on the *face of the cat*. But it shows more scatter black patterns around *cheek (臉頰) and mouth*.

1.3 (c)

Perform error diffusion with two different filter masks. Output the results as **result3.png**, and **result4.png**, respectively. Discuss these two masks based on the results.
You can find some masks **here** (from lecture slide 06. p23)

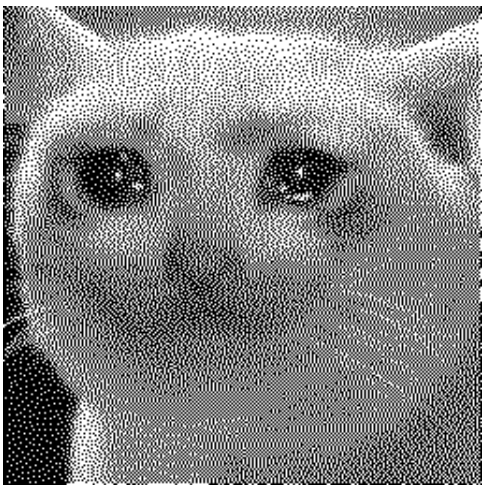

Figure 4: **result3.jpg** Error diffusion with mask Floyd Steinberg

Motivation The **error diffusion** is a practical algorithm to implement **blue noise dithering**.

Approach Follow the steps in **Lec 6 page 23—24**.

1. Generate **filter mask**. I implement **Floyd Steinberg** and **Jarvis** filter mask.
2. Error diffusion with serpentine scanning. As start from `i = 0`, I reverse order when `i % 2 == 1` is odd.

Performance of results In the end, I choose the settings with threshold $t = 0.5$.

For **Floyd Steinberg**, result of problem 1(c): **result3.jpg** Error diffusion with mask Floyd Steinberg.

For **Jarvis**, result of problem 1(c): **result4.jpg** Error diffusion with mask Jarvis.

Discussion Discuss **Floyd Steinberg** & **Jarvis** based on the results.

Floyd Steinberg: It seems $\times$ black patterns on the *neck*.

Jarvis: It seems more **twist** black patterns around the face. But I do not like its *left eye* with some **cluster** of whit patterns.

Both of **Floyd Steinberg** & **Jarvis** sketch *beards* more clear. And I feel **Floyd Steinberg** is more clear in its *six-line beards*.

I find out that baoundary of my image has some white line patterns. I consider that it occurs as I don't conduct padding of this image.

Other masks. I try other maske $\tilde{D} = \frac{1}{2}\begin{bmatrix} - & \bullet & 1 \\ - & 1 & 0 \end{bmatrix}$ result of problem 1(c): **result1c_diag** Error diffusion with mask $\tilde{D}$. This simple mask makes less contrast than **Jarvis** but

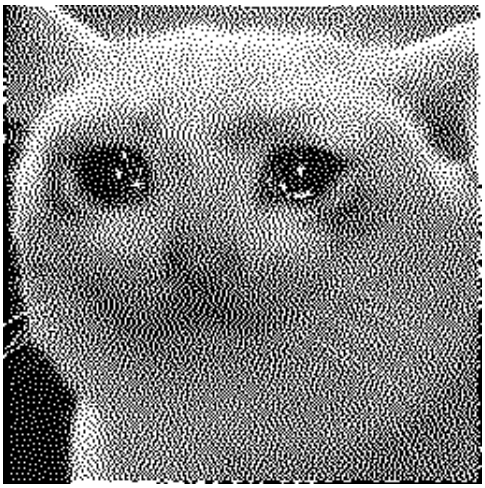

Figure 5: **result4.jpg** Error diffusion with mask Jarvis

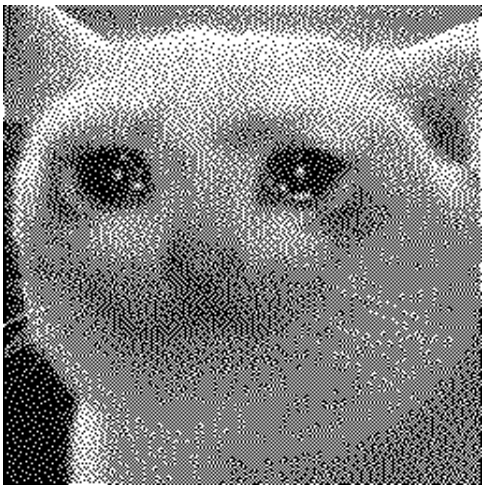

Figure 6: **result1c_diag** Error diffusion with mask $\tilde{D}$

seems better on its eyes for me.

1.4 (d)

Try to transfer **result1.png** to a dotted halftone/manga style binary image such as **sample1_dotted.png** in Figure 1.(c). Describe the steps in detail and show the result. You may need to utilize a function like **cv2.circle** to draw a circle.

Motivation It is interesting approach of **dotted halftone** / **manga style**.

Approach Rather than use `cv2.circle`, I use `PIL.ImageDraw` object with its `ellipse()` method.

1. Create new canvas as the same size of original binary image.
2. Design the kernel matrix $K_{3\times3}$ with siez 3.
3. Calculate **mean** of sub-array $\bar{H}$ by convolution of $K_{3\times3}$ then add white noise $N_{3\times3} \sim \mathcal{N}(0, 0.05)$.
4. Design **different radius** circle patterns based on the rules.

$$r = \begin{cases} 0, & \text{if } \bar{H} \leq 0.2 \\ 1, & \text{if } 0.2 < \bar{H} \leq 0.4 \\ 2, & \text{if } 0.4 < \bar{H} \leq 0.6 \\ 3, & \text{if } 0.6 < \bar{H} \leq 0.8 \\ 4, & \text{if } 0.8 < \bar{H} \leq 1 \end{cases}$$

   Note when radius $r = 4$, it means **square pattern** rather than **circle pattern** of sub-array.

5. Draw with `draw.ellipse((x-left, y-top), (x-right, y-bottom), radius, fill="white" )`

Performance of results In the end, based on **result1.jpg** Dithering with I_2.

Result of problem 1(d): **sample1_dotted.png** Dotted halftone style transfer.

Discussion Interesting style transfer methods. We could apply it on different dithering results, binary image. And design new patterns in the future.

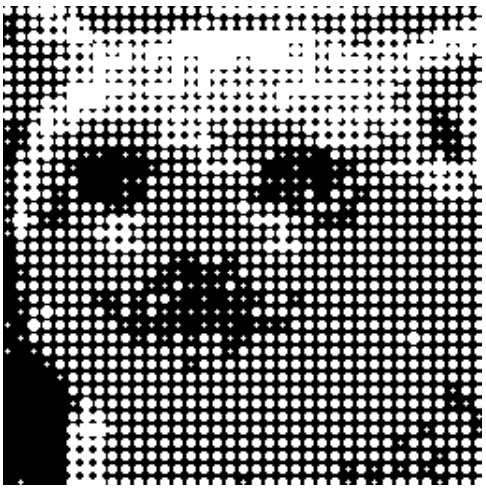

Figure 7: **sample1_dotted.png** Dotted halftone style transfer

2 Problem 2: FREQUENCY DOMAIN

In this problem, please perform Fourier transform and observe the relation between the spatial domain and the frequency spectrum. You may adopt tools for Fourier transform. The recommended tools are listed in the Appendix.

Original image **sample2.jpg** for question (a) (b), (c).

2.1 (a)

Perform Fourier transform on **sample2.png** to obtain its frequency spectrum and output it as **result5.png**. (Please take the log magnitude of the absolute value and center the low frequency part at the origin for visualization.)

Motivation **Frequency domain** could give us other respect of images. For visualization, **log transformation** could expand the dynamic range of low gray-level values and compress of high gray-level values.

Approach Follow the steps in **Lec 9 page 12—13**.

1. Centering and Fourier transform with

$$\mathcal{F}[f(j,k)(-1)^{j+k}] = F(u - \frac{M}{2}, v - \frac{N}{2})$$

   Take use of `np.fromfunction(lambda i, j: (-1)**(i + j), shape)`. Than `fft2` on it.

2. Take log and absolute value of $F[u - \frac{M}{2}, v - \frac{N}{2}]$ to get results $F'(u, v)$.

3. Normalize $F'(u, v)$ to $[0, 1]$ then scale to $[0, 255]$

Figure 8: **sample2.jpg**

Performance of results In the end, based on **sample2.jpg**.

Result of problem 2(a): **result5.jpg** Log axis of frequency domain.

Discussion

2.2 (b)

Based on the result of part (a), design and apply a low-pass filter in the frequency domain and transform the result back to the pixel domain by inverse Fourier transform. The resultant image is saved as **result6.png**. Please also design a low-pass filter in the pixel domain which behaves similarly to the one you design in the frequency domain. Output the result as **result7.png** and provide some discussions.

Motivation Filtering in frequency domain is easy to implement with Fourier transform. And it just design the filter matrix $h(j,k)$ **multiply** rather than **convolution** on frequency domain. So we could conduct the operation on overall image instead of sub-array on spatial domain.

Approach I choose **Gaussian filter**. As it is easy to specify and correspond between frequency and spatial domain.

1. Design Gaussian low-pass filter.

$$H_{\text{low}}(u,v) = \exp\left(\frac{-D(u,v)^2}{(2D_0^2)}\right)$$

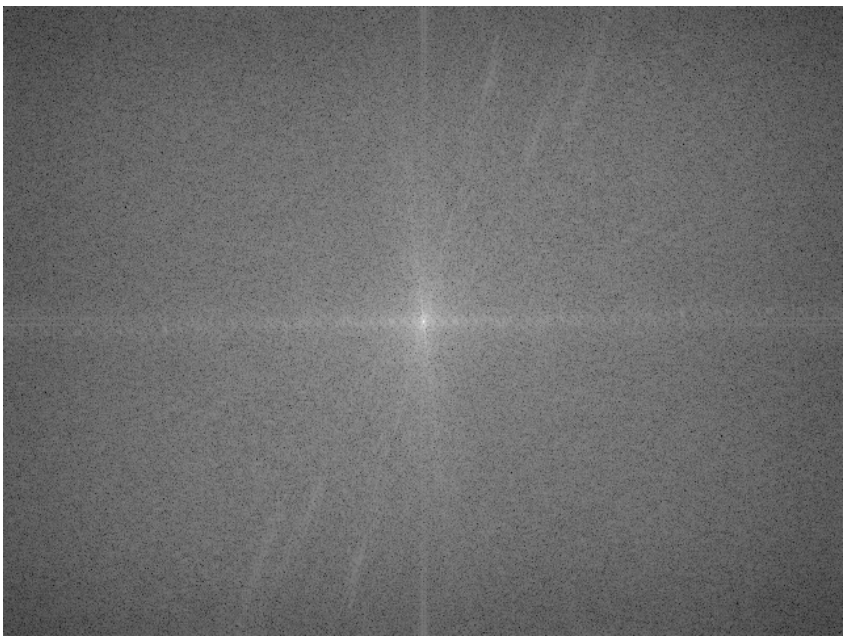

Figure 9: **result5.jpg** Log axis of frequency domain

where $D(u,v) = [(u - \frac{M}{2})^2 + (v - \frac{N}{2})^2]^{\frac{1}{2}}$
D_0 is cutoff, reflect standard deviation σ in Gaussin distribution.

2. Conduct Fourier transform $F(u,v)$ then mutiply $H_{\text{low}}(u,v)$ to get $G(u,v)$
3. Conduct inverse Fourier transform and shift (by -1^{j+k}) to get $g_{\text{low}}(j,k)$.

Correspond to frequency domain filtering, I implement spatial domain Gaussian filter.

1. Design Gaussian low-pass filter.

$$h_{\text{low}}(j,k) = \frac{1}{2\pi D_0^2} \exp\left(-\frac{(j - \frac{\text{kernel size}}{2})^2 + (v - \frac{\text{kernel size}}{2})^2}{2D_0^2}\right)$$

where kernel size,
D_0 is cutoff, reflect standard deviation σ in Gaussin distribution.

2. Conduct convolution by filter $g_{\text{low}}(j,k) = f(j,k) * h_{\text{low}}(j,k)$

Performance of results In the end, I choose the settings of $D_0 = 50$ on both frequency domain and spatial domain. And choose kernel size = 5 on spatial domain filter size.

For Gaussian low-pass filter on frequency domain, result of problem 2(b): **result6.jpg** Low-pass filter in frequency domain.

For Gaussian high-pass filter on spatial domain, result of problem 2(b): **result7.jpg** Low-pass filter in spatial domain.

Figure 10: **result6.jpg** Low-pass filter in frequency domain

Figure 11: **result7.jpg** Low-pass filter in spatial domain

Discussion Compare **result6.jpg** Low-pass filter in frequency domain with **result7.jpg** Low-pass filter in spatial domain.

result6.jpg Low-pass filter in frequency domain provide more **blurry** results although I choose the same D_0 (σ) on both domain. And I find out the white line on boundary. It could be my implementation error.

result7.jpg Low-pass filter in spatial domain could result **lattice effects** on each sub-region. I consider that it caused by the average/ convolution on each sub-array.

2.3 (c)

Based on the result of part (a), design and apply a high-pass filter in the frequency domain and transform the result back to the pixel domain by inverse Fourier transform. The resultant image is saved as **result8.png**. Please also design a high-pass filter in the pixel domain which behaves similarly to the one you design in the frequency domain. Output the result as **result9.png** and provide some discussions.

Motivation High-pass filter is another view in frequency domain. It correspond the **edge** on image.

Approach I choose **Gaussian filter**. And we could treat the high-pass filter as **original image minus low-pass filter**. For frequency domain,

1. Design Gaussian high-pass filter.

$$H_{\text{high}}(u,v) = 1 - H_{\text{low}}(u,v)$$

2. And follow process of frequency domain in 2.2.

Slightly different with frequency domain

1. Conduct spatial domain Gaussian low-pass filter in 2.2.
2. Original image minus low-pass result.

$$g_{\text{high}}(j,k) = f(j,k) - g_{\text{low}}(j,k)$$

As I find out that we couldn't change kernel filter before convolution.

Performance of results In the end, I choose the same settings as 2.2.

Result of problem 2(c): **result8.jpg** High-pass filter in frequency domain.

Result of problem 2(c): **result9.jpg** High-pass filter in spatial domain.

Figure 12: **result8.jpg** High-pass filter in frequency domain

Figure 13: **result9.jpg** High-pass filter in spatial domain

Figure 14: **sample3.jpg**

Discussion Compare **result8.jpg** High-pass filter in frequency domain with **result9.jpg** High-pass filter in spatial domain.

result8.jpg High-pass filter in frequency domain gives **thicker** and **fill** edges of image.

result9.jpg High-pass filter in spatial domain give **fragile** on the **large black region**.

Original image **sample3.jpg** for question (d) (e).

2.4 (d)

Perform Fourier Transform on **sample3.png** and output it as **result10.png**. Please discuss what you observe in **sample3.png** and **result10.png**.

Motivation It is relative simple conduct periodic noise reduction on frequency domain.

Approach The same steps as 2.1

Performance of results In the end, based on **sample3.jpg**.

Result of problem 2(d): **result10.jpg** Fourier Transform on **sample3.jpg**.

Discussion Observe in **sample3.jpg** and **result10.jpg** Fourier Transform on **sample3.jpg**. Here we could see the **vertical ripple** on the original image is corresponding to **two bright spots on horizontal center line**.

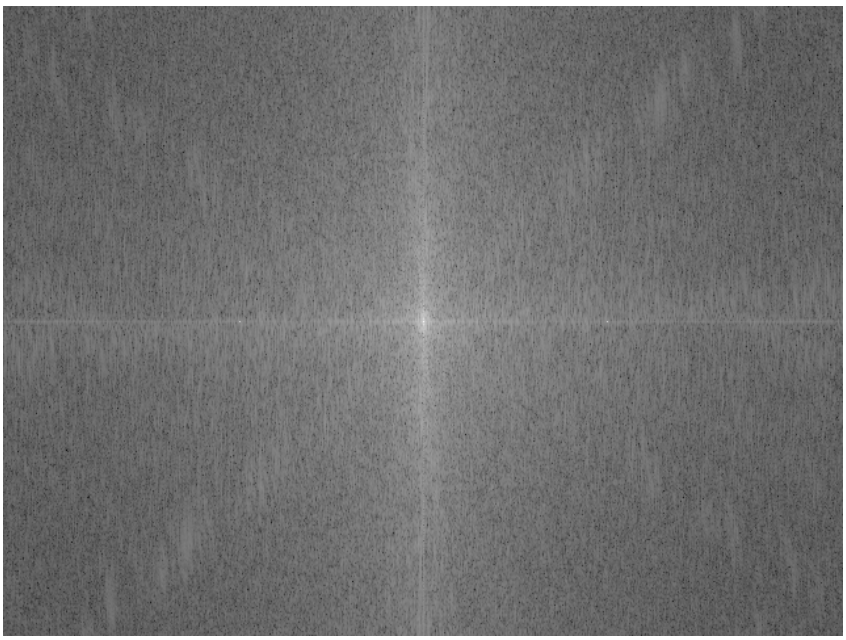

Figure 15: **result10.jpg** Fourier Transform on **sample3.jpg**

2.5 (e)

Try to remove the undesired pattern on **sample3.png** and output it as **result11.png**.

Motivation It is relative simple conduct periodic noise reduction on frequency domain.

Approach Follow the steps in **Lec 9 page 35**.

1. Use **butterworth** on two bright spots on horizontal center line on frequency domain.
2. Conduct inverse Fourier transform to recover original image.

Performance of results In the end, I choose the butterworth range with

- `M//2 - 1: M//2 + 2, 165:185`
- `M//2 - 1: M//2 + 2, 455:475`

where `M` is center horizontal line.

Result of problem 2(e): **result11.jpg** Noise cleaning of **sample3.jpg**.

Discussion Get critical removal points is crucial for denoising in frequency domain. Maybe we could develop **automatic process** to denoise ripple noise image in future.

Figure 16: **result11.jpg** Noise cleaning of **sample3.jpg**