2024 NTU DIP HW4

陳德維

May 7, 2024

Problem 1 : DIGITAL HALFTONING

(a) According to the dither matrix I2, please perform dithering on **sample1.png** to obtain a binary image **result1.png**.

Result

sample1 result1

Figure 1: Image Dithering w/ I_2

Approach

使用 spec 上提供的 dither matrix I_2:

$$I_2 = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

再透過這個公式從 dither matrix 生成 threshold matrix:

$$threshold(i,j) = \frac{255 * I_2(i,j) + 0.5}{2^2}$$

最後使用這個 threshold matrix，掃過整張圖決定哪些 pixel 是 255 哪些是 0 即完成。

(b) Expand the dither matrix I_2 to I_{256} (256×256) and use it to perform dithering on **sample1.png**. Output the result as **result2.png**. Compare **result1.png** and **result2.png** along with some discussions.

Result

sample1 result2

Figure 2: Image Dithering w/ I_{256}

Approach

使用 spec 上提供的 dither matrix I_2:

$$I_2 = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

再透過這個公式從 I_2 拓展到 I_{256}：

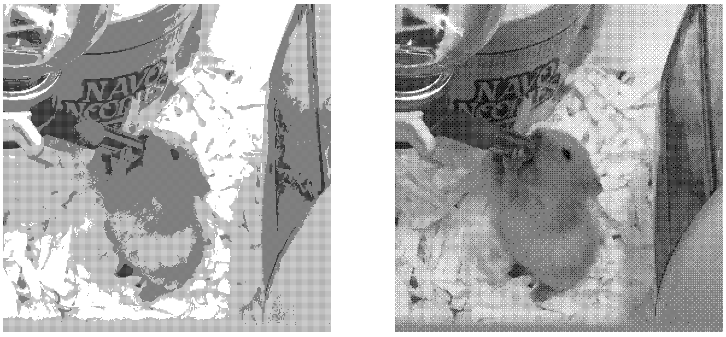
$$I_{2n} = \begin{bmatrix} 4I_n + 1 & 4I_n + 2 \\ 4I_n + 3 & 4I_n + 0 \end{bmatrix}$$

再透過這個公式從 dither matrix 生成 threshold matrix:

$$threshold(i, j) = \frac{255 * I_{256}(i, j) + 0.5}{256^2}$$

最後使用這個 threshold matrix，掃過整張圖決定哪些 pixel 是 255 哪些是 0 即完成。

Comparison



result1 result2

Figure 3: Image Dithering I_2 v.s. I_{256}

可以看到使用 I_2 的效果比起 I_{256} 來得差許多，推測是因為 $2x2$ 的範圍過小，導致 threshold 打開的標準沒辦法符合整張照片的趨勢，因此會有失真的狀況發生。

(c) Perform error diffusion with Floyd-Steinberg and Jarvis' patterns on **sample1.png**. Output the results as **result3.png** and **result4.png**, respectively. You may also try more patterns and show the results in your report. Discuss these patterns based on the results. You can find some masks here (from lecture slide 06. p21)

Result

sample1 result3

Figure 4: Error Diffusion w/ Floyd-Steinberg

Result


sample1


result4

Figure 5: Error Diffusion w/ Jarvis'

Approach

我採用了 Error Diffusion 以及 Serpentine scanning 的方式來進行掃描。首先我們將 pixel value 標準化至 $[0, 1]$，再來以 Serpentine scanning 一來一回的方式掃描各個 pixel，接下來利用 threshold 來決定其應為 0 還 1，並計算 error。最後透過給定的 error diffusion matrix 來決定要怎麼把 error 擴散出去。

其中 Floyd-Steinberg 的 error diffusion matrix 為：

$$D = \begin{bmatrix} 0 & 0 & 7/16 \\ 3/16 & 5/16 & 1/16 \end{bmatrix}$$

反向 scanning（右至左）時則為：

$$D = \begin{bmatrix} 7/16 & 0 & 0 \\ 1/16 & 5/16 & 3/16 \end{bmatrix}$$

Jarvis' 的 error diffusion matrix 為：

$$D = \frac{1}{48} \begin{bmatrix} 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

反向 scanning（右至左）時則為：

$$D = \frac{1}{48} \begin{bmatrix} 5 & 7 & 0 & 0 & 0 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

Discussion

從 result3 和 result4 中可以看到，相較之下 Jarvis’的看起來有較明顯的紋路，我個人是比較喜歡 Floyd-Steinberg 的結果。另外我也嘗試比較有無 Serpentine scanning 的差異，發現結果差異不會到很大，在一些平坦的區域會有比較明顯的 pattern 差異。

我也試了其他種類的 error diffusion matrix：

- Atkinson 的 error diffusion matrix 為：

$$D = \frac{1}{8}\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

sample1 Atkinson

Figure 6: Error Diffusion w/ Atkinson

- Burkes 的 error diffusion matrix 為：

$$D = \frac{1}{32}\begin{bmatrix} 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \end{bmatrix}$$

sample1 Burkes

Figure 7: Error Diffusion w/ Burkes

- Sierra 的 error diffusion matrix 為：

$$D = \frac{1}{32}\begin{bmatrix} 0 & 0 & 0 & 5 & 3 \\ 2 & 4 & 5 & 4 & 2 \\ 0 & 2 & 3 & 2 & 0 \end{bmatrix}$$

sample1 Sierra

Figure 8: Error Diffusion w/ Sierra

Problem 2 : SHAPE ANALYSIS

(a) Given a training set as shown in Figure 2, please deign a recognition system for license plates and perform it on **sample2.png**, **sample3.png** and **sample4.png** as shown in Figure 3, respectively. In other words, a license plate image serves as an input to the system, and the output is its corresponding characters. For example, if the input is **sample3.png**, your program is supposed to output ”FAE681”. Note that you don’t have to recognize the special character, ”-”. Please provide the flow chart along with the detailed descriptions of your algorithm. Also show the results and discuss on both the successful and failure cases.

Approach

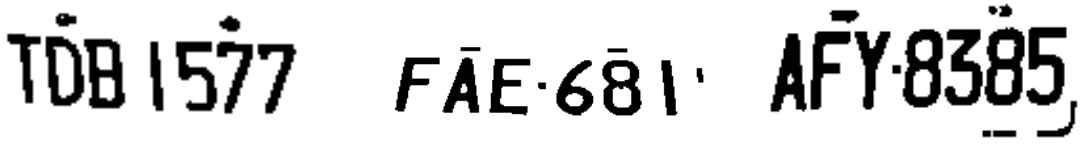
(1) **Thresholding**
由於原圖讀入時為灰階，我們先將它二值化成 0 和 255，我這邊選用 120 為 threshold，超過即為 255，反之為 0。

(2) **Convert to Black Text / White Background**
我先計算二值化後的圖片黑白 pixel 分別有多少，並使用了 heuristic 來決定目前這張圖有沒有黑白相反 (*i.e.*White Text / Black Background)。目的是將需要辨識的文字統一成黑色。

sample2 sample3 sample4

Figure 9: Convert to Black Text / White Background

(3) **Denoise**
可以看到現在字元都是黑色的了，但是旁邊仍然有許多非字元的黑色，因此在這個步驟我使用了一些 greedy 的方式以及搭配 morphlogical processing 來將周遭的黑色去掉。



sample2 sample3 sample4

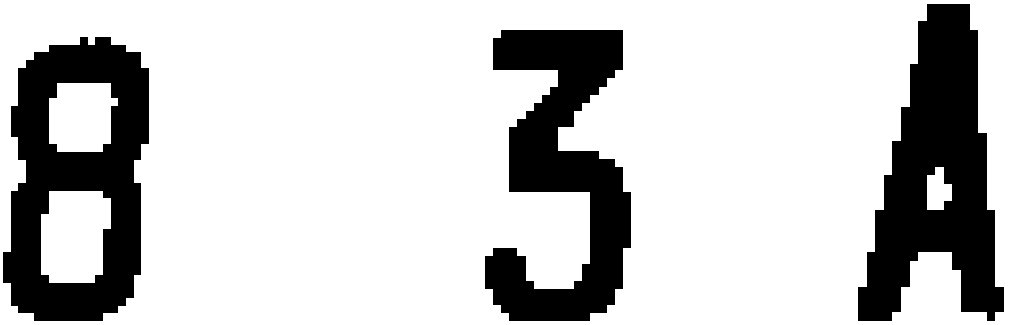
Figure 10: Denoise

(4) **Connected Component Labeling**
對這張圖的黑色部分進行 connected component labeling，要注意的是圖上仍有許多非目標的黑色，因此我決定一個 threshold 85，只要這個 connected component 累積的黑色 pixel 數量小於 threshold 則捨棄。最後結果如下：

sample2 sample3 sample4

Figure 11: Connected Component Labeling

(5) **Bounding Rectangle / Segmentation**
開始遍歷每一個 connected component，找到每一個 component 的四個角落，並將其 crop 出來。效果如下：



8 3 A

Figure 12: Bounding Rectangle / Segmentation

(6) **Train Set Preprocessing**
對 TrainSet 做上方相同的事，可得：

Figure 13: Connected component labeling (Train Set)

8 3 A

Figure 14: Bounding Rectangle / Segmentation

(7) **Feature Extraction**
我使用了 Bit Quads 來尋找 Average Area, Euler Number, Average Perimeter。將這些 feature 形成一個一維 vector 以供後續做比較。

- **Average Area (Duda's algorithm)**

$$A = \frac{\frac{1}{4}n\{Q_1\} + \frac{1}{2}n\{Q_2\} + \frac{7}{8}n\{Q_3\} + n\{Q_4\} + \frac{3}{4}n\{Q_D\}}{\#\ of\ pixels}$$

- **Euler Number (Eight Connectivity)**

$$E = \frac{1}{4}(n\{Q_1\} - n\{Q_3\} - 2n\{Q_D\})$$

- **Average Perimeter (Duda's algorithm)**

$$A = \frac{n\{Q_2\} + \frac{1}{\sqrt{2}}(n\{Q_1\} + n\{Q_3\} + 2n\{Q_D\})}{\#\ of\ pixels}$$

(8) **Inference**
分別對當前的 feature vector 與 train set 的 feature vector 做 L2-loss，但我有對 Euler Number 做加權，使其重要性提高。最後取 loss 最低的當作目標 character。

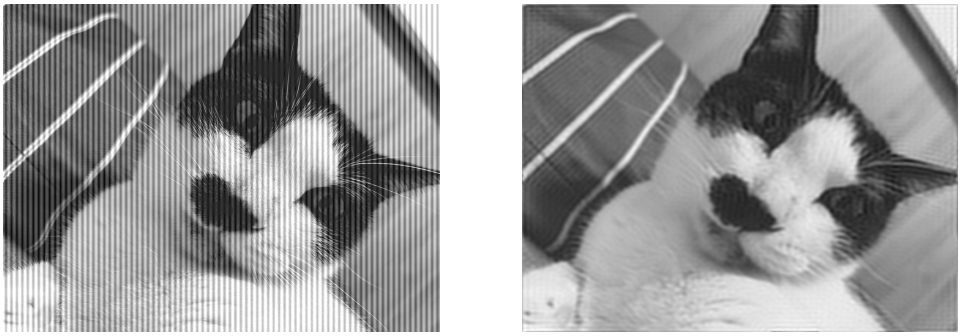
Result

在結果上我沒有取得正確的結果，我推測問題出在 feature 上的選擇，這次作業我遇到了一些事情，沒能好好的把它實做完成我覺得很可惜。我目前的想法是我應該可以去試著做 profiling, 2-axis projection, skeletonizing, 並搭配 hough transform。或許經過這些處理後，就可以達到很好的效果。(參考 https://www.slideserve.com/andrew/optical-character-recognition-for-handwritten-characters)

Problem 3 : FREQUENCY DOMAIN

(a) (bonus) Please remove the undesired pattern from **sample5.png** using a Fourier transform, and save the result as **result5.png**. Describe in detail how you accomplish this task.

Result



sample5 result5

Figure 15: Frequency Domain

Approach

首先我們先將 sample5 轉至 frequency domain，我們利用 np.fft.fft2() 來進行 2D Discrete Fourier Transform，再利用 np.fft.fftshift() 將 0-frequency 移至畫面中心。接下來再對其轉換後的結果做 Log Transformation 以方便觀察。

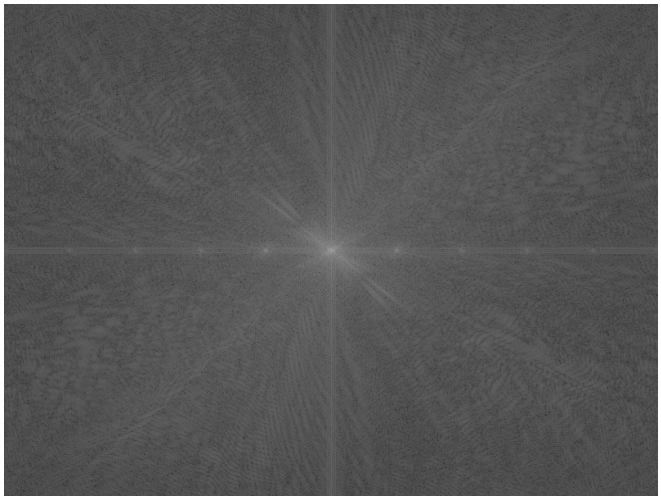


Figure 16: sample5 (Frequency Domain)

接下來，觀察其高頻區域可以看到十分混雜，因此我決定去實作上課提到的 3 種

low-pass filter 來對他進行 processing。最後再利用 np.fft.ifft2(np.fft.ifftshift()) 將其轉回 Image Domain 並取絕對值即可。

- Ideal Low-pass Filter

$$H(u,v) = \begin{cases} 1, & \text{if } D(u,v) \leq D_0 \\ 0, & \text{else} \end{cases}, \quad D(u,v) = ((u - \frac{M}{2})^2 + (v - \frac{N}{2})^2)^{\frac{1}{2}}$$

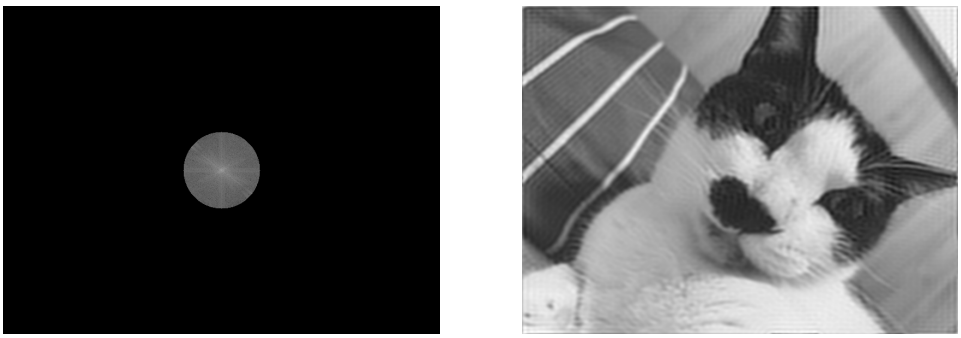
- Gaussian Low-pass Filter

$$H(u,v) = e^{\frac{-D^2(u,v)}{2D_0^2}}, \quad D(u,v) = ((u - \frac{M}{2})^2 + (v - \frac{N}{2})^2)^{\frac{1}{2}}$$

- Butterworth Low-pass Filter

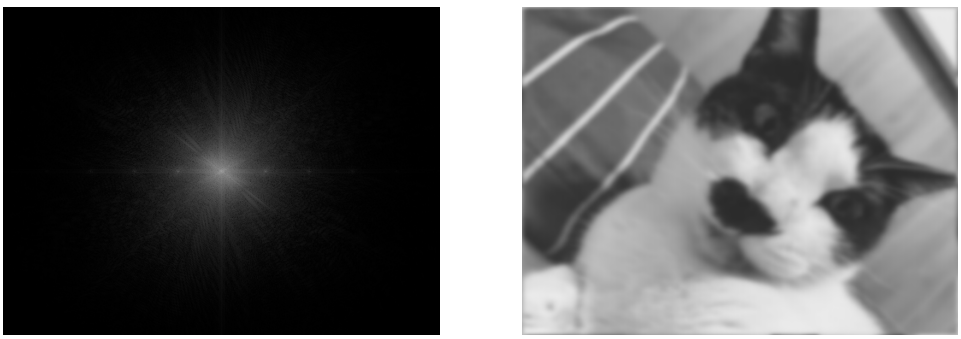
$$H(u,v) = \frac{1}{1 + (\frac{D(u,v)}{D_0})^{2n}}, \quad D(u,v) = ((u - \frac{M}{2})^2 + (v - \frac{N}{2})^2)^{\frac{1}{2}}$$

套用後，結果如下：



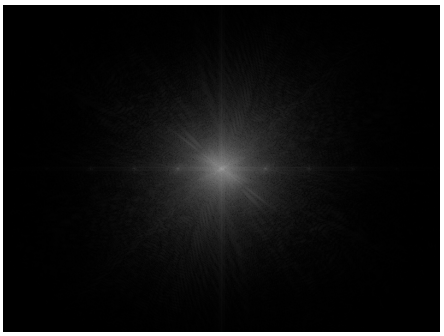
Frequency Domain Image Domain

Figure 17: Ideal Low-pass ($D_0 = 70$)



Frequency Domain Image Domain

Figure 18: Gaussian Low-pass ($D_0 = 25$)



Frequency Domain

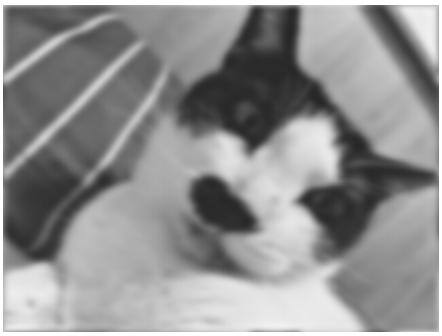


Image Domain

Figure 19: Butter-worth Low-pass ($D_0 = 25$, $n = 2$)

實測效果發現 Ideal Low-pass filter with $D_0 = 70$ **的效果最好**。