

# DIP Homework Assignment #4

姓名：鍾毓安

學號：B01902040

系級：資訊四

Email: [iamyuanchung@gmail.com](mailto:iamyuanchung@gmail.com)

繳交日期：2016.5.31

## Execution

To reproduce the result, simply execute README.m under Matlab environment.

## Problem 1: Shape Analysis

Sample1.raw is a gray-level image that contains some characters. Please design an algorithm to recognize different characters and count the number of occurrence of each character using TrainingSet.raw as the training set. Please provide the flow chart and details of your algorithm, and discuss the result in the report.

For problem 1, the following two functions were implemented:

1. shapeAnalysis.m: The function takes Sample1.raw and TrainingSet.raw, denoted as S1 and TS, respectively, as inputs and performs a series of steps, which will be described in detailed later in Figure 1-3, to deal with the required task. Finally, for each instance that appears in S1, the function outputs the class of shape it belongs to.
2. signSegment.m: The function takes S1 as input and segments S1 into small pieces recursively such that each piece is an independent instance to be classified. The function returns a cell array Ins, where the i-th instance (an image matrix) in S1 can be accessed by Ins{i}. Figure 1-1 depicts the concept of the recursive segmentation performed by signSegment.m.



Figure 1-1: Starting from the leftmost column of S1, the function finds the first column that contains at least one black pixel, as suggested by the red vertical line. Then, from that red vertical line, the function searches for the first column that contains all white pixels, as suggested by the blue vertical line. The segment between the red and the blue vertical lines is then taken as an independent input image by another signSegment.m and the same task described previously is performed again, except that for now the function starts from the top row and looks for the first row that contains at least one black pixel and so on.

Figure 1-2 displays some instances segmented by signSegment.m.



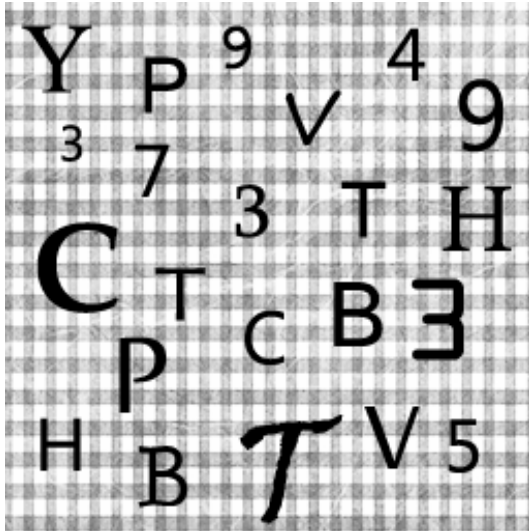
Figure 1-2: From left to right are Ins{1}, Ins{2}, Ins{11}, Ins{19}, and Ins{20}.

We can see that they can have very different sizes even they belong to the same category.

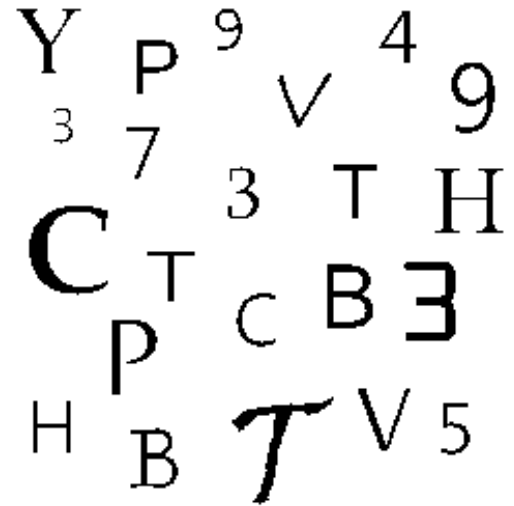
Undoubtedly, this will be one of the difficulties we will encounter during classification.

Figure 1-3 is the flow chart of shapeAnalysis.m. The following is the description of each step:

1. Sample1.raw, denoted as S1 is taken as input. However, the background of S1 contains noise. We first remove it and denote the clean version as S1\_clean.



S1

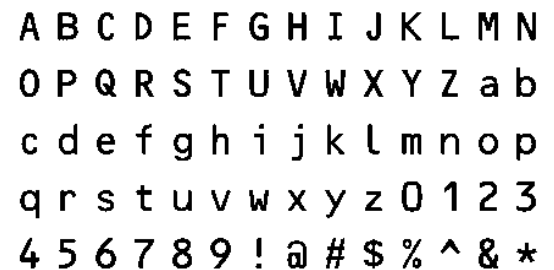


S1\_clean

2. S1\_clean is then fed in the function signSegment.m, which segments S1\_clean into separate instances to be classified. signSegment.m returns a cell array Ins, where each entry Ins{i} is the i-th instance (an image matrix). Some examples of the segmented instances are shown in Figure 1-2.
3. The TrainingSet.raw, denoted as TS, contains some pixels whose values are not 0 or 255. To make the later matching more convenient, TS is binarized such that the pixel values in the resultant image, denoted as TS\_bin, are either 0 or 255.



TS



TS\_bin



4. The classes of shapes provided in TS\_bin are so well-organized that we can separate them heuristically. By setting a window size and slicing it through TS\_bin, we obtain another cell array Cls, where each entry Cls{j} is the image matrix of the j-th class (counting from left to right, top to bottom). To make the later matching more convenient, each class Cls{j} is further trimmed such that the surrounding spaces are removed. Figure 1-4 displays some examples of classes before and after trimming.



Figure 1-4: Three pairs of before-after examples are shown.

The outer spaces are removed after  $\text{Cls}\{j\}$  is trimmed.

5. We now have two cell arrays:  $\text{Ins}$  and  $\text{Cls}$ . For each instance  $\text{Ins}\{i\}$ , we want to classify it into one of the seventy classes provided in TS. This is done by computing the similarity between  $\text{Ins}\{i\}$  with each  $\text{Cls}\{j\}$ :  $\text{Ins}\{i\}$  is first resized to the same size as  $\text{Cls}\{j\}$ . Then, the similarity between  $\text{Ins}\{i\}$  with current  $\text{Cls}\{j\}$  is measured by the overlapping percentage, that is, the number of pixels that have the same values divided by the area of  $\text{Cls}\{j\}$ . The larger the overlapping percentage is, the more similar they are.  $\text{Ins}\{i\}$  will then be classified as class  $j^* = \underset{j=1\sim70}{\operatorname{argmax}} \operatorname{Sim}(\text{Ins}\{i\}, \text{Cls}\{j\})$ . The reason why I use the overlapping percentage instead of the counts of pixels with the same values is that  $\text{Cls}\{j\}$  with larger size always has a better chance to win over another whose size is smaller, this cause the classification to be inaccurate.
6. By taking these steps, 16 out of 22 instances are classified correctly, that is, accuracy =  $16/22 \cong 73\%$ . Those six misclassified instances are somewhat originally hard to be classified and the misclassifications

are understandable, for example:  is classified as 8, and  is classified as 7.

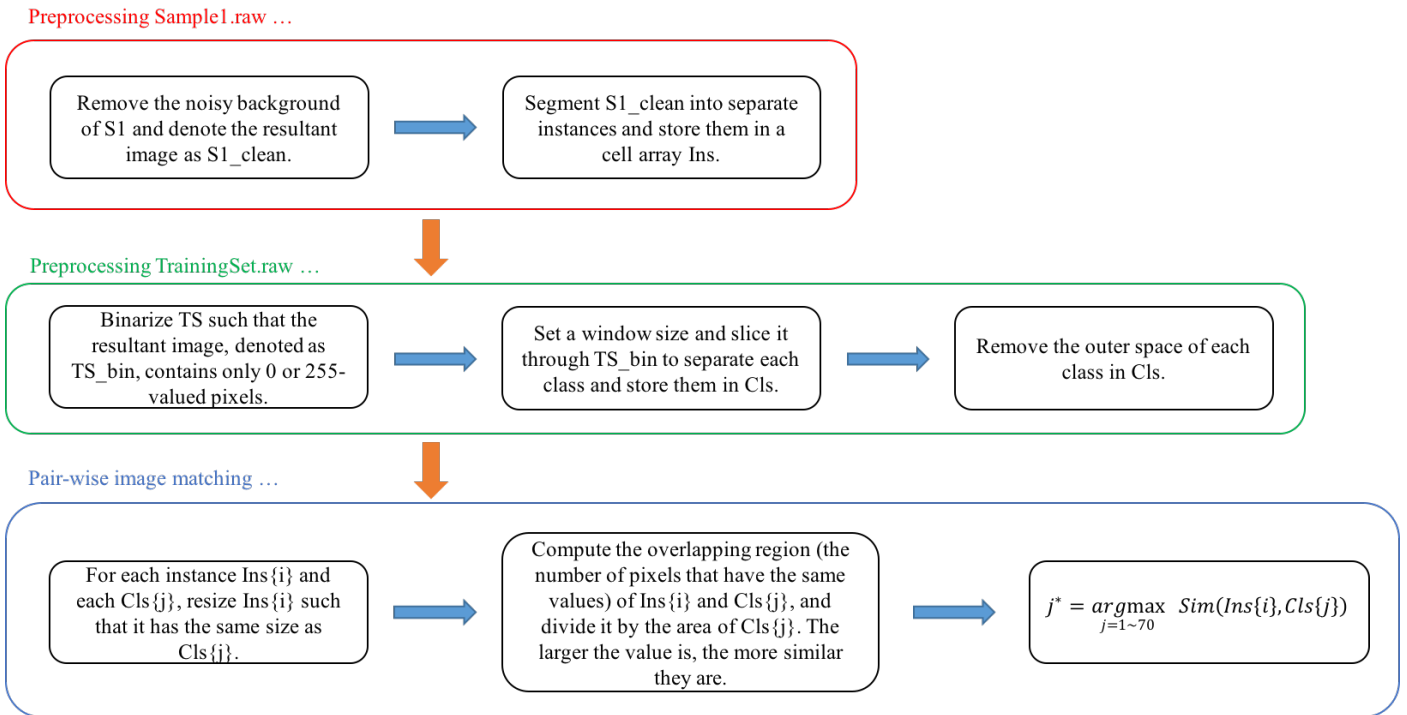


Figure 1-3: Flow chart of Problem 1. Note that the order of Processing Sample1.raw and Processing TrainingSet.raw does not matter, as long as  $\text{Ins}$  and  $\text{Cls}$  are generated before the Pair-wise image matching.

## Problem 2: Morphological Processing

Given a binary image Sample2.raw, please try to produce the same images as illustrated in Fig. 4 by adopting appropriate morphological processing. Please describe the designed algorithm in detail for each case.

- (a) Skeletonizing: skeletonizeImage.m skeletonizes Sample2.raw, denoted as  $S2$ , into the first expected image. Figure 2-1 is the flow chart skeletonizeImage.m.

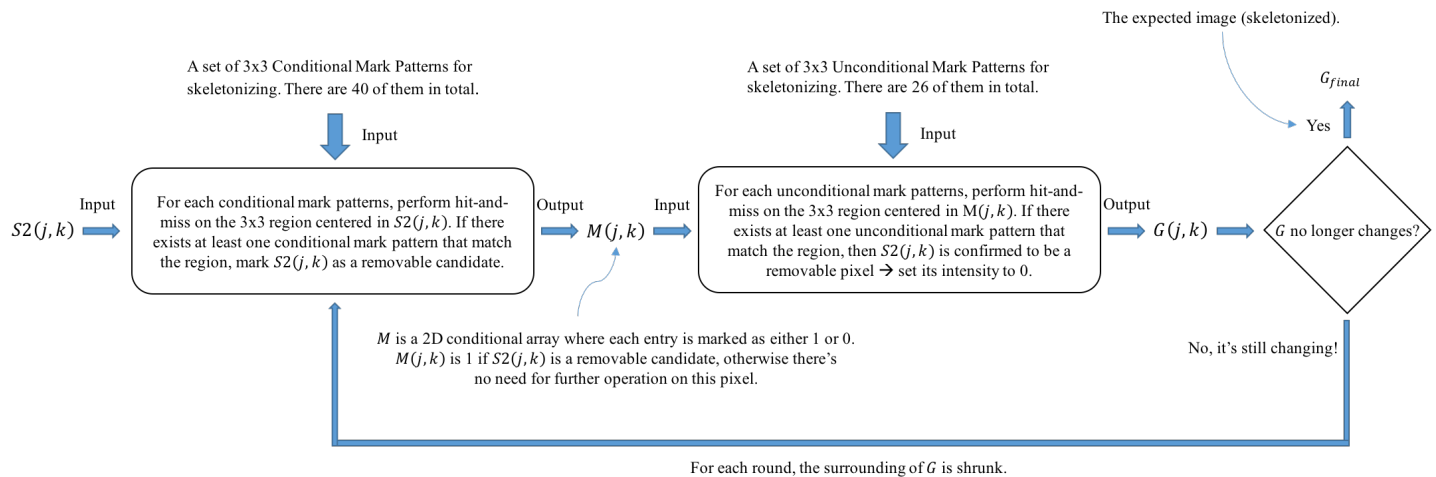


Figure 2-1: Flow chart of skeletonizeImage.m. The conditional and unconditional mark patterns are referenced from: <https://www.coursehero.com/file/6654786/patterntables/>.

The process of skeletonizing is very interesting, so I made the following animation, which shows the regions that is going to be removed in each round:

[https://www.csie.ntu.edu.tw/~b01902040/doc/DIP\\_hw4\\_Q2\\_Skeletonize.gif](https://www.csie.ntu.edu.tw/~b01902040/doc/DIP_hw4_Q2_Skeletonize.gif).

Figure 2-2 shows the skeletonized Sample2.raw.

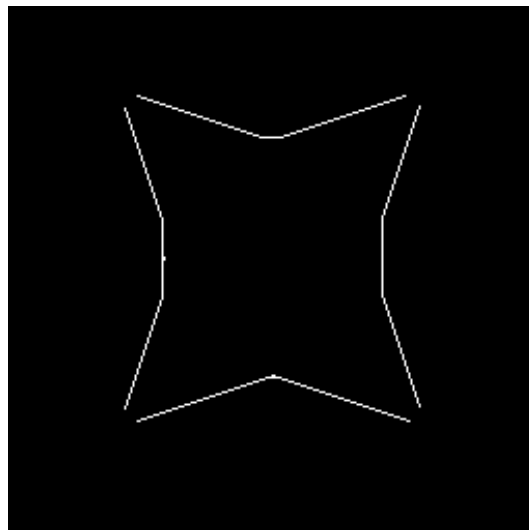


Figure 2-2: The skeletonized Sample2.raw

- (b) Boundary Extraction: For a given image  $F$ , its boundary  $\beta(F) = F - (F \ominus H)$ , where  $\ominus$  is the erosion operation and  $H$  is the structural element ( $3 \times 3$  foreground filter) with origin at its center. erodeImage.m takes an image  $F$  and a specified structural element  $H$  as inputs and perform erosion. extractBoundary.m simply minus  $F$  with the result of erodeImage( $F, H$ ), and the resultant image is the desired boundary.

Figure 2-3 shows the extracted boundary of Sample2.raw.

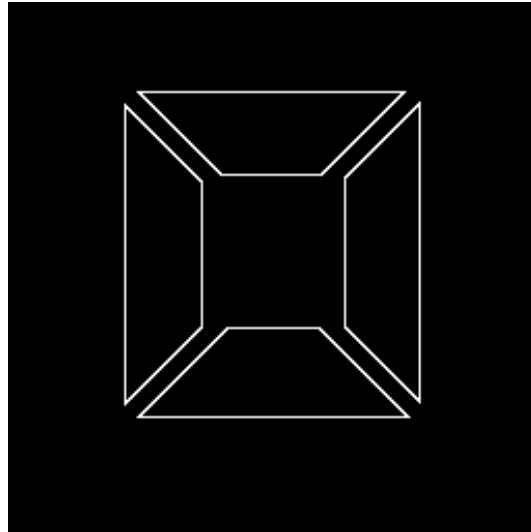
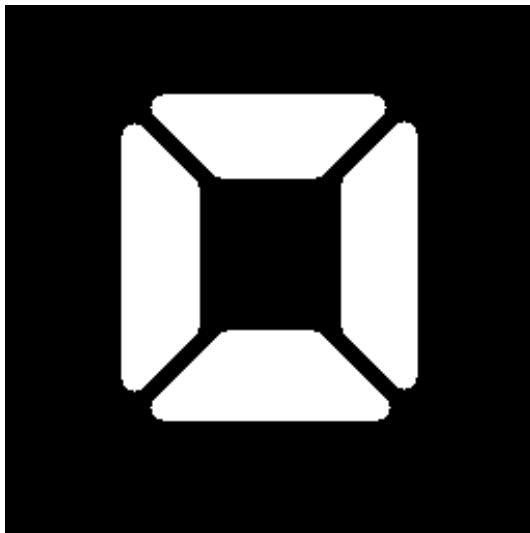


Figure 2-3: The boundary of Sample2.raw

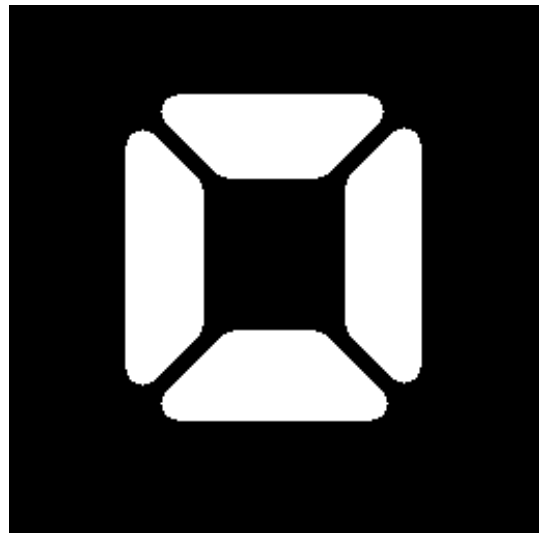
(c) Opening: For a given image  $F$  and a specified kernel  $H$ , the open operation performs as follows:

$$F \diamond H = [F \ominus H] \oplus H,$$

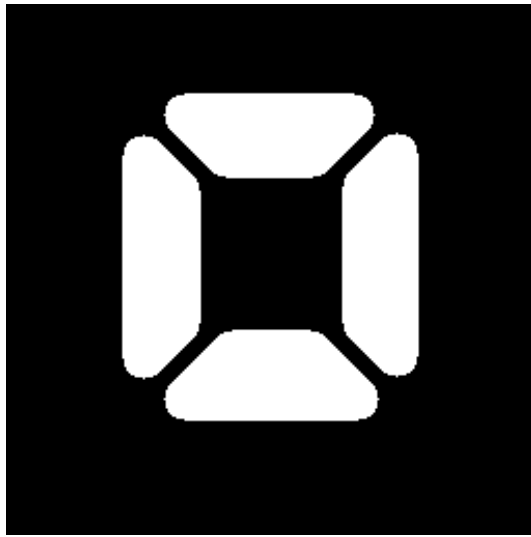
where  $\ominus$  denotes the erode operation and  $\oplus$  denotes the dilate operation. The dilateImage.m is further implemented to support the task. We need to define the kernel  $H$  used in the open operation, and this is done by function createRoundKernel.m: given a radius, the the function generates a round kernel such that the pixels within the radius are set as foreground pixels and others are set as background pixels. After the round kernel is generated, the third expected image can be obtained by calling the function opening.m, which first calls erodeImage.m, then calls dilateImage.m. Figure 2-4 displays four resultant images after the open operation using round kernels with radius = 6, 8, 10, 12.



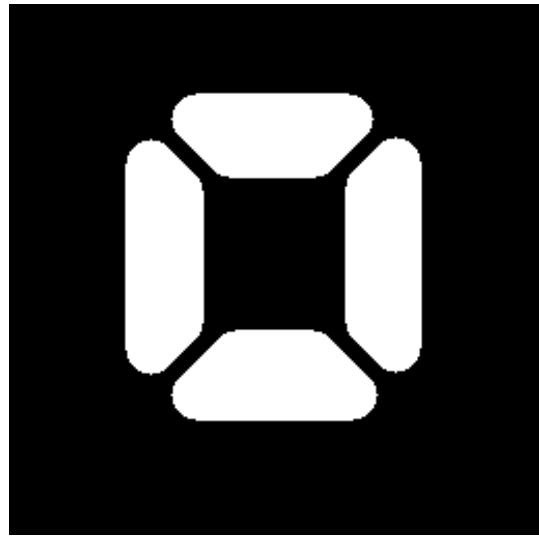
radius = 6



radius = 8



radius = 10



radius = 12

Figure 2-4: The resultant images after the open operation with different radius of the round kernel on Sample2.raw.

However, it seems to be difficult to observe the differences of the four images in Figure 2-4. Hence, I made the following animation, which reveals the unapparent differences when applying the open operation with different size of round kernels on Sample2.raw.

[https://www.csie.ntu.edu.tw/~b01902040/doc/DIP\\_hw4\\_Q2\\_Opening.gif](https://www.csie.ntu.edu.tw/~b01902040/doc/DIP_hw4_Q2_Opening.gif)

### **Problem 3: Texture Analysis**

Let's denote Sample3.raw as  $I$ . Please generate several images by the instructions below.

- (a) Transfer  $I$  to frequency domain by DFT (Discrete Fourier Transform) with centering and output the result as  $D$ .

The function DFT.m takes Sample3.raw, denoted as  $S3$ , as input and performs Discrete Fourier Transform with centering on it. The resultant frequency-domain image is denoted as  $D$ . To make  $D$  become a displayable Fourier spectrum here, first we need to compute the magnitude (simply apply the absolute operation) of  $D$ , then divide it by the maximum magnitude. Then, we use log transform  $\log_2(1 + D)$  to enhance the low-intensity pixels. Finally, we multiply  $D$  by 255 to map it back to the dynamic range that is comfortable for human eyes observation. Figure 3-1 is the resultant Fourier spectrum after 15 times of log transform.

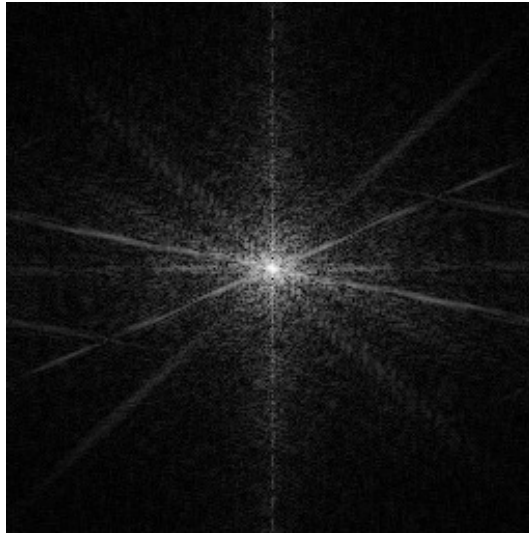
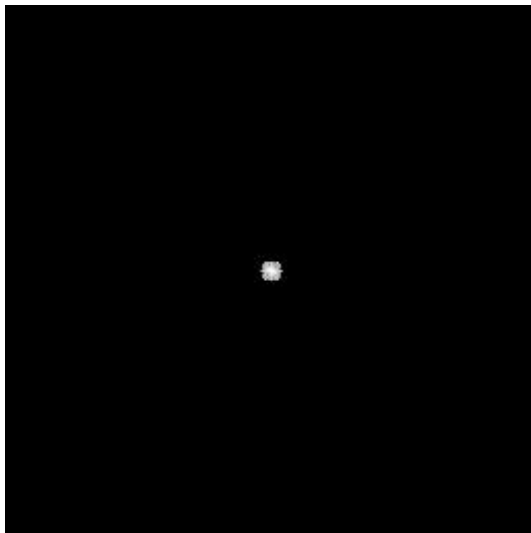


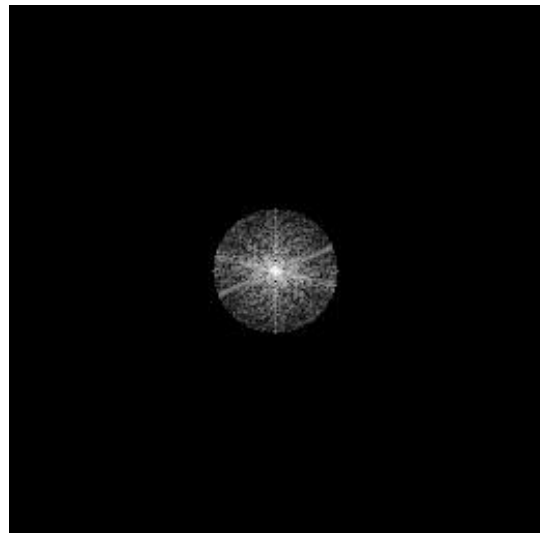
Figure 3-1: The Fourier spectrum of  $D$  after applying log transform for 15 times.

(b) Apply an ideal low-pass filter to  $D$  with  $D_0 = 5$  and 30. Output results as  $L_5$  and  $L_{30}$ , respectively.

Given  $D$  and a specified  $D_0$ , function `idealLowPass.m` performs the low-pass filtering. Figure 3-2 shows resultant  $L_5$  and  $L_{30}$ .



$L_5$

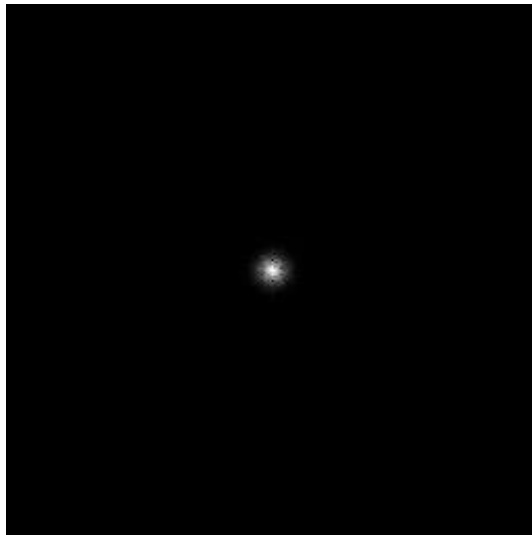


$L_{30}$

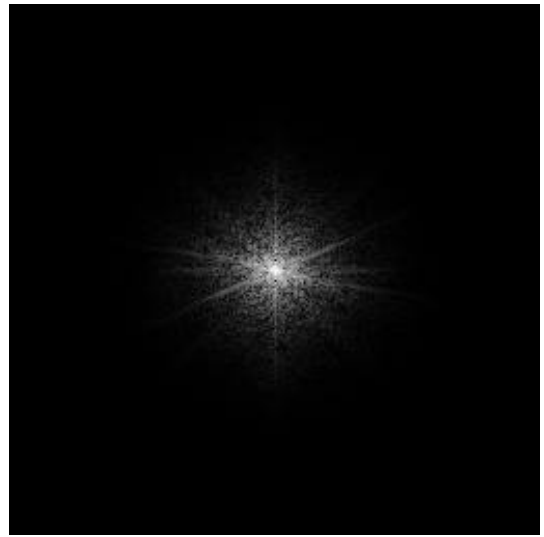
Figure 3-2: The Fourier spectrum of  $L_5$  and  $L_{30}$

(c) Apply a Gaussian low-pass filter to  $D$  with  $D_0 = 5$  and 30. Output results as  $G_5$  and  $G_{30}$ , respectively.

Given  $D$  and a specified  $D_0$ , function `gaussianLowPass.m` performs Gaussian low-pass filtering. Figure 3-3 shows resultant  $G_5$  and  $G_{30}$ .



$G_5$



$G_{30}$

Figure 3-3: The Fourier spectrum of  $G_5$  and  $G_{30}$

- (d) Transfer  $L_5$ ,  $L_{30}$ ,  $G_5$ , and  $G_{30}$  back to spatial domain by Inverse DFT. Please compare the results and provide some discussions in the report.

The function `invDFT.m` takes a frequency-domain image as input and convert it back to spatial domain. Figure 3-4 displays the results of converting  $D$ ,  $L_5$ ,  $L_{30}$ ,  $G_5$ , and  $G_{30}$  back to their corresponding spatial domain one by one.

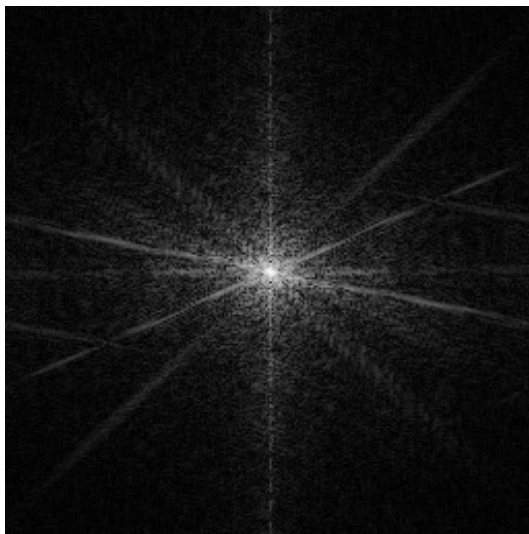


Figure 3-4-1: Inverse  $D$  back to spatial domain.

The inversed image is supposed to be the same as `Sample3.raw`



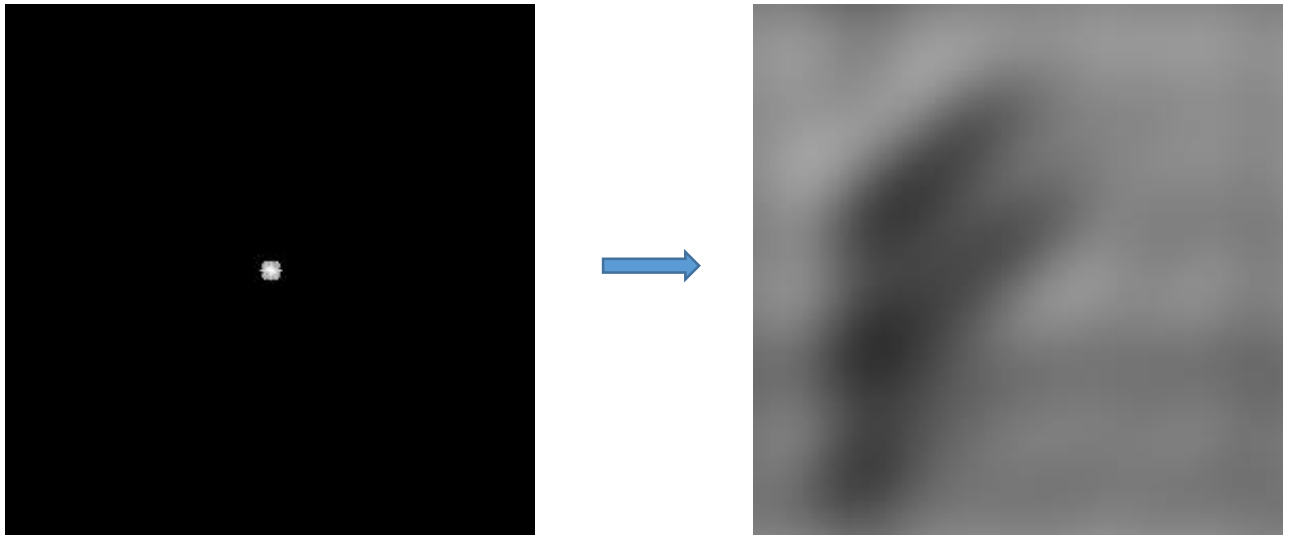


Figure 3-4-2: Inverse  $L_5$  back to spatial domain.

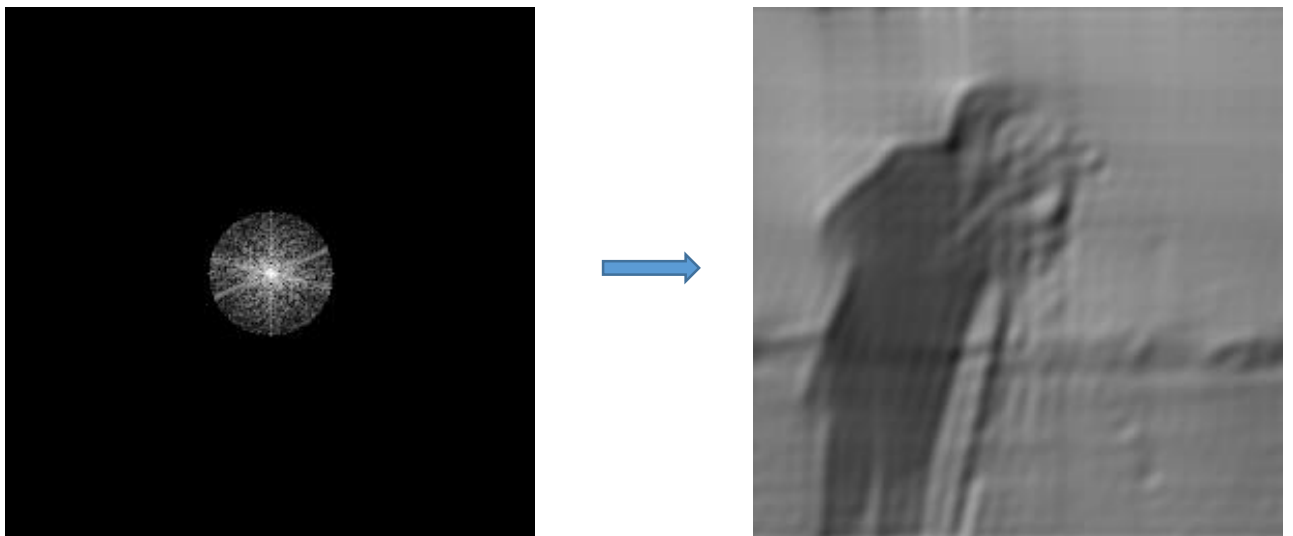


Figure 3-4-3: Inverse  $L_{30}$  back to spatial domain.

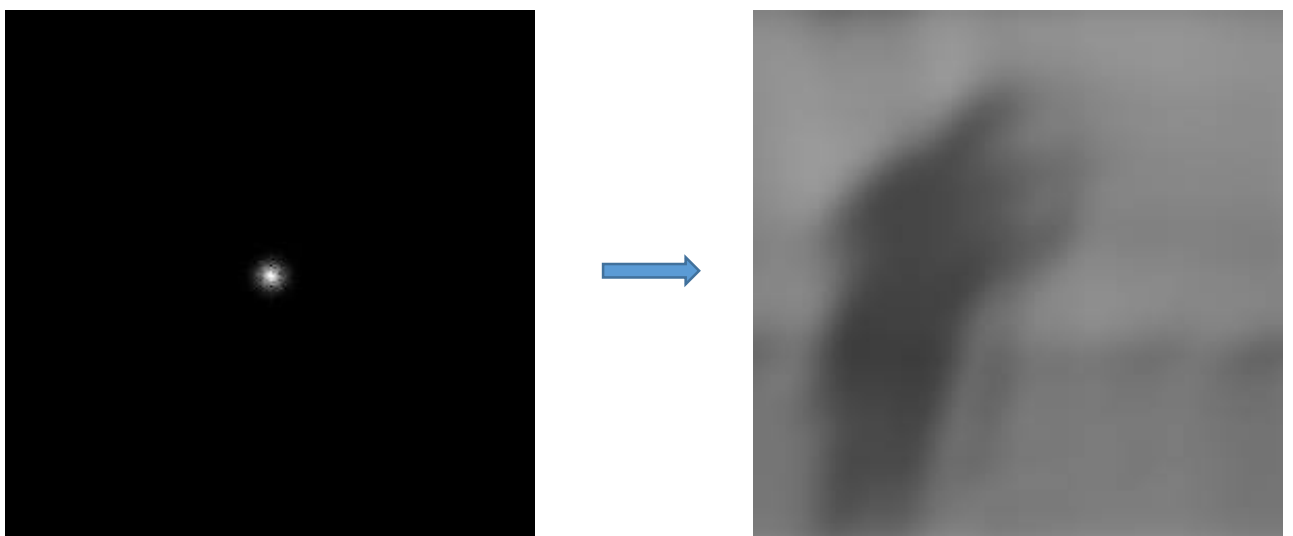


Figure 3-4-3: Inverse  $G_5$  back to spatial domain.

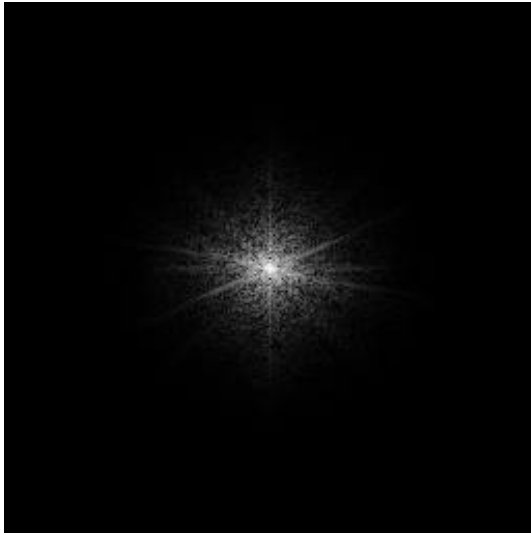


Figure 3-4-4: Inverse  $G_{30}$  back to spatial domain.