Assignment 2

COMP 478 Image Processing

Etienne Pham Do

40130483

Etienne Pham Do 40130483

Assignment 2

1.

Given that binary strings are broken strings of 1s with gaps of 0s for 1 to 5 pixels, a 5x5 kernel size would cover the largest gap of 0s that represents 5 pixels. Therefore, to ensure there are no gaps, the minimum kernel size is 5x5. For the threshold function to return a sharper version of the blurred image, the threshold value depends on the average value returned by the averaging filter. Its smallest average is found when a certain neighborhood of pixels from the blurred image contains only 2 pixels. This value r represents a gray level, which serves as an input for the thresholding function that should return a value s. For the blurred pixel to be sharp, r must be lower than s. Therefore, the threshold value must be set lower than r (or the smallest average given by the averaging mask).

2.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} => [1\ 1 - 4\ 1\ 1] + [1\ 1 - 4\ 1\ 1]^T = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -8 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} = \text{Let's call it A}$$

Then,

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & -8 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} = \text{Let's call it B}$$

⇨ A + B =

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & -16 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

This 5x5 mask must be built with uniform response in all directions: horizontal, vertical, and diagonal. The sum of all values in the filter must be equal to 0. This filter should yield a sharper image as there is more differentiation (more values in the matrix) in all directions unlike the aforementioned filters due to their smaller size.

3.

a)

$$F(\mu) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi\mu t}dt = \int_{0}^{W} Ae^{-j2\pi\mu t}dt$$

$$= \frac{-A}{j2\pi\mu}\left[e^{-j2\pi\mu t}\right]_0^W = \frac{-A}{j2\pi\mu}\left[e^{-j2\pi\mu W} - 1\right]$$

$$= \frac{-A}{j2\pi\mu}[\cos(2\pi\mu W) - jsin(2\pi\mu W) - 1]$$

Unlike the result from example 4.1, the main difference is that the result found from 0 ≤ t ≤ W contains an imaginary part as well as a cosine function that came from the identity: $e^{-j\theta} = \cos(\theta) - jsin(\theta)$

$$\int_0^1 e^{-j2\pi\mu t}dt$$

$$= \frac{-1}{j2\pi\mu}\left[e^{-j2\pi\mu t}\right]_0^1 = \frac{-1}{j2\pi\mu}\left[\ e^{-j2\pi\mu} - 1\right]$$

$$= \frac{-1}{j2\pi\mu}[\cos(2\pi\mu) - jsin(2\pi\mu) - 1]$$

b)

$$Fourier\big((f * g)(t)\big) = H(\mu)F(\mu)$$

$$= \left(\frac{AW}{\pi\mu W}[sin(\pi\mu W)]\right)^2 = \frac{A^2}{\pi^2\mu^2}[sin(\pi\mu W)]^2$$

<u>Programming</u>

```
import cv2

import numpy as np


#reading and blurring image using averaging filter of size 5x5

current_image = cv2.imread('Doc.tiff',0)

new_img = cv2.blur(current_image, (5,5))


cv2.imshow('image using avg filter',new_img)

cv2.waitKey(0);
```

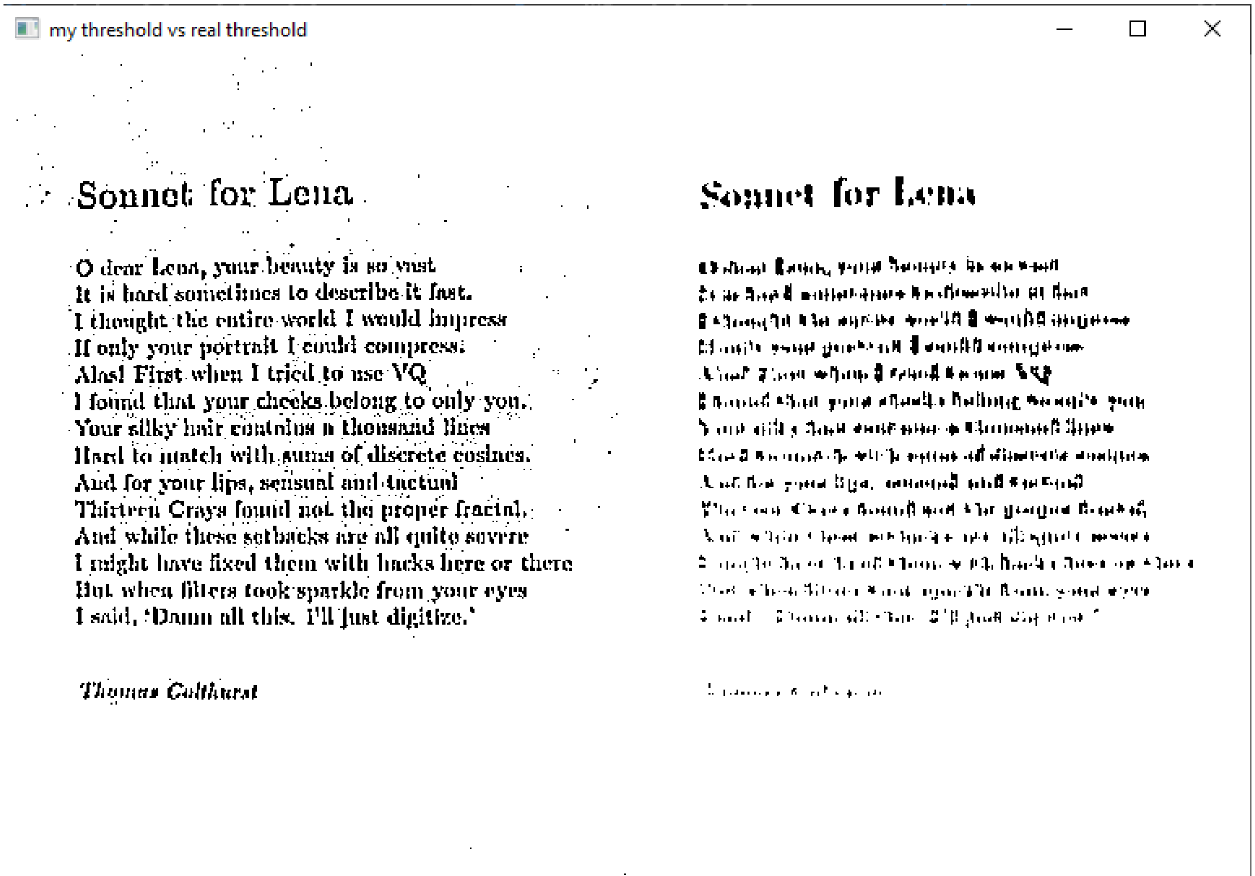
```
cv2.destroyAllWindows();
print('done showing image')

#threshold implementation and application
def get_threshold_value_on_pixel(c, weighted_avg):
    return weighted_avg - c

for col in range(len(new_img)):
    for row in range(len(new_img[col])):
        thresh_arr = get_threshold_value_on_pixel(3, new_img[col][row])
        new_img[col][row] = 0
        if current_image[col][row] > thresh_arr:
            new_img[col][row] = 255
print('threshold calculation done')

cv2.imshow('image using my threshold',new_img)
cv2.waitKey(0);
cv2.destroyAllWindows();
print('threshold image done')

#using built-in threshold function
img_2 = cv2.blur(current_image, (5,5))
img_from_builtin_adapt = cv2.adaptiveThreshold(img_2, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 5, 3)
#displaying the 2 images side-by-side
side_by_side_imgs = np.hstack((new_img, img_from_builtin_adapt))
cv2.imshow('my threshold vs real threshold',side_by_side_imgs)
cv2.waitKey(0);
cv2.destroyAllWindows()
```



Custom threshold: filter size = 5x5, parameter c = 3

Built-in threshold: filter size = 5x5, parameter c = 3

Reason: tested various values, but those 2 were close in terms of desirable results when comparing the output images side by side.

Result: the custom threshold implementation with the same values used by the built-in one seems to yield a sharper image, with a little bit a noise in the background, whereas the built-in has no noise but is a bit blurry.