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.

0 0 1 0 0

1 1 -8 1 1

0 0 1 0 0

0 0 1 0 0] = Let’s call it A

Then, [1 0 0 0 0 + [0 0 0 0 1 = [1 0 0 0 1 = Let’s call it B

0 1 0 0 0 0 0 0 1 0 0 1 0 1 0

0 0 -4 0 0 0 0 -4 0 0 0 0 -8 0 0

0 0 0 -1 0 0 1 0 0 0 0 1 0 1 0

0 0 0 0 1] 1 0 0 0 0] 1 0 0 0 1]

* A + B = [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]

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)

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:

b)

**Programming**

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)

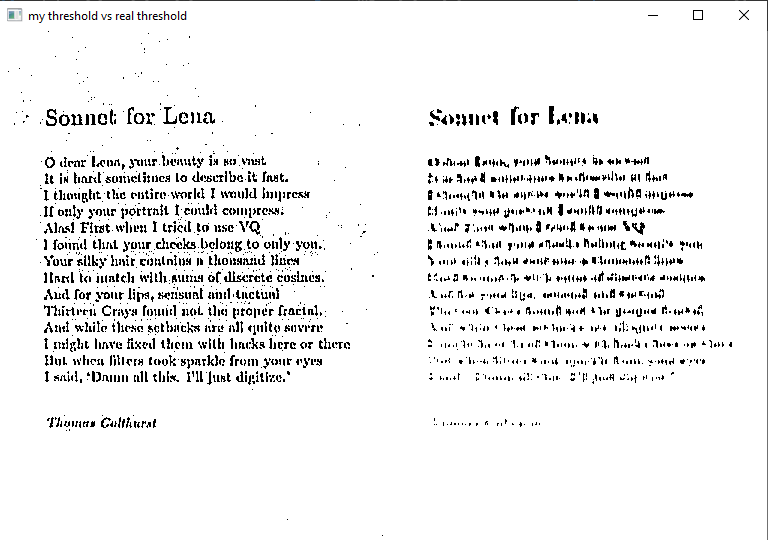
#displaying the 2 images side-by-side

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.