COMP 6771 Image Processing: Assignment 2

Student name: Yunqi Xu

Student id: 40130514

October 18, 2022

1. Question 1

   (a) The minimum size of the blur mask is a 7 * 7 filter. Since the averaging filter will calculate the average of the pixels contined in the neighborhood of the filter mask. No matter the coefficient value off the average filter is. If all pixels in the filter mask are zero, the output is zero. Based on the question, the gaps ranging from 1 to 5 pixels. So the minimum kernal size should greater than 5 to avoid the zero output. Also, the kernel size usually is odd number. So, 7 * 7 is the minimum size.

   (b) Based on the question, we want those ouput which the filter center located on the string or hole(and be filled) equal or greater than the threshold and keep those value. Suppose the center of the filter located at a center of a gap. In this case, the filter will have a minimum output that we want to convert it value. Also, the maximum size of gap is 5 pixels. If we want to use thresholding method to convert it back at the meanwhile keep string without small hole or noise, we need to guarantee that the minimum ouput greater than threshold. So, suppose we get S as the minumum useful output. We will set threshold closely less than S, but not less too much. To do this, we can keep string without small hole and abvoid to bring more noise during the threshold process.

2. Question 2

   The 5-by-5 Laplacian like filter has been shown in Eq. 1 below:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{1}$$

   The basic rule is the sum of all coefficient is 0.

   We will use $g(x, y) = f(x, y) - \nabla^2 f(x, y)$ to sharpen an image. Compared with filters in question, this one will output an image sharper result. Because, the center of the 5-by-5 filters have more weight than -4 or -8. So, the edge of the image will be further enhence.

3. Question2 Fourier Transform Queston

   (a) Based on the question, the Eq. 2 defines the variable t.

$$f(t) = \begin{cases} A & 0 \le t \le W \\ 0 & otherwase \end{cases} \tag{2}$$

$$
\begin{aligned}
F(\mu) &= \int_{-\infty}^{+\infty} f(t)e^{-j2\pi\mu t}dt \\
&= \int_{0}^{W} Ae^{-j2\pi\mu t}dt \\
&= \frac{-A}{j2\pi\mu}[e^{-j2\pi\mu t}]_0^W \\
&= \frac{-A}{j2\pi\mu}[e^{-j2\pi\mu W} - 1] \\
&= \frac{A}{j2\pi\mu}[1 - e^{-j2\pi\mu W}] \\
&= \frac{A}{j2\pi\mu}[e^{j\pi\mu W - e^{-j\pi\mu W}}]e^{-j\pi\mu W} \\
&= AW(\frac{sin(\pi\mu W)}{\pi\mu W})e^{-j\pi\mu W}
\end{aligned} \tag{3}
$$

Since $A = W = 1$, In the Example, the result is:

$$
F(\mu) = AW\frac{sin(\pi\mu W)}{\pi\mu W} = \frac{sin(\pi\mu)}{\pi\mu} \tag{4}
$$

And the result of Eq. 3 is:

$$
F(\mu) = \frac{sin(\pi\mu)}{\pi\mu}e^{-j\pi\mu} \tag{5}
$$

Compared with these two equation Eq. 4 and Eq. 5, the only different part is $e^{-j\pi\mu}$. So the case in this problem is a shifted part compared the resul of Example 4.1.

(b) Based on the property of Convolution:

$$
f(t) \star h(t) \Longleftrightarrow H(\mu)F(\mu)
$$

$$
f(t) = \begin{cases} A & \frac{-W}{2} \leq t \leq \frac{W}{2} \\ 0 & otherwase \end{cases} \tag{6}
$$

Based on the information provided by the right image in question, the Eq. 6 is the equation.

$$
\begin{aligned}
F(\mu) &= \int_{-\infty}^{+\infty} f(t)e^{-j2\pi\mu t}dt \\
&= \int_{\frac{-W}{2}}^{\frac{W}{2}} Ae^{-j2\pi\mu t}dt \\
&= \frac{-A}{j2\pi\mu}[e^{-j2\pi\mu t}]_{\frac{-W}{2}}^{\frac{W}{2}} \\
&= AW(\frac{sin(\pi\mu W)}{\pi\mu W})
\end{aligned} \tag{7}
$$

Based on the property of convolution and Eq. 7, the Fourier Transform of the tent function is:

$$\begin{aligned} F(\mu)F(\mu) &= A^2W^2\frac{sin^2(\pi\mu W)}{\pi^2\mu^2W^2} \\ &= A^2\frac{sin^2(\pi\mu W)}{\pi^2\mu^2} \end{aligned} \tag{8}$$

4. Question 3.

(a) The code has been shown below

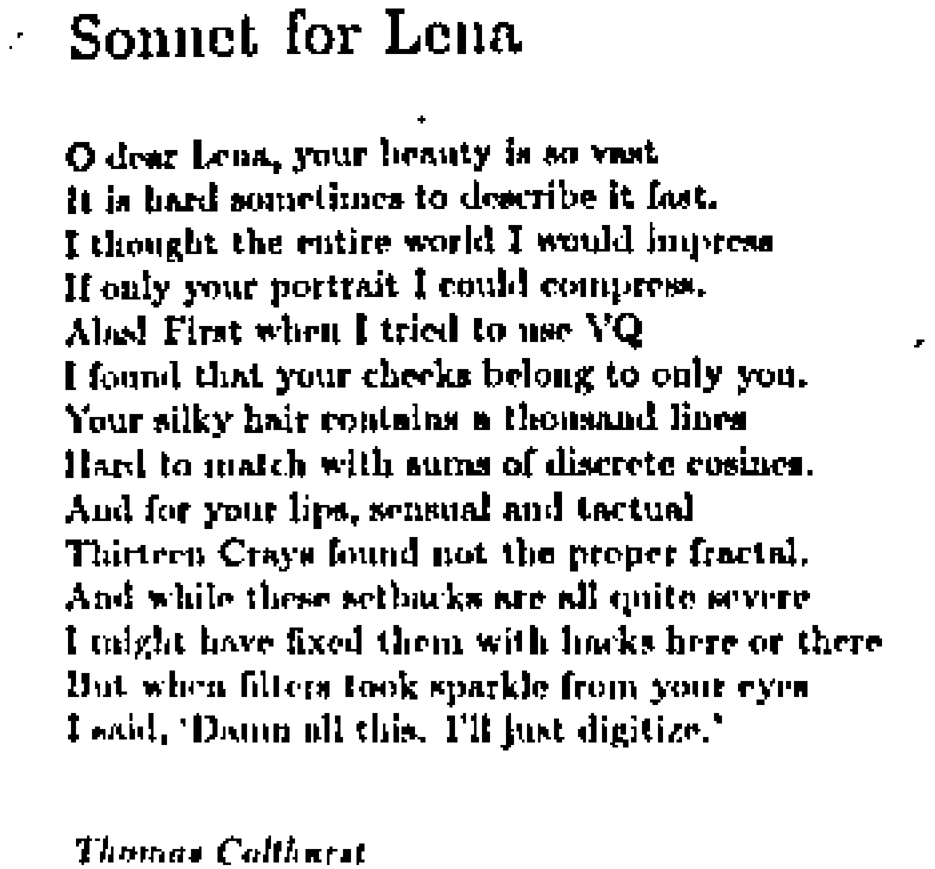
```
import numpy as np
import cv2


# read image
def readimg(path):
    return cv2.imread(path, 0)

# Generate a Gaussian filter
def GaussinFilter(sigma, size):
    m = (size - 1.) / 2.
    n = (size - 1.) / 2.
    y, x = np.ogrid[-m:m + 1, -n: n + 1]
    h = np.exp(-(x * x + y * y) / (2. * sigma * sigma))
    h[h < np.finfo(h.dtype).eps * h.max()] = 0
    all_sum = h.sum()
    if all_sum != 0:
        h /= all_sum
    return h



# Using gaussian to get the weighted aaverage.
def weightedSum(input_subimage, input_filter, filter_size):
    new_subimage = np.zeros((filter_size, filter_size))
    for y in range(filter_size):
        for x in range(filter_size):
            new_subimage[y, x] = input_subimage[y, x] * input_filter[y, x]
    return sum(sum(new_subimage))


def averagingFilter(input_image, input_filter, filter_size):
    image_height, image_width = input_image.shape[0], input_image.shape[1]
    output_image = np.zeros((image_height,image_width))
    padding_size = int((filter_size - 1) / 2)
    padding_short = np.zeros((padding_size, image_width))
    padding_long = np.zeros((image_height + (filter_size - 1), padding_size))
    padded_image = np.r_[padding_short, input_image]
    padded_image = np.r_[padded_image, padding_short]
    padded_image = np.c_[padding_long, padded_image]
    padded_image = np.c_[padded_image, padding_long]

    for y in range(padding_size, image_height + padding_size):
```

```
        for x in range(padding_size, image_width + padding_size):
            sub_image = padded_image[y - padding_size: y + (padding_size + 1), x - padding_size: x + (padding_size + 1)]
            output_image[y - padding_size, x - padding_size] = weightedSum(
                                            input_subimage=sub_image,
                                            input_filter=input_filter,
                                            filter_size=filter_size)
    return output_image



def calculateThreshold(input_image, C):
    image_height, image_width = input_image.shape
    output_image = np.zeros((image_height,image_width))
    for y in range(image_height):
        for x in range(image_width):
            output_image[y, x] = input_image[y, x] - C
    return output_image


# threshold the image
def threshold(input_image, max_value, threshold):
    output_image = np.zeros((input_image.shape[0], input_image.shape[1]))
    height, width = input_image.shape
    for y in range(height):
        for x in range(width):
            if input_image[y, x] >= threshold[y, x]:
                output_image[y, x] = max_value
            else:
                output_image[y, x] = 0
    return output_image


# The overall Adaptive thresholding
def adaptiveThresholding(input_image,
        max_value, adaptive_method,
                threshold_type, filter_size,
                 C, Gaussian_sigma=1.4):
    if adaptive_method == "Gaussian":
        filter = GaussinFilter(sigma=Gaussian_sigma,
                            size=filter_size)

    wa_image = averagingFilter(input_image=input_image, input_filter=filter, filter_size=filter_size)


    if threshold_type == "THRESH_BINARY":
        t_image = calculateThreshold(input_image=wa_image, C=C)

    threshold_image = threshold(input_image=input_image, max_value=max_value, threshold=t_image)

    output_image = np.array(threshold_image, dtype='uint8')

    return output_image

```

```
image = readimg("Doc.tiff")
output_image = adaptiveThresholding(
        input_image=image,
        max_value=255,
        adaptive_method="Gaussian",
        threshold_type="THRESH_BINARY",
        filter_size=9,
        C = 4.5,
        Gaussian_sigma=2)
cv2.imwrite('output.png', output_image)
```

(b) In this question, we choose the Gaussian filter as the coefficient when calculateng the weighted average. The reason is that, Mean filter is good at reducing random noise, but in our image, we want to split the foreground and background. And Gaussian filter can achieve a better result compared with Mean filter when dealing with the problem of spliting two different frequencies.

Also, The filter_size we choose is 11, since the letter in image are around 11 pixels for each of them. and the Gaussian_sigma = 3. Usualy, a higher gussian sigma will get a gaussian filter that can bring more neighborhood inforamtion to the center.

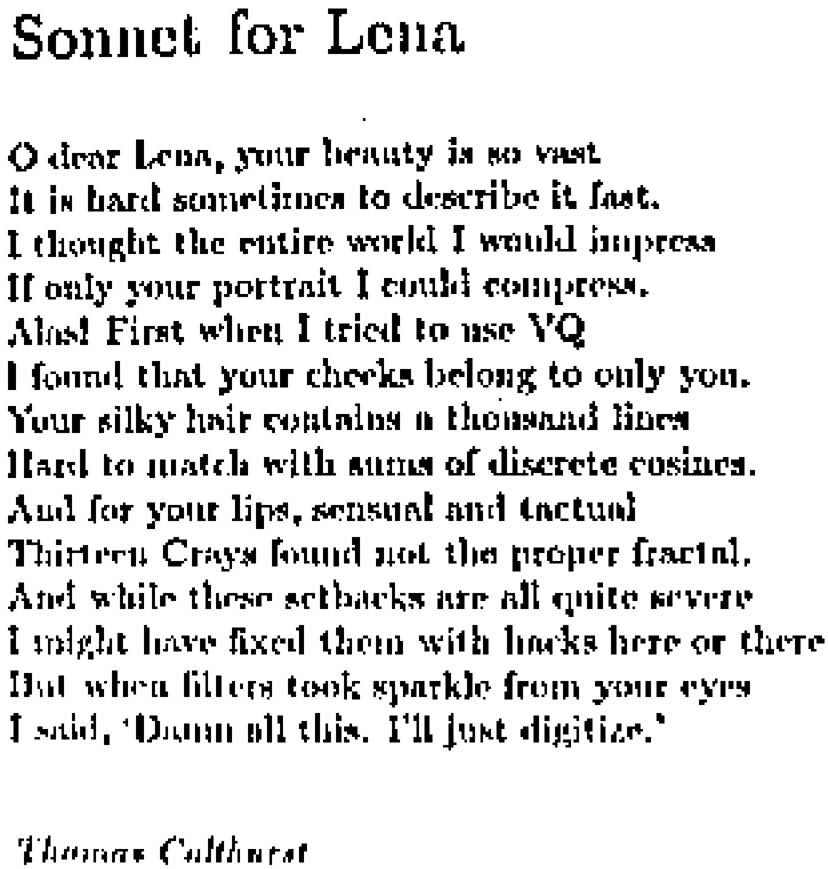


Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure 1: The output image of Python implement


Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure 2: The output image from open cv2.adaptiveThreshold

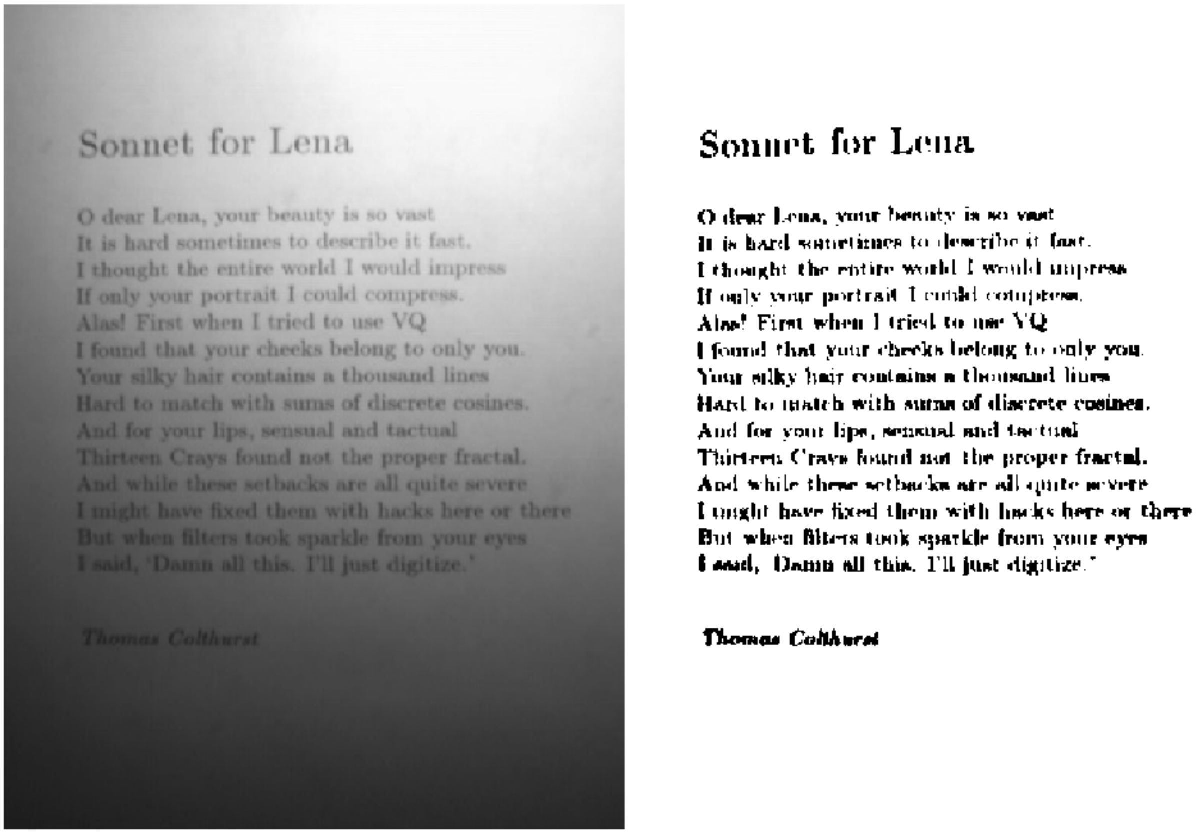


Figure 3: The output image from Matlab adaptthresh

The Fig. 1 is the output by using the code above. And the Fig. 2 is the output from the method which is provided by OpenCV cv2.adaptiveThreshold() and we use the same parameters as we used in the implemented code. The Fig. 3 is the output image from the Matlab adaptthresh() with the same parameters. All of these three function can separate the input image from the dark backgroud. The implemented code and the function from openCV can achieve a slightly better result may because there are more parameters that could be modified such as the sigma of gaussian.