COMP 478 DD

Piyush Pokharkar

40120654

Assignment 2

Theoretical Questions

1.a)

The box filter is the average of all the pixels in a neighborhood. In image (b), since the filter size was 25 pixels in width and the width of vertical bars is 5 pixels and space between them is 20 pixels, the average that the filter returns will always be the same value. Because 5 of the 25 columns will always have black pixels regardless of where it is moved horizontally. Therefore the average will always be the same, so we won't see any separation. The relative size of the other kernels doesn't matter because that affects on how smooth the image becomes. The frequency of image(b) does not accurately represent the frequency of the original image as it skips over every other one, showing no separation.

1.b)

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

General Rules:

The sum of the filter coefficients should be 0. The center is chosen to be a high value (usually negative) and nearby pixels are lower positive coefficients. The filter is symmetrical.

Compared to a real 3x3 Laplacian filter, this filter will result in sharper edges because the larger filter size and higher value at negative center focuses more high frequency components as there is a bigger contrast, resulting in sharper edges.

2.a.i)

$$D.F.T\{f(x,y)\} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{j2\pi\left(\frac{u_0 x}{M}+\frac{v_0 y}{N}\right)}e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{u_0 x}{M}+\frac{v_0 y}{N}\right)}e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}-\frac{u_0 x}{M}-\frac{v_0 y}{N}\right)}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{j2\pi\left(\frac{(u-u_0)x}{M}+\frac{(v-v_0)y}{N}\right)}$$

$$= F(u-u_0, v-v_0)$$

2.a.ii)

$$D.F.T.\{f(x-x_0, y-y_0\} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x-x_0, y-y_0)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{u_0 x}{M}+\frac{v_0 y}{N}\right)}$$

$$F(u,v) = e^{e^{-j2\pi\left(\frac{u_0 x}{M}+\frac{v_0 y}{N}\right)}}$$

2.b)

Fourier transform: $F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt$

Plug in f(t):

$$F(\omega) = \int_0^W 2e^{-j\omega t}dt$$

$$F(\omega) = 2\left|\frac{-e^{-j\omega t}}{j\omega}\right|_0^W$$

$$F(\omega) = 2\left[\frac{-e^{-j\omega W}}{j\omega} - \frac{-e^0}{j\omega}\right]$$

$$F(\omega) = 2\left(\frac{1-e^{-j\omega W}}{j\omega}\right)$$

Programming Questions (python)

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from scipy.ndimage import gaussian_filter



# Read images in gray scale
text = cv2.imread('text.png', cv2.IMREAD_GRAYSCALE)
H04 = cv2.imread('H04.bmp', cv2.IMREAD_GRAYSCALE)
Doc = cv2.imread('Doc.tiff', cv2.IMREAD_GRAYSCALE)

# Apply adaptive thresholding
def adaptive_thresholding(image, m, c):
    # Output image to store the thresholded result
    output_image = np.zeros_like(image)

    height, width = image.shape

    # Loop through  image
    for x in range(height):
        for y in range(width):
            # Calculate starting and ending indices for row and column based on given filter size (m)
            x1 = max(0, x - m // 2)
            x2 = min(height, x + m // 2)
            y1 = max(0, y - m // 2)
            y2 = min(width, y + m // 2)
```

```
            # Create filter with given size
            filter = image[x1:x2, y1:y2]

            # Weighted averaging filter (box filter)
            WA = np.mean(filter)

            # Calculate the threshold T(x, y)
            T = WA - c

            # Apply threshold transfer function
            output_image[x, y] = 255 if image[x, y] > T else 0

    return output_image


# Perform adaptive thresholding
at_text = adaptive_thresholding(text, 20, 6)
at_H04 = adaptive_thresholding(H04, 25, 25)
at_Doc = adaptive_thresholding(Doc, 8, 3)

# Plot
plt.subplot(2, 3, 1)
plt.imshow(at_text, cmap=plt.cm.gray)
plt.title('Text')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(at_H04, cmap=plt.cm.gray)
plt.title('H04')
```

```
plt.axis('off')


plt.subplot(2, 3, 3)
plt.imshow(at_Doc, cmap=plt.cm.gray)
plt.title('Doc')
plt.axis('off')


## Compare results with adaptthresh() function (from OpenCV)
# Perform adaptive thresholding (using adaptthresh equivalent)
adaptthresh_text = cv2.adaptiveThreshold(text, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 199, 5)
adaptthresh_H04 = cv2.adaptiveThreshold(H04, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 199, 5)
adaptthresh_Doc = cv2.adaptiveThreshold(Doc, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 199, 5)


# Plot
plt.subplot(2, 3, 4)
plt.imshow(adaptthresh_text, cmap=plt.cm.gray)
plt.title('Text (adaptthresh)')
plt.axis('off')


plt.subplot(2, 3, 5)
plt.imshow(adaptthresh_H04, cmap=plt.cm.gray)
plt.title('H04 (adaptthresh)')
plt.axis('off')


plt.subplot(2, 3, 6)
plt.imshow(adaptthresh_Doc, cmap=plt.cm.gray)
plt.title('Doc (adaptthresh)')
```

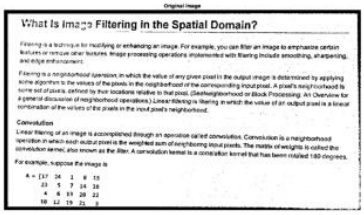
plt.axis('off')

plt.show()

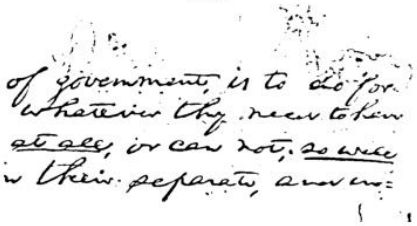
I used a box filter, because it was not necessary to use something like Gaussian to remove noise and make it smoother, but rather have more accurate filter to have clear and readable text.

When I applied adaptive thresholding, the images had way less shadows and stains, compared to using adaptiveThreshold from OpenCV, even after tweaking with the parameters. I also used mean filter in both cases to keep it consistent.

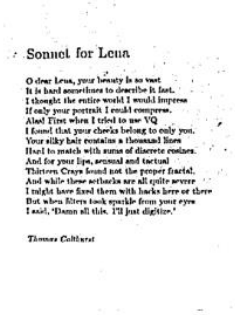
Text



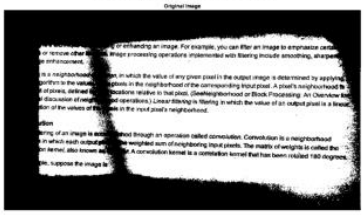
H04



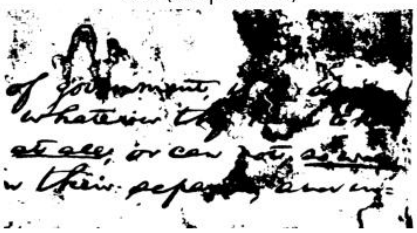
Doc



Text (adaptthresh)



H04 (adaptthresh)



Doc (adaptthresh)