# 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)

# 2.a.ii)

# 2.b)

Fourier transform:

Plug in f(t):

# 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.

