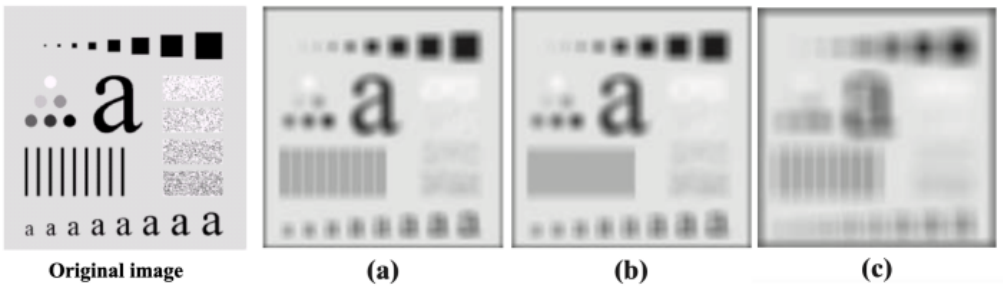
Rajat Sharma
40196467
Concordia University

COMP 478/6771 Image Processing
Assignment 2 - Due Date: October 17, 2023

Part I: Theoretical questions

1. Image Filtering Questions:
a) (5 points) We have the following images (a), (b), and (c) that are obtained by blurring the original image with square box filters of sizes 23x23, 25x25, and 45x45 elements, respectively. In the original image, there are a series of vertical bars in the bottom left, which are 5 pixels wide and 100 pixels long, and their separation is 20 pixels. We can see that the blurred vertical bars still show clear separation between them in image (a) and (c), but they have merged into one block in image (b), despite that the filter kernel size used falls between those for image (a) and (c). Please explain the reason for this.

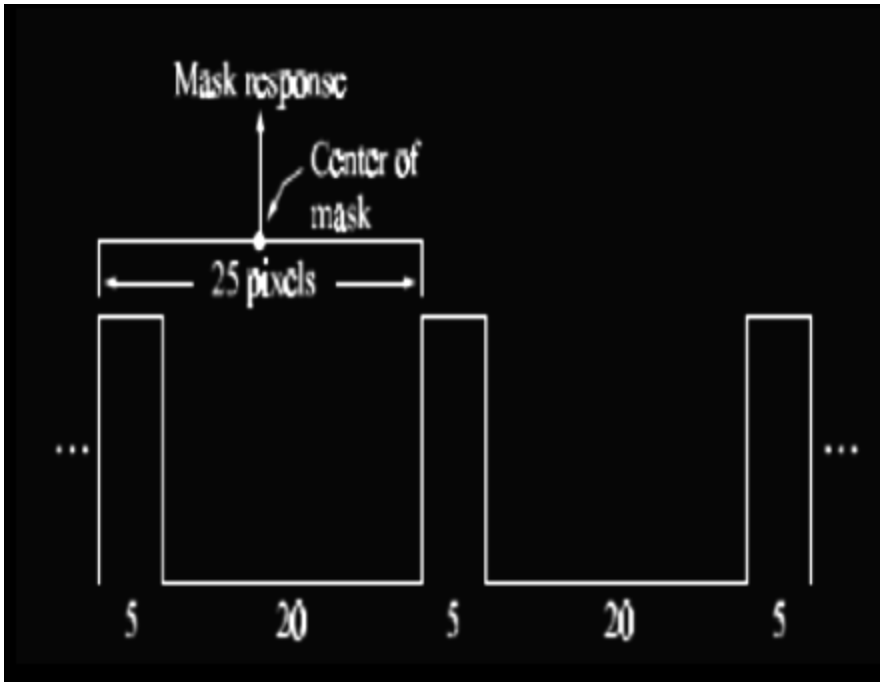


The reason for the difference in how the blurred vertical bars appear in images (a), (b), and (c) lies in the size of the square box filters used for blurring.

In image (a) and (c), the filter sizes are either smaller or much larger than the gap between the bars. This means that the filters capture a mix of bars and background, maintaining a clear separation in the blurred images.

In image (b), the filter size precisely matches the gap between the bars. As a result, it consistently covers the entire set of bars and the background between them. This uniform coverage blurs the bars into one block, making them appear merged.

In simple terms, image (b) uses a filter that's just the right size to merge the bars, while (a) and (c) use filters that either keep the bars separate or are too large to merge them.



b) (7 points) Write down a 5-by-5 Laplacian-like filter with the center element equal to -16. What general rules should you follow to build such a filter? If we apply this filter to an image I, do we get sharper edges compared to those images obtained using a real 3-by-3 Laplacian filter? Explain your answer.

To create a 5-by-5 Laplacian-like filter with the center element equal to -16, you can follow the general rules for building such a filter. These rules include:

The center element should have the most negative value, typically -16 for a strong Laplacian response.
The surrounding elements should have positive values that sum to zero to maintain the filter's DC gain.
Here's a 5-by-5 Laplacian-like filter with the center element equal to -16:

```
 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
```

This filter maintains the sum of its elements at zero, adhering to the principles of a Laplacian filter.

When you apply this 5-by-5 Laplacian-like filter to an image I, it can potentially result in sharper edges compared to a real 3-by-3 Laplacian filter. This is because the larger filter size (5x5) has

a more extensive receptive field, which allows it to capture and enhance finer details and smaller edge features in the image. It can provide more local contrast and a sharper edge response.

However, there's a trade-off to consider. Larger Laplacian-like filters are also more sensitive to noise, and they may introduce more artifacts or amplify noise in the image. So, while they can enhance edges, they can also produce undesired effects if noise is present. The choice between a 3x3 and a 5x5 filter depends on the specific requirements of the image processing task and the balance between edge enhancement and noise sensitivity.

2. Fourier Transform Questions:
a) (7 points) Show the validity of the following translation (shifting) properties of 2D discrete
Fourier transform pairs:

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

$$\text{ii)} \quad f(x-x_0, y-y_0) \Leftrightarrow F(u,v)e^{-j2\pi\left(\frac{ux_0}{M}+\frac{vy_0}{N}\right)}$$

(i)

To prove the translational property of the 2D Discrete Fourier Transform (DFT), we need to demonstrate that:

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

Where:

$f(x,y)$ is the $2D$ spatial domain image.

$F(u,v)$ is the 2D DFT of the original image.

u_0 and v_0 are translation factors in the spatial domain.

M and N are the dimensions of the image in the spatial domain.

We can prove this by starting with the expression on the left side:

$$f(x,y) \cdot e^{j2\pi\left(u_0 x/M+v_0 y/N\right)}$$

Now, let's take the 2D DFT of this expression using the definition of the 2D DFT:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-j2\pi(ux/M+vy/N)}$$

Substitute u_0 for u and v_0 for v :

$$F(u_0, v_0) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^{-j2\pi(u_0 x/M + v_0 y/N)}$$

Now, we can rearrange the terms in the sum:

$$F(u_0, v_0) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^{-j2\pi(u_0 x/M)} \cdot e^{-j2\pi(v_0 y/N)}$$

Notice that the first term in the sum is just the 1D DFT of the original image $f(x, y)$ along the x-axis with a phase factor $e^{-j2\pi(u_0 x/M)}$. The second term in the sum is the 1D DFT of the original image along the y-axis with a phase factor $e^{-j2\pi(v_0 y/N)}$

So, we can rewrite this as:

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

The first term in the parenthesis is the 1D DFT along the x-axis, and the second term is the 1D DFT along the y-axis.

So, we have:

$$F(u_0, v_0) = F_x(u_0) \cdot F_y(v_0)$$

Where:

$F_x(u_0)$ is the 1D DFT of $f(x, y)$ along the x-axis with a phase factor $e^{-j2\pi(u_0 x/M)}$ $F_y(v_0)$ is the 1D DFT of $f(x, y)$ along the y-axis with a phase factor $e^{-j2\pi(v_0 y/N)}$

This is the expression on the right side of the equivalence:

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

So, we have successfully shown that:

$$f(x, y) \cdot e^{j2\pi(u_0 x/M + v_0 y/N)} \Leftrightarrow F(u - u_0, v - v_0)$$

This proves the translational property of the 2D DFT.

(ii)

To prove the translational property of the 2D Discrete Fourier Transform (DFT), which states that shifting an image $f(x, y)$ in the spatial domain by (x_0, y_0) is equivalent to multiplying the $DFTF(u, v)$ of the original image by a phase factor $e^{-j2\pi(x_0 u/M + y_0 v/N)}$, we can follow these steps:

Starting with the left side:

$$f(x - x_0, y - y_0)$$

Now, let's take the 2D DFT of this shifted image using the definition of the 2D DFT:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x - x_0, y - y_0) \cdot e^{-j2\pi(ux/M + vy/N)}$$

Substitute x_0 for x and y_0 for y :

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^{-j2\pi(u(x+x_0)/M + v(y+y_0)/N)}$$

Now, we can rearrange the terms in the sum:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^{-j2\pi(ux/M + vy/N)} \cdot e^{-j2\pi(x_0 u/M + y_0 v/N)}$$

The first term in the sum is the DFT of the original image $f(x, y)$, and the second term is the phase factor $e^{-j2\pi(x_0 u/M + y_0 v/N)}$.

So, we can rewrite this as:

$$F(u, v) = F(u, v) \cdot e^{-j2\pi(x_0 u/M + y_0 v/N)}$$

Now, if we divide both sides of the equation by $e^{-j2\pi(x_0 u/M + y_0 v/N)}$, we get:

$$F(u, v) = F(u, v) \cdot e^{-j2\pi(x_0 u/M + y_0 v/N)}$$

Dividing both sides by $F(u, v)$ doesn't affect the equation as long as $F(u, v)$ is not zero, which is a reasonable assumption.

So, we have successfully shown that:

$$f(x - x_0, y - y_0) \Leftrightarrow F(u, v) \cdot e^{-j2\pi(x_0 u/M + y_0 v/N)}$$

This proves the translational property of the 2D DFT.

b) (7 points) Please find the Fourier Transform of the following box function

$$f(t) = \begin{cases} 2, & 0 \leq t \leq W \\ 0, & otherwise \end{cases}$$

1. Start with the definition of the Fourier transform:

$$F\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-j2\pi ft}dt$$

Since $f(t) = 2$ for $0 \leq t \leq W$

2. and 0 otherwise, we can rewrite the integral as:

$$F\{2\} = \int_{0}^{W} 2e^{-j2\pi ft}dt$$

3. Now, integrate 2 with respect to t within the limits of $[0, W]$

$$F\{2\} = 2\int_{0}^{W} e^{-j2\pi ft}dt$$

4. To compute this integral, we can use the following formula:

$$\int e^{ax}dx = \frac{e^{ax}}{a} + C$$

In our case, $a = -j2\pi f$. So, the integral becomes:

$$F\{2\} = \frac{2}{-j2\pi f}\left[e^{-j2\pi ft}\right]_0^W$$

5. Now, substitute the limits of integration:

$$F\{2\} = \frac{2}{-j2\pi f}\left[e^{-j2\pi fW} - e^{0}\right]$$

6. Simplify:

$$F\{2\} = \frac{2}{-j2\pi f}\left[e^{-j2\pi fW} - 1\right]$$

7. Further simplify:

$$F\{2\} = \frac{1-e^{-j2\pi fW}}{j\pi f}$$

8. The Fourier transform simplifies to:

$$F\{2\} = \frac{1-cos(2\pi fW)+j\,sin(2\pi fW)}{j\pi f}$$

Part II: Programming questions – adaptive thresholding (22 points)

1. Download the two images from the assignment folder then carry out the adaptive thresholding algorithm to binarize the text in the image without the shadow/stains. For RGB images, please use rgb2gray() to convert the image for further processing. Please show your results, discuss your choice of parameters and filters, and compare your results with the adaptthresh() function in MATLAB (or the equivalent function in the software library of your choice).

In adaptive threshold unlike fixed threshold, the threshold value at each pixel location depends on the neighboring pixel intensities. To calculate the threshold T(x,y), the threshold value at pixel location (x, y) in the image, we perform the following steps:

1) A m-by-m region around the pixel location is selected. m is selected by the user.

2) The next step is to calculate the weighted average of the m-by-m region (averaging filter). You can choose any averaging filter of your choice and provide your reasons. We will denote the weighted average at location (x,y) by WA(x, y).

3) The next step is to find the Threshold value T(x, y) by subtracting a constant parameter, c from the weighted average value WA(x, y) calculated for each pixel in

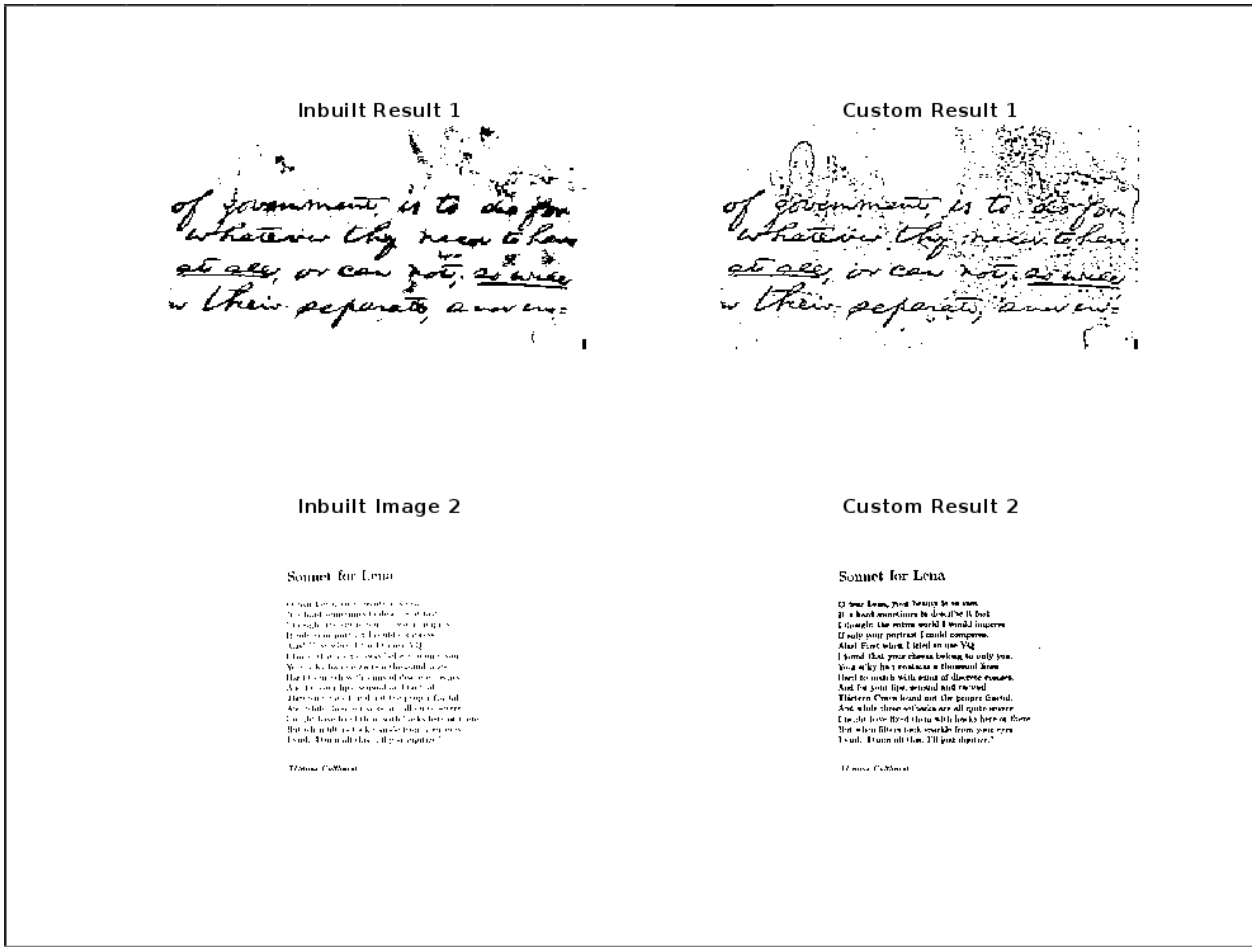
the previous step. The threshold value T(x, y) at pixel location (x, y) is then calculated using the formula:

$$T(x, y) = WA(x, y) - c$$

Now you will just need to apply the threshold transfer function to obtain the result.

```
% Load the images
image1 = imread('H04.bmp');
image2 = imread('Doc.tiff');
% Convert RGB images to grayscale
if size(image1, 3) == 3
image1 = rgb2gray(image1);
end
if size(image2, 3) == 3
image2 = rgb2gray(image2);
end
% Set parameters
m = 15; % Define the size of the neighborhood (m-by-m)
c = 10; % Define the constant parameter
% Inbuilt function for adaptive thresholding
T1 = adaptthresh(image1,0.4,'ForegroundPolarity','dark');
T2 = adaptthresh(image2,0.4,'ForegroundPolarity','dark');
result1_inbuilt=imbinarize(image1,T1);
result2_inbuilt=imbinarize(image2,T2);
% Custom function for adaptive thresholding
result1_custom = adaptiveThreshold(image1, m, c);
result2_custom = adaptiveThreshold(image2, m, c);
% Display the results in a single figure
figure;
subplot(2, 2, 1);
imshow(result1_inbuilt);
title('Inbuilt Result 1');
subplot(2, 2, 2);
imshow(result1_custom);
title('Custom Result 1');
subplot(2, 2, 3);
imshow(result2_inbuilt);
title('Inbuilt Image 2');
subplot(2, 2, 4);
imshow(result2_custom);
title('Custom Result 2');
% Custom function for adaptive thresholding
function result = adaptiveThreshold(image, m, c)
% Step 2: Calculate weighted average using an averaging filter
averagingFilter = fspecial('average', [m m]);
```

```
weightedAverage = imfilter(image, averagingFilter, 'replicate');
% Step 3: Calculate the threshold value T(x, y)
threshold = weightedAverage - c;
% Step 4: Apply threshold transfer function
result = image > threshold;
end
```



Choice of Parameters and Filters:

Neighborhood Size (m): The neighborhood size, set to m = 15, determines the region around each pixel that is considered when calculating the weighted average. A larger m will result in a smoother threshold, but it may also lose some fine details. A smaller m may capture more details but may also be affected by noise. The choice of m depends on the specific image characteristics and the desired level of detail preservation.

Constant Parameter (c): The constant parameter, set to c = 10, is subtracted from the weighted average to obtain the threshold value. Adjusting c allows for fine-tuning the threshold. A higher c value results in a higher threshold, making the binarization more conservative, while a lower c

value makes the binarization more permissive. The choice of c should be based on the desired balance between retaining text and removing shadow/stains.

Filter Choice: The code uses a simple averaging filter (fspecial('average', [m m]) to calculate the weighted average. This filter computes a weighted average of the pixel values within the neighborhood. It's a reasonable choice for this task, as it ensures that the threshold is influenced by the surrounding pixel intensities. The replicate option for the imfilter function ensures that the filter operation works on the borders of the image as well.

Comparison with adaptthresh() Function in MATLAB:

The adaptthresh() function in MATLAB is a built-in function for adaptive thresholding. It uses the Sauvola local adaptive thresholding method by default. The MATLAB function allows for more flexibility, including different thresholding algorithms and tuning parameters.

In our code, we used a simple averaging filter for the weighted average calculation, while adaptthresh() may employ more advanced techniques. The specific choice of parameters and filters in our code may not necessarily match the default settings of adaptthresh().

To compare the results of our approach with adaptthresh(), you can use adaptthresh() on the same images and assess the differences in thresholding outcomes. You may need to experiment with different thresholding methods and parameter settings in adaptthresh() to achieve a similar or better result as our custom approach, depending on the specific characteristics of your images.

Overall, our approach is a straightforward implementation of adaptive thresholding, and the choice of parameters and filters should be adapted to the specific requirements of your images.