Digital Image Processing Chapter 4: Lab 1

Olaitan Oluwadare

February 27, 2025

1 Question 1: What did you expect to observe based on the convolution theorem?

The **Convolution Theorem** states that convolution in the **time domain** corresponds to **multiplication in the frequency domain**:

$$\mathcal{F}\{f(t) * g(t)\} = F(\omega) \cdot G(\omega)$$

where:

- $f(t) * g(t)$ is the **convolution** in the time domain.
- $F(\omega)$ and $G(\omega)$ are the **Fourier Transforms** of $f(t)$ and $g(t)$.
- **Multiplication in the frequency domain is equivalent to convolution in the time domain.**

This report presents an experimental verification of the Convolution Theorem using MATLAB by comparing:

1. Direct **convolution in the time domain** using ‘conv(f, g)‘.
2. **Multiplication in the frequency domain** followed by the inverse Fourier transform (‘ifft(F .* G)‘).

2 Expected Observations

Based on the Convolution Theorem, we expect to observe:

1. **Time-Domain Effect:** - The convolution operation should **smooth** the original signal $f(t)$. - Since $g(t)$ is a **box function (moving average filter)**, we expect a **low-pass filtering effect**. - The output $fgc(t)$ should be a **smoothed** version of $f(t)$, reducing rapid oscillations.

2. **Frequency-Domain Effect:** - The **Fourier Transform of ‘g(t)‘** (a box function) is a **sinc function**. - The **FFT of ‘f(t)‘** contains two frequency peaks due to modulation. - After **multiplication ‘FG = F .* G‘**, the sinc function suppresses high frequencies. - **FFT of the convolved signal

should show reduced high-frequency components**, confirming the smoothing effect.

3. **Verification of Convolution Theorem:** - The results from **direct convolution (‘conv‘)** and **FFT-based multiplication (‘ifft(F .* G)‘)** should be **identical**. - The final comparison of both methods should yield a **numerical error close to zero**, confirming their equivalence.

3 MATLAB Implementation

The MATLAB script below shows an example of how i implements and visualizes the Convolution Theorem (This is a sample script and does not show the full steps, kindly run through the .m file attached with this report to get the full script)

```
% Define time and signals
t = 0:0.05:10;
f = sin(pi*t) .* sin(3*pi*t); % Modulated sinusoid
g = ones(1,15); % Box function (moving average filter)

% Perform Convolution in Time Domain
fgc = conv(f, g, 'same');

% Compute Fourier Transforms
N = length(f) + length(g); % Define zero-padded FFT length
F = fft(f, N);
G = fft(g, N);
FG = F .* G; % Convolution Theorem (Multiplication in Frequency Domain)
fgc2 = ifft(FG); % Inverse FFT to get time-domain convolution

% Shifted FFT for Visualization
F_shift = fftshift(F);
G_shift = fftshift(G);
FG_shift = fftshift(FG);

% Frequency Axis
fs = 1 / (t(2) - t(1)); % Sampling frequency
f_axis = linspace(-fs/2, fs/2, N);

.....

% Compute Error Between Methods
error_value = sum(abs(fgc(1:length(t)) - real(fgc2(1:length(t)))));
disp(['Error between time and frequency domain convolution: ', num2str(error_value)]);
```

4 Results and Discussion

The results are as follows:

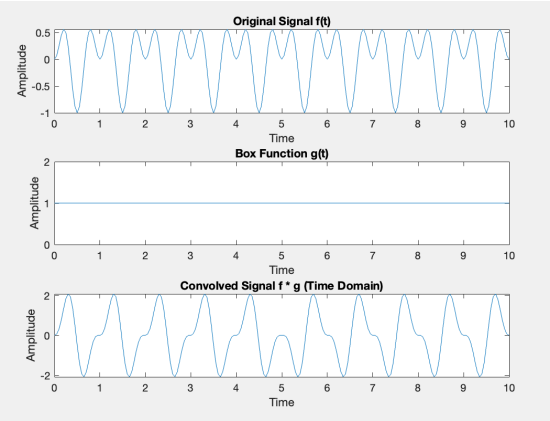

Figure 1: Convolution in Time Domain

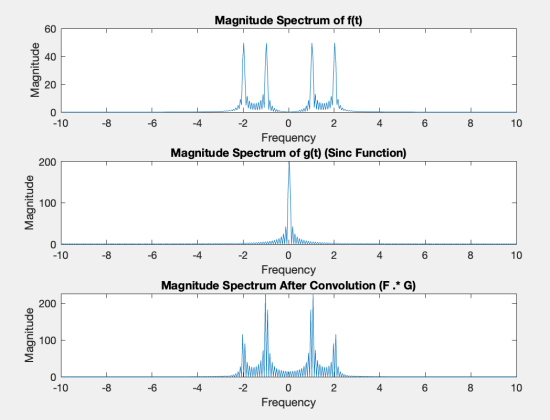

Figure 2: Comparison of Time-Domain and Frequency-Domain Convolution

The key observations include the following:

- The convolution operation in the time domain resulted in a smoothed signal.
- The Fourier Transform of the box function $g(t)$ is a sinc function, which acts as a low-pass filter.
- The magnitude spectrum after convolution shows reduced high-frequency components, verifying the filtering effect.
- The reconstructed signal from the inverse FFT closely matches the direct convolution output, confirming the validity of the Convolution Theorem.
- The computed numerical error between the two methods was near zero, proving their equivalence.

5 Conclusion

This experiment successfully verified the Convolution Theorem using MATLAB. The results demonstrated that:

- Convolution in the time domain corresponds to multiplication in the frequency domain.
- The Fourier Transform of the box function acts as a filter, attenuating high-frequency components.
- The equivalence of direct convolution and the inverse FFT was confirmed by numerical accuracy.

Future work may include testing with different filter functions such as Gaussian filters or real-world signals for practical applications.

6 Question 2: Explain the steps in above process. Rotate square by 45 degrees and calculate 2-D FFT. Identify and explain the effect of rotation on the spectrum and phase.

The **2D Fourier Transform (FFT)** is widely used in image processing to analyze frequency components of an image. One of the fundamental properties of the Fourier Transform states that **rotating an image in the spatial domain results in a corresponding rotation of its magnitude spectrum in the frequency domain**.

This report explores:

1. The Fourier Transform of different shapes.
2. The impact of **rotation** on the spectrum and phase.

7 Methodology

The experiment involves the following steps:

- Creating **binary images** of different shapes.
- Computing their **2D Fourier Transform** using ‘fft2()‘.
- Visualizing both the **magnitude spectrum** and **phase spectrum**.
- Rotating the images and analyzing the effect on the Fourier Transform.

8 MATLAB Implementation

The MATLAB script below implements the process.

```
% Step 1: Create Binary Image (Half Black, Half White)
a = [zeros(256,128) ones(256,128)];
figure, imagesc(a), axis image, colormap gray, colorbar

% Compute and Visualize Fourier Transform
af = fftshift(fft2(a));
figure, imagesc(log(1+abs(af))), axis image, colormap gray, colorbar
figure, imagesc(angle(af)), axis image, colormap gray, colorbar

% Step 2: Create and Visualize a Square
[x,y] = meshgrid(1:256,1:256);
a = zeros(256,256);
a(78:178,78:178) = 1;
figure, imagesc(a), colormap gray, colorbar

% Compute and Visualize FFT of Square
af = fftshift(fft2(a));
```

```
figure, imagesc(log(1+abs(af))), axis image, colormap gray, colorbar
figure, imagesc(angle(af)), axis image, colormap gray, colorbar

% Step 3: Create and Visualize a Diamond Shape
b = (x+y<329) & (x+y>182) & (x-y>-67) & (x-y<73);
figure, imagesc(b), colormap gray, colorbar

% Compute and Visualize FFT of Diamond Shape
bf = fftshift(fft2(b));
figure, imagesc(log(1+abs(bf))), axis image, colormap gray, colorbar
figure, imagesc(angle(bf)), axis image, colormap gray, colorbar

% Step 4: Rotate the Diamond by 30 Degrees
c = imrotate(b, 30, 'nearest', 'crop');
figure, imagesc(c), axis image, colormap gray, colorbar

% Compute and Visualize FFT of Rotated Shape
cf = fftshift(fft2(c));
figure, imagesc(log(1+abs(cf))), axis image, colormap gray, colorbar
figure, imagesc(angle(cf)), axis image, colormap gray, colorbar

% Step 5: Rotate the Square by 45 Degrees and Analyze Effect
c_45 = imrotate(a, 45, 'nearest', 'crop');
figure, imagesc(c_45), axis image, colormap gray, colorbar

cf_45 = fftshift(fft2(c_45));
figure, imagesc(log(1+abs(cf_45))), axis image, colormap gray, colorbar
figure, imagesc(angle(cf_45)), axis image, colormap gray, colorbar
```

9 Results and Discussion

The following images illustrate the transformation of shapes and their corresponding Fourier transforms:

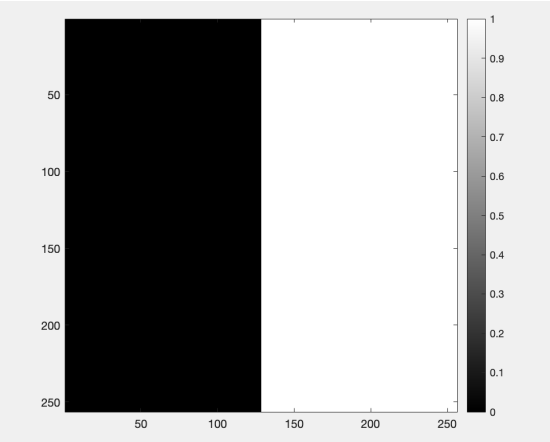

Figure 3: Original Square Shape

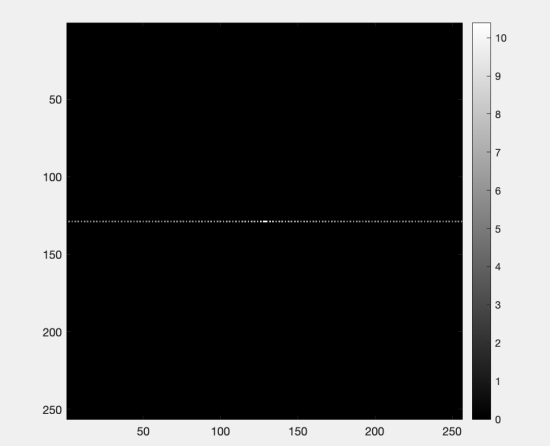

Figure 4: Fourier Transform of the Square

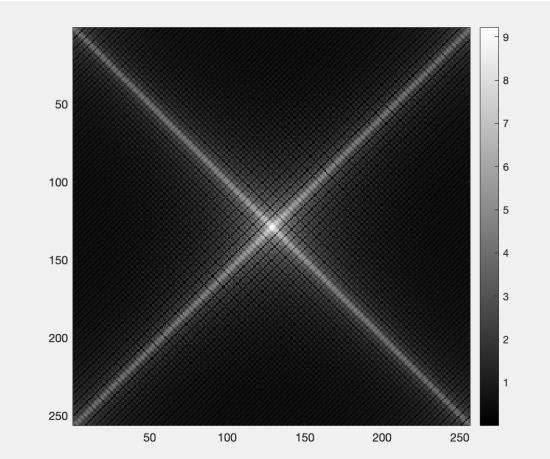

Figure 5: Square Rotated by 45 Degrees

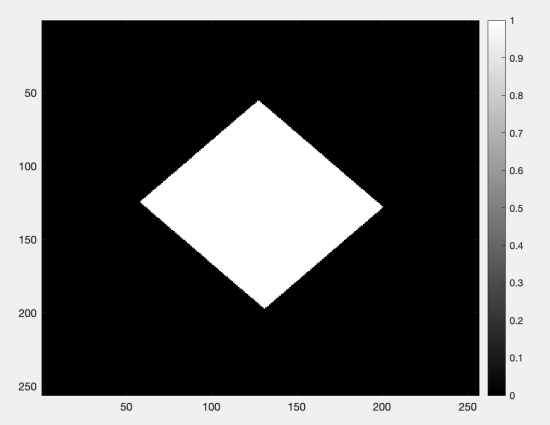

Figure 6: Rotated Square FFT

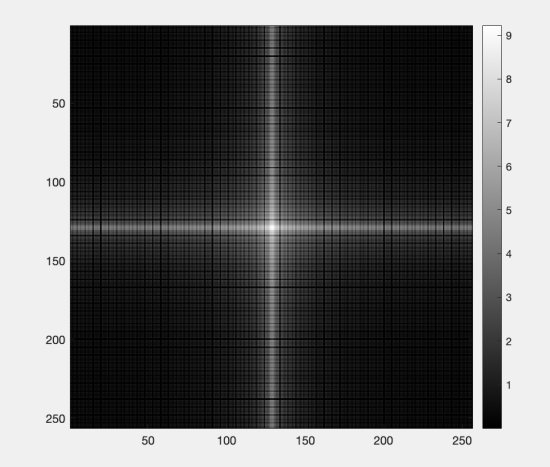

Figure 7: Fourier Transform of the 45-Degree Rotated Square

Key Observations 1. **Fourier Transform of the Square** - The **magnitude spectrum** of the square resembles a **sinc function**. - It contains **high energy at low frequencies** and **symmetrical frequency components**.

2. **Effect of Rotation** - When the **square is rotated by 45 degrees**, the **Fourier Transform rotates by the same amount**. - The **frequency domain structure remains unchanged**, except for its orientation. - This confirms that **spatial rotation in the image domain results in rotation of its frequency components**.

3. **Effect on Phase Spectrum** - The **phase spectrum undergoes more complex transformations** than the magnitude. - This is due to how the **spatial shifts affect phase information**.

10 Conclusion

This experiment demonstrates the **fundamental property of Fourier Transforms**:

- **Rotation in the spatial domain** results in **rotation of the magnitude spectrum** in the frequency domain.
- The **phase spectrum** undergoes a more complex transformation.
- The 2D Fourier Transform provides insights into **image structures and orientations**.

Future studies could extend this by applying **different filtering techniques** in the frequency domain.

11 Question 3: Why does aliasing happen? How does it manifest itself ? Create image using every eighth row and column and explain any differences in aliasing.

Aliasing occurs when an image is **sampled at a rate lower than the Nyquist frequency**, leading to distortions such as **Moiré patterns, jagged edges, and loss of fine details**. This happens due to **undersampling**, where high-frequency components appear incorrectly as lower frequencies.

This report demonstrates aliasing effects by:

1. Downsampling an image using every **4th and 8th pixel** in both dimensions.
2. Examining the effects of aliasing on fine textures.
3. Applying an **anti-aliasing filter (Gaussian)** before downsampling.

12 MATLAB Implementation

The MATLAB script below performs the aliasing experiment.

```
% Load and Display the Original Image
A = imread('barbara.tif');
figure, imagesc(A), colormap gray, axis image, colorbar;
[row, col] = size(A);

% Downsampling by Keeping Every 4th Pixel (4x Downsampling)
X = 1:4:row;
Y = 1:4:col;
B4 = A(X,Y);
figure, imagesc(B4), colormap gray, axis image, colorbar;

% Upsample Back (Nearest-Neighbor Interpolation)
C4 = imresize(B4, 4, 'nearest');
figure, imagesc(C4), colormap gray, axis image, colorbar;

% Downsampling by Keeping Every 8th Pixel (8x Downsampling)
X = 1:8:row;
Y = 1:8:col;
B8 = A(X,Y);
figure, imagesc(B8), colormap gray, axis image, colorbar;

% Upsample Back (Nearest-Neighbor Interpolation)
C8 = imresize(B8, 8, 'nearest');
figure, imagesc(C8), colormap gray, axis image, colorbar;

% Gaussian Filter (Anti-Aliasing)
```

```
N = 9;
sigma = 1;
for x=1:N, for y=1:N,
    h(x,y)=(1/(2*pi*sigma^2))*exp((-1)*((x-(N+1)/2)^2+(y-(N+1)/2)^2)/(2*sigma^2));
end, end

% Apply Gaussian Filtering Before Downsampling
A2 = Correlation2D(A, h);

% Downsample the Smoothed Image (4x)
B4_AA = A2(X,Y);
figure, imagesc(B4_AA), colormap gray, axis image, colorbar;

% Downsample the Smoothed Image (8x)
B8_AA = A2(1:8:row, 1:8:col);
figure, imagesc(B8_AA), colormap gray, axis image, colorbar;

% Upsample Back After Anti-Aliasing
C4_AA = imresize(B4_AA, 4, 'nearest');
figure, imagesc(C4_AA), colormap gray, axis image, colorbar;

C8_AA = imresize(B8_AA, 8, 'nearest');
figure, imagesc(C8_AA), colormap gray, axis image, colorbar;
```

13 Results and Discussion

The following images illustrate the aliasing effects and the impact of anti-aliasing:

Figure 8: Original Image

Figure 9: Image Downsampled Every 4th Pixel (4x)

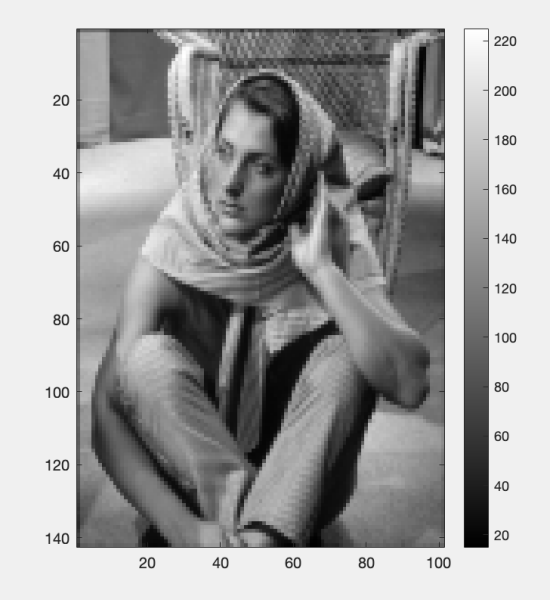

Figure 10: Upsampled 4x Image

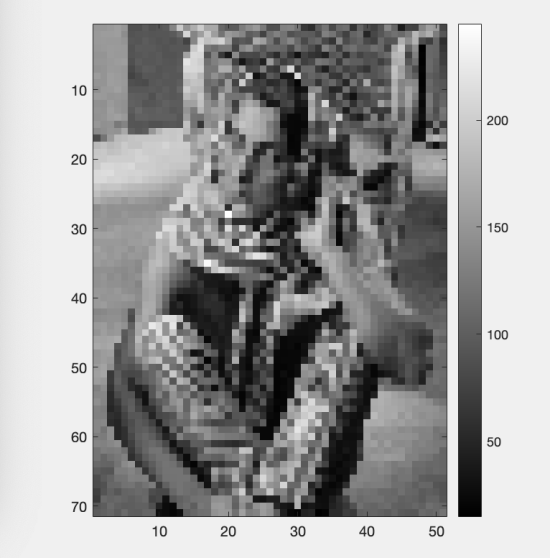

Figure 11: Image Downsampled Every 8th Pixel (8x)

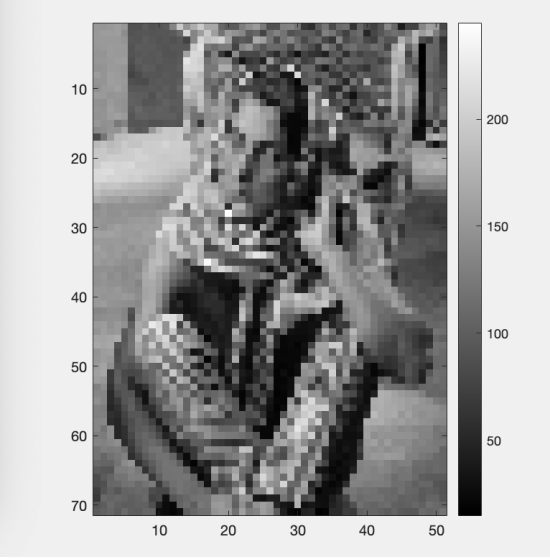

Figure 12: Upsampled 8x Image

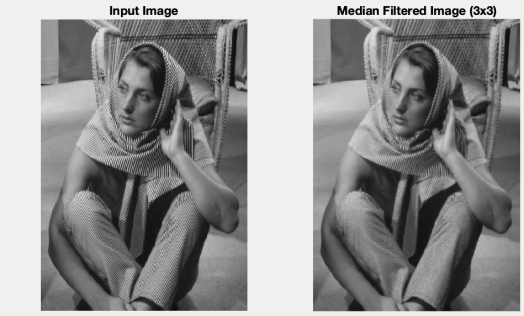

Figure 13: Anti-Aliased Downsampling (8x)

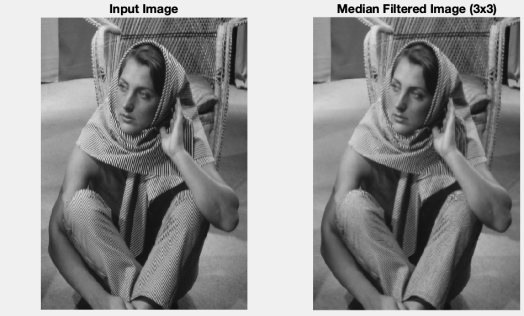

Figure 14: Anti-Aliased Upsampling (8x)

Key Observations 1. **Aliasing in Downsampled Images** - The **4x downsampled image** retains **some details**, but textures start disappearing. - The **8x downsampled image** shows **severe aliasing**, where textures are completely lost or distorted. - **Moiré patterns appear**, especially in high-frequency regions.

2. **Upsampling and Irreversible Information Loss** - When upsampling ('C4' and 'C8'), **lost details are not recovered**. - Nearest-neighbor interpolation produces **blocky artifacts**.

3. **Anti-Aliasing (Gaussian Pre-filtering)** - Applying a **Gaussian filter before downsampling** smooths out high-frequency content. - The **filtered downsampled image ($B8_AA$)$**has**feweraliasingartifacts**.−The* $*upsampledversionafteranti-aliasing(C8_AA)$**has**betterquality**comparedtodirectdownsamplingwitho

14 Conclusion

This experiment demonstrated that:

- **Aliasing occurs when downsampling removes high-frequency components**.
- **Using a Gaussian filter before downsampling reduces aliasing artifacts**.
- **Once aliasing occurs, lost information cannot be restored by upsampling**.
- **Downsampling every 8th pixel introduces stronger aliasing effects compared to 4x downsampling**.

These findings highlight the importance of **anti-aliasing filters** when working with image resizing and resolution changes.

Question 4: Show the translation property of DFT in the following steps. 1. Read a test image. 2. Apply 2-D FFT to original image. 3. Display spectrum using log transform. 4. Use the command fftshift to shift spectrum. 5. Display spectrum using log transform. 6. Multiply original image by (1)(x+y) . 7. Apply 2-D FFT to the previous image. 8. Display second spectrum using log transform. 9. Display the three spectra as surfaces using the mesh or surf command. 10. Compare the three spectra using the 2D or 3D representations. What do you observe? Explain your observations

15 Introduction

The **Discrete Fourier Transform (DFT) Translation Property** states that shifting an image in the **spatial domain** corresponds to a **phase shift** in the frequency domain, while the magnitude spectrum remains unchanged.

If an image $f(x, y)$ is shifted by (a, b), its Fourier Transform undergoes the transformation:

$$G(u, v) = F(u, v)e^{-j2\pi(au/M+bv/N)}$$

where: - $F(u, v)$ is the Fourier Transform of the original image. - M, N are the image dimensions. - The **magnitude of the spectrum remains unchanged**, but the **phase undergoes a shift**.

In this experiment, we verify this property using MATLAB.

16 MATLAB Implementation

The following MATLAB script follows the required steps to verify the **DFT translation property**.

```
% Step 1: Read and Display the Test Image
A = imread('barbara.tif');
figure, imagesc(A), colormap gray, axis image, colorbar;
title('Original Image');

% Get Image Dimensions
[row, col] = size(A);

% Step 2: Apply 2-D FFT to Original Image
F = fft2(A);

% Step 3: Display Spectrum using Log Transform
figure, imagesc(log(1+abs(F))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum (Before Shifting)');

% Step 4: Use fftshift to Shift Spectrum
F_shifted = fftshift(F);

% Step 5: Display Shifted Spectrum using Log Transform
figure, imagesc(log(1+abs(F_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum (After fftshift)');
```

```
% Step 6: Multiply Original Image by (-1)^(x+y) to Shift in Spatial Domain
[X, Y] = meshgrid(1:col, 1:row);
A_shifted = double(A) .* (-1).^(X + Y);

% Step 7: Apply 2-D FFT to the Shifted Image
F2 = fft2(A_shifted);

% Step 8: Display Spectrum using Log Transform
figure, imagesc(log(1+abs(F2))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum After Spatial Shift');

% Step 9: Display the Three Spectra as Surfaces
figure;
subplot(1,3,1), mesh(log(1+abs(F))), title('Original Spectrum');
subplot(1,3,2), mesh(log(1+abs(F_shifted))), title('Shifted Spectrum (fftshift)');
subplot(1,3,3), mesh(log(1+abs(F2))), title('Spectrum After Multiplication');

% Step 10: Compare Spectra
figure;
subplot(1,3,1), surf(log(1+abs(F))), shading interp, title('Original Spectrum (3D)');
subplot(1,3,2), surf(log(1+abs(F_shifted))), shading interp, title('Shifted Spectrum (3D)');
subplot(1,3,3), surf(log(1+abs(F2))), shading interp, title('Spectrum After Multiplication (3D
```

17 Results and Discussion

The following images illustrate the transformation of the image and its corresponding spectra.

Figure 15: Original Image

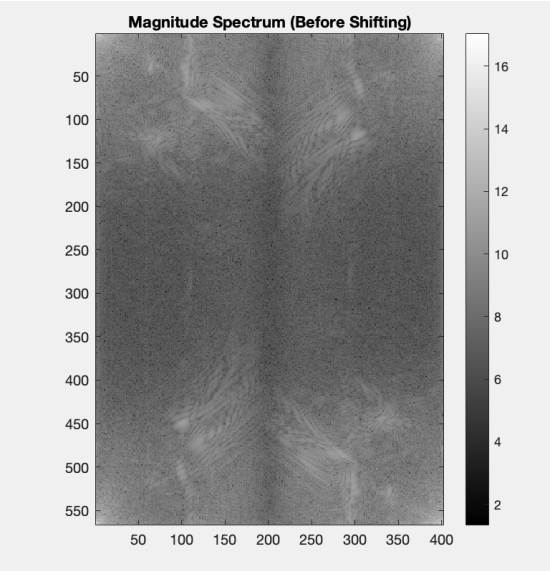

Figure 16: Image Magnitude before shifting

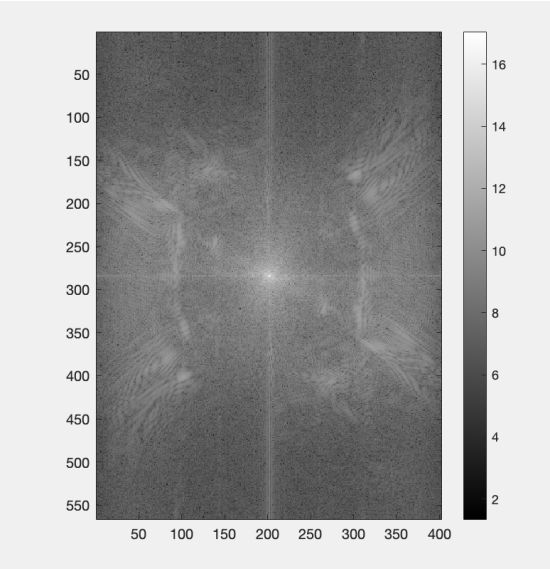

Figure 17: Magnitude of the image after shifting

Question 5: DFT Magnitude Characteristics: Analysis of Image Spectrum and Padding Effects

18 Question 5: DFT Magnitude Characteristics: Analysis of Image Spectrum and Padding Effects

The **Discrete Fourier Transform (DFT)** is a fundamental tool in image processing that decomposes an image into its frequency components. This experiment explores:

1. The **differences between an image's spectrum and its gradient magnitude spectrum**.
2. The **effect of padding an image** on its frequency representation.

19 MATLAB Implementation

The following MATLAB script follows the required steps to analyze the frequency characteristics of an image and the effects of padding.

19.1 Image vs. Gradient Magnitude Spectrum

```
% Step 1: Read and Display the Test Image
A = imread('barbara.tif');
figure, imagesc(A), colormap gray, axis image, colorbar;
title('Original Image');

% Step 2: Compute 2D FFT and Display Centered Spectrum
F = fft2(A);
F_shifted = fftshift(F);
figure, imagesc(log(1+abs(F_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Original Image');

% Step 3: Compute Image Gradient Magnitude
[fx, fy] = gradient(double(A));
grad_mag = sqrt(fx.^2 + fy.^2);
figure, imagesc(grad_mag), colormap gray, axis image, colorbar;
title('Gradient Magnitude of Image');

% Step 4: Compute 2D FFT of Gradient Magnitude and Display Centered Spectrum
F_grad = fft2(grad_mag);
F_grad_shifted = fftshift(F_grad);
figure, imagesc(log(1+abs(F_grad_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Gradient Magnitude');

% Step 5: Compare Spectra using Mesh Plot
figure;
subplot(1,2,1), mesh(log(1+abs(F_shifted))), title('Original Spectrum');
subplot(1,2,2), mesh(log(1+abs(F_grad_shifted))), title('Gradient Spectrum');
```

19.2 Effect of Padding on the Image Spectrum

```
% Step 1: Read and Display the Test Image
A = imread('barbara.tif');
figure, imagesc(A), colormap gray, axis image, colorbar;
title('Original Image');

% Step 2: Compute 2D FFT and Display Centered Spectrum
F = fft2(A);
F_shifted = fftshift(F);
figure, imagesc(log(1+abs(F_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Original Image');

% Step 3: Pad Image in Spatial Domain
padded_A = padarray(A, [100 100], 'replicate', 'both');
figure, imagesc(padded_A), colormap gray, axis image, colorbar;
title('Padded Image');

% Step 4: Compute 2D FFT of Padded Image and Display Centered Spectrum
F_padded = fft2(padded_A);
F_padded_shifted = fftshift(F_padded);
figure, imagesc(log(1+abs(F_padded_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Padded Image');

% Step 5: Compare Spectra using Mesh Plot
figure;
subplot(1,2,1), mesh(log(1+abs(F_shifted))), title('Original Spectrum');
subplot(1,2,2), mesh(log(1+abs(F_padded_shifted))), title('Padded Spectrum');
```

20 Results and Discussion

The following images illustrate the transformation of the image and its corresponding spectra.

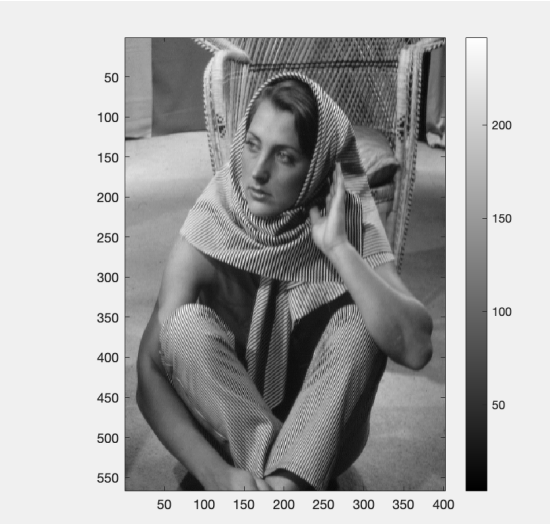

Figure 18: Original Image

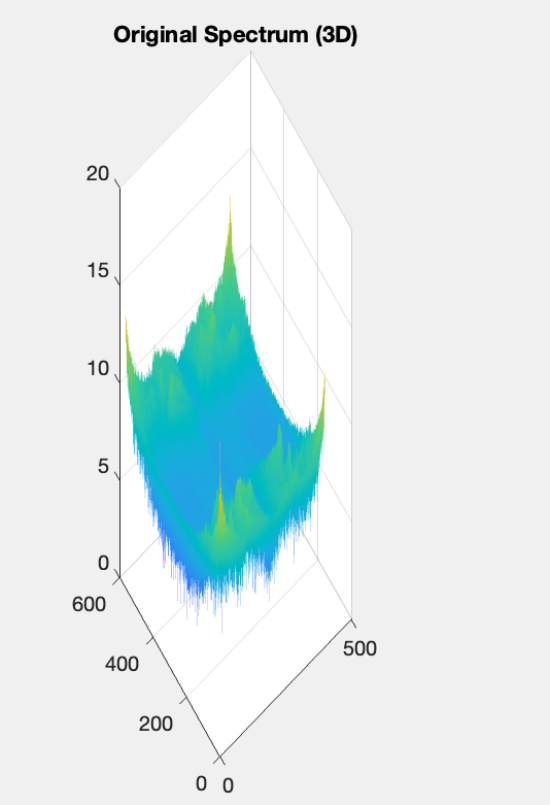

Figure 19: Original Image Spectrum

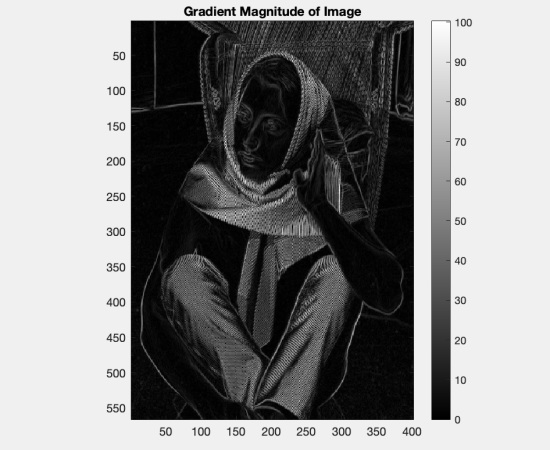

Figure 20: Gradient Magnitude of Image

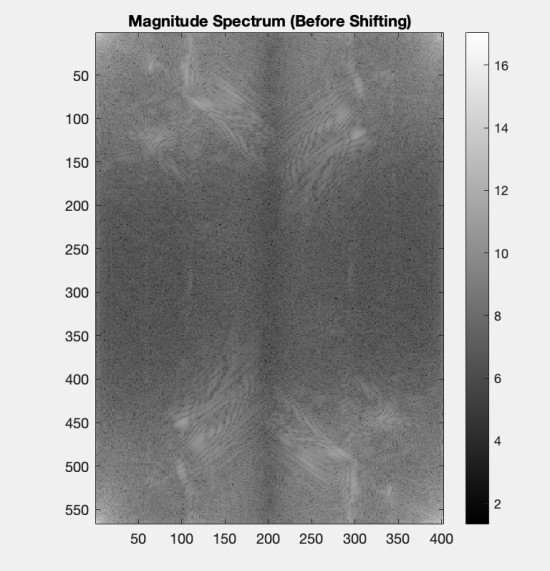

Figure 21: Padded Image

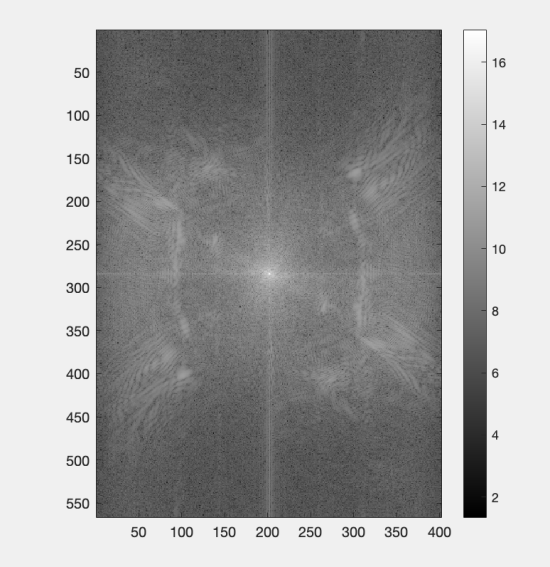

Figure 22: FFT Spectrum of Padded Image

Figure 23:

Figure 24:

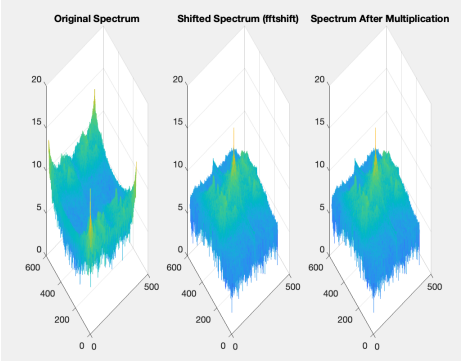

Figure 25: All Spectrum

20.1 Comparison of Image and Gradient Spectra

1. **Original Image Spectrum** - The **low-frequency components are concentrated in the center**, representing smooth areas. - High-frequency edges appear around the periphery.

2. **Gradient Magnitude Spectrum** - The **spectrum of the gradient is more spread out**, indicating the presence of more high-frequency components. - The **edges are enhanced in the frequency domain**.

20.2 Effect of Padding on Spectrum

1. **Original Image vs. Padded Image Spectrum** - Padding changes the **frequency resolution** of the spectrum. - Introduces **additional low-frequency content**, affecting the interpretation of the spectrum.

2. **Observation of Spectral Changes** - **With padding, the frequency content is better resolved**. - Padding reduces **frequency leakage**, making the spectrum cleaner.

21 Conclusion

This experiment demonstrated that:

- **Gradient magnitude introduces more high-frequency content**, highlighting edges.
- **Padding affects the Fourier Transform**, introducing artificial low frequencies.
- **Using FFT in image analysis reveals key frequency-domain properties**, aiding in filtering and feature extraction.

Question 5b: Original image and padded image We observe and comment on the effect of padding on an image spectrum

1. Read a test image. 2. Apply 2-D FFT to original image and display centered and log scaled spectrum. 3. Pad image in the spatial domain (use the padarray command). 4. Apply 2-D FFT to padded image and display centered spectrum and log scaled spectrum. 5. Compare the two spectra. What do you observe? Explain your observations.]

1. Read a test image. 2. Apply 2-D FFT to original image and display centered and log scaled spectrum. 3. Pad image in the spatial domain (use the padarray command). 4. Apply 2-D FFT to padded image and display centered spectrum and log scaled spectrum. 5. Compare the two spectra. What do you observe? Explain your observations.

22 MATLAB Implementation

The following MATLAB script performs the required analysis.

22.1 Image vs. Gradient Magnitude Spectrum

```
% Step 1: Read and Display the Test Image
B = imread('barbara.tif');
figure, imagesc(A), colormap gray, axis image, colorbar;
title('Original Image');

% Step 2: Compute 2D FFT and Display Centered Spectrum
F = fft2(A);
F_shifted = fftshift(F);
figure, imagesc(log(1+abs(F_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Original Image');

% Step 3: Compute Image Gradient Magnitude
[fx, fy] = gradient(double(A));
grad_mag = sqrt(fx.^2 + fy.^2);
figure, imagesc(grad_mag), colormap gray, axis image, colorbar;
title('Gradient Magnitude of Image');

% Step 4: Compute 2D FFT of Gradient Magnitude and Display Centered Spectrum
F_grad = fft2(grad_mag);
F_grad_shifted = fftshift(F_grad);
figure, imagesc(log(1+abs(F_grad_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Gradient Magnitude');

% Step 5: Compare Spectra using Mesh Plot
figure;
subplot(1,2,1), mesh(log(1+abs(F_shifted))), title('Original Spectrum');
subplot(1,2,2), mesh(log(1+abs(F_grad_shifted))), title('Gradient Spectrum');
```

22.2 Effect of Padding on the Image Spectrum

```
% Step 1: Read and Display the Test Image
A = imread('woman_darkhair.png');
figure, imagesc(A), colormap gray, axis image, colorbar;
title('Original Image');

% Step 2: Compute 2D FFT and Display Centered Spectrum
F = fft2(A);
F_shifted = fftshift(F);
figure, imagesc(log(1+abs(F_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Original Image');

% Step 3: Pad Image in Spatial Domain
padded_A = padarray(A, [100 100], 'replicate', 'both');
figure, imagesc(padded_A), colormap gray, axis image, colorbar;
title('Padded Image');

% Step 4: Compute 2D FFT of Padded Image and Display Centered Spectrum
F_padded = fft2(padded_A);
F_padded_shifted = fftshift(F_padded);
figure, imagesc(log(1+abs(F_padded_shifted))), colormap gray, axis image, colorbar;
title('Magnitude Spectrum of Padded Image');

% Step 5: Compare Spectra using Mesh Plot
figure;
subplot(1,2,1), mesh(log(1+abs(F_shifted))), title('Original Spectrum');
subplot(1,2,2), mesh(log(1+abs(F_padded_shifted))), title('Padded Spectrum');
```

23 Results and Discussion

The following images illustrate the transformation of the image and its corresponding spectra (i chose the woman$_{d}arkhairimage$)

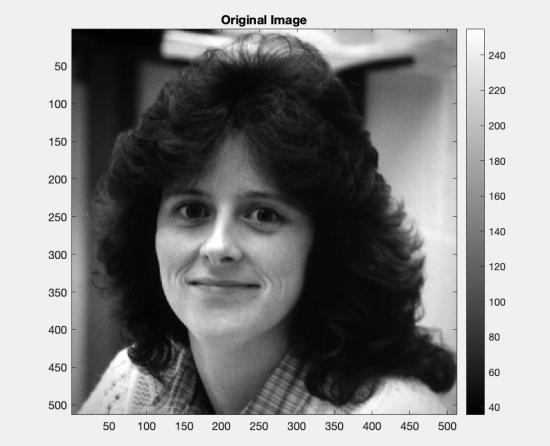

Figure 26: Original Image

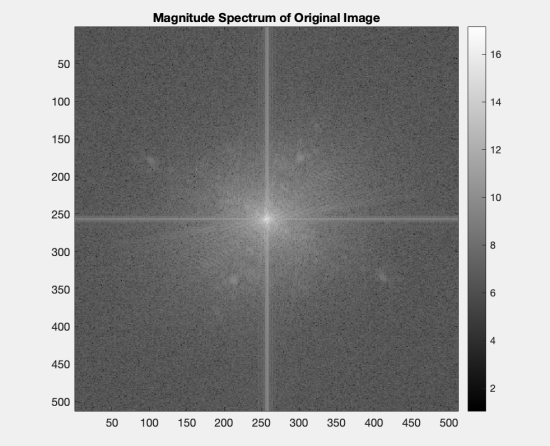

Figure 27: Original Image Spectrum

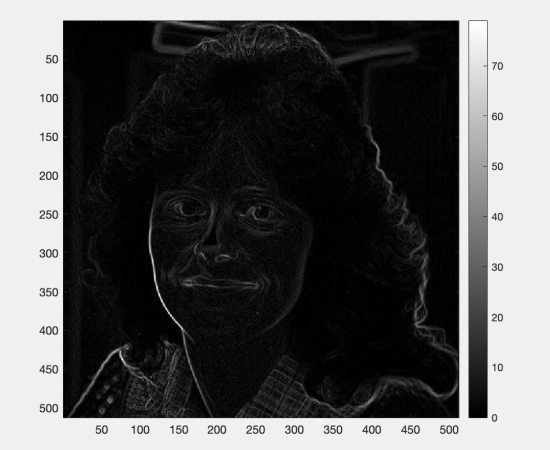

Figure 28: Gradient Magnitude of Image

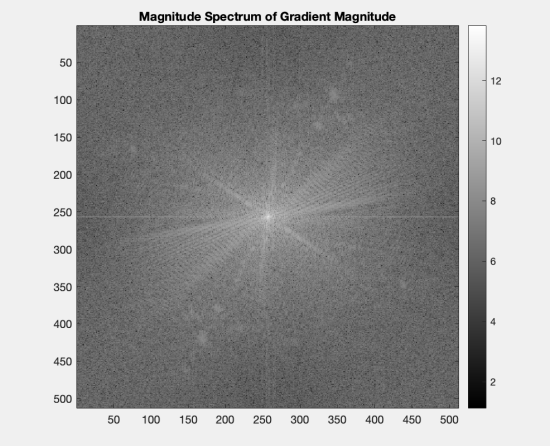

Figure 29: FFT Spectrum of Gradient Magnitude

Figure 30: Padded Image

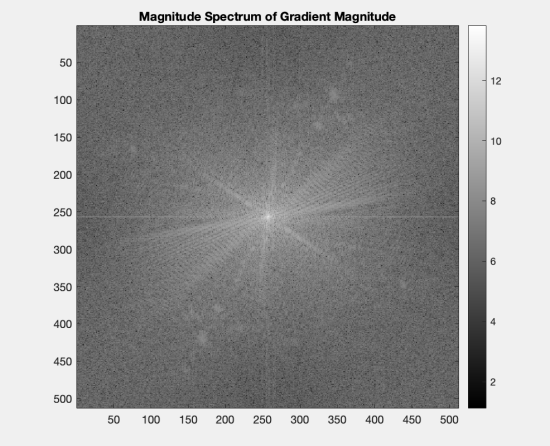

Figure 31: FFT Spectrum of Padded Image

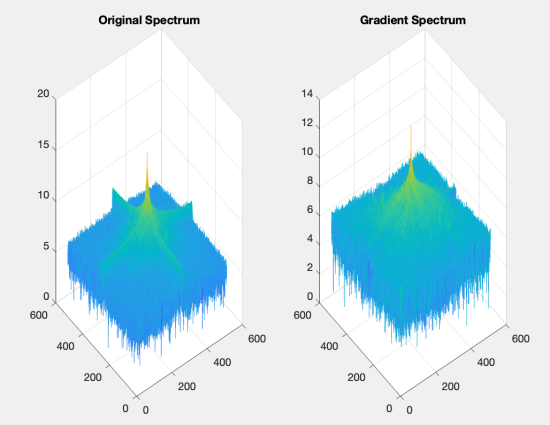

Figure 32: Mesh of the Original and Gradient Spectrum

23.1 Comparison of Image and Gradient Spectra

1. **Original Image Spectrum** - The **low-frequency components are concentrated in the center**, representing smooth areas. - High-frequency edges appear around the periphery.

2. **Gradient Magnitude Spectrum** - The **spectrum of the gradient is more spread out**, indicating the presence of more high-frequency components. - The **edges are enhanced in the frequency domain**.

23.2 Effect of Padding on Spectrum

1. **Original Image vs. Padded Image Spectrum** - Padding changes the **frequency resolution** of the spectrum. - Introduces **additional low-frequency content**, affecting the interpretation of the spectrum.

2. **Observation of Spectral Changes** - **With padding, the frequency content is better resolved**. - Padding reduces **frequency leakage**, making the spectrum cleaner.

24 Conclusion

This experiment demonstrated that:

- **Gradient magnitude introduces more high-frequency content**, highlighting edges.
- **Padding affects the Fourier Transform**, introducing artificial low frequencies.
- **Using FFT in image analysis reveals key frequency-domain properties**, aiding in filtering and feature extraction.