Assignment 7

Mahdi Tabatabaei - Heliya Shakeri

Biomedical Signal and Image Processing Lab

Nastaran Nouri

December 4, 2024

Contents

Question 1

Analyze the given image (t1.jpg) by reading it using the `imread` command. Display the image using `imshow`, select the first slice of the image for processing, and compute the 1D Discrete Fourier Transform (DFT) of the 128th row. Plot the magnitude and phase of the DFT. Then, convert the image to a double format and compute the 2D FFT. Visualize the results with and without the `fftshift` for better interpretation.

```
%% Q1
% Read the image
img = imread(data_path + "/S1_Q1_utils/t1.jpg");

% Display the image
figure
imshow(img);
title('S1-Q1-utils/t1.jpg')
saveas(gcf, results_path + "fig.q1.1.png");

% Select the first slice
img_d = img(:, :, 1);
img_d_slcted_row = img_d(128, :);

% Length of the selected row
N = length(img_d_slcted_row);

% DFT of the selected row
dfft_row = fft(img_d_slcted_row, N);

% fftshift to center the zero-frequency component
dfft_row = fftshift(dfft_row);


figure('units', 'normalized', 'outerposition', [0 0 1 1]);

% Plot the magnitude of the DFT
subplot(2, 1, 1);
scale = linspace(-pi, pi, N);
plot(scale, abs(dfft_row));
xlim([scale(1) scale(end)]);
title('Magnitude of DFT of the row 128');
xlabel('Frequency');
grid on;

% Plot the phase of the DFT
subplot(2, 1, 2);
plot(scale, angle(dfft_row));
xlim([scale(1) scale(end)]);
title('Phase of DFT of the row 128');
xlabel('Frequency');
grid on;
saveas(gcf, results_path + "fig.q1.2.png");

% Convert the image slice to double for FFT2
fft2_img = fft2(double(img_d));

figure('units', 'normalized', 'outerposition', [0 0 1 1]);

% Plot the original first slice of the image
```

```
    subplot(1, 3, 1);
    imshow(img_d);
    title('First Slice of Image');

    % fftshift and the log magnitude
    subplot(1, 3, 2);
    fft_rot_2 = log10(abs(fftshift(fft2_img)));
    fft_rot_2 = fft_rot_2 / max(max(abs(fft_rot_2)));
    imshow(fft_rot_2);
    title('2d FFT with fftshift');

    % Log magnitude without fftshift
    subplot(1, 3, 3);
    fft_rot_2 = log10(abs(fft2_img));
    fft_rot_2 = fft_rot_2 / max(max(abs(fft_rot_2)));
    imshow(fft_rot_2);
    title('2d FFT without fftshift');

    saveas(gcf, results_path + "fig.q1.3.png");
```

Source Code 1: Fourier Transform Analysis of t1.jpg

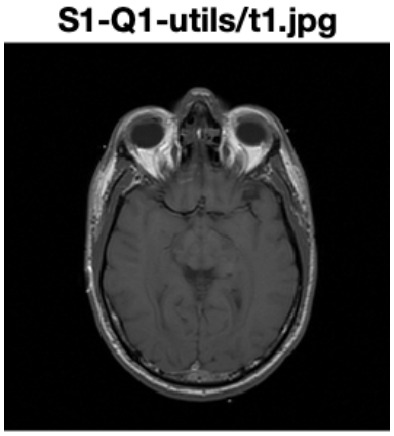

Figure 1: Original Image (t1.jpg)

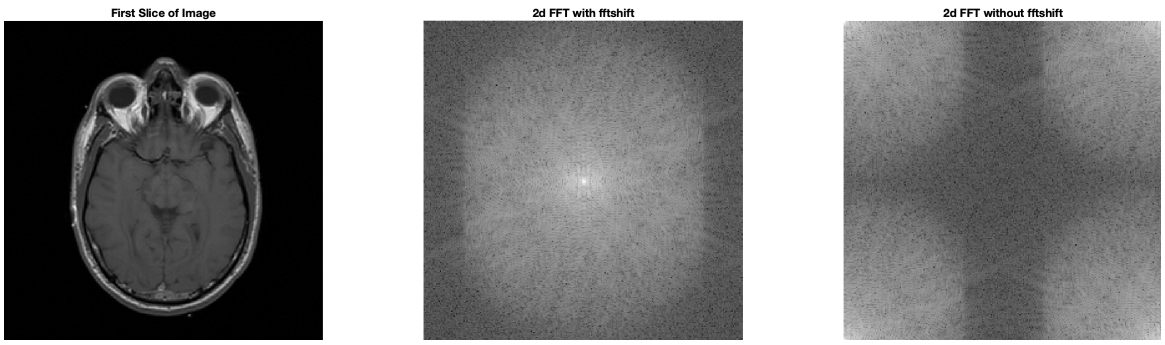

Figure 2: 2D FFT with and without `fftshift`

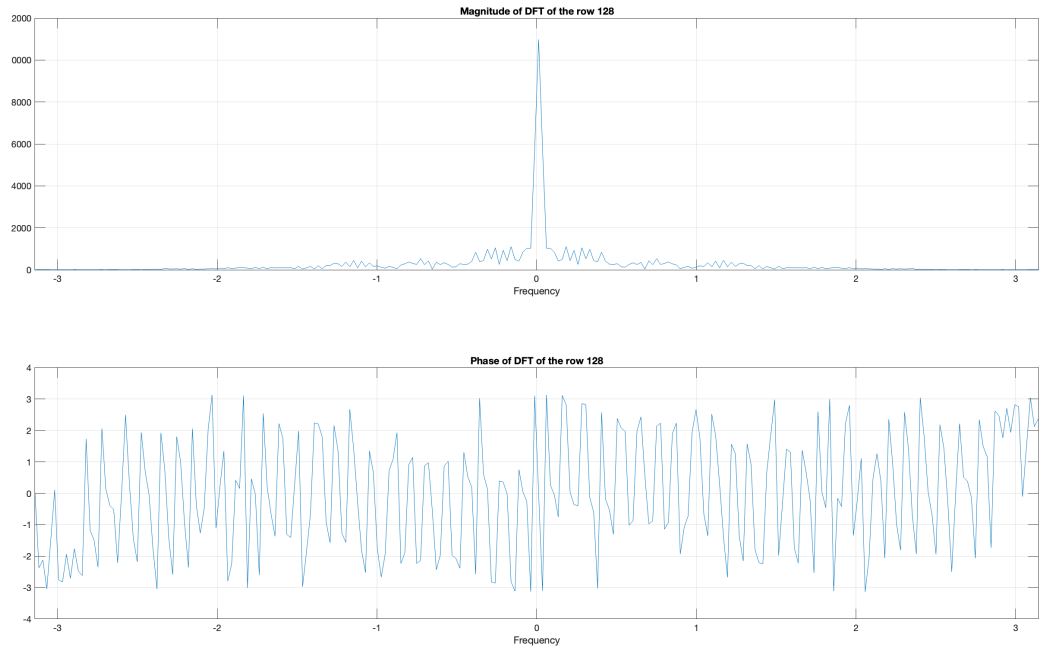

Figure 3: Magnitude and Phase of DFT for the 128th Row of the Image

Soloution

The DFT of the 128th row of the image provides information about its frequency components. The magnitude plot shows the energy distribution, while the phase plot illustrates the alignment of the frequencies. When performing 2D FFT, the `fftshift` centers the zero-frequency component, making the visualization more interpretable. Without `fftshift`, the result is not centered, which can be less intuitive for analysis.

Question 2: Image Convolution Using Fourier Transform

Perform convolution of two matrices, G and L, using Fourier Transform and visualize the results. Then, apply convolution between a given image and G, and display the results for analysis.

```
%% Q2
% G matrix
L = 256;
[X,Y] = ndgrid(-L/2:L/2-1, -L/2:L/2-1);
G = zeros(L,L);
G(X.^2 + Y.^2 <= L-1) = 1;

% L matrix
F = zeros(L,L);
F(100,50) = 1;
F(120,48) = 2;

% Convolution of G and F using Fourier Transform
fft_G = fft2(G);
fft_F = fft2(F);
fft_out = fft_G.*fft_F;
conv_img = fftshift(ifft2((fft_out)));
conv_img = conv_img/max(max(conv_img));

```

```
    % Plot G, F, and their convolution result
    figure('units','normalized','outerposition',[0 0 1 1])
    subplot(1,3,1)
    imshow(G)
    title('G Matrix')
    subplot(1,3,2)
    imshow(F)
    title('L Matrix')
    subplot(1,3,3)
    imshow(conv_img)
    title('Convolution of Images')

    saveas(gcf,results_path + "fig.q2.1.png");

    % pd image and conv with G
    img = imread(data_path+"/S1_Q2_utils/pd.jpg");
    img_d = double(img(:,:,1));

    % Normalization
    img_d = img_d/max(max(img_d));

    % Convolution of the pd image and G using Fourier Transform
    fft_img = fft2(img_d);
    fft_out = fft_img.*fft_G;
    conv_img = fftshift(ifft2(fft_out));
    conv_img = 255*conv_img/max(max(abs(conv_img)));
    conv_img = uint8(conv_img);

    % Plot the original image and the convolution result
    figure
    subplot(1,2,1)
    imshow(img_d)
    title('S1-Q2-utils/pd.jpg')

    subplot(1,2,2)
    imshow(conv_img)
    title('pd convolved with G')
    saveas(gcf,results_path + "fig.q2.2.png");
```

Source Code 2: Convolution of Matrices and Image with Fourier Transform

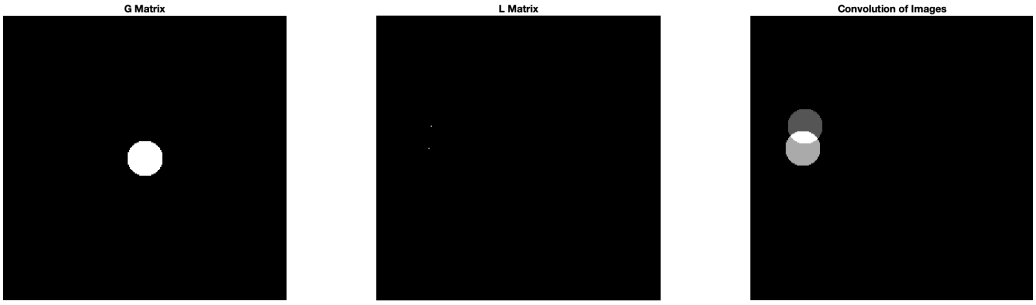

Figure 4: G, L, and Their Convolution Result

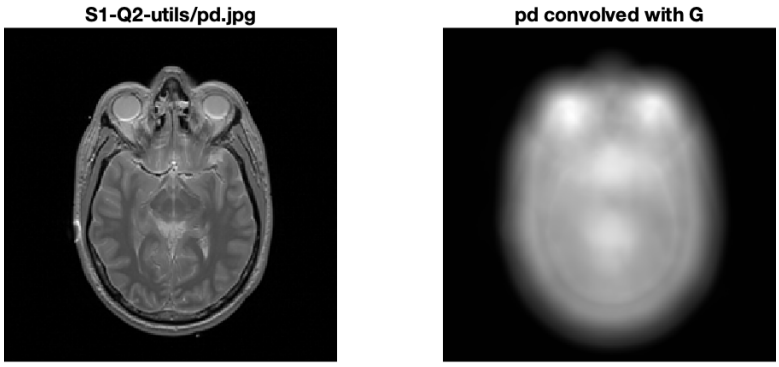

Figure 5: The Original pd Image and Its Convolution with G

Soloution

The convolution of the G and L matrices produces a circular result centered at the positions defined by L, with the intensity influenced by the values at those positions. This is expected as G represents a low-pass filter, allowing only the low-frequency components of L to pass through.
When applied to the input image (pd.jpg), the convolution with G highlights low-frequency features of the image, effectively smoothing it. The output removes high-frequency details, such as edges, while retaining the core structure of the image. This is consistent with the low-pass filtering property of G, which restricts the result to frequencies corresponding to the circular area defined by G's radius.
The resulting convolved image demonstrates blurred boundaries, as seen in the second figure. This outcome aligns with the expectation of a low-pass filter's behavior in both the spatial and frequency domains.

Question 3: Zero-Padding and Zoom in Frequency Domain

Use the concept of zero-padding in the frequency domain to magnify the spatial domain representation of an image. The process involves padding the frequency representation of the image with zeros to increase its resolution and then performing an inverse Fourier transform to magnify the image.

```
%% Q3
% The input image
img = imread(data_path+"/S1_Q3_utils/ct.jpg");
[d1, d2, d3] = size(img);

% 2D FFT of the image and shift zero frequency to the center
fft_img = fftshift(fft2(img));
zoom_s = 2;
new_img = zeros(zoom_s*d1, zoom_s*d2, 3);

%offset for centering the original image in the larger matrix
ms = round(d1*(zoom_s/2 - 0.5));
ns = round(d2*(zoom_s/2 - 0.5));

% Place the frequency content in the center of the zero-padded matrix
new_img(ms:(d1+ms-1), ns:(d2+ns-1), :) = fft_img;

% Inverse FFT to return to the spatial domain
```

```
    final = abs(ifft2(ifftshift(new_img)));
    f_fin = final(ms:(d1+ms-1), ns:(d2+ns-1), :);

    % Plot the original and zoomed images
    figure('units','normalized','outerposition',[0 0 1 1])
    subplot(1,2,1)
    imshow(img,[])
    title('Original Image');

    subplot(1,2,2)
    imshow(uint8(f_fin*zoom_s^2),[]);
    title('Zoomed Image');
    saveas(gcf,results_path + "fig.q3.1.png");
```

Source Code 3: Zero-Padding and Zoom in Frequency Domain

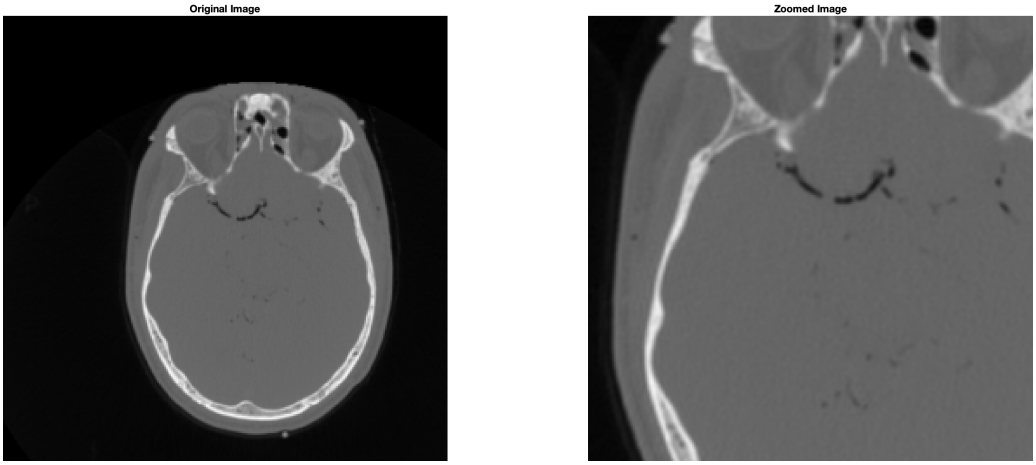

Figure 6: Original and Zoomed Image Using Zero-Padding in Frequency Domain

Soloution

Zero-padding in the frequency domain is a technique used to increase the resolution of the spatial domain image by interpolating additional information during the inverse Fourier transform. This is equivalent to magnifying the image without introducing new details. The zero-padded frequency domain representation introduces additional samples in the spatial domain, effectively ”zooming in” on the original image.

Q4-1: Spatial Shift Using Fourier Domain

The goal of this question is to create a spatial shift for the original image using processing in the Fourier domain and then return to the spatial domain.

Read the image `ct.jpg` from the folder `S1_Q4_utils`, and consider the first slice of the original image. Then, perform a spatial shift by 20 units to the right and 40 units downward.

Carefully perform this operation in the Fourier domain and then return to the spatial domain. Finally, plot the shifted Fourier kernel and analyze the result.

```
% Step 1: Read the image
image = imread('S1_Q4_utils/ct.jpg');
image_gray = rgb2gray(image); % Convert to grayscale if necessary

% Display the original image
figure;
imshow(image_gray);
title('Original Image');

% Step 2: Define the spatial shift
[m, n] = size(image_gray); % Get the size of the image
shift_x = 20; % Shift 20 units to the right
shift_y = 40; % Shift 40 units downward

% Create the Fourier shift kernel
[x, y] = meshgrid(0:(n-1), 0:(m-1));
shift_kernel = exp(-2j * pi * (shift_x * x / n + shift_y * y / m));

% Step 3: Compute the Fourier Transform of the image
image_fft = fft2(double(image_gray));

% Apply the shift in the Fourier domain
shifted_fft = image_fft .* shift_kernel;

% Step 4: Perform the inverse Fourier Transform to get the shifted image
shifted_image = real(ifft2(shifted_fft));

% Display the shifted image
figure;
imshow(shifted_image, []);
title('Shifted Image');

kernel_abs= real(shift_kernel);
figure
imshow(kernel_abs,[])
```

Source Code 4: Frequency Analysis of Clean and Noisy ECG Segments

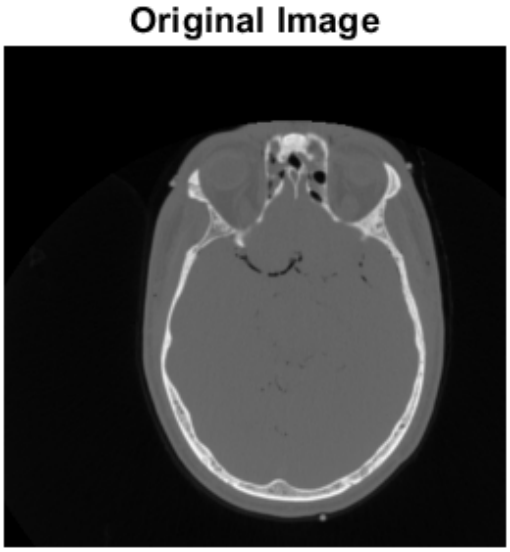

Figure 7: Original Image

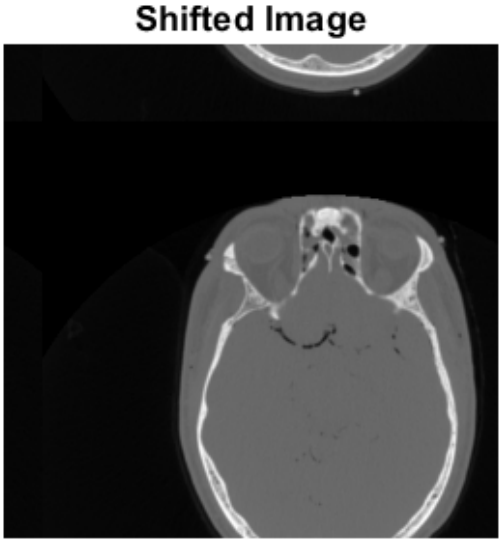

Figure 8: Shifted Image

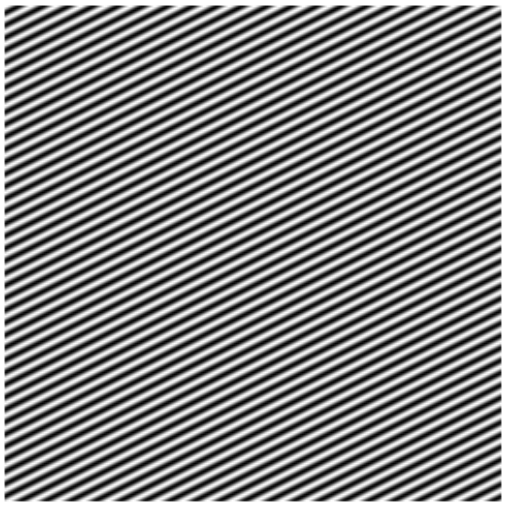

Figure 9: Fourier Transform of Kernel

Soloution

The 2D Fourier Transform highlights unique properties of spatial transformations, providing a clear way to interpret patterns in the frequency domain.
In 2D Fourier analysis, the transform represents a set of coefficients, a_k, for each spatial frequency and direction. These coefficients are mapped in the 2D frequency domain, defined by the axes (u_k, v_k), which correspond to horizontal and vertical spatial frequencies. These coefficients are significant, as they form the sinusoidal basis functions that reconstruct the image in the spatial domain. Their magnitude and phase specify the characteristics of the image.
The diagonal lines in the provided figure indicate that the image contains a shift in two directions. The angle of these lines relative to the horizontal axis determines the orientation of the diagonal patterns, highlighting the presence of spatial shifts in both directions.

Q4-2: Image Rotation and Fourier Domain Analysis

Read the image `ct.jpg` from the folder `S1_Q4_utils` and consider the first slice of it.

Rotate the image by 30 degrees using the `imrotate` function. Plot the original image and the rotated image side by side in the spatial domain.

Compute the Fourier Transform of both the original image and the rotated image. Plot their Fourier Transforms side by side in the frequency domain.

Finally, display the rotated image in the frequency domain (its Fourier Transform) and analyze the results.

```
% Step 1: Read the image and extract the first slice
image = imread('S1_Q4_utils/ct.jpg');
image_gray = rgb2gray(image); % Convert to grayscale if necessary

% Display the original image and rotated image side by side
angle = 30; % Rotation angle in degrees
rotated_image = imrotate(image_gray, angle, 'bilinear', 'crop');

figure;
```

```
    subplot(1, 2, 1);
    imshow(image_gray, []);
    title('Original Image');

    subplot(1, 2, 2);
    imshow(rotated_image, []);
    title(['Rotated Image (', num2str(angle), ' Degrees)']);

    % Step 2: Compute the Fourier Transforms of the original and rotated
    images
    image_fft = fftshift(fft2(double(image_gray)));
    rotated_image_fft = fftshift(fft2(double(rotated_image)));

    % Compute the magnitude spectrum for visualization
    original_fft_magnitude = log(1 + abs(image_fft));
    rotated_fft_magnitude = log(1 + abs(rotated_image_fft));

    % Display the Fourier Transforms side by side
    figure;
    subplot(1, 2, 1);
    imshow(original_fft_magnitude, []);
    title('Fourier Transform (Original Image)');

    subplot(1, 2, 2);
    imshow(rotated_fft_magnitude, []);
    title(['Fourier Transform (Rotated Image - ', num2str(angle), ' Degrees)'
    ]);

    % Step 3: Display the rotated Fourier domain
    % The rotated Fourier Transform is already computed in 'rotated_image_fft
    '.
    % We will directly plot its magnitude spectrum again for clarity.

    figure;
    imshow(log(1 + abs(rotated_image_fft)), []);
    title(['Rotated in Fourier Domain (', num2str(angle), ' Degrees)']);
```

Source Code 5: Frequency Analysis of Clean and Noisy ECG Segments

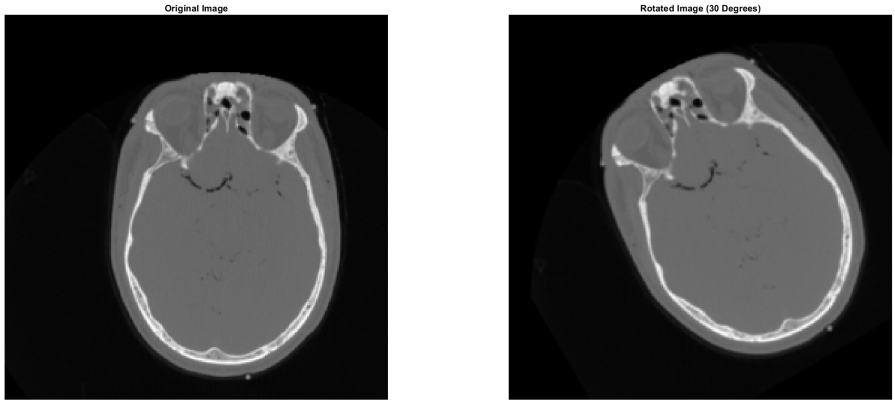

Figure 10: Original and Rotated Image

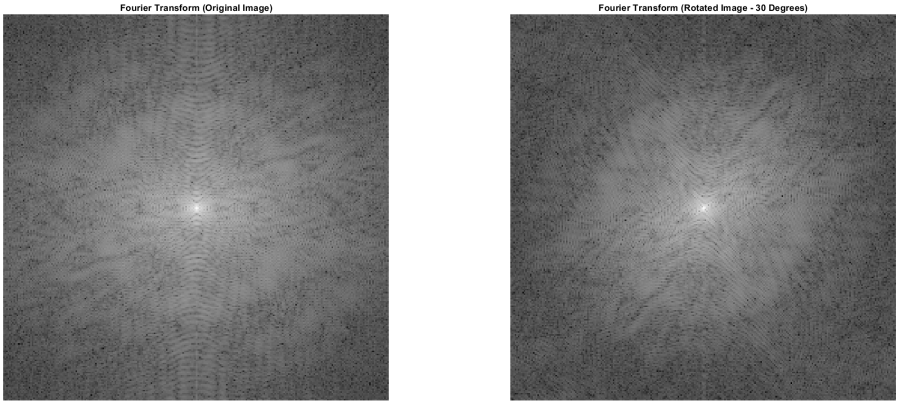

Figure 11: Original and Rotated Image in Frequency Domain

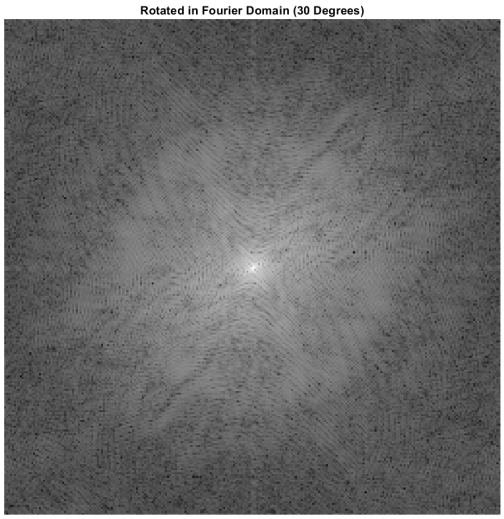

Figure 12: Rotated Image in Frdquency Domain

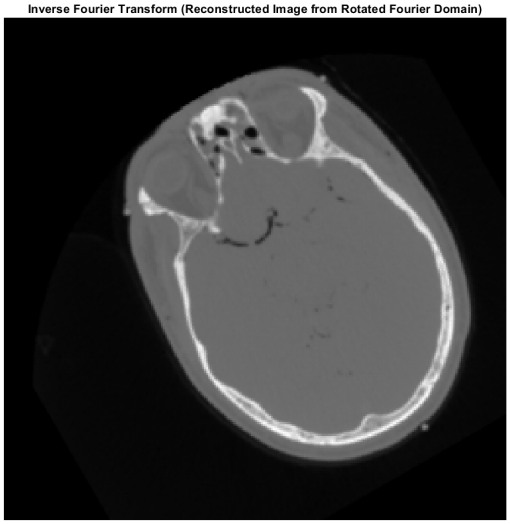

Figure 13: Rotated Image in Time Domain

Soloution

Description of Figure 4:

- The first image shows the original CT scan of the head in the spatial domain.
- The second image displays the same CT scan rotated by 30 degrees, achieved through spatial transformation.

Analysis of Figure 4:

- The rotation operation introduces a geometrical change in the spatial representation of the image while preserving its intensity values.
- This operation demonstrates how transformations in the spatial domain modify the orientation of the image but not its inherent content.

Description of Figure 5:

- The left image represents the Fourier Transform (frequency domain representation) of the original image.
- The right image represents the Fourier Transform of the rotated image.

Analysis of Figure 5:

- The Fourier Transform of the original image reveals symmetrical patterns, with intensity concentrated near the center (low frequencies) and diminishing outward (high frequencies).
- The Fourier Transform of the rotated image shows the same frequency components but with a corresponding rotation in the frequency domain to match the spatial transformation.
- This rotation property is consistent with Fourier Transform theory: spatial domain rotations result in equivalent frequency domain rotations.

Q5: Gradient Calculation and Edge Detection

The goal of this task is to produce results similar to Figure 4-1. To this end, read the image `t1.jpg` from the folder `S1_Q5_utils`. Then, using the `circshift` function, compute the central difference derivatives in both vertical and horizontal directions.

Next, calculate the gradient magnitude for the image and display all the results, including:

- Original image,
- Vertical derivative,
- Horizontal derivative,
- Gradient magnitude.

Finally, analyze the results in terms of detecting horizontal, vertical, and diagonal edges alongside the original image.

```
% Step 1: Read the Image
image = imread('S1_Q5_utils/t1.jpg');
image_gray = rgb2gray(image); % Convert to grayscale if necessary

% Display the original image
figure;
subplot(1, 4, 1);
imshow(image_gray, []);
title('Original Image');

% Step 2: Compute Central Differences Using circshift
% Vertical derivative (dy)
dy = circshift(image_gray, [-1, 0]) - circshift(image_gray, [1, 0]);

% Horizontal derivative (dx)
dx = circshift(image_gray, [0, -1]) - circshift(image_gray, [0, 1]);

% Display the vertical derivative
subplot(1, 4, 2);
imshow(dy, []);
title('Vertical Derivative (dy)');

% Display the horizontal derivative
subplot(1, 4, 3);
imshow(dx, []);
title('Horizontal Derivative (dx)');

% Step 3: Compute Gradient Magnitude
gradient_magnitude = sqrt(double(dx).^2 + double(dy).^2);

% Display the gradient magnitude
subplot(1, 4, 4);
imshow(gradient_magnitude, []);
title('Gradient Magnitude');
```

Source Code 6: Classification Criteria for Normal and Abnormal ECG

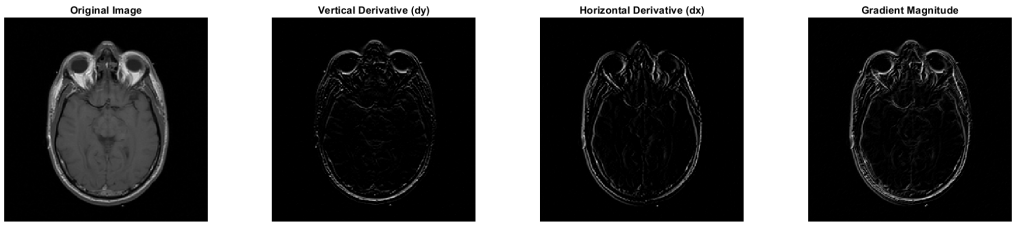

Figure 14: Images

Soloution

Analysis:

- The **Original Image** represents the grayscale CT scan of the brain, showing structural details.
- The **Vertical Derivative (dy)** highlights edges in the vertical direction, such as horizontal features in the image.
- The **Horizontal Derivative (dx)** emphasizes edges in the horizontal direction, making vertical structures prominent.
- The **Gradient Magnitude** combines both horizontal and vertical derivatives, producing a comprehensive representation of all edges and structural boundaries in the image.
- Overall, this visualization demonstrates how image derivatives and gradients can effectively capture edge information and emphasize significant structural features in medical images.

Q6: Advanced Edge Detection Methods

Investigate advanced image edge detection methods, such as the **Canny** and **Sobel** methods. Apply these methods to the image from the previous question and analyze the results.

Finally, compare the results of these methods with the central difference method used earlier and discuss the differences.

```
% Step 1: Read the image
image = imread('S1_Q5_utils/t1.jpg');
image_gray = rgb2gray(image); % Convert to grayscale if necessary

% Display the original image
figure;
imshow(image_gray, []);
title('Original Image');

% Step 2: Apply Sobel edge detection
sobel_edges = edge(image_gray, 'sobel');

% Display Sobel edge detection result
figure;
imshow(sobel_edges, []);
title('Sobel Edge Detection');

% Step 3: Apply Canny edge detection
```

```
    canny_edges = edge(image_gray, 'canny');

    % Display Canny edge detection result
    figure;
    imshow(canny_edges, []);
    title('Canny Edge Detection');

    % Step 4: Compare with previous gradient magnitude
    % Compute gradient magnitude using central difference (reuse previous code)
    dx = circshift(image_gray, [0, -1]) - circshift(image_gray, [0, 1]);
    dy = circshift(image_gray, [-1, 0]) - circshift(image_gray, [1, 0]);
    gradient_magnitude = sqrt(double(dx).^2 + double(dy).^2);

    % Display the comparison of all methods
    figure;
    subplot(1, 3, 1);
    imshow(gradient_magnitude, []);
    title('Gradient Magnitude');

    subplot(1, 3, 2);
    imshow(sobel_edges, []);
    title('Sobel Method');

    subplot(1, 3, 3);
    imshow(canny_edges, []);
    title('Canny Method');
```

Source Code 7: Classification Criteria for Normal and Abnormal ECG

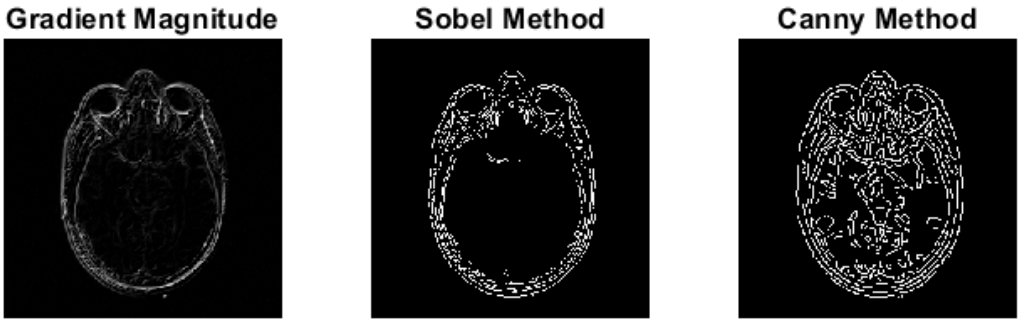

Figure 15: Comparison

Soloution

Sobel Edge Detection The Sobel edge detection method calculates gradients in the horizontal and vertical directions to emphasize regions of high spatial frequency that correspond to edges. The Sobel operator uses two convolution kernels:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

These kernels highlight edges by computing a weighted gradient in the image. The Sobel method is computationally efficient and primarily detects prominent edges, but it may be less effective for fine details. **Canny Edge Detection** The Canny edge detection method is more advanced and consists of the following steps:

1. Smoothing: The image is first smoothed using a Gaussian filter to reduce noise.
2. Non-Maximum Suppression: Spurious edges are removed by keeping only the local maxima of the gradient.
3. Hysteresis Thresholding: Two thresholds are applied to detect strong and weak edges. Weak edges connected to strong edges are preserved, while isolated weak edges are discarded.
4. Gradient Calculation: The gradient magnitude and direction are computed using finite differences.

The Canny method is more robust than Sobel, as it includes noise reduction and a systematic approach to preserve only meaningful edges.

Comparison of Results

- The Sobel method is simple and effective for detecting strong edges but may struggle with noise and fine details.
- The Canny method produces cleaner edges by incorporating noise suppression and multi-stage edge detection.
- Compared to the central difference method, both Sobel and Canny provide more structured and reliable edge detection, with Canny being the most advanced for identifying precise edges.