IMAGE AS A 2D - SIGNAL

By Team 1:

Gnanada Vanarpathi -SE24UCAM003

Gnapika Chebrolu -SE24UCAM007

Shoa Ahmed -SE24UCAM039

Pratheeksha Sumesh - SE24UCAM040

Sruchitha Gandhi -SE24UCAM041

Vasavi Agarwal -SE24UCAM062

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our professor, Dr. Rakesh Kumar, for his invaluable guidance, encouragement, and continuous support throughout this project. We extend our thanks to all faculty members and classmates for their suggestions. Special thanks to our institution, Mahindra University, for giving us the opportunity to work on this project.

Contents

Introduction

What is a 2D signal?

A two-dimensional (2D) signal is a signal that varies with respect to two independent variables, typically represented by spatial coordinates x and y. It can be expressed as f(x,y), where the value of f denotes the signal's amplitude or intensity at a particular position.

In simple terms, while a one-dimensional (1D) signal like sound varies only with time, a 2D signal varies across a surface or plane. The most common example of a 2D signal is a digital image, which contains brightness or color information for each point in a rectangular grid of pixels.

Difference Between 1D and 2D Signals :

A **one-dimensional (1D) signal** is a function that varies with a single independent variable, most commonly time. Examples include sound waves or temperature readings, represented as f(t).A **two-dimensional (2D) signal**, on the other hand, varies with two independent spatial variables, x and y, and is expressed as f(x,y).

While 1D signals show how amplitude changes over time, 2D signals show how intensity or color varies over an area.
This difference is fundamental because many operations used for 1D signals, such as filtering or Fourier transformation, can be extended into two dimensions for image processing applications.

Image as a 2D signal

In signal processing, an image can be considered as a 2D function :

$$f(x, y)$$

Here,

- x and y denote the spatial coordinates (horizontal and vertical positions), and
- f(x,y) denotes the **intensity** or **brightness** value of the image at that pixel.

For a grayscale image, the intensity at each pixel is represented by the single function f(x,y), where the value typically ranges from 0 (black) to 255 (white). Thus, the entire image can be viewed as a 2D matrix of intensity values.

In the case of a color (RGB) image, there are three such 2D signals — one for each color channel: Red, Green, and Blue. These are denoted as $f_R(x, y)$, $f_G(x, y)$, and $f_B(x, y)$ respectively. Together, they form a three-dimensional data structure:

$$f(x, y) = [\, f_R(x, y), f_G(x, y), f_B(x, y) \,]$$

Each channel carries the intensity information for its respective color. When these three channels are combined, they produce the full-color image that we see.

In MATLAB, an RGB image is represented as a three-dimensional matrix of size M×N×3 , where M and N are the number of rows and columns (pixels), and the third dimension corresponds to the color planes. Each color channel can be processed separately or together, depending on the desired operation such as shifting, scaling, rotation, filtering, or Fourier transformation.

Thus, an RGB image can be understood as a combination of three 2D signals, making image processing a direct application of 2D signal analysis techniques in the spatial domain.

Applications :

Image processing based on 2D signal analysis has wide applications in real-world systems such as medical imaging, satellite data processing, facial recognition, remote sensing, robotics, and industrial quality control. Techniques like filtering and Fourier transforms are used to enhance, analyze, and extract meaningful information from images for automated decision-making.

Types of Digital Images :

Digital images can be categorized according to the number of intensity or color components stored per pixel:

- **Binary Images:** Contain only two intensity levels, 0 and 1, representing black and white.
- **Grayscale Images:** Each pixel has a single intensity value ranging from 0 to 255, representing different shades of gray.
- **RGB (Color) Images:** Each pixel consists of three intensity values corresponding to the Red, Green, and Blue channels. These three 2D signals combine to form a full-color image.
- **Indexed Images:** Use a color lookup table (palette) where each pixel stores an index pointing to a predefined color value.

Different types of images are used based on the application—binary images for segmentation, grayscale for edge detection, and RGB for color analysis.

Concepts of Sampling and Quantization :

1] Sampling

Sampling is the process of converting a continuous image into a **discrete** form by measuring its intensity values at specific intervals in the spatial domain.

The number of samples taken per unit distance determines the **resolution** of the image.

- Higher sampling → better resolution (more detail)
- Lower sampling → loss of detail or blocky appearance

Mathematically, sampling converts the continuous function f(x,y) into discrete pixel values f[m,n], where m and n are integer pixel coordinates.

2] Quantization

Quantization involves mapping the continuous range of intensity values into a **finite set of discrete levels**.
For example, in an 8-bit image, intensity values are quantized to 256 levels (0–255).

- Higher quantization levels → smoother tone transitions
- Lower quantization levels → posterization or color banding

Together, **sampling** and **quantization** make it possible to represent and store an image digitally.

Digital Image Formation :

A **digital image** is formed by capturing a continuous visual scene using an image sensor (such as a camera or scanner).
The sensor converts light energy from the real world into an electrical signal, which is then **sampled** and **quantized** to create a digital version.

Sampling determines how many pixels are used to represent the image (spatial resolution), while quantization assigns discrete intensity levels to those pixels.

The result is a digital image that can be represented as a matrix of numbers and processed using computers.This conversion from the continuous world to a discrete digital form is what allows software like MATLAB to manipulate and analyze images efficiently.

Basic Image Properties :

Every digital image has a set of basic properties that define its structure and quality:

1] Resolution: The total number of pixels in the image, usually expressed as width × height.

2] Aspect Ratio: The ratio of width to height, which determines the image's shape.

3] Bit Depth: The number of bits used to represent each pixel's intensity. An 8-bit image allows 256 intensity levels per channel.

4] Color Model: The mathematical model used to represent colors, such as RGB, HSV, or CMY.

5] File Format: Defines how image data is stored (e.g., JPEG, PNG, TIFF).

Understanding these properties helps determine how an image will respond to operations such as filtering, scaling, and compression.

Coding: Basic Image Operations in MATLAB :

Read and Display Image:

```
img = imread('peppers.png');      % Read the RGB image
imshow(img);                      % Display the image
title('Original RGB Image');
```

Convert to Grayscale

```
gray = rgb2gray(img);             % Convert to grayscale
figure;
imshow(gray);
title('Grayscale Image');
```

Plot Image Histogram

```
figure;
imhist(gray);                     % Display histogram
title('Histogram of Grayscale Image');
xlabel('Pixel Intensity');
ylabel('Frequency');
```

The peppers.png image was used as the test image. The original image was first read and displayed using the imread and imshow functions. It was then converted into a grayscale image using rgb2gray. Finally, the intensity distribution of the grayscale image was visualized through its histogram using the imhist function.

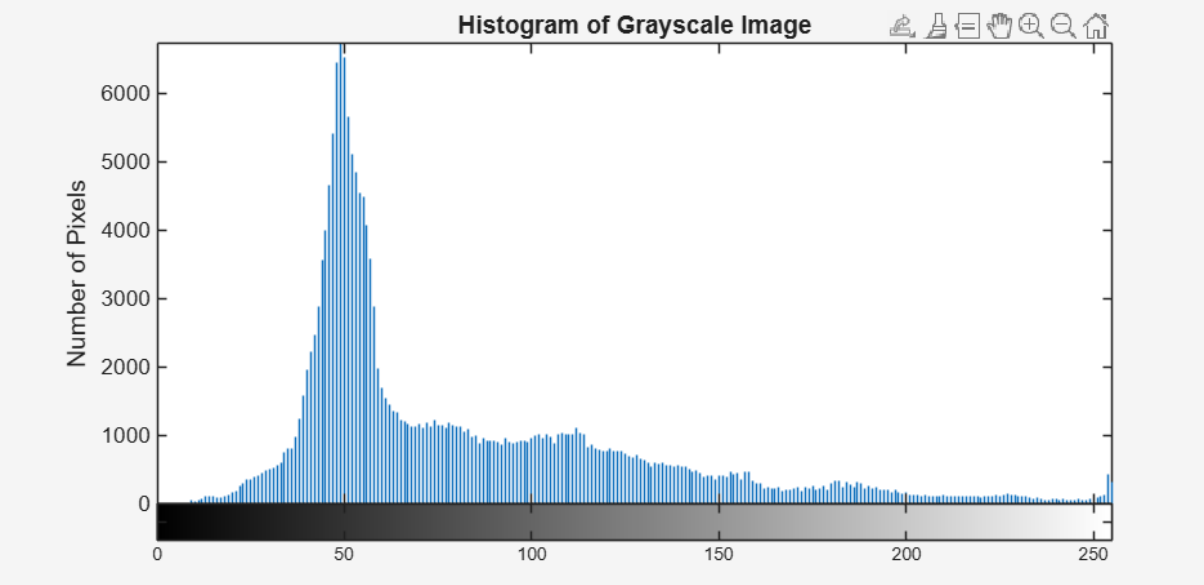
- The **original RGB image** shows rich color variations, representing three 2D signals (R, G, B).
- The **grayscale image** combines color channels into a single intensity signal f(x,y)f(x, y)f(x,y).
- The **histogram** displays the frequency of each intensity level, illustrating how pixel brightness is distributed across the image.

Output: Image grayscale demonstration:

Original RGB Image :

Grayscale Image :

Histogram :



Scaling an Image

- An image $f(x, y)$is a 2-D signal.
- **Scaling** means changing the spacing of pixels along the x and y axes:

$$g(x, y) = f(ax, by)$$

where

  - $a, b < 1$: **enlargement** (zoom in)
  - $a, b > 1$: **reduction** (zoom out)
- In MATLAB, imresize performs this operation by interpolation.
- When scaling up, new pixel values are estimated (interpolated).
- When scaling down, pixels are dropped, reducing resolution.

Enlarge image

```
scale_up = imresize(img, 2.5);        % Scale by 250%
figure, imshow(scale_up);             % Display the image
```

Shrink image

```
scale_down = imresize(img, 0.5);          % Scale by 50%
figure, imshow(scale_down);               % Display the image
title('Scaled Down (Zoomed Out) Image');
```

Output : Image Scaling Demonstration:

- Original RGB image

- Scaled up (zoomed in) Image

- Scaled Down (zoomed out) Image

Interpretation:

From the scaling operation, it is observed that enlarging an image increases its apparent size but may cause slight blurring due to interpolation, while reducing it decreases the resolution as some pixel information is lost. This transformation clearly illustrates the behavior of an image as a two-dimensional signal—when spatial coordinates are stretched, the image expands, and when they are compressed, the image contracts. Scaling thus plays a vital role in various applications such as zooming, resizing, pattern recognition, and multi-resolution analysis in image processing.

Shifting and Translation:

Objective:

To implement horizontal and vertical image translation using MATLAB and analyze its mathematical behavior as a 2D signal transformation**.**

Mathematical Model:

The mathematical model for image translation is given by:

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

where

- x_0 : shift along horizontal axis
- y_0 : shift along vertical axis

Translation moves the image spatially without changing pixel intensity. If $x_0 > 0$, the image moves right;if $y_0 > 0$; it moves downward.

Code:

```
clc; clear; close all;

img = imread('peppers.png');


% Figure 2.1 – Original Image

figure, imshow(img);

title('Figure 2.1: Original RGB Image (Peppers)');

saveas(gcf, 'Figure_2_1_Original.png');


% Figure 2.2 – Horizontal Shift
```

```
% Figure 2.3 – Vertical Shift
shift_v = circshift(img, [60, 0]);
figure, imshow(shift_v);
title('Figure 2.3: Vertically Shifted Image (Down by 60 pixels)');
saveas(gcf, 'Figure_2_3_VerticalShift.png');
% Figure 2.4 – Both Shifts
```

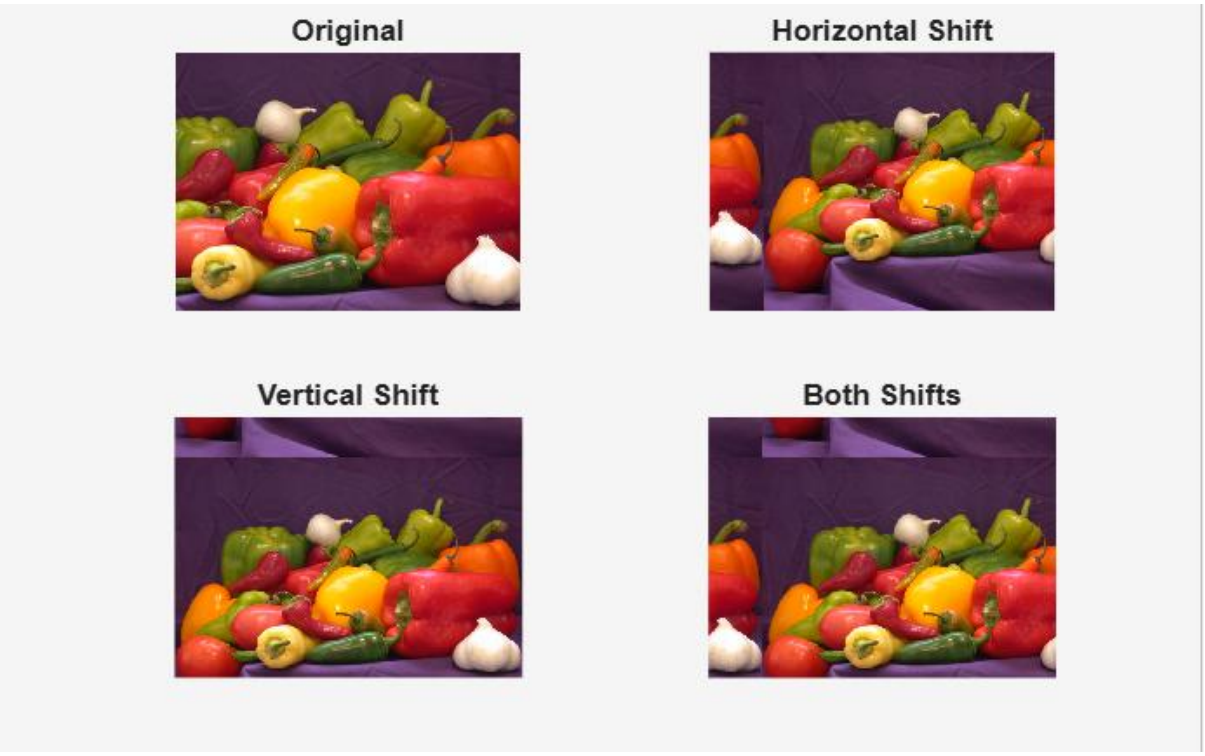
```
% Figure 2.5 – Combined Comparison
figure;
subplot(2,2,1), imshow(img), title('Original');
subplot(2,2,2), imshow(shift_h), title('Horizontal Shift');
subplot(2,2,3), imshow(shift_v), title('Vertical Shift');
```

Explanation:

In translation, the pixel values are not altered - only their spatial positions are changed.

This demonstrates the property of a 2D signal where spatial coordinates can be shifted without affecting signal amplitude. The operation is helpful in tasks like motion estimation, alignment, and object tracking.

Output: Image Translation Demonstration



Applications:

- Satellite image alignment – to match images taken at different times.
- Motion tracking – detects movement between consecutive frames.
- Image stitching – aligns photos to create panoramas.
- Medical imaging – used for aligning MRI or CT scan slices.

Rotation and Reflection :

This section focuses on two important image processing transformations Rotation and Reflection (Flipping).

Rotation changes the orientation of an image, while reflection creates a mirror version of it. Both are essential for correcting image orientation, analyzing symmetry, and enhancing visual understanding

Rotation:

Image rotation is a geometric transformation that turns an image around a fixed point (usually the origin) by an angle θ\thetaθ.

Each pixel's new position after rotation is calculated using the equations:

Mathematical Background :

When an image is rotated by an angle θ, every pixel *(x,y)* moves to a new position (x′,y′) according to the equations:

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

Or equivalently,

$$f'(x,y) = f(x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta)$$

These formulas describe how rotation preserves the image's structure while changing its orientation.

Code:

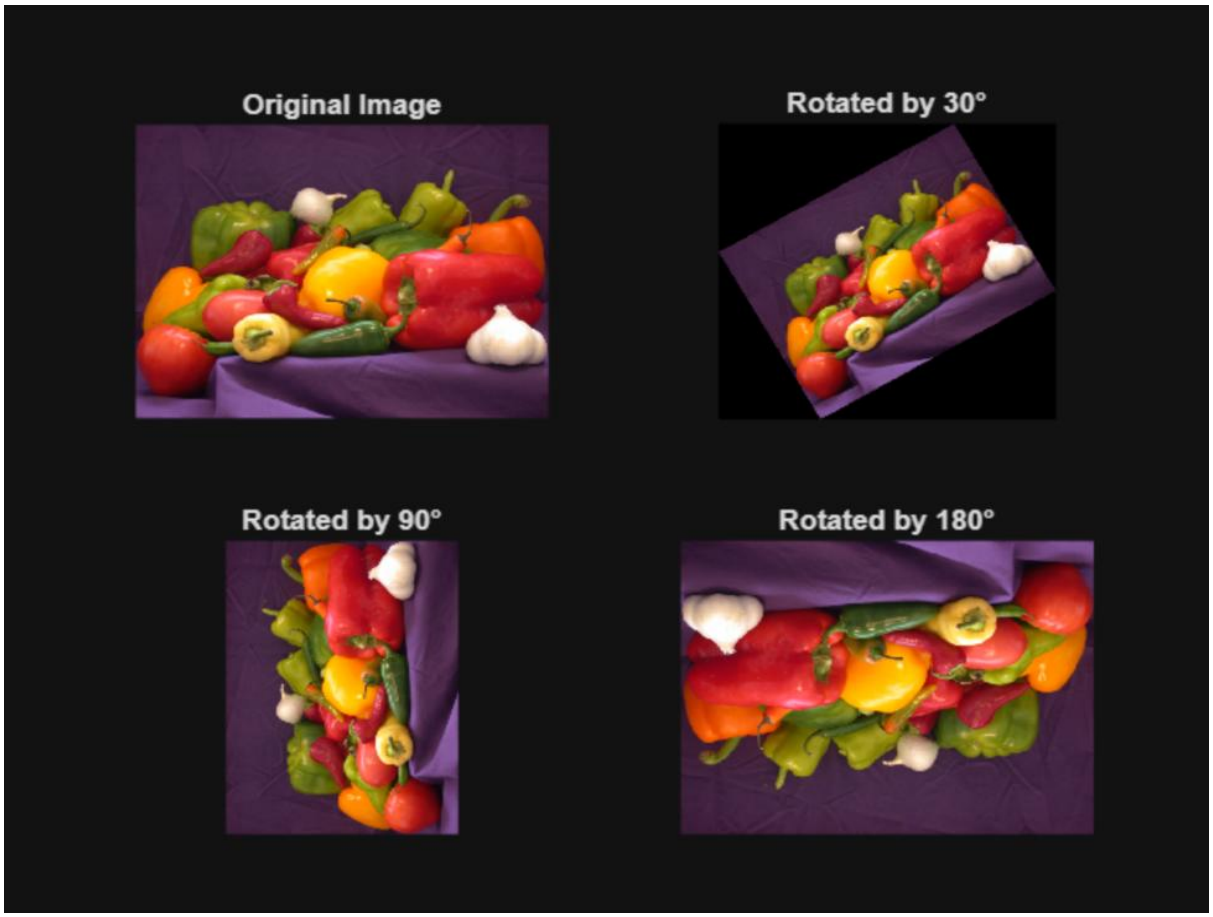
```
rotated30 = imrotate(img, 30);    % Rotate the image by 30 degrees
rotated90 = imrotate(img, 90);    % Rotate the image by 90 degrees
rotated180 = imrotate(img, 180);  % Rotate the image by 180
degrees

figure;
subplot(2,2,1); imshow(img); title('Original Image');
subplot(2,2,2); imshow(rotated30); title('Rotated by 30°');
subplot(2,2,3); imshow(rotated90); title('Rotated by 90°');
subplot(2,2,4); imshow(rotated180); title('Rotated by 180°');
```

Explanation:

- imrotate(image, angle) rotates the image **counterclockwise** by the specified angle (in degrees).
- subplot(m,n,p) divides the window into a grid of $m \times n$ small plots and selects position p for the next image.
- Here we use subplot(2,2,…) so we can show four images together.

Output: Image Rotation Demonstration:



Applications:

- Image orientation correction
- Data augmentation in machine learning
- Medical and satellite image analysis
- Robotics vision and pattern recognition

Reflection:

Image reflection (also called *flipping*) creates a mirror version of an image.
There are two main types of reflection:

- **Horizontal Flip:** Mirrors the image from left to right.
- **Vertical Flip:** Mirrors the image from top to bottom.

These transformations can be achieved by reversing the order of pixel coordinates along specific axes.

Code:

```
% Perform horizontal and vertical reflections
flip_horizontal = fliplr(img);   % Flip left to right
flip_vertical   = flipud(img);   % Flip top to bottom

% Display flipped images
figure;
subplot(1,3,1); imshow(img); title('Original Image');
subplot(1,3,2); imshow(flip_horizontal); title('Horizontally Flipped');
subplot(1,3,3); imshow(flip_vertical); title('Vertically Flipped');
```

Explanation:

- fliplr(img) — flips the image along the vertical axis (left ↔ right).
- flipud(img) — flips the image along the horizontal axis (top ↔ bottom).
- Reflection is useful in symmetry analysis, mirror effects, and geometric transformations.

Output :Image Reflection Demonstration:

Applications:

- Symmetry detection and pattern analysis
- Creating mirror effects in design
- Data preprocessing for machine learning
- Enhancing visual aesthetics

The rotation and reflection transformations demonstrate how image geometry can be manipulated using MATLAB.

These operations are fundamental in digital image processing, providing the basis for more advanced techniques like scaling, translation, and affine transformations.

Image Filtering in the spatial domain

Image filtering is one of the most fundamental operations in digital image processing. Filtering refers to the process of modifying or enhancing an image by emphasizing certain information while suppressing others. Since a digital image is treated as a **two-dimensional signal**, filtering techniques are applied to manipulate pixel intensity values based on their spatial arrangement. Common objectives of filtering include **noise reduction**, **smoothing**, **edge enhancement**, and **feature extraction**.

In spatial domain filtering, operations are performed directly on pixel values in the image plane. The output value of a pixel is determined by applying a mathematical operation to the pixel and its neighboring pixels, defined by a small matrix called a **filter mask** or **kernel**. The manner in which the kernel interacts with the image is governed by **convolution** or **correlation**, which are essential concepts in signal processing.

Code:

```
I = imread('peppers.png');

figure; imshow(I); title('Orginal RGB Image');

grayI = rgb2gray(I);

figure; imshow(grayI); title('Greyscale Image');
```

Output:Image filtering demonstration

- Original Image

- Grey scale Image

Convolution and correlation in image processing

Correlation

Correlation is used to measure similarity between a filter kernel and a region of the image. Given an image **f(x,y)** and a filter kernel **h(i,j),** the correlation operation is expressed as:

$$g(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} f(x+i, y+j)h(i,j)$$

Here, each output pixel g(x,y)g(x,y)g(x,y) is obtained by sliding the kernel over the image and computing a weighted sum of neighboring pixel values.

Correlation **does not reverse** the filter kernel.

Convolution

Convolution is similar to correlation, but the filter kernel is rotated by **180 degrees** before the weighted sum is computed. Convolution is mathematically defined as:

$$g(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} f(x+i, y+j)h(-i,-j)$$

Convolution is the operation formally used in signal processing and aligns with the Fourier Transform property:

convolution in spatial domain ↔ Multiplication in frequency domain

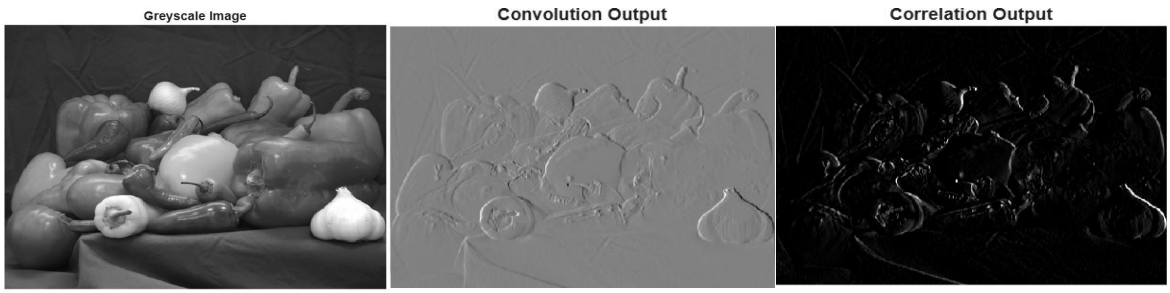
In image processing, many kernels are symmetric, so convolution and correlation produce identical results in practice. However, the distinction remains important in theory.

```
kernel = [1 0 -1; 1 0 -1; 1 0 -1];   % Vertical edge detector


% Convolution (mask flipped)
I_conv = conv2(double(grayI), double(kernel), 'same');
figure; imshow(I_conv, []); title('Convolution Output');


% Correlation (mask not flipped)
I_corr = imfilter(grayI, kernel, 'replicate');
figure; imshow(I_corr, []); title('Correlation Output');
```

Output : Image convolution Demonstration



Spatial Domain Filters

Spatial filtering modifies images through direct pixel manipulation using masks (kernels). These filters are categorized into two main classes:

Linear spatial Filters

Linear filters compute each output pixel as a **linear combination of its neighborhood**. They are widely used for smoothing (blurring) and sharpening.

Smoothing (Low-Pass) Filters

These filters reduce variations in pixel values, thereby suppressing noise and reducing detail. They are often used in applications like medical imaging, preprocessing, and object detection.

(a)Mean Filter

The mean filter replaces each pixel with the **average** of its neighbors. For a 3×3 filter:

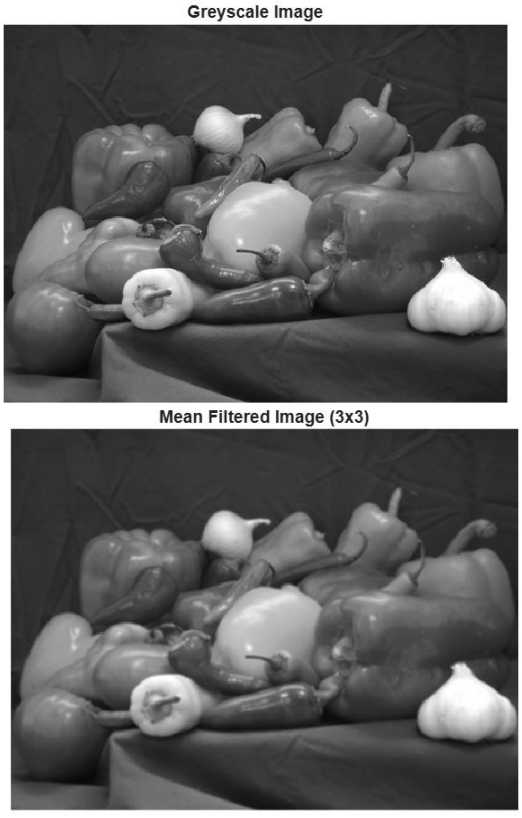
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

It reduces random noise but also causes **loss of fine details**.

Code:

```
meanFilter = fspecial('average', [3 3]);
I_mean = imfilter(grayI, meanFilter, 'replicate');
figure; imshow(I_mean); title('Mean Filtered Image (3x3)');
```

Output :Low – Pass mean filter Demonstration



(b)Gaussian smoothing filter

The Gaussian filter applies a weighted average using a Gaussian distribution, giving more importance to pixels near the center:

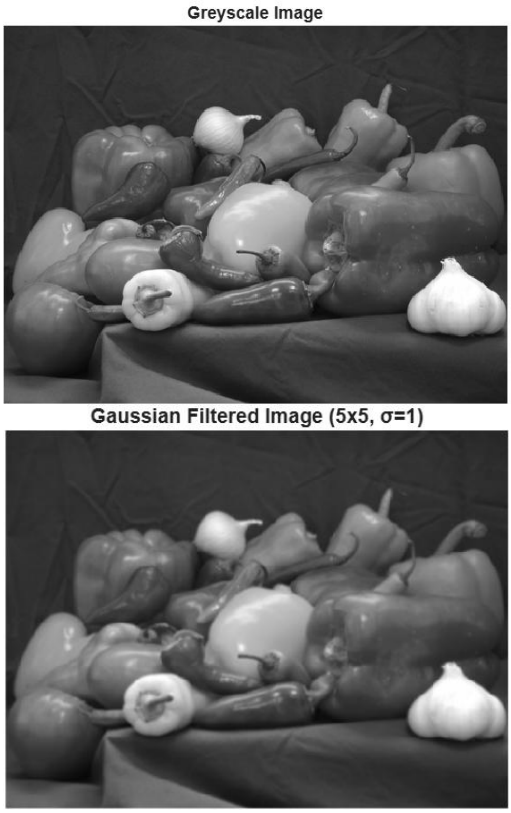
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

It produces natural-looking blur and is preferred for denoising.

Code:

```
gaussFilter = fspecial('gaussian', [5 5], 1.0);
  I_gauss = imfilter(grayI, gaussFilter, 'replicate');
figure; imshow(I_gauss); title('Gaussian Filtered Image (5x5, σ=1)');
```

Output: Gaussian Filtered image demonstration:



Sharpening (High-Pass) Filters

Sharpening filters highlight **edges** and **transitions**. They enhance high-frequency components.

(a)Laplacian Filter

The Laplacian detects regions of rapid intensity change:

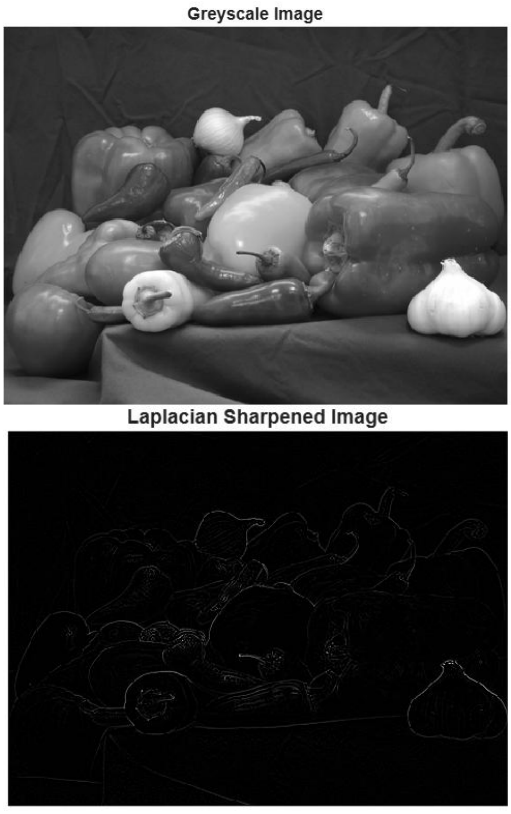
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -10 \\ 0 & -1 & \end{bmatrix}$$

Often used along with the original image

Code

```
lapFilter = fspecial('laplacian', 0.2);
I_lap = imfilter(grayI, lapFilter, 'replicate');
figure; imshow(I_lap, []); title('Laplacian Sharpened Image');
```

Output:Laplacian filter demonstration:



$$g(x,y) = f(x,y) + \alpha \cdot \nabla^2 f(x,y)$$

(b)Sobel and Prewitt Filter (Edge Detection)

These filters detect directional edges:

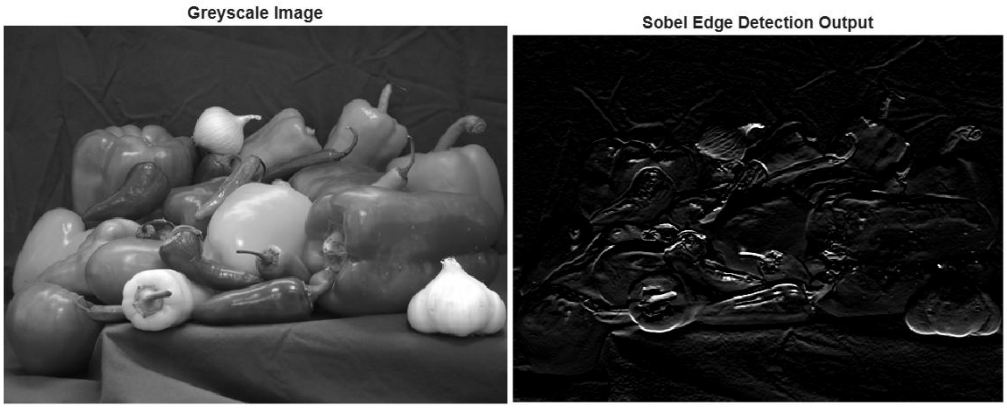
Horizontal sobel:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Code:

```
sobelFilter = fspecial('sobel');
I_sobel = imfilter(grayI, sobelFilter, 'replicate');
figure; imshow(I_sobel, []); title('Sobel Edge Detection Output');
```

Output: Sobel edge detection demonstration:



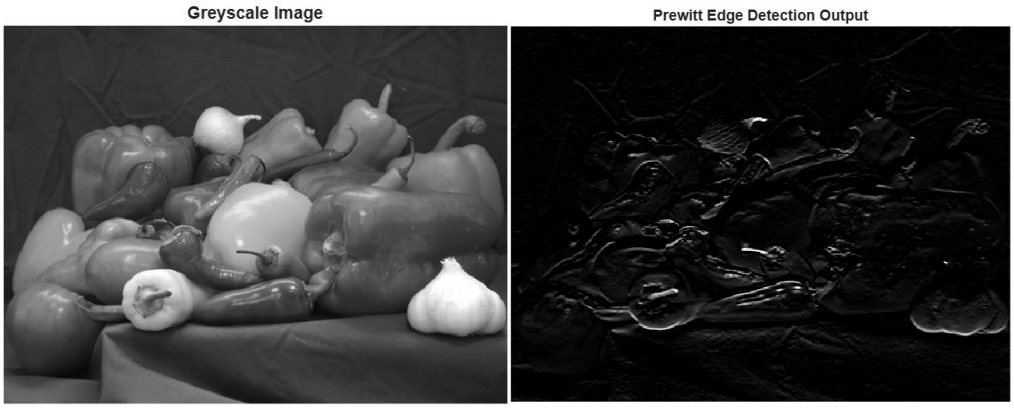
Vertical sobel:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Code:

```
prewittFilter = fspecial('prewitt');
I_prewitt = imfilter(grayI, prewittFilter, 'replicate');
figure; imshow(I_prewitt, []); title('Prewitt Edge Detection Output');
```

Output: Prewitt Edge Detection Demonstration:



They are essential in feature detection and segmentation tasks.

Non-Linear Spatial Filters

Non-linear filters do not rely on weighted neighborhood sums. They are especially useful for removing **impulse noise (salt-and-pepper noise)**.

Median Filter

The median filter replaces each pixel with the **median** value of its neighborhood. It preserves sharp edges better than mean filtering.

Example neighborhood:

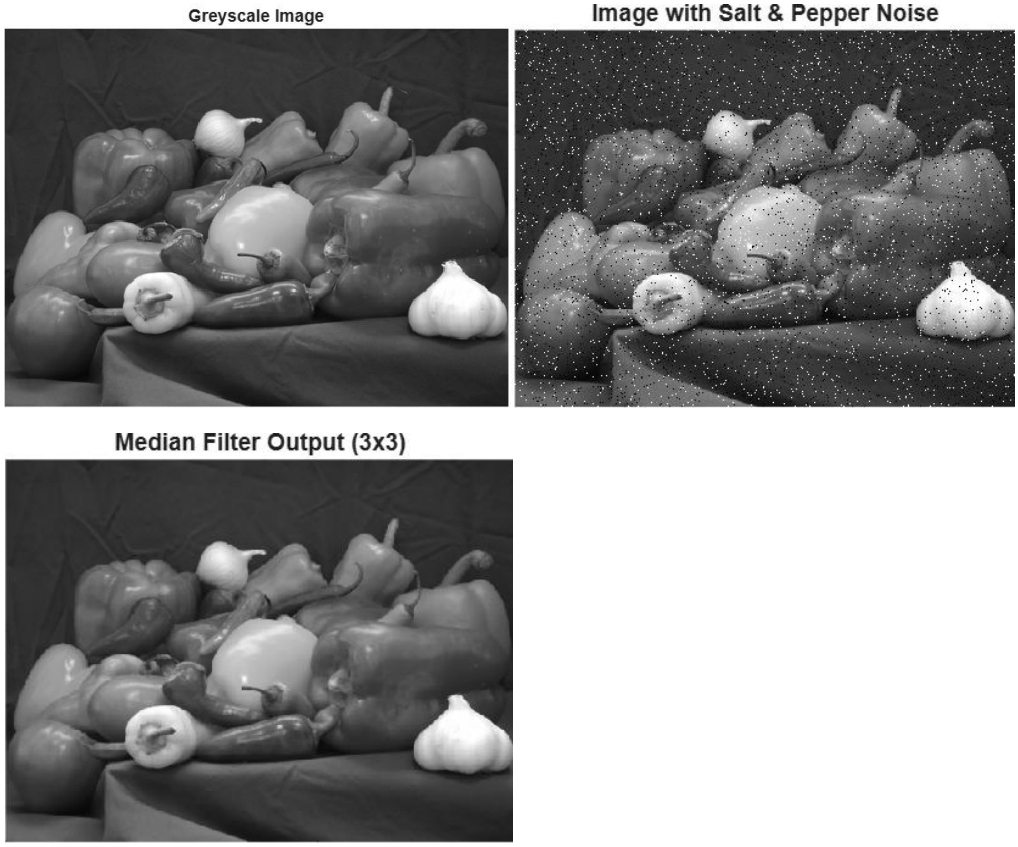
```
3 255 4
2 5   6
7 8   9
Median = 6
```

```
I_noisy = imnoise(grayI,'salt & pepper',0.04);  % Add noise for demonstration

figure; imshow(I_noisy); title('Image with Salt & Pepper Noise');
```

```
I_med = medfilt2(I_noisy,[3 3]);
figure; imshow(I_med); title('Median Filter Output (3x3)');
```

Output:Median filter demonstration:



Thus, the impulse value 255 is removed without blurring edges.

Application of Spatial Filtering

1. **Noise removal :**

   Mean and median filters smooth fluctuations

2. **Image enhancement:**

   Sharpening filters highlight important structures

3. **Edge detection:**

   Sobel, Prewitt, Laplacian extract boundaries

4. **Medical imaging:**

   Smoothing removes sensor artifacts

5. **Computer vision & object detection:**

   Feature maps obtained using convolution kernels

Frequency Domain Analysis:

Theoretical Foundation

Frequency Domain Representation

A 2D digital image f(x,y) can be equivalently represented in the frequency domain through the Discrete Fourier Transform (DFT). This transformation decomposes the image into constituent sinusoidal components, revealing frequency content not apparent in the spatial domain.

- Forward Transform (Analysis):

$$F(u,v) = \sum_{x=0}^{M-1} \quad \sum_{y=0}^{N-1} \quad f(x,y) \cdot \exp\left(-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right)$$

- Inverse Transform (Synthesis):

$$f(x,y) = \frac{1}{MN} \sum_{x=0}^{M-1} \quad \sum_{y=0}^{N-1} \quad F(u,v) \cdot \exp\left(-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right)$$

f(x,y) - spatial domain image

(**M×N -** pixels)

F(u,v) - frequency domain representation (complex-valued)

(u,v) - frequency coordinates

(x,y) - are spatial coordinates.

Complex Representation:

Each coefficient F(u,v) contains magnitude and phase:

$$F(u,v) = |F(u,v)| \cdot exp(j\phi(u,v))$$

Magnitude :

$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)}$$

(Indicates amplitude frequency component)

Phase:

$$\phi(u,v) = arctan\left(\frac{I(u,v)}{R(u,v)}\right)$$

(Indicates spatial positioning)

Frequency Interpretation

- Low frequencies (near center) represent smooth regions and gradual intensity changes.
- High frequencies (far from center) represent edges, fine details, and abrupt changes.
- DC component F(0,0) is the zero frequency term representing average image intensity.

Analysis 1: Transformation

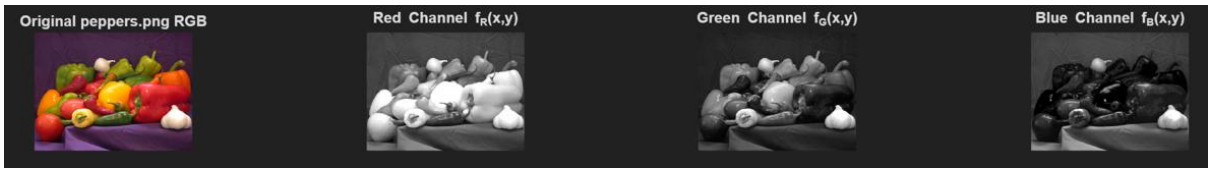
Theory:

The 2D DFT transforms spatial domain representation to frequency domain, enabling analysis of frequency content and efficient filtering operations.

Implementation:

```
F_R = fftshift(fft2(R));                    % 2D FFT red, DC centered
F_G = fftshift(fft2(G));                    % 2D FFT green, DC centered
F_B = fftshift(fft2(B));                    % 2D FFT blue, DC centered
```

Output: Image transformation demonstration:



Results:

- Frequency domain representation $F(u,v)$obtained for all channels.
- Complex coefficients contain complete image information.
- Transformation is lossless and reversible.

Inference:

Successful transformation enables subsequent frequency-based analysis and filtering operations. Each RGB channel transformed independently confirms multi-channel processing capability.

Analysis 2: Magnitude Spectrum

Theory

Magnitude spectrum $|F(u,v)|$ reveals frequency content distribution. Natural images typically show energy concentration at low frequencies.

Computation:

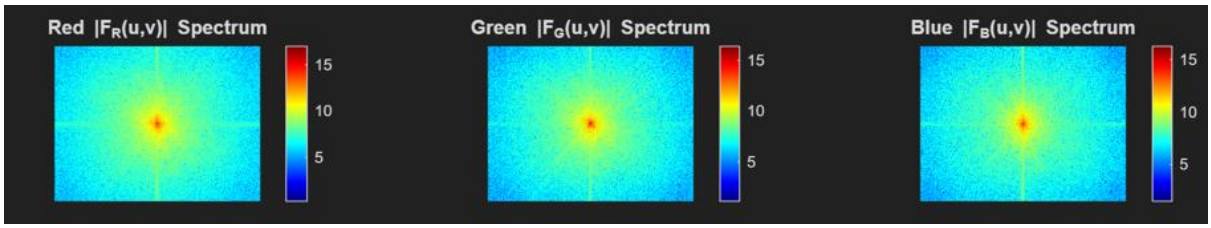
$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)}$$

Log scaling for visualization: $log(1 + |F(u,v)|)$

Implementation:

```
mag_R = abs(F_R);                    % Magnitude spectrum red |F_R(u,v)|
mag_G = abs(F_G);                    % Magnitude spectrum green |F_G(u,v)|
mag_B = abs(F_B);                    % Magnitude spectrum blue |F_B(u,v)|
log_mag_R = log(1+mag_R);            % Log scale for visualization red
log_mag_G = log(1+mag_G);            % Log scale for visualization green
log_mag_B = log(1+mag_B);            % Log scale for visualization blue
```

Plots:



Result:

Channel	**Spectrum Charecteristics**
Red	Bright center , gradual fade to edges
Green	Similar to red,slightly broader
Blue	Bright center,overall dimmer

Inference

All channels exhibit **bright central region** indicating dominant low-frequency content. This confirms peppers.png is a natural image with smooth regions dominating over high-frequency details. Peripheral dimming indicates minimal high-frequency energy.

Analysis 4: Energy Distribution

Theory

- Power spectrum :

$$P(u,v) = |F(u,v)|^2$$

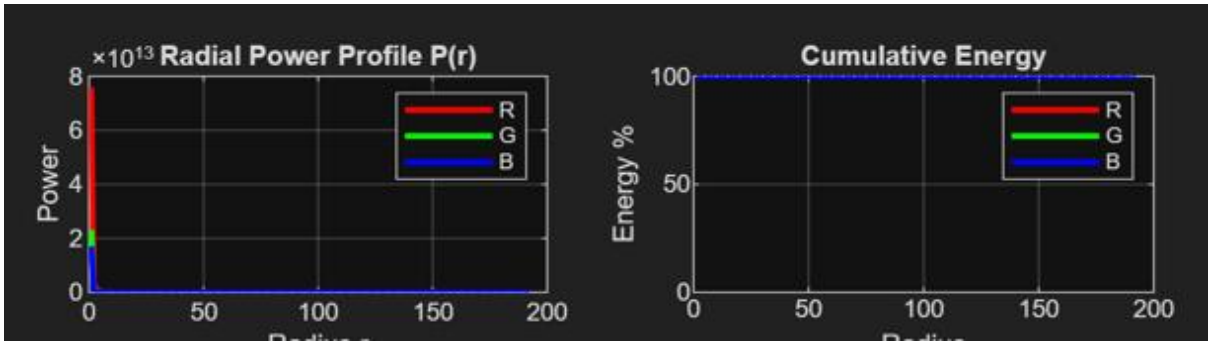
- Radial distance from DC :

$$D(u,v) = \sqrt{\left(u - \frac{M}{2}\right)^2 + \left(v - \frac{N}{2}\right)^2}$$

Implentation:

```
power_R = mag_R.^2;                                    % Power spectrum red P_R(u,v)=|F_R|²
power_G = mag_G.^2;                                    % Power spectrum green P_G(u,v)=|F_G|²
power_B = mag_B.^2;                                    % Power spectrum blue P_B(u,v)=|F_B|²
[U,V] = meshgrid(1:N,1:M);                             % Frequency coordinate grid
D = sqrt((U-center(2)).^2+(V-center(1)).^2);            % Radial distance D(u,v) from DC
max_r = floor(min(M,N)/2);                             % Maximum radius to analyze
rad_R = zeros(1,max_r);                                % Radial profile red storage
rad_G = zeros(1,max_r);                                % Radial profile green storage
rad_B = zeros(1,max_r);                                % Radial profile blue storage
for r=1:max_r                                          % Loop through each radius
    mask=(D>=r-0.5)&(D<r+0.5);                         % Circular ring mask at radius r
    rad_R(r)=mean(power_R(mask));                      % Average power at radius r (red)
    rad_G(r)=mean(power_G(mask));                      % Average power at radius r (green)
    rad_B(r)=mean(power_B(mask));                      % Average power at radius r (blue)
end
cum_R = cumsum(rad_R.^2)/sum(rad_R.^2)*100;            % Cumulative energy % red
cum_G = cumsum(rad_G.^2)/sum(rad_G.^2)*100;            % Cumulative energy % green
cum_B = cumsum(rad_B.^2)/sum(rad_B.^2)*100;            % Cumulative energy % blue
idx90_R = find(cum_R>=90,1);                           % Radius containing 90% energy red
idx90_G = find(cum_G>=90,1);                           % Radius containing 90% energy green
idx90_B = find(cum_B>=90,1);                           % Radius containing 90% energy blue
fprintf('90%% Energy Radius - R: %d, G: %d, B: %d\n', idx90_R, idx90_G, idx90_B);
```

Plots:



Results:

Metric	Red	Green	Blue
90% Energy Radius	1	1	1
% of Max Frequency	0.52%	0.52%	0.52%

Key Finding: 90% of total energy concentrated within radius 1 of DC component.

Inference

Extreme energy concentration in lowest frequencies (<1% of spectrum) indicates image is exceptionally smooth with gradual intensity variations. High-frequency content (edges/details) contributes minimally to total energy. This is characteristic of natural photographs with dominant low-frequency structure and explains high compressibility of natural images. Radial power profile shows steep exponential decay, confirming natural image $1/f^2$ power law characteristic.

Analysis 5: Filtering Operations

Theory:

- Convolution Theorem :

$$f(x,y) \otimes h(x,y) \Leftrightarrow F(u,v) \cdot H(u,v)$$

(Spatial convolution Frequency multiplication)

- Filtering:

$$G(u,v) = F(u,v) \cdot H(u,v)$$

$$g(x, y) = F^{-1}[G(u, v)]$$

Implementation:

```
D0=30;                                                    % Cutoff frequency D₀
H_LP=exp(-D.^2/(2*D0^2));                                 % Gaussian low-pass H_LP(u,v)
H_HP=1-H_LP;                                              % Gaussian high-pass H_HP(u,v)
G_LP_R = F_R.*H_LP;                                       % Filtered spectrum red LP: G=F·H
G_HP_R = F_R.*H_HP;                                       % Filtered spectrum red HP: G=F·H
img_LP_R = real(ifft2(ifftshift(G_LP_R)));                % Inverse FFT red LP → spatial
img_HP_R = real(ifft2(ifftshift(G_HP_R)));                % Inverse FFT red HP → spatial
G_LP_G = F_G.*H_LP;                                       % Filtered spectrum green LP
G_LP_B = F_B.*H_LP;                                       % Filtered spectrum blue LP
img_LP_G = real(ifft2(ifftshift(G_LP_G)));                % Inverse FFT green LP
img_LP_B = real(ifft2(ifftshift(G_LP_B)));                % Inverse FFT blue LP
img_LP = cat(3,uint8(img_LP_R),uint8(img_LP_G),uint8(img_LP_B)); % Reconstruct RGB low-pass image
```

Low-Pass Filtering (Smoothing)

Filter design:

Gaussian low-pass filter with cutoff $D_0 = 30$:

$$H_{LP}(u, v) = exp\left(\frac{-D^2(u,v)}{2D^2_0}\right)$$

Image:

Results:

- High frequencies attenuated (H(u,v) → 0 for large D).
- Spatial result shows blurred/smoothed image.

- Edge sharpness reduced while overall structure preserved.

Inference:

Low-pass filtering successfully removes high-frequency components (edges, noise) while preserving low-frequency structure. Demonstrates smoothing effect through frequency-selective attenuation. Validates convolution theorem: multiplication in frequency domain equivalent to convolution in spatial domain with computational efficiency gain.

High-Pass Filtering (Edge Enhancement)

Filter design:

Gaussian high-pass filter:

$$H_{HP}(u,v) = 1 - exp\left(\frac{-D^2(u,v)}{2D_0^2}\right)$$

Image:

Results:

- Low frequencies suppressed (H(u,v) → 0 for small D).
- High frequencies preserved (H(u,v) → 1 for large D).
- Spatial result shows edges and boundaries only. Smooth regions removed.

Inference:

High-pass filtering isolates high-frequency content, effectively performing edge detection. Removal of low frequencies eliminates smooth regions, leaving only abrupt intensity changes (edges). Demonstrates that edges are high-frequency phenomena in frequency domain.

Bibliography:

- Digital Image Processing Using Matlab (Gonzalez)

Conclusion

This project successfully demonstrated the fundamental concept of treating digital images as two-dimensional signals and applying signal processing techniques for their analysis and manipulation. Through practical implementation in MATLAB, we explored essential operations including geometric transformations (scaling, translation, rotation, and reflection), spatial domain filtering using various kernels for smoothing and edge detection, and frequency domain analysis using the Discrete Fourier Transform.