



MSc in Digital Humanities
Image Processing/Analysis

Image Processing/Analysis

Anestis Koutsoudis, Chairi Kiourt, Christos Tsinos, George Pavlidis

Image Enhancement

Christos Tsinos

e-mail: ctsinos@uoa.gr



Overview

- Introduction
- Categories of Image Enhancement
- Spatial Domain Techniques
- Frequency Domain Techniques
- Edge Detection
- Color Image Enhancement

Introduction

- **Definition:** Image enhancement is the process of improving the visual appearance of an image or making it more suitable for specific applications, such as analysis or interpretation
- **Goals**
 - improve contrast
 - highlight details
 - remove noise
 - enhance specific features within an image

Introduction

- **Applications**
 - **Medical Imaging:** Enhancing MRI scans and X-rays for accurate diagnosis
 - **Satellite Imagery:** Improving clarity for land classification and environmental monitoring
 - **Photography:** Refining images for professional and artistic purposes
 - **Forensics:** Clarifying CCTV footage for investigations
 - **Remote Sensing:** Enhancing images for agricultural and climate studies.

Categories of Image Enhancement

- **Spatial Domain Techniques:** Directly manipulate the pixel values of an image to achieve enhancement
 - Contrast adjustment
 - Histogram equalization
 - Spatial filtering
- **Advantages:** Simple to implement and computationally efficient
- **Limitations:** May not effectively enhance features in complex images

Categories of Image Enhancement

- **Frequency Domain Techniques:** Modify the image's frequency representation (Fourier Transform) to enhance specific features
 - **Low-pass Filtering:** Smooths images by removing high-frequency noise
 - **High-pass Filtering:** Enhances edges and fine details by emphasizing high-frequency components
- **Advantages:** Effective in emphasizing or suppressing certain image features
- **Limitations:** Requires complex transformations and may introduce artifacts.

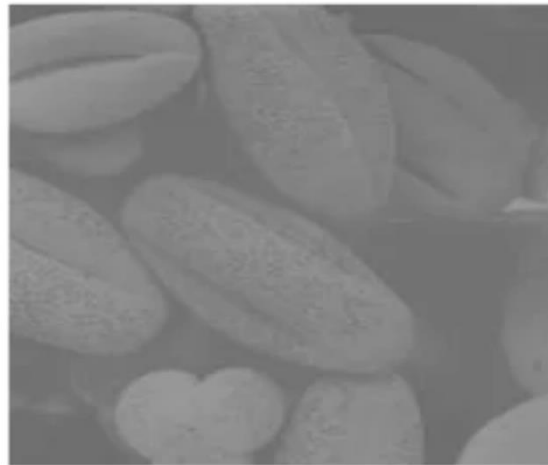
Categories of Image Enhancement

- **Hybrid Techniques:** Combine spatial and frequency domain approaches to achieve optimal enhancement
 - Multi-scale enhancement methods that apply filtering in both domains for improved results
- **Adaptive Techniques:** Modify enhancement parameters dynamically based on local image characteristics
 - Adaptive histogram equalization (AHE)
 - Contrast-limited adaptive histogram equalization (CLAHE).

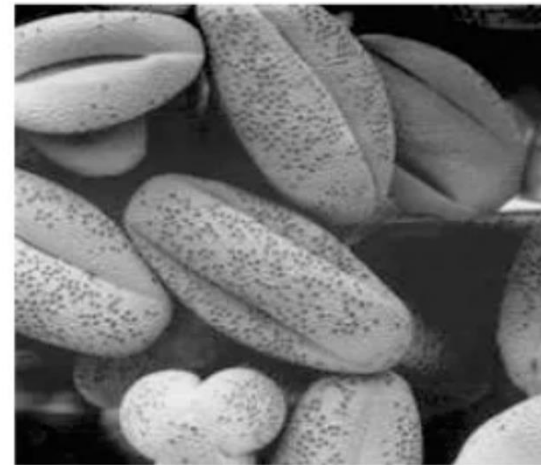
Spatial Domain Techniques

- **Contrast Adjustment:** Modifying pixel intensity values to improve image visibility

Formula: $I' = \alpha I + \beta$ (where I is the input image, α is contrast gain, β is brightness adjustment)



Original Image



Contrast Enhanced Image

Spatial Domain Techniques

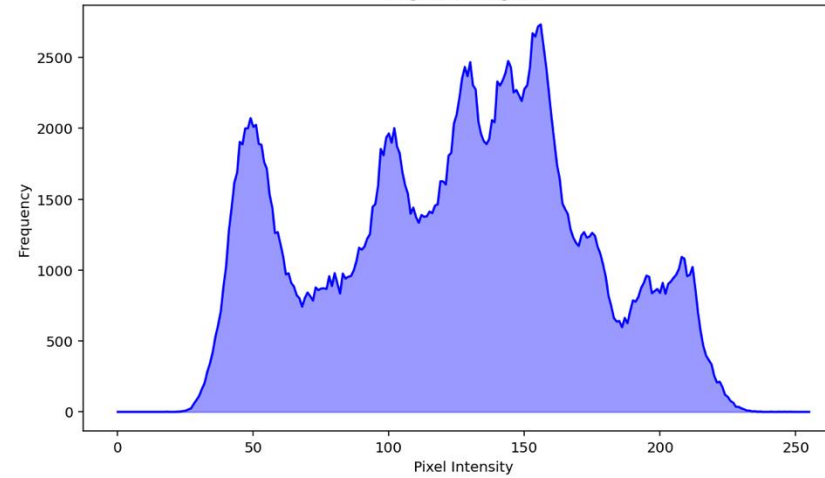
- **Histogram Equalization:** A technique for improving image contrast.
- **Mathematical Concept:**
 - Compute histogram of pixel intensities
 - Calculate cumulative distribution function (CDF)
 - Map original pixel values to new intensity values based on the CDF
- **Advantages:** Effective for enhancing contrast in grayscale images.

Spatial Domain Techniques

Original Image



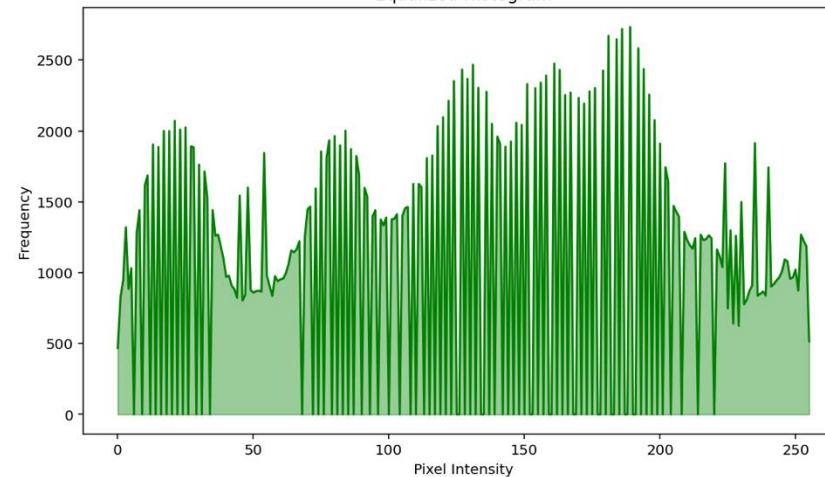
Original Histogram



Equalized Image



Equalized Histogram



Spatial Domain Techniques

- **Image smoothing** is a fundamental technique in image processing aimed at reducing noise and fine details to enhance the visual quality of images
- By applying smoothing filters, we can achieve a more uniform appearance, making it easier to analyse and interpret the image.

Before



After



Spatial Domain Techniques

- **Common Image Smoothing Techniques**
- **Mean (Average) Filtering:** This method replaces each pixel's value with the average of its neighbouring pixels
 - It effectively reduces random noise but may blur edges
- **Median Filtering:** Each pixel's value is replaced by the median value of its neighbouring pixels
 - This method is particularly effective at removing salt-and-pepper noise without blurring edges.

Spatial Domain Techniques

- **Gaussian Filtering:** This technique applies a Gaussian function to weight neighbouring pixels, giving more importance to those closer to the centre
 - It provides a balance between noise reduction and edge preservation
- **Bilateral Filtering:** This advanced method considers both spatial distance and pixel intensity differences, allowing it to smooth images while preserving edges
 - It effectively reduces noise while maintaining sharp edges.

Spatial Domain Techniques

- **Example:** Comparison of Mean-Gaussian Filtering under Gaussian noise

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```
# Function to add Gaussian noise
```

```
def add_gaussian_noise(image, mean=0, sigma=25):
```

```
    """
```

```
    Add Gaussian noise to the image.
```

```
    :param image: input image.
```

```
    :param mean: mean of the Gaussian noise.
```

```
    :param sigma: standard deviation of the Gaussian noise.
```

```
    :return: noisy image.
```

```
    """
```

```
    gaussian_noise = np.random.normal(mean, sigma, image.shape)
```

```
    noisy_image = np.uint8(np.clip(image + gaussian_noise, 0, 255))
```

```
    return noisy_image
```



Spatial Domain Techniques

```
image = cv2.imread('peppers.jpg', cv2.IMREAD_COLOR)

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Add Gaussian noise to the image
noisy_image = add_gaussian_noise(gray_image)

# Apply Mean filter (3x3 kernel)
mean_filtered = cv2.blur(noisy_image, (7, 7))

# Apply Gaussian filter (kernel size 5x5 and sigma = 1.0)
gaussian_filtered = cv2.GaussianBlur(noisy_image, (7, 7), 1.0)

# Display the original, noisy, and filtered images for comparison
plt.figure(figsize=(10, 8))
```

```
plt.subplot(2, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

```
plt.subplot(2, 3, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image')
plt.axis('off')
```

```
plt.subplot(2, 3, 3)
plt.imshow(mean_filtered, cmap='gray')
plt.title('Mean Filtered')
plt.axis('off')
```

```
plt.subplot(2, 3, 4)
plt.imshow(gaussian_filtered,
cmap='gray')
plt.title('Gaussian Filtered')
plt.axis('off')
```

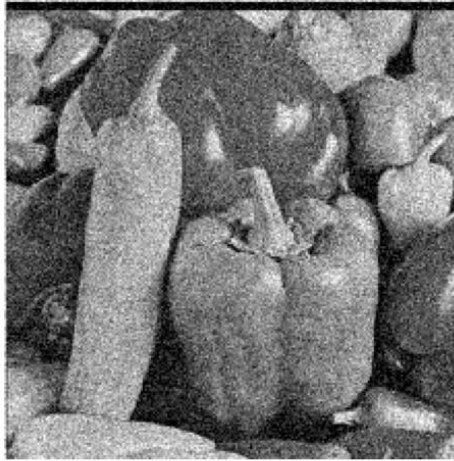
```
plt.tight_layout()
plt.show()
```

Spatial Domain Techniques

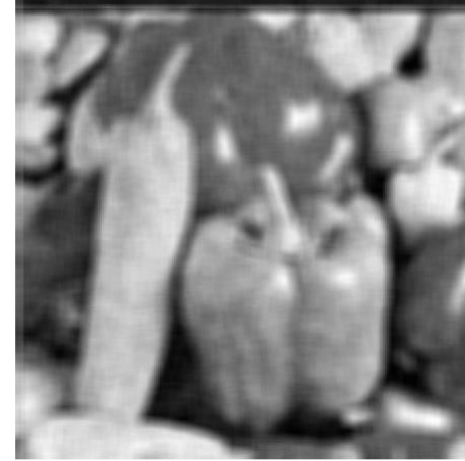
Original Image



Noisy Image



Mean Filtered



Gaussian Filtered



Spatial Domain Techniques

- **Example:** Comparison of Mean-Median-Gaussian Filtering under Salt-and-Pepper noise

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```
# Function to add Salt and Pepper noise
```

```
def add_salt_and_pepper_noise(image, salt_prob=0.01, pepper_prob=0.01):
```

```
    """
```

```
    Add Salt and Pepper noise to an image.
```

```
    :param image: input image.
```

```
    :param salt_prob: probability of salt noise (white).
```

```
    :param pepper_prob: probability of pepper noise (black).
```

```
    :return: noisy image.
```

```
    """
```

```
    noisy_image = image.copy()
```

```
# Salt noise (white pixels)
```

```
    num_salt = int(salt_prob * image.size)
```

```
    salt_coords = [np.random.randint(0, i-1, num_salt) for i in image.shape]
```

```
    noisy_image[salt_coords[0], salt_coords[1]] = 255
```

```
# Pepper noise (black pixels)
```

```
    num_pepper = int(pepper_prob * image.size)
```

```
    pepper_coords = [np.random.randint(0, i-1, num_pepper) for i in image.shape]
```

```
    noisy_image[pepper_coords[0], pepper_coords[1]] = 0
```

```
    return noisy_image
```



Spatial Domain Techniques

```
image = cv2.imread('cameraman.jpg', cv2.IMREAD_COLOR)

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Add Salt and Pepper noise to the image
noisy_image = add_salt_and_pepper_noise(gray_image)

# Apply Mean filter (3x3 kernel)
mean_filtered = cv2.blur(noisy_image, (3, 3))

# Apply Gaussian filter (kernel size 5x5 and sigma = 1.0)
gaussian_filtered = cv2.GaussianBlur(noisy_image, (5, 5), 1.0)

# Apply Median filter (3x3 kernel)
median_filtered = cv2.medianBlur(noisy_image, 3)

# Display the original, noisy, and filtered images for comparison
plt.figure(figsize=(10, 8))
```

```
plt.subplot(2, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

```
plt.subplot(2, 3, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image (Salt & Pepper)')
plt.axis('off')
```

```
plt.subplot(2, 3, 3)
plt.imshow(mean_filtered, cmap='gray')
plt.title('Mean Filtered')
plt.axis('off')
```

```
plt.subplot(2, 3, 4)
plt.imshow(gaussian_filtered, cmap='gray')
plt.title('Gaussian Filtered')
plt.axis('off')
```

```
plt.subplot(2, 3, 5)
plt.imshow(median_filtered, cmap='gray')
plt.title('Median Filtered')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Spatial Domain Techniques

Original Image



Noisy Image (Salt & Pepper)



Mean Filtered



Gaussian Filtered



Median Filtered



Spatial Domain Techniques

- **Image sharpening** is a technique used to enhance the edges and fine details in an image
- Unlike smoothing (which blurs an image to remove noise), sharpening enhances the contrast between neighbouring pixels to make the image appear clearer and crisper
- **Methods:** High-pass filter, Laplacian filter, Unsharp Masking, Sobel Operator

Spatial Domain Techniques

- **High-pass filtering** involves subtracting the low-frequency components (blurring) from the original image, leaving behind the high-frequency details (edges, textures)
- **How it works**
 - Low-pass filtering (such as Gaussian blur) is applied to the original image, and the result is subtracted from the original image.
 - This leaves the high-frequency components, which represent the sharp edges and fine details
- **Effect:** The edges and finer details of the image are emphasized, leading to a sharper image

$$\text{Sharpened Image} = \text{Original Image} - \text{Low-pass filtered Image}$$

Spatial Domain Techniques

- **The Laplacian filter** is a second-order derivative filter that highlights areas of rapid intensity change (edges) in an image
- **How it works:**
 - It detects regions where there is a sharp contrast between neighbouring pixels (edges)
 - The Laplacian operator is a convolution mask, typically a 3x3 kernel like:

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

- **Effect:** When applied, this filter highlights the edges in the image, enhancing the sharpness

$$\text{Sharpened Image} = \text{Original Image} + \text{Laplacian Filtered Image}$$

Spatial Domain Techniques

- **Unsharp masking** is a popular sharpening technique that uses a blurred version of the original image to enhance its details
- **How it works:**
 - First, the image is blurred (usually with a Gaussian filter)
 - Then, the difference between the original and blurred image is calculated
 - This difference (the "mask") is added to the original image to enhance the details
- **Effect:** The edges are enhanced while keeping the rest of the image relatively unaffected

$$\text{Sharpened Image} = \text{Original Image} + (\text{Original Image} - \text{Blurred Image})$$

Spatial Domain Techniques

- **Sobel Operator:** The Sobel operator is often used in edge detection but can also be used for sharpening. It computes the gradient of the image intensity at each pixel, giving a measure of the image's edges
- **How it works:** The Sobel operator uses two kernels for detecting edges in both horizontal and vertical directions

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- **Effect:** It highlights the edges of the image, making it appear sharper.

Spatial Domain Techniques

- Image Sharpening methods comparison

```
import cv2
import matplotlib.pyplot as plt

# Function to apply high-pass filter (subtracting Gaussian blur
from the original)
def high_pass_filter(image, kernel_size=5, sigma=1):
    blurred_image = cv2.GaussianBlur(image, (kernel_size,
kernel_size), sigma)
    high_pass = cv2.subtract(image, blurred_image)
    return high_pass

# Function to apply Laplacian filter
def laplacian_filter(image):
    laplacian = cv2.Laplacian(image, cv2.CV_64F)
    laplacian = cv2.convertScaleAbs(laplacian) # Convert to
absolute values
    return laplacian
```



Spatial Domain Techniques

Function to apply unsharp masking

```
def unsharp_masking(image, kernel_size=5, sigma=1,
strength=1.5):
    blurred_image = cv2.GaussianBlur(image, (kernel_size,
kernel_size), sigma)
    mask = cv2.subtract(image, blurred_image)
    sharpened_image = cv2.addWeighted(image, 1 + strength,
mask, strength, 0)
    return sharpened_image
```

Function to apply Sobel operator (for edge detection and sharpening)

```
def sobel_operator(image):
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3) #
Detect horizontal edges
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3) #
Detect vertical edges
    sobel_edges = cv2.sqrt(sobel_x**2 + sobel_y**2)
    sobel_edges = cv2.convertScaleAbs(sobel_edges) # Convert
to absolute values
    return sobel_edges
```



Spatial Domain Techniques

```
image = cv2.imread('cameraman.jpg', cv2.IMREAD_COLOR) # Apply Sobel operator
sobel_sharpened = sobel_operator(gray_image)

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # Display the original image and all sharpened images for
comparison
plt.figure(figsize=(12, 10))

# Apply High-pass filter
high_pass_sharpened = high_pass_filter(gray_image)

# Apply Laplacian filter
laplacian_sharpened = laplacian_filter(gray_image)

# Apply Unsharp Masking
unsharp_sharpened = unsharp_masking(gray_image)
```

Spatial Domain Techniques

```
plt.subplot(3, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

```
plt.subplot(3, 3, 2)
plt.imshow(high_pass_sharpened, cmap='gray')
plt.title('High-pass Filter')
plt.axis('off')
```

```
plt.subplot(3, 3, 3)
plt.imshow(laplacian_sharpened, cmap='gray')
plt.title('Laplacian Filter')
plt.axis('off')
```

```
plt.subplot(3, 3, 4)
plt.imshow(unsharp_sharpened, cmap='gray')
plt.title('Unsharp Masking')
plt.axis('off')
```

```
plt.subplot(3, 3, 5)
plt.imshow(sobel_sharpened, cmap='gray')
plt.title('Sobel Operator')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```

Spatial Domain Techniques

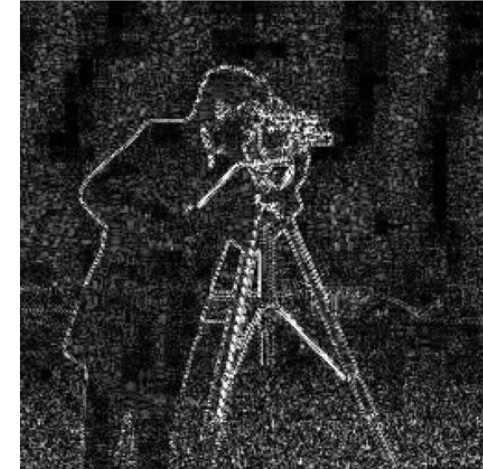
Original Image



High-pass Filter



Laplacian Filter



Unsharp Masking



Sobel Operator



Frequency Domain Techniques

Steps in Frequency Domain Image Enhancement:

1. **Transform the image to the frequency domain** using Discrete Fourier Transform or another suitable transform
2. **Apply the filter (low-pass, high-pass, etc.)** to manipulate the frequency components
3. **Transform the image back to the spatial domain** using the inverse Fourier transform
4. **Analyse and evaluate** the enhanced image based on the desired characteristics (sharpness, noise reduction, contrast, etc.)

Frequency Domain Techniques

- The **Discrete Fourier Transform (DFT)** is commonly used to convert an image from the spatial domain to the frequency domain
- After applying the DFT, the resulting image can be analysed in terms of its frequency components
- **High-pass filters** allow high-frequency components (fine details, edges) to pass through while attenuating low-frequency components (smooth areas)
- This enhances edges and fine details, making the image appear sharper

Frequency Domain Techniques

- **Common high-pass filters**
 - Ideal high-pass filter
 - Butterworth high-pass filter
 - Gaussian high-pass filter
- **Low-pass filters** allow low-frequency components (smooth and large structures) to pass through while attenuating high-frequency components
- This can smooth out an image, reduce noise, or blur the image

Frequency Domain Techniques

- **Common low-pass filters**
 - Ideal low-pass filter
 - Butterworth low-pass filter
 - Gaussian low-pass filter
- **Band-pass filters** allow a specific range of frequencies to pass, blocking both low and high frequencies
- This technique can be used to enhance certain features of an image that are within a specific frequency range

Frequency Domain Techniques

- **Image Sharpening:** Frequency domain image sharpening enhances the high-frequency components of an image, making the details more pronounced and improving the sharpness
- This can be achieved using high-pass filters or by subtracting a low-pass filtered version of the image from the original image
- **Image De-noising:** Filtering techniques in the frequency domain can be used to remove noise.
- Low-frequency noise can be reduced by applying a high-pass filter, whereas high-frequency noise can be reduced by using a low-pass filter.

Frequency Domain Techniques

- **Noise Reduction:** Some frequency domain techniques help in removing specific types of noise by filtering out certain frequency ranges
- **Computational Complexity:** Fourier transforms can be computationally expensive for large images, so optimized algorithms like the **Fast Fourier Transform (FFT)** are often used to speed up processing
- **Artifacts:** Some frequency domain methods, such as ideal filters, can introduce undesirable artifacts (e.g., ringing or blockiness) when applied to images

Frequency Domain Techniques

- **Example:** Butterworth Filters in the Frequency Domain

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('building.jpg', 0) # Read in grayscale
if image is None:
    raise ValueError("Image not found. Please check the file
path.")

# Function to compute the 2D Fourier Transform
def fourier_transform(img):
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    return fshift
```

```
# Function to compute the inverse Fourier Transform
def inverse_fourier_transform(fshift):
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    return np.abs(img_back)

# Function to compute the magnitude spectrum
def magnitude_spectrum(fshift):
    return np.log(np.abs(fshift) + 1)
```

Frequency Domain Techniques

Butterworth Low-Pass Filter

```
def butterworth_low_pass(shape, cutoff, order=2):  
    rows, cols = shape  
    u = np.arange(0, rows)  
    v = np.arange(0, cols)  
    idx = np.meshgrid(u, v, indexing='ij')  
    dist = np.sqrt((idx[0] - rows//2)**2 + (idx[1] - cols//2)**2)  
    filter_lp = 1 / (1 + (dist / cutoff)**(2 * order))  
    return filter_lp
```

Butterworth High-Pass Filter

```
def butterworth_high_pass(shape, cutoff, order=2):  
    rows, cols = shape  
    u = np.arange(0, rows)  
    v = np.arange(0, cols)  
    idx = np.meshgrid(u, v, indexing='ij')  
    dist = np.sqrt((idx[0] - rows//2)**2 + (idx[1] - cols//2)**2)  
    filter_hp = 1 / (1 + (cutoff / dist)**(2 * order))  
    filter_hp[dist == 0] = 0 # Avoid division by zero at the center  
    return filter_hp
```

Apply Butterworth filters and return the filtered images

```
def apply_filter(fshift, filter):  
    fshift_filtered = fshift * filter  
    img_back_filtered =  
    inverse_fourier_transform(fshift_filtered)  
    return img_back_filtered
```

Frequency Domain Techniques

Convert the image to frequency domain

```
fshift = fourier_transform(image)
```

Create Butterworth filters

```
cutoff_frequency = 30 # Define cutoff frequency for the filters
```

```
order = 2 # Order of the Butterworth filter
```

```
lp_filter = butterworth_low_pass(image.shape,  
cutoff_frequency, order)
```

```
hp_filter = butterworth_high_pass(image.shape,  
cutoff_frequency, order)
```

Apply the filters

```
img_lp = apply_filter(fshift, lp_filter)
```

```
img_hp = apply_filter(fshift, hp_filter)
```

Plot the frequency responses and filtered images

```
plt.figure(figsize=(15, 15))
```

Plot original image and its magnitude spectrum

```
plt.subplot(3, 3, 1)
```

```
plt.imshow(image, cmap='gray')
```

```
plt.title("Original Image")
```

```
plt.axis('off')
```

```
plt.subplot(3, 3, 2)
```

```
plt.imshow(magnitude_spectrum(fshift), cmap='gray')
```

```
plt.title("Magnitude Spectrum (Original)")
```

```
plt.axis('off')
```

Plot Low-Pass Filter and its response

```
plt.subplot(3, 3, 3)
```

```
plt.imshow(lp_filter, cmap='gray')
```

```
plt.title("Butterworth Low-Pass Filter")
```

```
plt.axis('off')
```

Frequency Domain Techniques

```
plt.subplot(3, 3, 4)
plt.imshow(magnitude_spectrum(fshift * lp_filter),
cmap='gray')
plt.title("Magnitude Spectrum (Low-Pass Filtered)")
plt.axis('off')
```

```
# Plot High-Pass Filter and its response
plt.subplot(3, 3, 5)
plt.imshow(hp_filter, cmap='gray')
plt.title("Butterworth High-Pass Filter")
plt.axis('off')
```

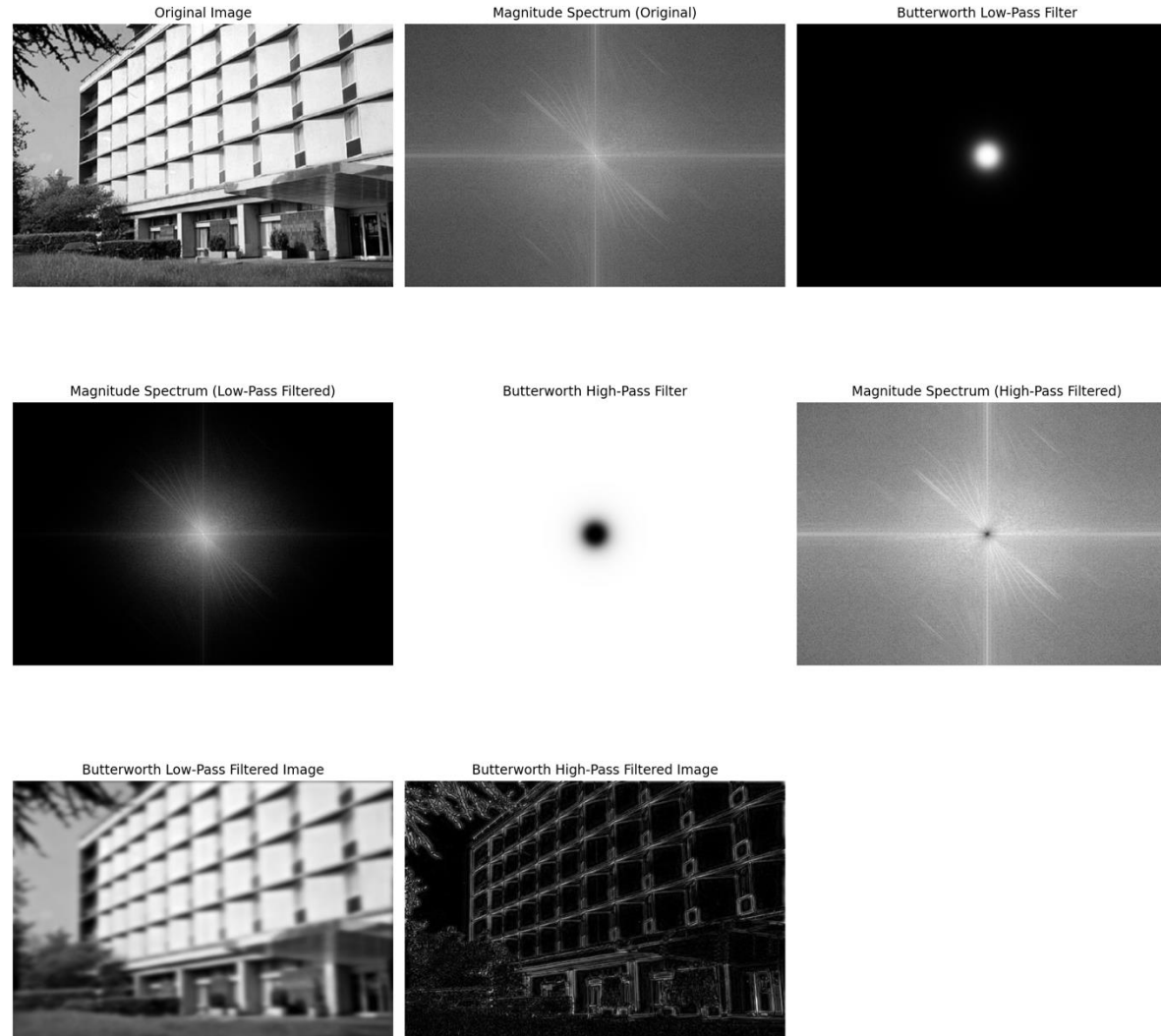
```
plt.subplot(3, 3, 6)
plt.imshow(magnitude_spectrum(fshift * hp_filter),
cmap='gray')
plt.title("Magnitude Spectrum (High-Pass Filtered)")
plt.axis('off')
```

```
# Plot filtered images
plt.subplot(3, 3, 7)
plt.imshow(img_lp, cmap='gray')
plt.title("Butterworth Low-Pass Filtered Image")
plt.axis('off')
```

```
plt.subplot(3, 3, 8)
plt.imshow(img_hp, cmap='gray')
plt.title("Butterworth High-Pass Filtered Image")
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```

Frequency Domain Techniques



Edge Detection

- **Edge detection** is a fundamental technique in image processing and computer vision, used to identify boundaries within images by detecting sharp discontinuities in intensity
- These edges often correspond to object boundaries, changes in texture, and other significant features in an image
- There are several edge detection techniques, each with its strengths and application areas

Edge Detection

- The **Sobel operator** is a simple and widely used edge detection method
- It uses convolution with two kernels (one for detecting horizontal edges and one for detecting vertical edges)
- It computes the gradient of the image intensity in both the horizontal and vertical directions
 - The magnitude of the gradient at each pixel gives an estimate of the edge strength
 - The direction of the gradient provides the orientation of the edge

Edge Detection

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- It is sensitive to edges in both the horizontal and vertical directions, making it effective for detecting general edges
- **Advantages:** Simple and fast
- **Limitations:** Sensitive to noise, does not provide very precise edge locations

Edge Detection

- The **Prewitt operator** is similar to the Sobel operator but uses different kernels
- It is also used to detect edges in both horizontal and vertical directions.
- Like Sobel, it calculates the gradient of the image, but the kernels have different coefficients (lower weights on the diagonal).

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

- The Prewitt operator is often used in applications that require detecting vertical and horizontal edges with a slight advantage over Sobel in certain situations
- **Advantages:** Simple and fast
- **Limitations:** Similar to Sobel, it is sensitive to noise

Edge Detection

- The **Canny edge** detector is one of the most widely used edge detection algorithms due to its precision and ability to detect weak edges while minimizing noise
- **Step 1:** Apply Gaussian smoothing to reduce noise
- **Step 2:** Compute the gradient using the Sobel operator to identify regions of high intensity change
- **Step 3:** Apply non-maximum suppression to thin out the edges by keeping only the local maxima in the gradient direction.

Edge Detection

- **Step 4:** Use double thresholding to classify edges into strong, weak, and non-edges
- **Step 5:** Perform edge tracking by hysteresis, where weak edges connected to strong edges are kept, and others are discarded
- Canny edge detector is known for its effectiveness in detecting edges while reducing noise and producing thin edges

Edge Detection

- **Advantages:** Provides precise edge detection with low noise sensitivity and thin edges
- **Limitations:** Computationally more expensive compared to simpler methods like Sobel and Prewitt
- **Example:** Comparison of Sobel/Prewitt/Canny Edge Detectors

Edge Detection

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```
# Load the image
image = cv2.imread('building.jpg', 0) # Read in grayscale
if image is None:
    raise ValueError("Image not found. Please check the file
path.")
```

```
def prewitt_operator(image):
    # Prewitt kernels for detecting edges in the x and y direction
    kernel_x = np.array([[1, 0, -1],
                        [1, 0, -1],
                        [1, 0, -1]])

    kernel_y = np.array([[1, 1, 1],
                        [0, 0, 0],
                        [-1, -1, -1]])

    grad_x = cv2.filter2D(image, -1, kernel_x)
    grad_y = cv2.filter2D(image, -1, kernel_y)

    # Convert to float64 before magnitude calculation
    grad_x = np.float64(grad_x)
    grad_y = np.float64(grad_y)

    magnitude = cv2.magnitude(grad_x, grad_y)
    return magnitude
```

Edge Detection

Sobel Operator (using OpenCV)

```
def sobel_operator(image):
```

```
    grad_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
```

```
    grad_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
```

Ensure the gradients are in float64 for magnitude calculation

```
    grad_x = np.float64(grad_x)
```

```
    grad_y = np.float64(grad_y)
```

```
    magnitude = cv2.magnitude(grad_x, grad_y)
```

```
    return magnitude
```

Canny Edge Detection (using OpenCV)

```
def canny_operator(image):
```

```
    return cv2.Canny(image, 100, 200) # 100 and 200 are  
thresholds for Canny
```

Apply edge detection methods

```
sobel_edges = sobel_operator(image)
```

```
prewitt_edges = prewitt_operator(image)
```

```
canny_edges = canny_operator(image)
```

Edge Detection

```
# Plot the results
plt.figure(figsize=(12, 12))

# Original Image
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Sobel Edge Detection
plt.subplot(2, 2, 2)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel Edge Detection")
plt.axis('off')
```

```
# Prewitt Edge Detection
plt.subplot(2, 2, 3)
plt.imshow(rewitt_edges, cmap='gray')
plt.title("Prewitt Edge Detection")
plt.axis('off')

# Canny Edge Detection
plt.subplot(2, 2, 4)
plt.imshow(canny_edges, cmap='gray')
plt.title("Canny Edge Detection")
plt.axis('off')

plt.tight_layout()
plt.show()
```

Edge Detection

Original Image



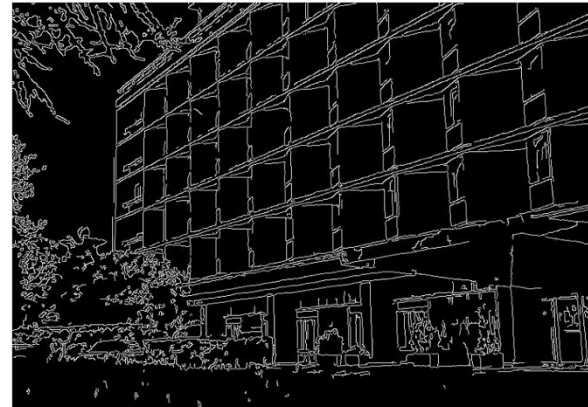
Sobel Edge Detection



Prewitt Edge Detection



Canny Edge Detection



Edge Detection

Original Image



Sobel Edge Detection



Prewitt Edge Detection



Canny Edge Detection



Colour Image Enhancement

- **Colour image enhancement** techniques are used to improve the visual quality of an image by enhancing certain aspects such as brightness, contrast, and colour balance
- In addition to traditional grayscale image enhancement, color image enhancement involves manipulating the colour channels (Red, Green, and Blue channels in RGB images or Hue, Saturation, and Value channels in HSV images) to improve the perception and details of the image

Colour Image Enhancement

- Histogram equalization is a technique used to improve the contrast of an image by adjusting the intensity distribution of pixels
- Instead of applying histogram equalization directly to the entire image (which can distort colour), you apply it separately to each colour channel (Red, Green, and Blue in the RGB colour model)
- **Procedure**
 - Separate the image into its individual RGB channels
 - Apply histogram equalization to each channel
 - Merge the enhanced channels back together to form the final enhanced image

Colour Image Enhancement

- **Advantages:** Improves contrast and visibility of details in an image
- **Limitations:** May lead to unnatural color changes or over-enhancement of certain regions

Colour Image Enhancement

- **Contrast stretching** is a simple technique to improve image contrast by linearly stretching the pixel values over a specified range
- **Application to Colour Images:** Contrast stretching is typically applied to each channel (R, G, and B) separately to avoid distortions in the colour balance
- **Procedure:**
 - Find the minimum and maximum pixel values in each colour channel
 - Stretch the pixel values between the minimum and maximum values to a specified range (usually 0 to 255 for an 8-bit image)

Colour Image Enhancement

- **Advantages:** Simple and fast method to enhance contrast
- **Limitations:** Can result in loss of fine details in some regions

Colour Image Enhancement

- **Gamma correction** is used to adjust the brightness and contrast of an image
- It is useful in correcting the non-linear relationship between the pixel values and perceived brightness

$$I_{\text{new}} = \text{max} \times \left(\frac{I}{\text{max}} \right)^{\gamma}$$

Where:

- I_{new} is the output image after gamma correction.
- I is the original pixel value.
- max is the maximum pixel value (255 for an 8-bit image).
- γ is the gamma value. If $\gamma > 1$, the image becomes darker; if $\gamma < 1$, the image becomes brighter.

Colour Image Enhancement

- **Application to Colour Images:** Gamma correction can be applied to each RGB channel separately to adjust the brightness and contrast of the image
- **Advantages:** Allows for flexible adjustment of brightness and contrast
- **Limitations:** Requires careful tuning of the gamma value to avoid unnatural effects

Colour Image Enhancement

- **Colour image manipulation** involves adjusting the individual colour channels (R, G, B) to enhance specific colour properties
- **Procedure:**
 - Boost the intensity of a specific channel (e.g., enhancing the red channel to make the image appear warmer).
 - Reduce the intensity of certain channels (e.g., reducing the blue channel to warm up the image).
- **Application to Colour Images:** Channel manipulation can be used to balance colours or adjust the temperature of the image

Colour Image Enhancement

- **Advantages:** Simple and effective for fine-tuning colours
- **Limitations:** Can lead to unnatural colour balance if overused
- **Example:** Contrast Stretching/Gamma Correction/ Warmer image

Colour Image Enhancement

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread("lenna.jpeg") # Replace with your image
path
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert
to RGB for correct color display

# Function for contrast stretching
def contrast_stretching(image):
    # Convert image to float type for processing
    image = np.float32(image)

    # Compute min and max values per channel
    min_val = np.min(image, axis=(0, 1), keepdims=True)
    max_val = np.max(image, axis=(0, 1), keepdims=True)

    # Stretch contrast to full 0-255 range
    stretched = (image - min_val) / (max_val - min_val) * 255.0
    stretched = np.clip(stretched, 0, 255).astype(np.uint8)

    return stretched
```

Colour Image Enhancement

```
# Function for gamma correction
def gamma_correction(image, gamma=1.5):
    # Build a lookup table to speed up computation
    inv_gamma = 1 / gamma
    table = np.array([(i / 255.0) ** inv_gamma * 255 for i in
range(256)]).astype("uint8")

    # Apply gamma correction using the lookup table
    return cv2.LUT(image, table)
```

```
# Function for making an image warmer
def make_warmer(image, intensity=20):
    # Split into color channels
    r, g, b = cv2.split(image)

    # Increase red and decrease blue slightly
    r = np.clip(r + intensity, 0, 255)
    b = np.clip(b - intensity, 0, 255)

    # Merge channels back together
    warmer_image = cv2.merge([r, g, b])
    return warmer_image

# Apply enhancement techniques
contrast_stretched = contrast_stretching(image)
gamma_corrected = gamma_correction(image, gamma=1.5)
warmer_image = make_warmer(image)
```

Colour Image Enhancement

```
# Display results  
plt.figure(figsize=(12, 8))
```

```
plt.subplot(2, 2, 1)  
plt.imshow(image)  
plt.title("Original Image")  
plt.axis('off')
```

```
plt.subplot(2, 2, 2)  
plt.imshow(contrast_stretched)  
plt.title("Contrast Stretching")  
plt.axis('off')
```

```
plt.subplot(2, 2, 3)  
plt.imshow(gamma_corrected)  
plt.title("Gamma Correction ( $\gamma=1.5$ )")  
plt.axis('off')
```

```
plt.subplot(2, 2, 4)  
plt.imshow(warmer_image)  
plt.title("Warmer Image Effect")  
plt.axis('off')
```

```
plt.tight_layout()  
plt.show()
```

Colour Image Enhancement

Original Image



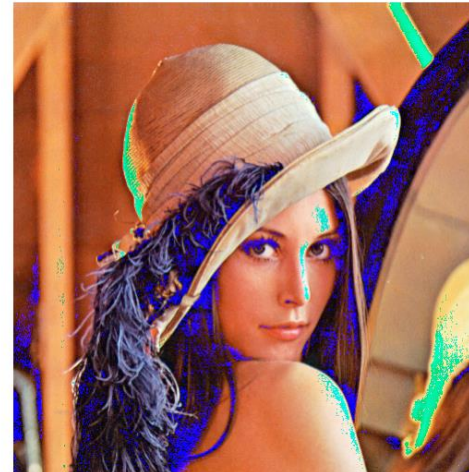
Contrast Stretching



Gamma Correction ($\gamma=1.5$)



Warmer Image Effect





Thank you for your attention!