



MSc in Digital Humanities
Image Processing/Analysis

Image Processing/Analysis

Anestis Koutsoudis, Chairi Kiourt, Christos Tsinos, George Pavlidis

Image Transformations

Christos Tsinos

e-mail: ctsinos@uoa.gr

Overview

- Introduction
- Understand the fundamentals of digital image processing
- Explore different types of image transformations and their applications
- Gain hands-on experience with tools and techniques for implementing transformations through python examples

Introduction to Digital Image Processing

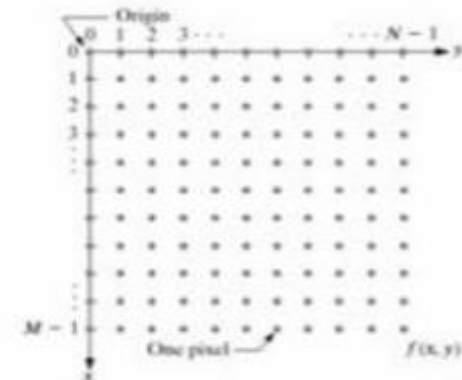
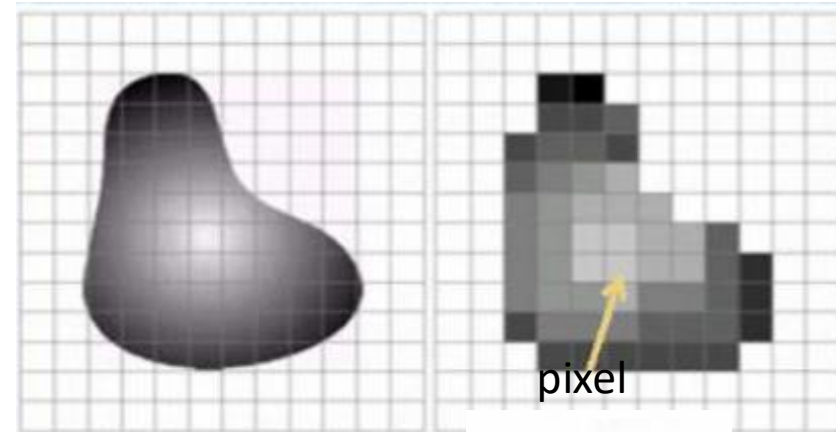
- **Definition:** Digital Image Processing (DIP) involves the use of computer algorithms to perform operations on digital images, enhancing their quality, extracting meaningful information, or transforming them for further use.
- **Example:** Correcting lighting in photographs or detecting edges in medical scans.
- **Key Difference from Analog Processing:** DIP operates on discrete numerical data (pixels), while analog processing works on continuous signals.

Digital Images

- **Pixels:** Smallest unit of an image, often represented as a matrix of intensity values.
- Each pixel has a value corresponding to brightness or color.
- Images can be grayscale (single intensity value) or color (three components: RGB or other color spaces).
- **Resolution:** Refers to the total number of pixels in an image, which determines its detail.

Digital Images

- An image can be defined as a two-dimensional array specially arranged in rows and columns
- A digital image consists of a finite number of pixes, each of which has a specific value at a specific position



Types of Digital Images

- **BINARY IMAGE:** A binary image contains only two-pixel values, i.e., 0 & 1, where 0 represents black and 1 represents white. This type of image is also known as **Monochrome**
- **GREYSCALE IMAGE:** A greyscale image is a black-and-white image with various shades of grey
- **8-BIT COLOR FORMAT:** This is the most popular image format. It features 256 different shades of colors and is commonly known as a greyscale image. In this format: **0** represents black - **255** represents white - **127** represents grey

Types of Digital Images

- **16-BIT COLOR FORMAT:** This is a coloured image format that contains 65,536 different colours. It is also known as the **High Colour Format**. Unlike greyscale images, the colour distribution in this format is different. A 16-bit format is actually divided into three further components: **Red**, **Green**, and **Blue** – commonly referred to as the well-known **RGB format**
- **32-BIT COLOR FORMAT** is also met in many applications

Why is Digital Image Processing Important?

- Foundational technology for many modern applications:
- **Medical Imaging:** Enhancing MRI or CT scan clarity to assist in diagnosis
- **Remote Sensing:** Satellite images used for environmental monitoring or urban planning
- **Artificial Intelligence:** Enabling object detection, recognition, and analysis in real-world settings

Why is Digital Image Processing Important?

- **Photography & Media:** Automatic editing features like red-eye correction, filters, and HDR.
- **Security Systems:** Face detection, fingerprint recognition, and license plate recognition.

Why Transformations are important to DIP?

- Image transformations are essential in image processing for various reasons, as they enable manipulation, enhancement, and analysis of images to meet specific goals or solve problems.
- **Improving Image Quality:**
 - **Noise Reduction:** Filters like low-pass transformations help remove unwanted noise from images.
 - **Contrast Enhancement:** Intensity transformations like histogram equalization enhance the contrast, making details more visible.
 - **Brightness Adjustment:** Useful for correcting underexposed or overexposed images.
 - **Example:** Enhancing the visibility of faint details in X-ray or satellite images.

Why Transformations are important to DIP?

- **Preparing Data for Analysis**
 - **Normalization:** Transformations like scaling and resizing ensure uniform data for machine learning models.
 - **Segmentation:** Thresholding separates objects from the background for feature extraction and pattern recognition.
 - **Colour Space Conversion:** Changing colour representations (e.g., RGB to HSV) simplifies tasks like object detection or colour-based segmentation.
 - **Example:** Converting an image to grayscale for edge detection algorithms.

Why Transformations are important to DIP?

- **Enabling Image Understanding:**
 - **Edge Detection:** Frequency transformations and high-pass filtering highlight edges and boundaries.
 - **Feature Extraction:** Geometric transformations help identify patterns or align objects for comparison.
 - **Object Recognition:** Geometric corrections ensure objects are positioned correctly for accurate recognition.
 - **Example:** Aligning faces in a dataset for facial recognition models.

Types of Digital Image Transformations

- **Geometric Transformations:**
 - Translation
 - Rotation
 - Scaling
 - Affine and Perspective transformations
- **Intensity Transformations:**
 - Brightness adjustment
 - Contrast stretching
 - Thresholding
- **Frequency Domain Transformations:**
 - Fourier Transform
 - Filtering (low-pass, high-pass)
- **Color Transformations:**
 - RGB to Grayscale
 - Color space conversions (e.g., HSV, LAB)

Tools and Technologies

- **Software Tools:**

- MATLAB
- Python libraries (OpenCV, PIL, NumPy, scikit-image)
- Adobe Photoshop (for non-programmatic editing)

- **Hardware Considerations:**

- High-performance GPUs for real-time transformations.
- Image sensors for accurate data acquisition.

Geometric Transformations

- **Translation:** Moves an image by a specified distance
- **Rotation:** Rotates the image around a specific point or centre
- **Scaling:** Resizes the image by a scaling factor
- **Affine Transformation:** Combines translation, rotation, and scaling while preserving lines and parallelism
- **Perspective Transformation:** Adjusts the view of an image as if observed from a different angle

Geometric Transformations

- **Translation:**

Translation shifts an image in the x and y directions. The translation matrix T is applied to every pixel to move the image by a specified number of pixels in both directions:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

Where:

- t_x is the shift in the horizontal direction.
- t_y is the shift in the vertical direction.

Geometric Transformations

- **Translation:**

Pixel Transformation:

For any pixel at coordinates (x, y) , its new position (x', y') after translation is given by:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = T \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

So, the new pixel position becomes:

$$x' = x + t_x \quad \text{and} \quad y' = y + t_y$$

Geometric Transformations

- **Effect on Image:**
 - The entire image is shifted along the x-axis and along the y-axis
 - Pixels move in the direction of translation, with the image remaining unchanged in shape or size
 - This is useful in aligning images or moving objects within the image.

Geometric Transformations

- **Rotation:**

The rotation matrix R rotates an image by an angle θ around a specific point, usually the center of the image:

$$R = \begin{bmatrix} \cos \theta & -\sin \theta & (1 - \cos \theta)x_c + \sin \theta y_c \\ \sin \theta & \cos \theta & (1 - \cos \theta)y_c - \sin \theta x_c \end{bmatrix}$$

Where (x_c, y_c) is the center of rotation, and θ is the angle of rotation.

Pixel Transformation:

A pixel at coordinates (x, y) will rotate around the center (x_c, y_c) by an angle θ to a new position (x', y') :

$$x' = (x - x_c) \cos \theta - (y - y_c) \sin \theta + x_c$$

$$y' = (x - x_c) \sin \theta + (y - y_c) \cos \theta + y_c$$

Geometric Transformations

- **Effect on Image:**
 - The image is rotated around a point, typically its centre.
 - Depending on the angle of rotation, parts of the image may get clipped, especially for non-90-degree rotations.
 - It's used for tasks like rotating scanned images or objects in computer vision.

Geometric Transformations

- **Scaling:**

Scaling enlarges or shrinks an image by a factor of s_x and s_y in the x and y directions, respectively:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix}$$

Where:

- s_x is the scaling factor in the x direction.
- s_y is the scaling factor in the y direction.

Geometric Transformations

Pixel Transformation:

Each pixel (x, y) in the original image will be scaled to a new position (x', y') according to the scaling matrix:

$$x' = s_x \cdot x$$

$$y' = s_y \cdot y$$

Effect on Image:

- If s_x and s_y are greater than 1, the image is enlarged.
- If s_x and s_y are between 0 and 1, the image is shrunk.
- This transformation is widely used in resizing images.

Geometric Transformations

- **Affine:**

An affine transformation combines translation, scaling, rotation, and shearing in one matrix.

The transformation matrix A is a 2x3 matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{bmatrix}$$

This matrix allows transformations such as shearing or skewing.

Pixel Transformation:

A pixel at (x, y) will be mapped to (x', y') as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Geometric Transformations

- **Affine -Effect on Image:**

- The image can be skewed, rotated, scaled, or sheared, while maintaining parallel lines and affine properties.
- Used for tasks like correcting distortions or aligning images.

- **Perspective:**

The perspective transformation uses a 3x3 matrix P , which maps points to simulate the effect of a 3D perspective. The matrix is:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

Geometric Transformations

Pixel Transformation:

For a pixel at (x, y) , its new position (x', y') is calculated as:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = P \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The homogeneous coordinates allow for 3D-like effects on 2D images, resulting in distortion of straight lines into vanishing points.

- **Effect on Image:**
 - Perspective transformation simulates a 3D effect by distorting the image to appear as if viewed from an angle or different viewpoint
 - This is commonly used for rectifying images or creating 3D effects

Geometric Transformations - Summary

- **Translation** shifts the entire image, moving each pixel by the same amount in the x and y directions
- **Rotation** moves pixels around a central point, changing their orientation
- **Scaling** changes the size of the image, either shrinking or enlarging the pixels.
- **Affine Transformation** combines these effects, preserving parallelism and allowing for more complex deformations.

Geometric Transformations - Summary

- **Perspective Transformation** introduces depth, making the image appear as if it was taken from a different viewpoint.
- These mathematical transformations are critical for tasks such as image registration, alignment, and editing. Each transformation manipulates the pixel coordinates, which can be computed and applied to the entire image using matrix operations.

Geometric Transformations - Examples

- **What we need:**
 - Python Interpreter
 - Python Editor
 - Python Libraries
 - opencv-python (cv2) – image processing tools
 - matplotlib – plotting images
 - numpy – linear algebra operations
 - urllib, ssl, certify – downloading images from OpenCV's repository

Geometric Transformations - Examples

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import urllib.request
import ssl
import certifi

# Load an image from OpenCV's GitHub sample repository
url = "https://raw.githubusercontent.com/opencv/opencv/master/samples/data/lena.jpg"
context = ssl.create_default_context(cafile=certifi.where())
req = urllib.request.urlopen(url, context=context)
image_data = cv2.imdecode(np.asarray(bytearray(req.read()), dtype=np.uint8), cv2.IMREAD_COLOR)
image_rgb = cv2.cvtColor(image_data, cv2.COLOR_BGR2RGB)
```

Geometric Transformations - Examples

Translation

tx, ty = 50, 30 # Shift right by 50, down by 30

translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])

translated = cv2.warpAffine(image_data, translation_matrix, (image_data.shape[1], image_data.shape[0]))

Rotation

angle = 45 # Rotate by 45 degrees

center = (image_data.shape[1] // 2, image_data.shape[0] // 2)

rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)

rotated = cv2.warpAffine(image_data, rotation_matrix, (image_data.shape[1], image_data.shape[0]))

Scaling

scaled = cv2.resize(image_data, None, fx=0.4, fy=0.4, interpolation=cv2.INTER_LINEAR)

Geometric Transformations - Examples

Affine Transformation

```
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
```

```
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])
```

```
affine_matrix = cv2.getAffineTransform(pts1, pts2)
```

```
affine_transformed = cv2.warpAffine(image_data, affine_matrix, (image_data.shape[1], image_data.shape[0]))
```

Perspective Transformation

```
pts1 = np.float32([[56, 65], [368, 52], [28, 387], [389, 390]])
```

```
pts2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
```

```
perspective_matrix = cv2.getPerspectiveTransform(pts1, pts2)
```

```
perspective_transformed = cv2.warpPerspective(image_data, perspective_matrix, (300, 300))
```

Geometric Transformations - Examples

```
# Plot the results
fig, axs = plt.subplots(2, 3, figsize=(15, 10))
axs[0, 0].imshow(image_rgb)
axs[0, 0].set_title("Original Image")
axs[0, 0].axis("off")

axs[0, 1].imshow(cv2.cvtColor(translated, cv2.COLOR_BGR2RGB))
axs[0, 1].set_title("Translated")
axs[0, 1].axis("off")

axs[0, 2].imshow(cv2.cvtColor(rotated, cv2.COLOR_BGR2RGB))
axs[0, 2].set_title("Rotated")
axs[0, 2].axis("off")

axs[1, 0].imshow(cv2.cvtColor(scaled, cv2.COLOR_BGR2RGB))
axs[1, 0].set_title("Scaled")
axs[1, 0].axis("off")
```

Geometric Transformations - Examples

```
axs[1, 1].imshow(cv2.cvtColor(affine_transformed,  
cv2.COLOR_BGR2RGB))  
axs[1, 1].set_title("Affine Transformed")  
axs[1, 1].axis("off")
```

```
axs[1, 2].imshow(cv2.cvtColor(perspective_transformed,  
cv2.COLOR_BGR2RGB))  
axs[1, 2].set_title("Perspective Transformed")  
axs[1, 2].axis("off")
```

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

Geometric Transformations - Examples

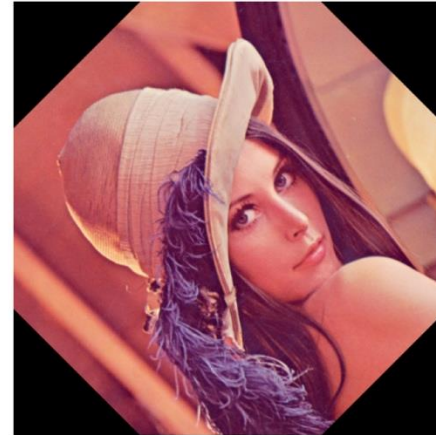
Original Image



Translated



Rotated



Scaled



Affine Transformed



Perspective Transformed



Intensity and Frequency Domain Transformations

- **Intensity Transformations:**
 - Histogram equalization for enhancing low-light images
 - Thresholding for segmentation in medical imaging
- **Frequency Transformations:**
 - Use of Fourier Transform to remove noise or compress images
 - Filtering to highlight edges or smooth regions

Intensity Transformations – Histogram Equalization

- **Histogram equalization** is a technique for improving the contrast of an image by redistributing its intensity levels over the entire range

- **Steps:**

1. **Histogram Calculation:**

The histogram $h(i)$ is a frequency distribution of pixel intensities i in the image.

2. **Cumulative Distribution Function (CDF):**

Compute the cumulative sum of the histogram values:

$$c(i) = \sum_{j=0}^i h(j)$$

Intensity Transformations – Histogram Equalization

3. Mapping Intensities:

Map the original intensity levels to new ones based on the CDF:

$$I' = \text{round} \left(\frac{c(I) - c_{\min}}{(N \times M) - c_{\min}} \times (L - 1) \right)$$

Where:

- $N \times M$ = total number of pixels
- L = number of intensity levels (usually 256)

Intensity Transformations – Histogram Equalization

```
import cv2
import numpy as np
import urllib.request
import matplotlib.pyplot as plt
import ssl
import certifi

# Load image from URL
url = "https://raw.githubusercontent.com/opencv/opencv/master/samples/data/lena.jpg"
context = ssl.create_default_context(cafile=certifi.where())
req = urllib.request.urlopen(url, context=context)
image_array = np.asarray(bytearray(req.read()), dtype=np.uint8)
image = cv2.imdecode(image_array, cv2.IMREAD_GRAYSCALE)
```

Intensity Transformations – Histogram Equalization

Apply Histogram Equalization

```
equalized_image = cv2.equalizeHist(image)
```

Calculate Histograms

```
original_hist = cv2.calcHist([image], [0], None, [256], [0, 256])
```

```
equalized_hist = cv2.calcHist([equalized_image], [0], None, [256], [0, 256])
```

Plot Original and Equalized Images with their Histograms

```
plt.figure(figsize=(15, 10))
```

Original Image

```
plt.subplot(2, 2, 1)
```

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

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

```
plt.axis("off")
```

Intensity Transformations – Histogram Equalization

```
# Original Histogram  
plt.subplot(2, 2, 2)  
plt.title("Original Histogram")  
plt.plot(original_hist, color='blue')  
plt.fill_between(range(256), original_hist.flatten(), color='blue',  
alpha=0.4)  
plt.xlabel("Pixel Intensity")  
plt.ylabel("Frequency")
```

```
# Equalized Image  
plt.subplot(2, 2, 3)  
plt.title("Equalized Image")  
plt.imshow(equalized_image, cmap="gray")  
plt.axis("off")
```

Intensity Transformations – Histogram Equalization

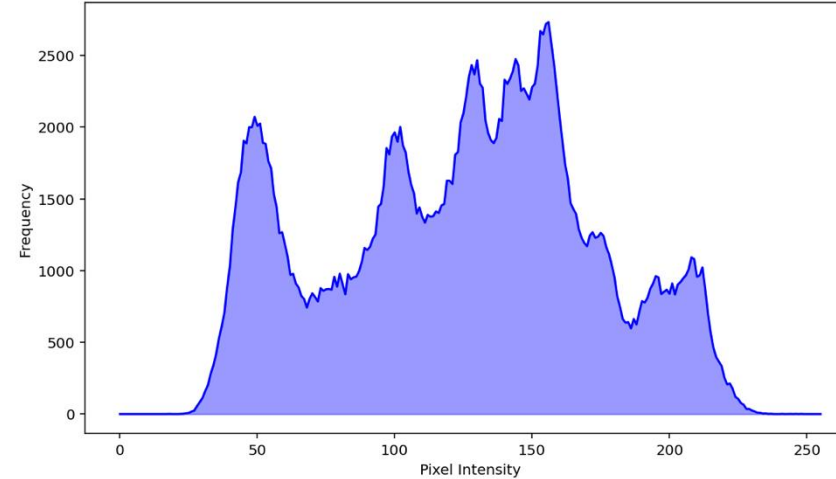
```
# Equalized Histogram  
plt.subplot(2, 2, 4)  
plt.title("Equalized Histogram")  
plt.plot(equalized_hist, color='green')  
plt.fill_between(range(256), equalized_hist.flatten(), color='green',  
alpha=0.4)  
plt.xlabel("Pixel Intensity")  
plt.ylabel("Frequency")  
  
plt.tight_layout()  
plt.show()
```

Intensity Transformations – Histogram Equalization

Original Image



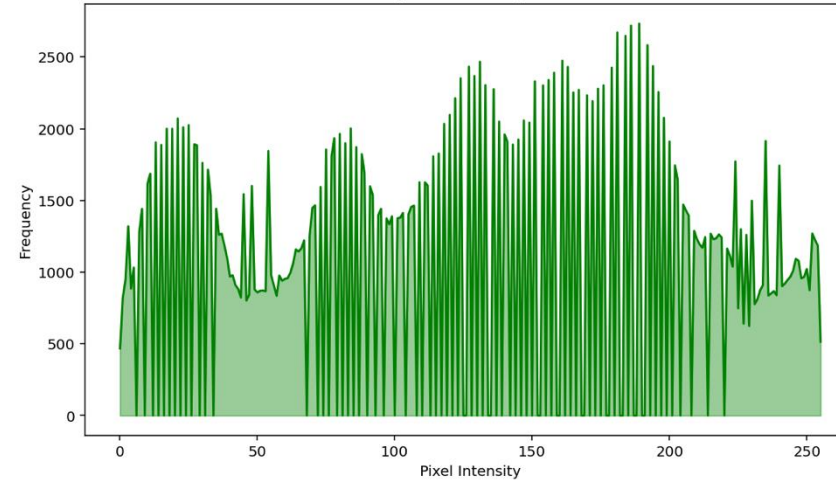
Original Histogram



Equalized Image



Equalized Histogram



Intensity Transformations – Thresholding

- Thresholding is a technique for segmenting an image by converting grayscale pixels into binary values based on a threshold T
- **Simple Thresholding:**

$$I'(x, y) = \begin{cases} 255 & \text{if } I(x, y) \geq T \\ 0 & \text{otherwise} \end{cases}$$

- **Adaptive Thresholding:** Instead of a global threshold value T , the threshold is calculated for each pixel based on the mean intensity of its neighbourhood.

Intensity Transformations – Thresholding

```
import cv2
import numpy as np
import urllib.request
import matplotlib.pyplot as plt
import ssl
import certifi

# Load image from URL
url =
"https://raw.githubusercontent.com/opencv/opencv/master/sa
mples/data/building.jpg"
context = ssl.create_default_context(cafile=certifi.where())
req = urllib.request.urlopen(url, context=context)
image_array = np.asarray(bytearray(req.read()),
dtype=np.uint8)
image = cv2.imdecode(image_array, cv2.IMREAD_GRAYSCALE)
```

Intensity Transformations – Thresholding

Simple Thresholding

```
_, binary_image = cv2.threshold(image, 128, 255,  
cv2.THRESH_BINARY)
```

Adaptive Thresholding

```
adaptive_thresh = cv2.adaptiveThreshold(image, 255,  
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)
```

Plot original and thresholded images

```
plt.figure(figsize=(15, 5))  
plt.subplot(1, 3, 1)  
plt.title("Original Image")  
plt.imshow(image, cmap="gray")  
plt.axis("off")
```

Intensity Transformations – Thresholding

```
plt.subplot(1, 3, 2)  
plt.title("Simple Thresholding")  
plt.imshow(binary_image, cmap="gray")  
plt.axis("off")
```

```
plt.subplot(1, 3, 3)  
plt.title("Adaptive Thresholding")  
plt.imshow(adaptive_thresh, cmap="gray")  
plt.axis("off")  
plt.show()
```

Intensity Transformations – Thresholding

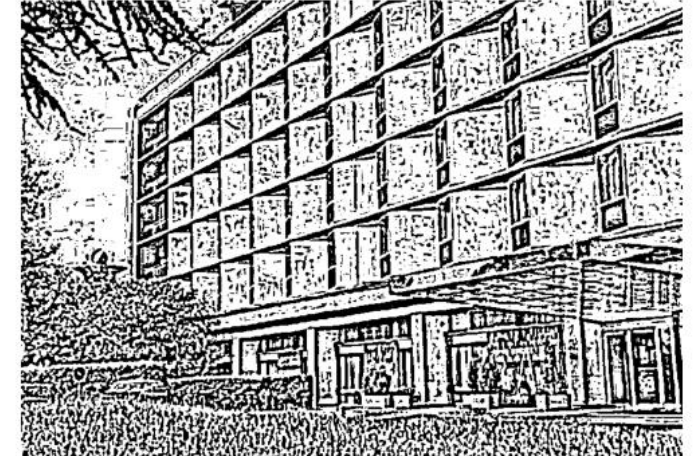
Original Image



Simple Thresholding



Adaptive Thresholding



Fourier Transform Basics

- In the **spatial domain**, an image is represented by pixel intensities
- In the **frequency domain**, the image is represented as a combination of sinusoidal waves of varying frequencies

Discrete Fourier Transform (DFT) of an image $f(x, y)$:

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

Fourier Transform Basics

Where:

- $f(x, y)$: Pixel intensity at (x, y) .
- M, N : Image dimensions.
- $F(u, v)$: Frequency component at (u, v) .
- Inverse Fourier Transform (to return to the spatial domain):

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

Fourier Transform Basics

- **Frequency Components:**
 - **Low frequencies** represent slow variations (e.g., smooth areas).
 - **High frequencies** represent rapid changes (e.g., edges and fine details).

Filtering in the Frequency Domain

Steps:

1. Apply Fourier Transform to the image to convert it into the frequency domain.
2. Create a filter mask that represents the desired filtering operation (e.g., low-pass, high-pass, band-pass).
3. Multiply the frequency-domain representation of the image with the filter mask.
4. Apply the Inverse Fourier Transform to convert the filtered image back to the spatial domain.

Filtering in the Frequency Domain –Types of Filters

- **Low-Pass Filter**
 - Allows low frequencies to pass and attenuates high frequencies.
 - Used for blurring or noise reduction
- **High-Pass Filter**
 - Allows high frequencies to pass and attenuates low frequencies.
 - Used for edge detection and sharpening
- **Band-Pass Filter**
 - Allows frequencies within a specific range to pass.

Filtering in the Frequency Domain

```
import cv2
import numpy as np
import urllib.request
import matplotlib.pyplot as plt
import ssl
import certifi

# Load image from URL
url = "https://raw.githubusercontent.com/opencv/opencv/master/samples/data/building.jpg"
context = ssl.create_default_context(cafile=certifi.where())
req = urllib.request.urlopen(url, context=context)
image_array = np.asarray(bytearray(req.read()), dtype=np.uint8)
image = cv2.imdecode(image_array, cv2.IMREAD_GRAYSCALE)
```

Filtering in the Frequency Domain

Fourier Transform

```
dft = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
```

Magnitude Spectrum

```
magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1]))
```

Create Low-Pass Filter (LPF)

```
rows, cols = image.shape
crow, ccol = rows // 2, cols // 2
mask_lpf = np.zeros((rows, cols, 2), np.uint8)
mask_lpf[crow-30:crow+30, ccol-30:ccol+30] = 1
```

Apply LPF

```
fshift_lpf = dft_shift * mask_lpf
magnitude_lpf = 20 * np.log(cv2.magnitude(fshift_lpf[:, :, 0], fshift_lpf[:, :, 1]))
lpf_result = cv2.idft(np.fft.ifftshift(fshift_lpf))
lpf_result = cv2.magnitude(lpf_result[:, :, 0], lpf_result[:, :, 1])
```



Filtering in the Frequency Domain

```
# Create High-Pass Filter (HPF)
mask_hpf = np.ones((rows, cols, 2), np.uint8)
mask_hpf[crow-30:crow+30, ccol-30:ccol+30] = 0

# Apply HPF
fshift_hpf = dft_shift * mask_hpf
magnitude_hpf = 20 * np.log(cv2.magnitude(fshift_hpf[:, :, 0], fshift_hpf[:, :, 1]))
hpf_result = cv2.idft(np.fft.ifftshift(fshift_hpf))
hpf_result = cv2.magnitude(hpf_result[:, :, 0], hpf_result[:, :, 1])

# Plot Results
plt.figure(figsize=(18, 12))

# Original Image
plt.subplot(3, 3, 1)
plt.title("Original Image")
plt.imshow(image, cmap="gray")
plt.axis("off")
```

Filtering in the Frequency Domain

```
# Magnitude Spectrum (Original)
plt.subplot(3, 3, 2)
plt.title("Original Magnitude Spectrum")
plt.imshow(magnitude_spectrum, cmap="gray")
plt.axis("off")
```

```
# Low-Pass Filter Result
plt.subplot(3, 3, 3)
plt.title("Low-Pass Filter Result")
plt.imshow(lpf_result, cmap="gray")
plt.axis("off")
```

```
# Magnitude Spectrum After LPF
plt.subplot(3, 3, 4)
plt.title("Magnitude Spectrum After LPF")
plt.imshow(magnitude_lpf, cmap="gray")
plt.axis("off")
```

Filtering in the Frequency Domain

```
# High-Pass Filter Result
plt.subplot(3, 3, 5)
plt.title("High-Pass Filter Result")
plt.imshow(hpf_result, cmap="gray")
plt.axis("off")

# Magnitude Spectrum After HPF
plt.subplot(3, 3, 6)
plt.title("Magnitude Spectrum After HPF")
plt.imshow(magnitude_hpf, cmap="gray")
plt.axis("off")

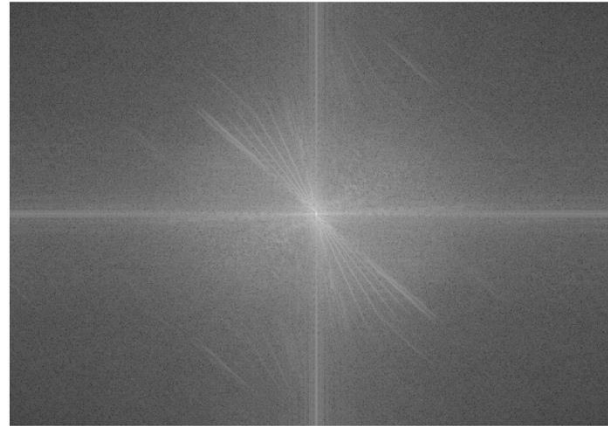
plt.tight_layout()
plt.show()
```

Filtering in the Frequency Domain

Original Image



Original Magnitude Spectrum



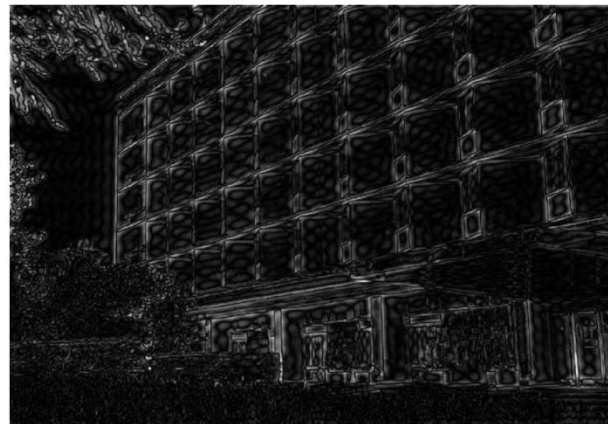
Low-Pass Filter Result



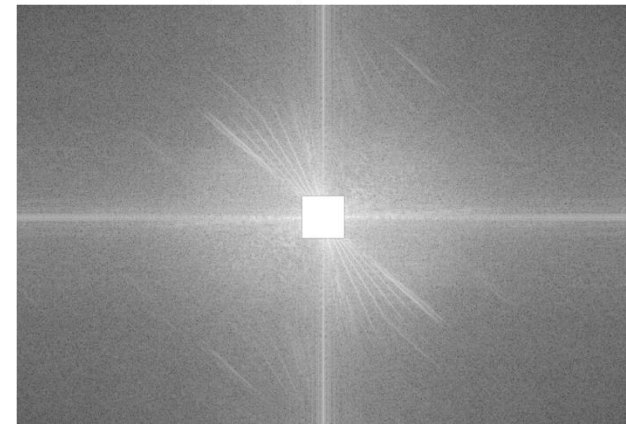
Magnitude Spectrum After LPF



High-Pass Filter Result



Magnitude Spectrum After HPF



References

- **Textbooks**
 - "Digital Image Processing" by Gonzalez and Woods.
 - "Computer Vision: Algorithms and Applications" by Richard Szeliski.
- **Online Resources**
 - OpenCV documentation and examples



Thank you for your attention!