**1. What is Image Processing?**

Image processing involves performing operations on images to enhance them, extract features, or prepare them for other applications like computer vision or machine learning. It can be broken into two types:

* **Analog Image Processing:** Applied to hard copies like photographs.
* **Digital Image Processing:** Manipulation of images in digital form using computers.

Common goals:

* Noise reduction
* Image enhancement
* Image restoration
* Object detection and recognition

**2. Applications of Image Processing**

* **Medical Field:** MRI scan enhancement, tumor detection.
* **Remote Sensing:** Earth observation via satellite images.
* **Industrial Automation:** Detecting defects in manufactured products.
* **Security:** Biometric recognition, surveillance systems.
* **Robotics:** Navigation, visual feedback.
* **Multimedia:** Image compression (JPEG), enhancement, filters in apps.

**3. Image Representation**

**Pixels**

An image is made of tiny elements called **pixels (picture elements)**. Each pixel represents an intensity value.

**Bit Depth**

Defines how many different intensity levels a pixel can represent:

* **1-bit:** 2 levels (black and white)
* **8-bit:** 256 levels of grayscale (0=black, 255=white)
* **24-bit color image:** 8 bits for each R, G, B channel → 256 × 256 × 256 colors.

**Grayscale vs. Color Images**

* **Grayscale Image:** One channel, each pixel has intensity between 0–255.
* **Color Image:** 3 channels (RGB or BGR), each pixel has a tuple like (R, G, B).

**4. OpenCV and PIL (Pillow)**

**OpenCV:**

* C++ library with Python bindings.
* Efficient for real-time computer vision applications.
* Handles image/video processing, machine learning, and more.

**Pillow (PIL):**

* Simpler Python imaging library.
* Good for basic tasks (image open, crop, rotate, save).
* Useful in web apps, scripting, and when GUI isn’t required.

**5. Loading, Displaying, and Saving Images**

**With OpenCV:**

import cv2

img = cv2.imread('image.jpg') # Load image

cv2.imshow('Image', img) # Display image

cv2.imwrite('output.jpg', img) # Save image

cv2.waitKey(0) # Wait for keypress

cv2.destroyAllWindows()

*Note:* OpenCV loads images in **BGR** format by default.

**With PIL:**

from PIL import Image

img = Image.open('image.jpg') # Load

img.show() # Show using default viewer

img.save('output.jpg') # Save

**6. Image Attributes**

* shape: Dimensions (Height × Width × Channels)
* size: Total number of pixels
* dtype: Data type of each pixel (usually uint8)

print(img.shape) # (height, width, channels)

print(img.size) # total number of pixels

print(img.dtype) # data type

**7. Transformations: Scaling, Translation, Rotation, Affine**

**Scaling (Resizing)**

resized = cv2.resize(img, (300, 300))

* INTER\_LINEAR or INTER\_AREA used for interpolation.

**Translation (Shifting Image)**

M = np.float32([[1, 0, 50], [0, 1, 100]]) # Shift x by 50, y by 100

translated = cv2.warpAffine(img, M, (cols, rows))

**Rotation**

M = cv2.getRotationMatrix2D(center, angle, scale)

rotated = cv2.warpAffine(img, M, (width, height))

**Affine Transformation**

Uses 3 points from the source and maps them to destination points.

M = cv2.getAffineTransform(pts1, pts2)

dst = cv2.warpAffine(img, M, (cols, rows))

**8. Flipping, Resizing, and Cropping**

**Flipping:**

cv2.flip(img, 0) # Vertical flip

cv2.flip(img, 1) # Horizontal flip

**Resizing:**

resized = cv2.resize(img, (new\_w, new\_h))

**Cropping:**

cropped = img[y1:y2, x1:x2]

This selects a rectangular portion.

**9. RGB to Grayscale Conversion**

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

Grayscale uses luminance values from RGB: Y=0.299R+0.587G+0.114BY = 0.299R + 0.587G + 0.114B

**10. Histograms**

* A graph showing the number of pixels for each intensity level (0–255).
* Helps analyze contrast, brightness, dynamic range.

import matplotlib.pyplot as plt

plt.hist(gray.ravel(), 256, [0, 256])

**11. Histogram Equalization**

Improves the global contrast of an image.

equalized = cv2.equalizeHist(gray)

**Applications:**

* Better detail in low-light images
* Satellite imaging
* Enhancing X-ray scans

**12. Erosion and Dilation**

**Erosion**

* Removes noise, erodes object boundaries.

kernel = np.ones((3,3), np.uint8)

erosion = cv2.erode(img, kernel, iterations=1)

**Dilation**

* Expands object boundaries.

dilation = cv2.dilate(img, kernel, iterations=1)

Useful in:

* Removing small noise
* Filling gaps
* Connecting disjoint parts

**13. Object Shape Filtering and Border Detection**

**Shape Filtering**

Use **contours** + conditions to filter unwanted objects based on area or shape:

for cnt in contours:

area = cv2.contourArea(cnt)

if area > 500:

cv2.drawContours(img, [cnt], -1, (0, 255, 0), 2)

**Border Detection**

Find object boundaries using **contours**:

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cv2.drawContours(img, contours, -1, (255, 0, 0), 2)

# Image Processing Tutorial with OpenCV and Pillow

# 1. Setup

import cv2

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

# Load sample image (you can replace this with your own image path)

img = cv2.imread(cv2.samples.findFile('lena.jpg'))

# Display the image using OpenCV

cv2.imshow('Original Image', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

# 2. Image Attributes

print("Shape (Height, Width, Channels):", img.shape)

print("Size (Total Pixels):", img.size)

print("Data Type:", img.dtype)

# 3. Basic Operations

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

cv2.imshow('Grayscale Image', gray)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Resize

resized = cv2.resize(img, (200, 200))

cv2.imshow('Resized Image', resized)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Crop

cropped = img[100:250, 100:250]

cv2.imshow('Cropped Image', cropped)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Flip

flipped\_h = cv2.flip(img, 1)

flipped\_v = cv2.flip(img, 0)

cv2.imshow('Flipped Horizontal', flipped\_h)

cv2.imshow('Flipped Vertical', flipped\_v)

cv2.waitKey(0)

cv2.destroyAllWindows()

# 4. Geometric Transformations

# Scaling

scaled = cv2.resize(img, None, fx=0.5, fy=0.5, interpolation=cv2.INTER\_AREA)

cv2.imshow('Scaled Down', scaled)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Translation

rows, cols = img.shape[:2]

M = np.float32([[1, 0, 100], [0, 1, 50]]) # x shift = 100, y shift = 50

translated = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow('Translated Image', translated)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Rotation

M = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)

rotated = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow('Rotated Image', rotated)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Affine Transform

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

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

M = cv2.getAffineTransform(pts1, pts2)

affine = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow('Affine Transform', affine)

cv2.waitKey(0)

cv2.destroyAllWindows()

# 5. Histogram and Equalization

plt.figure()

plt.title("Grayscale Histogram")

plt.xlabel("Pixel value")

plt.ylabel("Frequency")

plt.hist(gray.ravel(), 256, [0, 256])

plt.show()

# Histogram Equalization

equalized = cv2.equalizeHist(gray)

cv2.imshow('Equalized Image', equalized)

cv2.waitKey(0)

cv2.destroyAllWindows()

# 6. Morphological Operations

# Create binary image

\_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)

# Erosion

kernel = np.ones((5,5), np.uint8)

erosion = cv2.erode(binary, kernel, iterations=1)

cv2.imshow('Eroded Image', erosion)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Dilation

dilation = cv2.dilate(binary, kernel, iterations=1)

cv2.imshow('Dilated Image', dilation)

cv2.waitKey(0)

cv2.destroyAllWindows()

# 7. Contours and Shape Filtering

# Find contours

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

img\_copy = img.copy()

for contour in contours:

area = cv2.contourArea(contour)

if area > 500:

cv2.drawContours(img\_copy, [contour], -1, (0, 255, 0), 2)

cv2.imshow('Contours Filtered by Area', img\_copy)

cv2.waitKey(0)

cv2.destroyAllWindows()