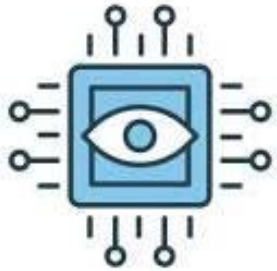
Computer Vision: From Basics to Expert Level

Advanced Computer Vision & Image Processing with Python



Dated: 04 Aug 2025

Instructor : Fawad Bahadur

Training Objectives & Outcomes

1. Understand the fundamentals of image processing and computer vision
2. Install and configure OpenCV in a virtual environment
3. Read, display, and manipulate images using OpenCV
4. Apply key techniques such as filtering, thresholding, resizing, and blending
5. Detect shapes, edges, contours, and regions of interest in images
6. Develop practical mini-projects using image processing techniques

Introduction to OpenCV

What is OpenCV?

- OpenCV stands for Open-Source Computer Vision Library
- A powerful library for **image processing**, **computer vision**, and **machine learning**
- Enables real-time image and video analysis across platforms

History & Background

- Developed by Intel in 1999
- Became open-source and is now maintained by OpenCV.org
- Key tool in AI, robotics, automation, and deep learning

1999 → OpenCV → Open-Source → Widely Used

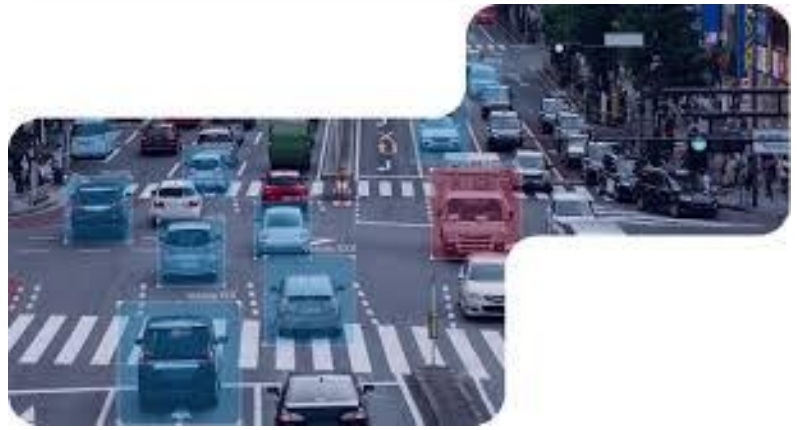
Core Features of OpenCV

Image Processing

Resize, filter, blur, rotate, transform, and enhance images

Video Analysis

Read, write, and process videos frame-by-frame in real-time

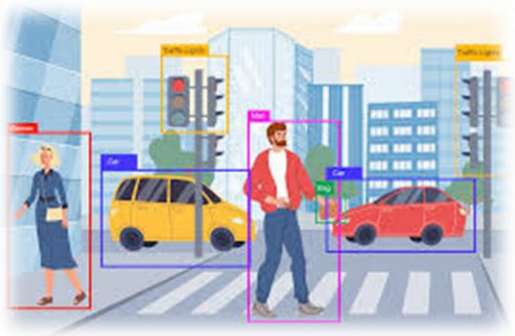


Object Detection & Recognition

Face detection, motion tracking, and feature matching

Camera Integration

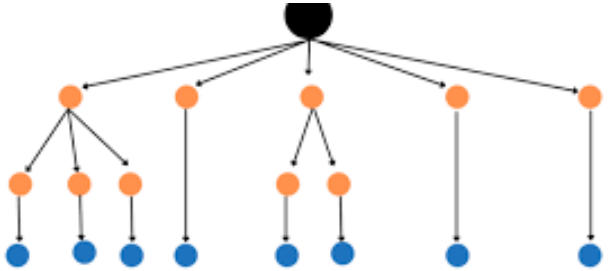
Access and process webcam/video stream feeds



Conti...

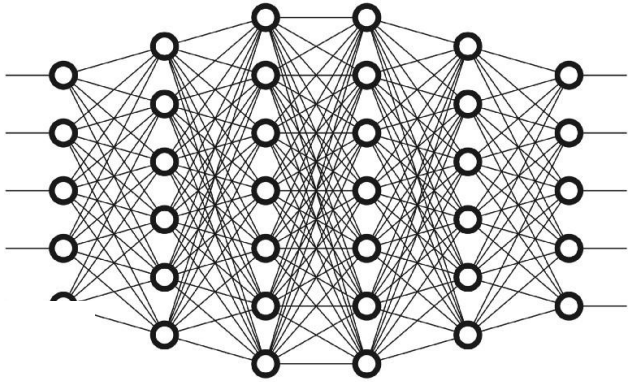
Machine Learning Module

Includes built-in classifiers (SVM, KNN, Decision Trees)



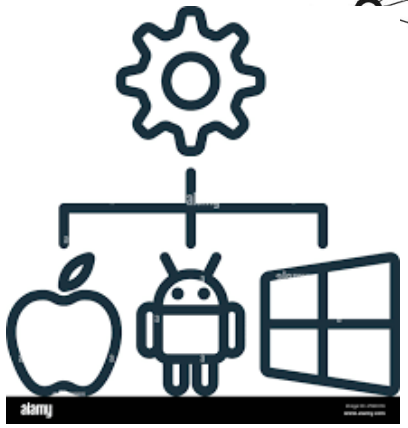
Deep Learning Support

Run pre-trained models (TensorFlow, Caffe, PyTorch, ONNX)



Platform Independence

Works on Windows, macOS, Linux, Android, and iOS



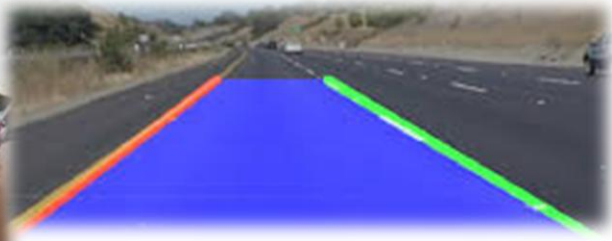
Applications of OpenCV

Face & Object Detection

Used in surveillance, biometrics, and camera apps

Autonomous Vehicles

Lane detection, object recognition, and navigation systems

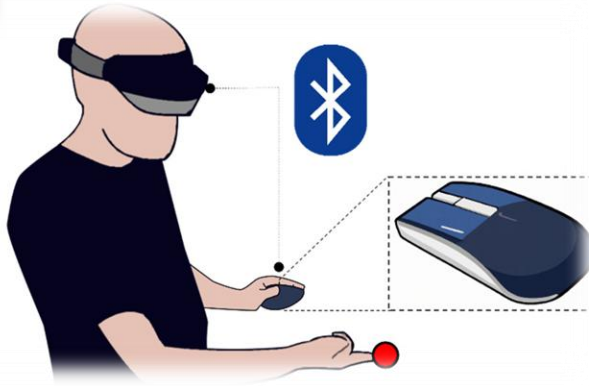


Robotics

Vision-based navigation, gesture control, and automation

Augmented Reality (AR)

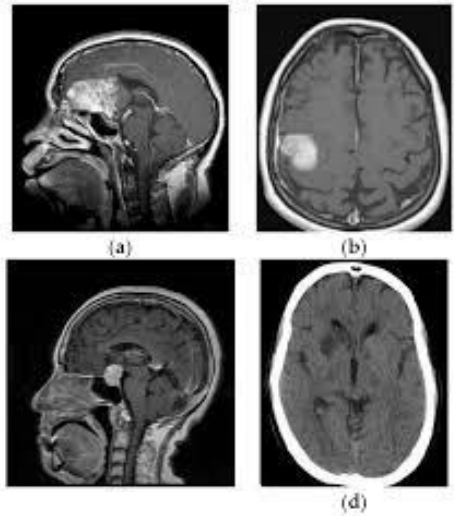
Real-time marker tracking and image overlay



Conti...

Healthcare & Medical Imaging

Image enhancement, tumor detection, diagnostic tools



Industrial Automation

Quality inspection, motion analysis, barcode scanning

Gaming & Gesture Recognition

Interactive gaming using motion or facial input

Digital Image Processing

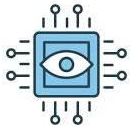
Focus: Manipulating images to enhance quality or extract features.

Goal: Improve image for human interpretation or further analysis.

Tasks: Filtering, noise reduction, compression, edge detection, segmentation.

Output: Processed image (e.g., sharpened, segmented regions).

Example: Removing blur from a medical X-ray.



Computer Vision

Focus: Enabling machines to "understand" visual data like humans.

Goal: High-level interpretation, decision-making, or automation.

Tasks: Object detection, facial recognition, scene understanding, autonomous navigation.

Output: Descriptions, labels, or actions (e.g., "cat detected," "turn left").

Example: A self-driving car identifying pedestrians and traffic signs.

Installing OpenCV

Method 1: Using pip (Recommended for most users)

Bash

```
!pip install opencv-python
```

Method 2: Using Anaconda

Bash

```
conda install -c conda-forge opencv
```

Installation

Reading and Displaying Images with OpenCV

1. Import OpenCV

Python

```
import cv2
```

2. Read an Image

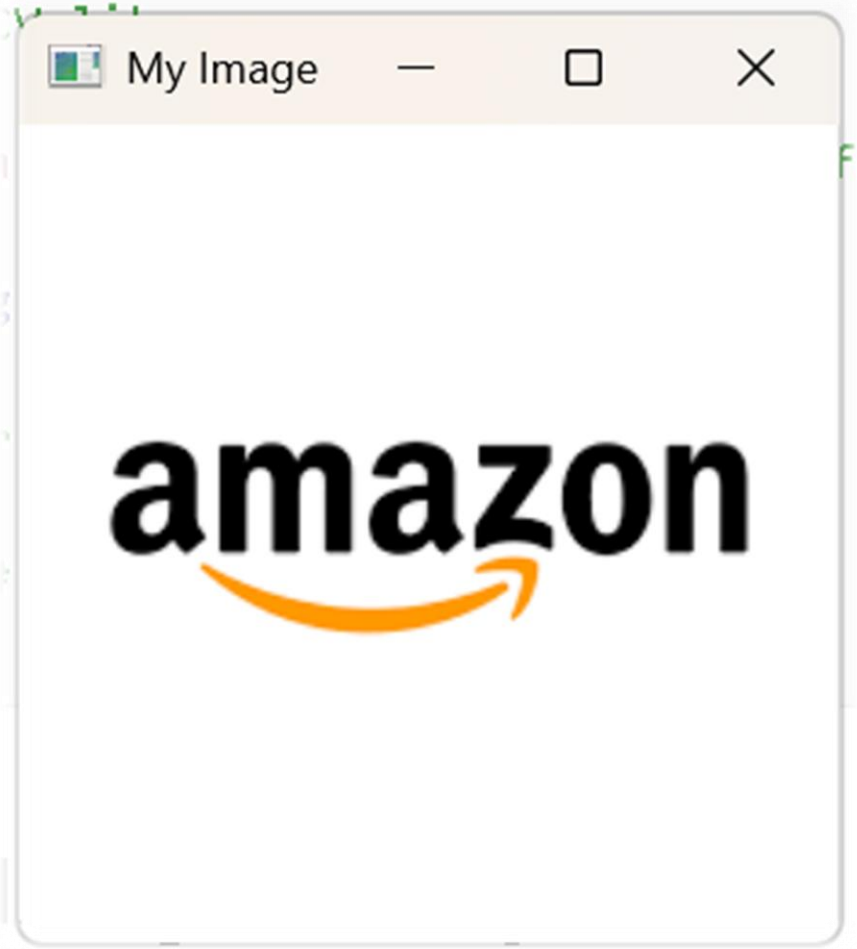
Python

```
img = cv2.imread("image.jpg")  # Reads image in BGR format
```

3. Display the Image

Python

```
cv2.imshow("My Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Notes

- cv2.imread() can return None if the file path is incorrect
- cv2.waitKey(0) waits for a key press
- Always use cv2.destroyAllWindows() to close display windows

Writing and Saving Images in OpenCV

Saving Images to Disk

1. Import and Read Image

Python

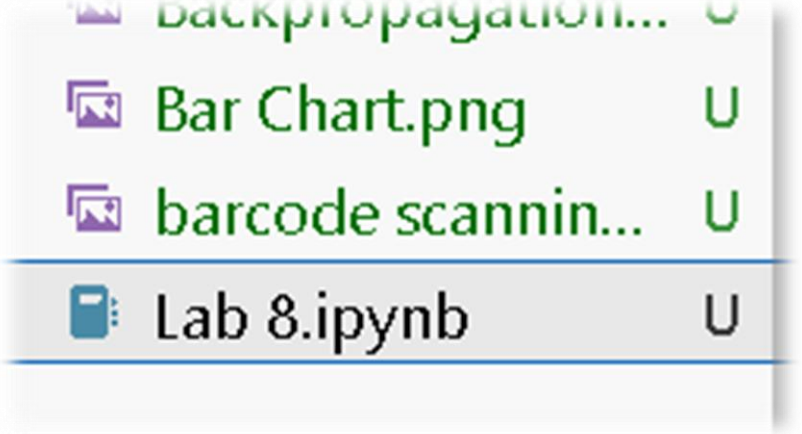
```
import cv2img = cv2.imread("input.jpg")
```

2. Save Image

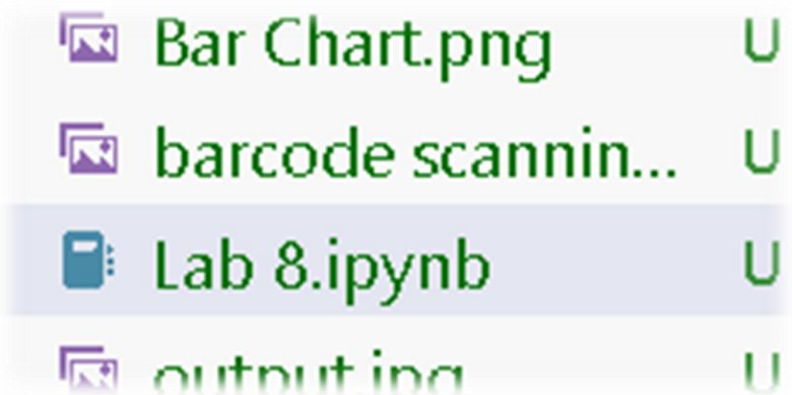
Python

```
cv2.imwrite("output.jpg", img)
```

Before



After



Notes

- Format depends on file extension: .jpg, .png, .bmp, etc.
- Ensure destination folder exists or use full path
- Returns True if saved successfully, otherwise False

Working with Video in OpenCV

1. Open Video Source

Python

```
import cv2cap = cv2.VideoCapture(0)
# 0 for webcam, or path to video file
```

2. Read & Display Frames

Python

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow("Video", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

3. Release Resources

```
pythoncap.release()
cv2.destroyAllWindows()
```

Notes

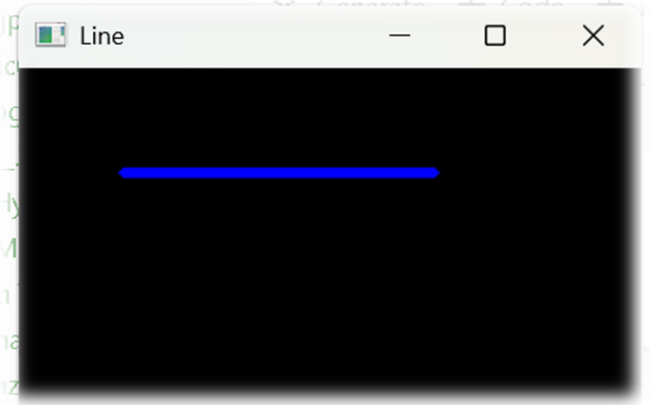
- Use 0 for default webcam, 1 or file path for others
- cv2.waitKey(1) controls frame speed
- Press ‘q’ to exit video window

Drawing Shapes and Text on Images

1. Line

Python

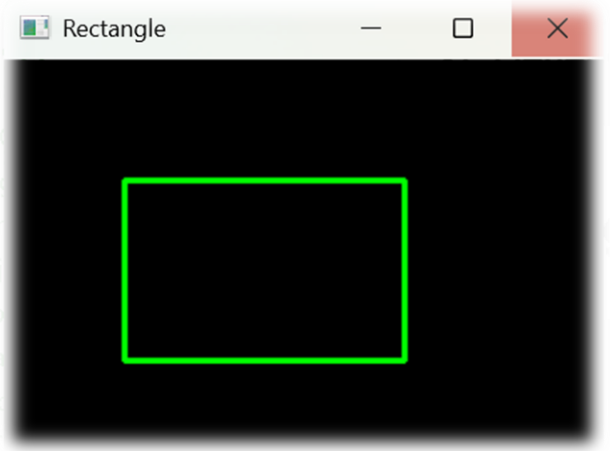
```
cv2.line(img, (50, 50), (200, 50), (255, 0, 0), 3)
```



2. Rectangle

Python

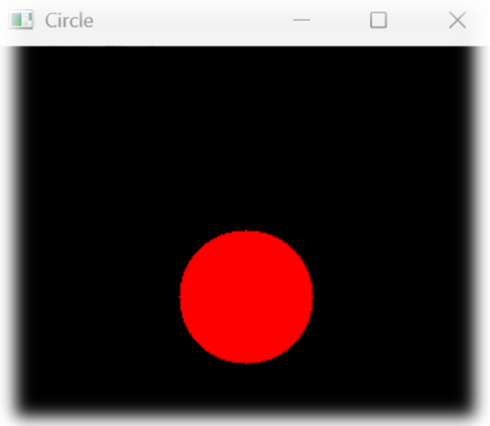
```
cv2.rectangle(img, (60, 60), (200, 150), (0, 255, 0), 2)
```



3. Circle

Python

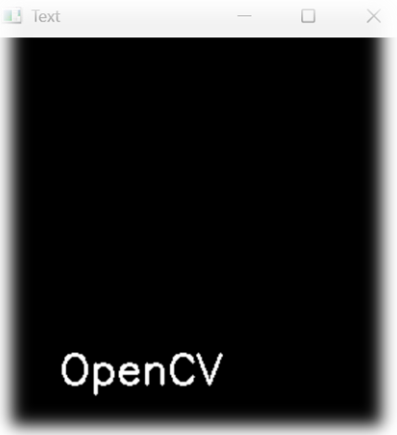
```
cv2.circle(img, (150, 150), 40, (0, 0, 255), -1)  # -1 = filled
```



4. Put Text

Python

```
cv2.putText(img, "OpenCV", (50, 250), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
```



Notes

- Coordinates are in (x, y) format
- Color is in BGR (Blue, Green, Red)
- Thickness: use -1 for filled shapes

Image Resizing, Cropping, and Rotation

1. Resizing

Python

```
resized = cv2.resize(img, (300, 200))  # Width x Height
```

Maintain aspect ratio using scaling:

Python

```
resized = cv2.resize(img, None, fx=0.5, fy=0.5)
```

2. Cropping

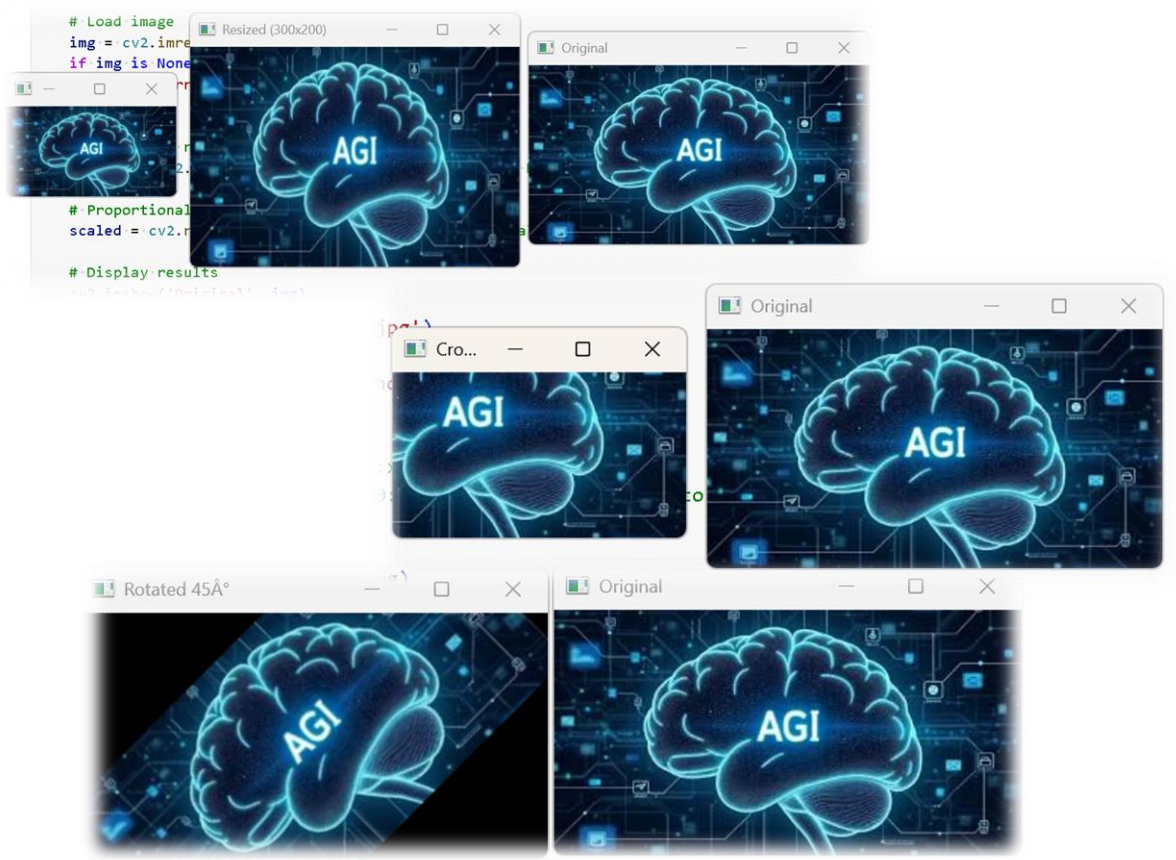
Python

```
cropped = img[50:200, 100:300]  # img[y1:y2, x1:x2]
```

3. Rotation

Python

```
(h, w) = img.shape[:2]
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, 45, 1.0)
# 45° rotationrotated = cv2.warpAffine(img, M, (w, h))
```



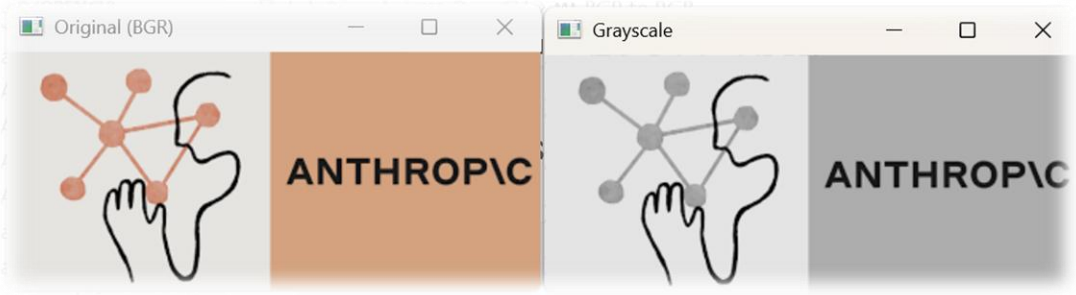
Notes

- Resizing can distort images if aspect ratio not maintained
- Cropping is simply slicing the NumPy array
- Rotation uses affine transform centered on the image

Color Spaces and Conversions

What is a Color Space?

A way to represent colors. Different color spaces are used for different computer vision tasks.



Common Color Conversions:

1. BGR to Grayscale

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

2. BGR to RGB

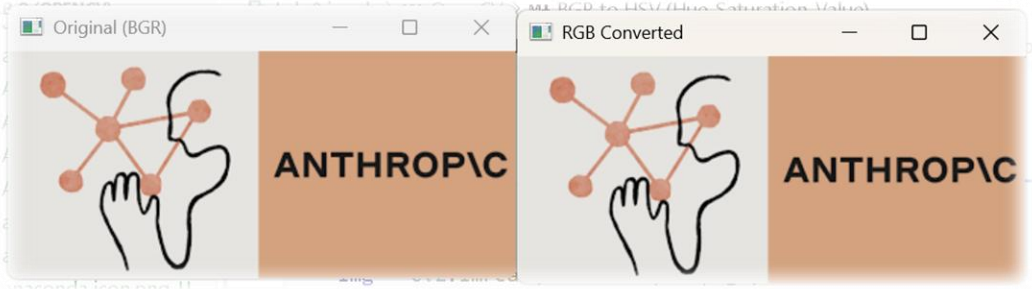
```
rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

3. BGR to HSV

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

4. BGR to LAB

```
lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
```



Use Cases

- **Grayscale:** Simplifies processing
- **HSV:** Better for color filtering
- **RGB:** For displaying with matplotlib
- **LAB:** For lighting-independent processing

Image Thresholding Techniques

What is Thresholding?

A method to segment an image by converting it to binary form (black & white) based on pixel intensity.

1. Simple Thresholding

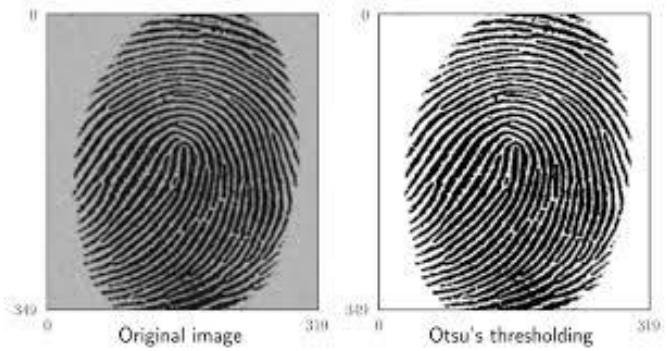
```
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
```

Other Modes:

1. -cv2.THRESH_BINARY_INV
2. cv2.THRESH_TRUNC
3. cv2.THRESH_TOZERO

2. Adaptive Thresholding

```
adaptive = cv2.adaptiveThreshold(
    gray, 255,
    cv2.ADAPTIVE_THRESH_MEAN_C,
    cv2.THRESH_BINARY,     11, 2)
```



3. Otsu's Binarizatio

```
_, otsu = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

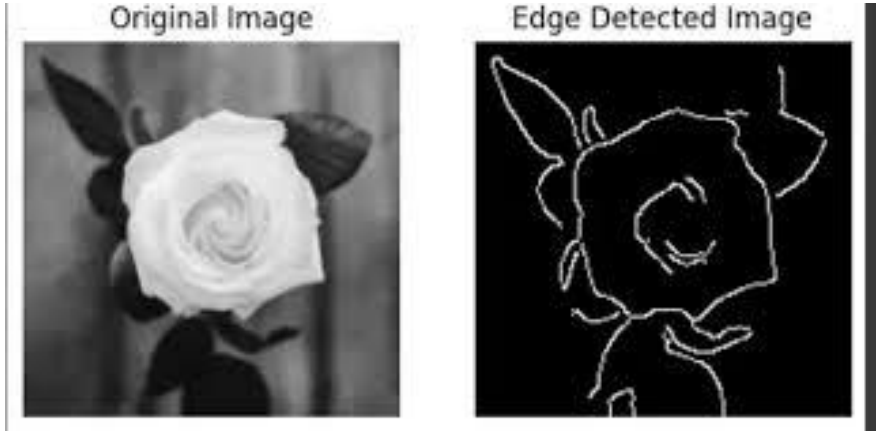
Use Cases

- Object detection
- Text extraction
- Background removal

Edge Detection Techniques

What is Edge Detection?

Edge detection highlights the boundaries and outlines within an image — essential for object detection, shape analysis, etc.



Use Cases

- Contour detection
- Preprocessing for segmentation
- Shape/feature recognition

1. Canny Edge Detection

Python

```
edges = cv2.Canny(img, 100, 200)
```

- Low threshold: 100
- High threshold: 200
- Most widely used and accurate edge detection

Conti...

2. Sobel Operator

Python

```
sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)
sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)
```

Detects edges in horizontal and vertical directions

3. Laplacian Operator

Python

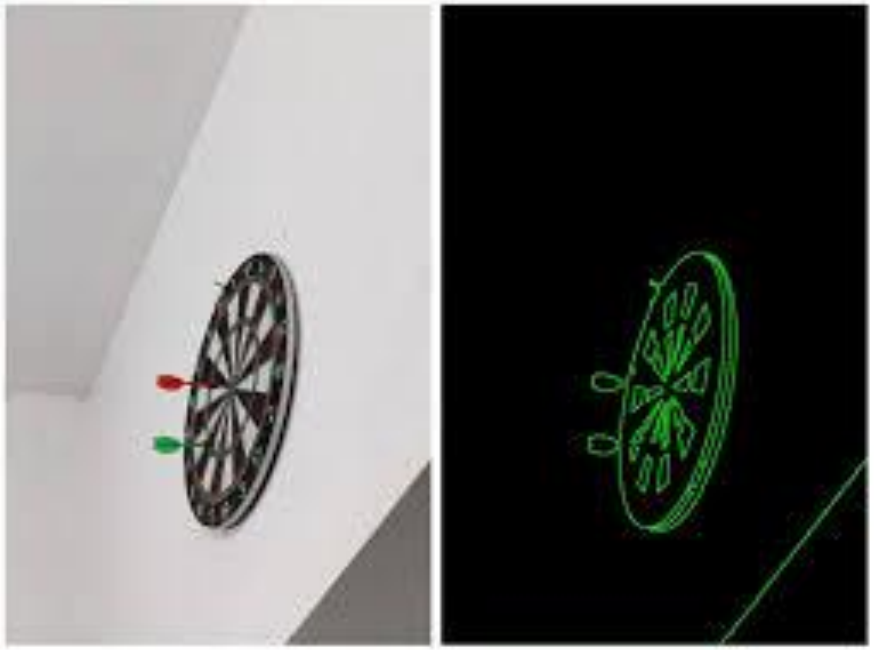
```
laplacian = cv2.Laplacian(gray, cv2.CV_64F)
```

- Captures all-direction edges

Contour Detection and Drawing

What are Contours?

Contours are continuous curves joining all the points along a boundary with the same color or intensity — useful for shape analysis and object detection.



Use Cases

- Object counting
- Shape recognition
- Measuring object area/perimeter

Drawing Contours

Python

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

Draws all contours in green with line thickness of 2.

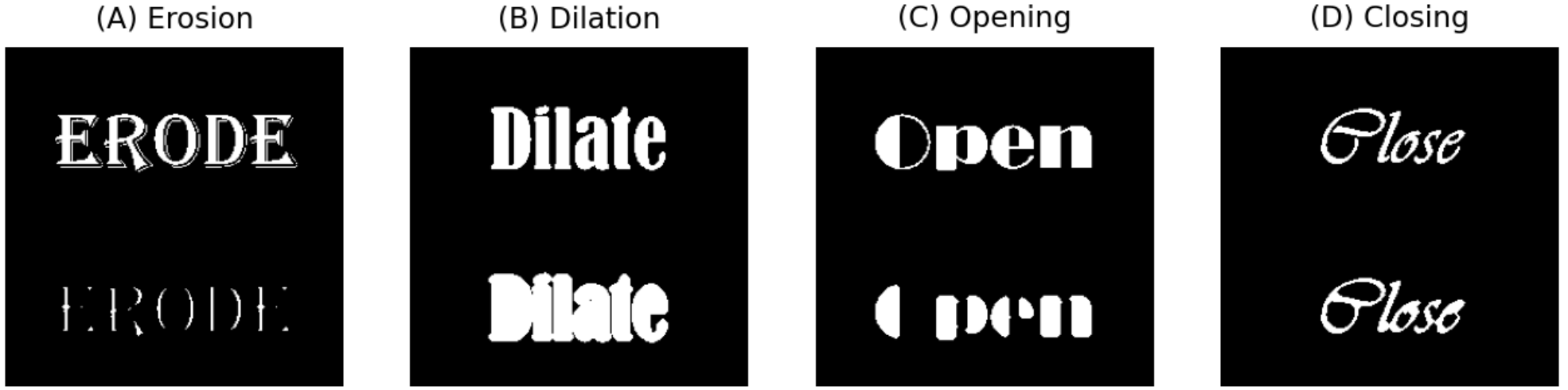
Morphological Transformations

What are Morphological Operations?

Morphological transformations apply a structuring element to process images based on shapes. Common in noise removal and shape manipulation.

Applications

- Noise cleaning
- Filling gaps
- Object separation



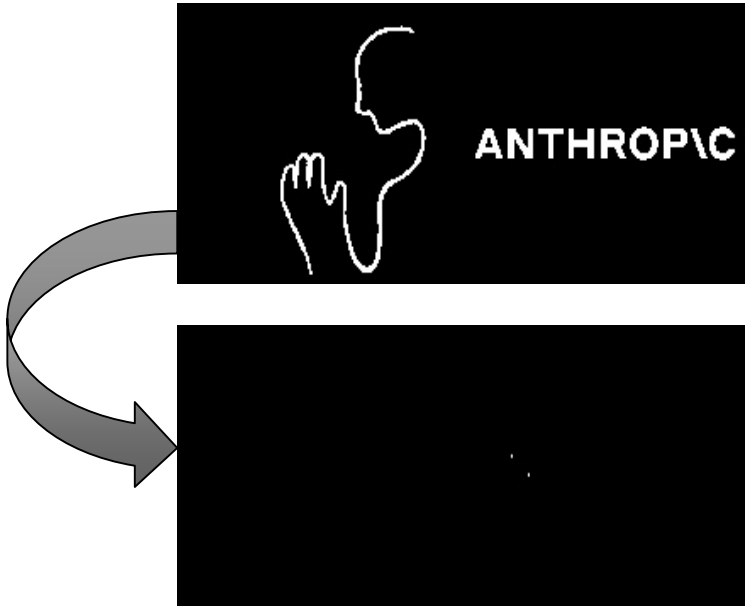
Morphological Transformations

1. Erosion

Python

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

- Removes boundaries of objects
- Useful for removing small white noises

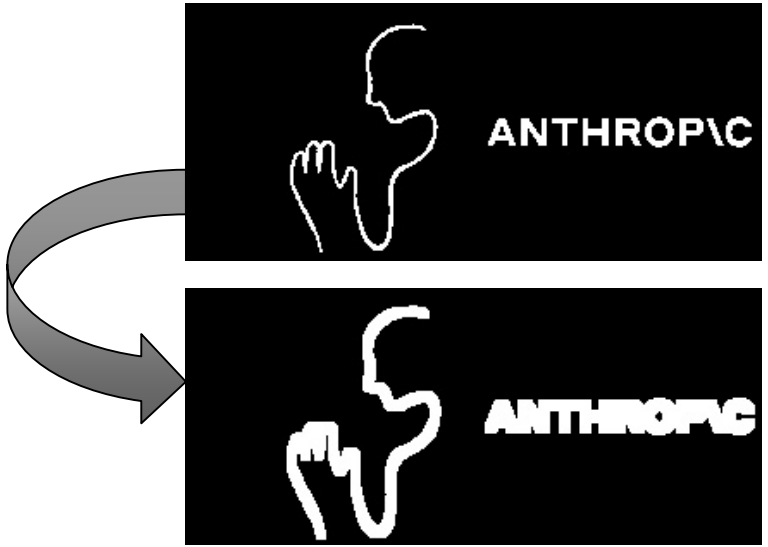


2. Dilation

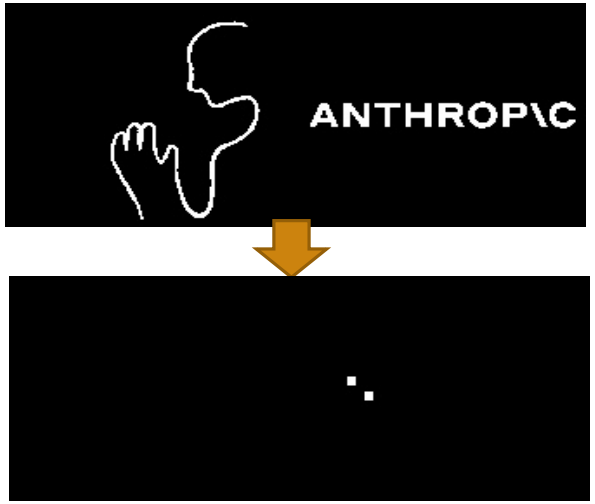
Python

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

- Adds pixels to object boundaries
- Enhances object size



Morphological Transformations

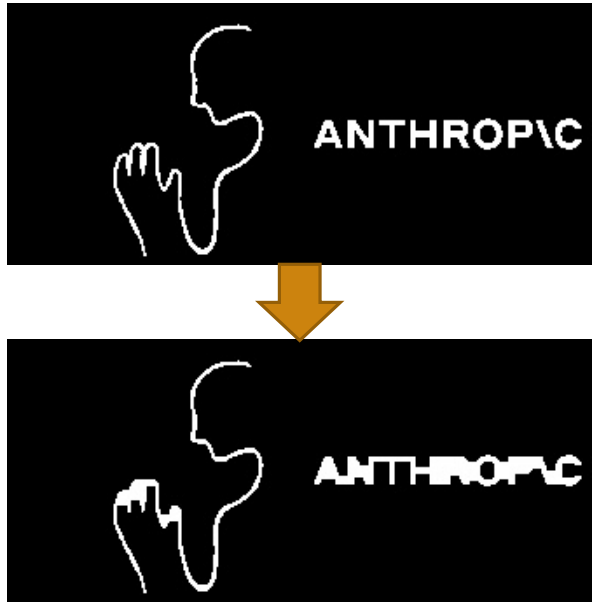


3. Opening (Erosion followed by Dilation)

Python

```
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

- Removes noise while preserving shape



4. Closing (Dilation followed by Erosion)

Python

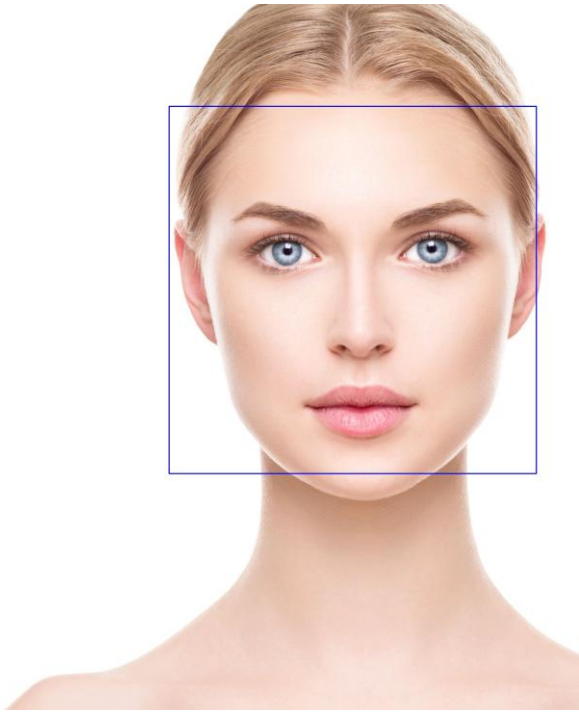
```
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

- Closes small holes inside objects

Face Detection with Haar Cascades

What are Haar Cascades?

A machine learning–based approach used for object detection, especially faces, using features like edges, lines, and rectangles.



How It Works:

- Haar features extracted from the image
- Trained classifiers detect patterns
- Cascade classifier improves detection speed

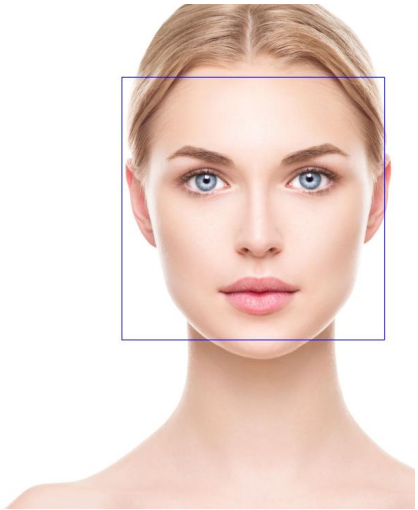
Conti...

Steps in OpenCV:

Python

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
```



Notes

- Use grayscale images for detection
- **Parameters:** scaleFactor, minNeighbors
- Pre-trained XML models available in OpenCV

Object Detection with Pre-trained Models

What is Object Detection?

Object detection involves locating and classifying multiple objects in an image or video stream.

Components of Object Detection

Classification: What is in the image?
Localization: Where is the object?
Detection: What & where together!

Popular Pre-trained Models in OpenCV:

- MobileNet-SSD
- YOLO (You Only Look Once)
- Faster R-CNN

Use Cases

- Real-time surveillance
- Smart cameras
- Traffic monitoring

Image Blending Techniques

What is Image Blending?

Combining two images to produce a visually smooth transition or overla

Types of Blending

1. Simple Addition
2. Weighted Blending
3. Mask-Based Blending

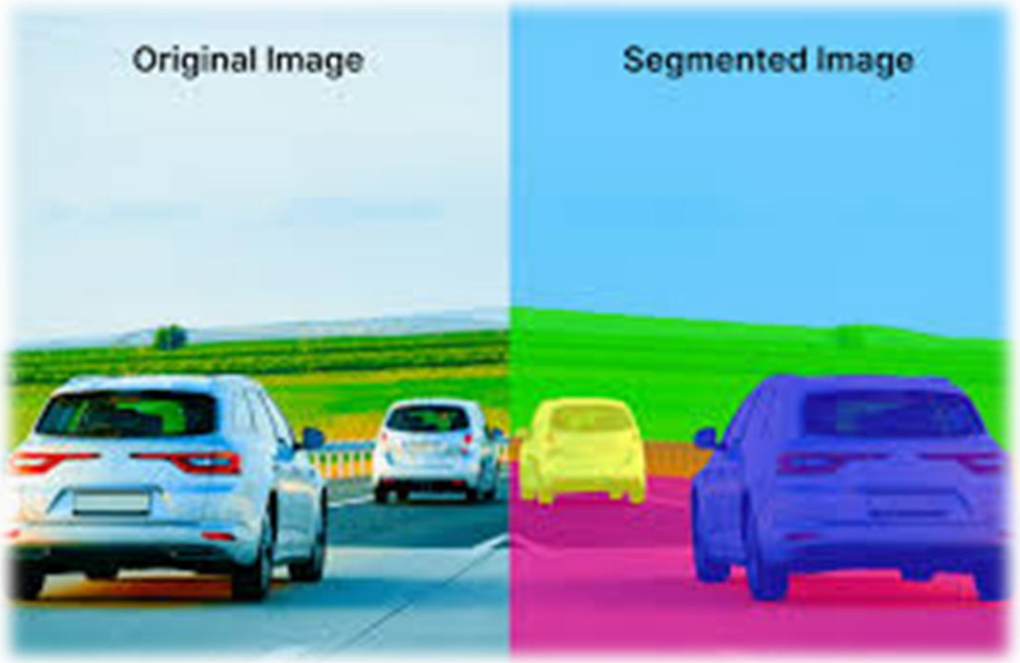
Use Cases

- Overlaying logos
- Transition effects
- Panorama stitching
- Exposure fusion

Image Segmentation in Computer Vision

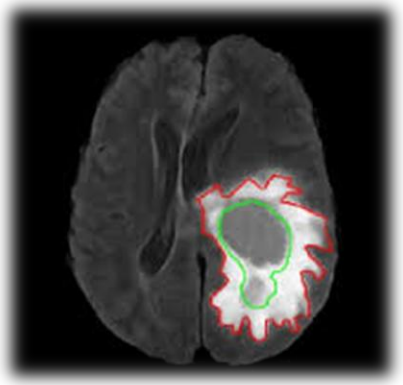
What is Image Segmentation?

Image segmentation is the process of dividing an image into meaningful regions or segments for easier analysis.



Purpose

- Isolate objects from background
- Aid in object detection & recognition
- Used in medical imaging, self-driving cars, etc.



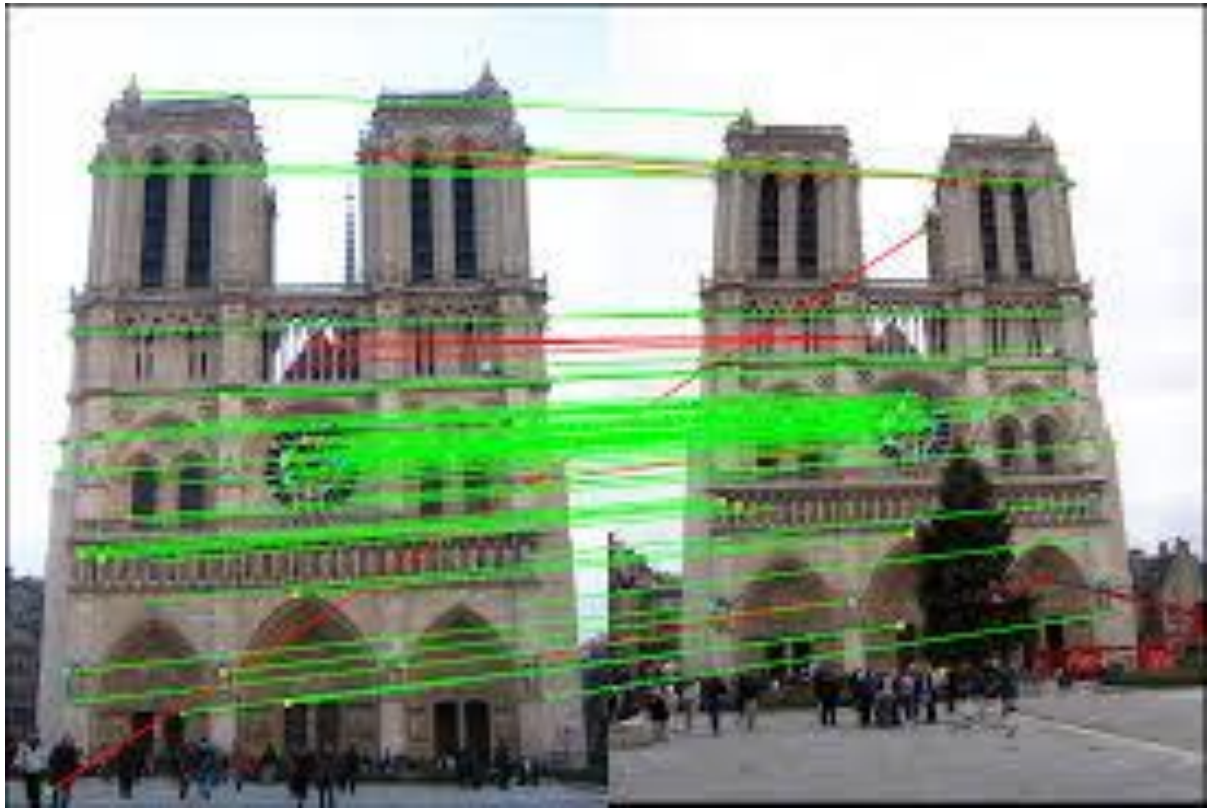
Use Cases

- Medical image analysis
- Face & object recognition
- Lane detection in autonomous driving

Feature Detection in Images

What is Feature Detection?

Feature detection identifies key points or areas of interest in an image that are unique and can be tracked or matched across images.



Popular Feature Detection Algorithms

- Harris Corner Detector
- Shi-Tomasi (Good Features to Track)
- SIFT (Scale-Invariant Feature Transform)
- ORB (Oriented FAST and Rotated BRIEF)

Use Cases

- Image stitching
- Object recognition
- Motion tracking
- Augmented reality

THANK YOU