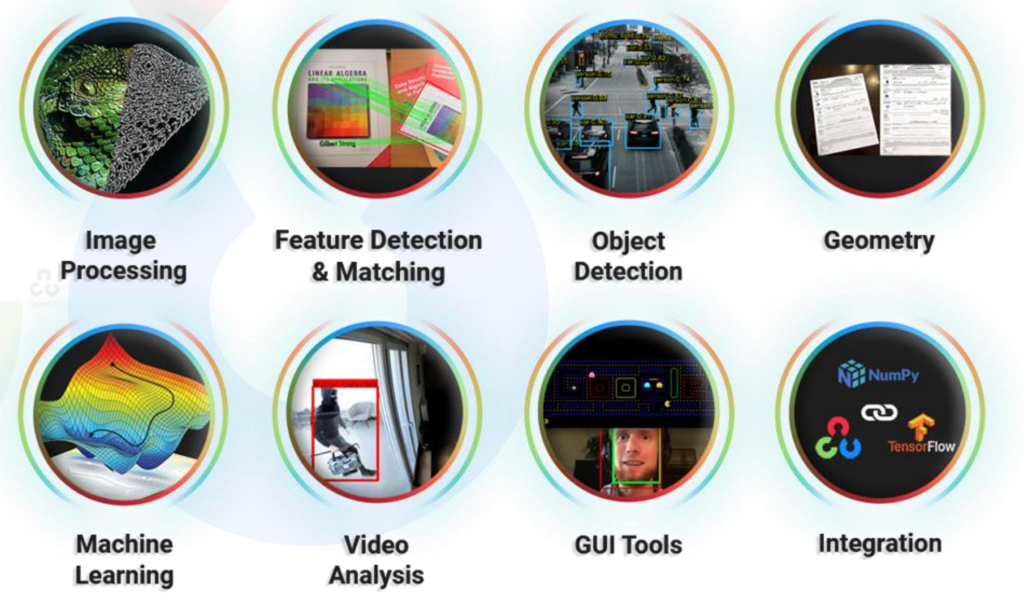
Introduction to OpenCV

A Powerful Open-Source Library for Computer Vision & Image Processing

Presented by :Farooq Shehzad

Overview

- Introduction to OpenCV and its importance
- How to install and set up OpenCV
- Basic image processing techniques
- Edge detection and feature detection
- Object and face detection using OpenCV
- OpenCV applications in deep learning
- Real-world use cases and applications



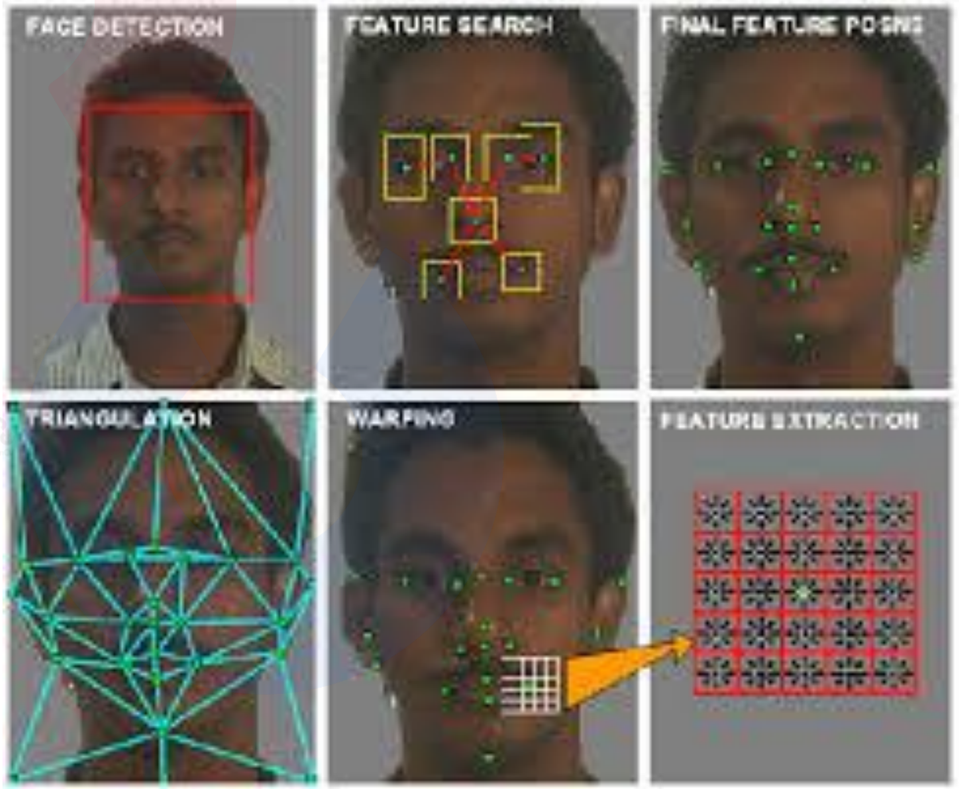
What is OpenCV?

Definition:

OpenCV (Open-Source Computer Vision Library) is an open-source library used for **image processing, computer vision, and machine learning.**
It provides tools for real-time image and video analysis.

Brief History:

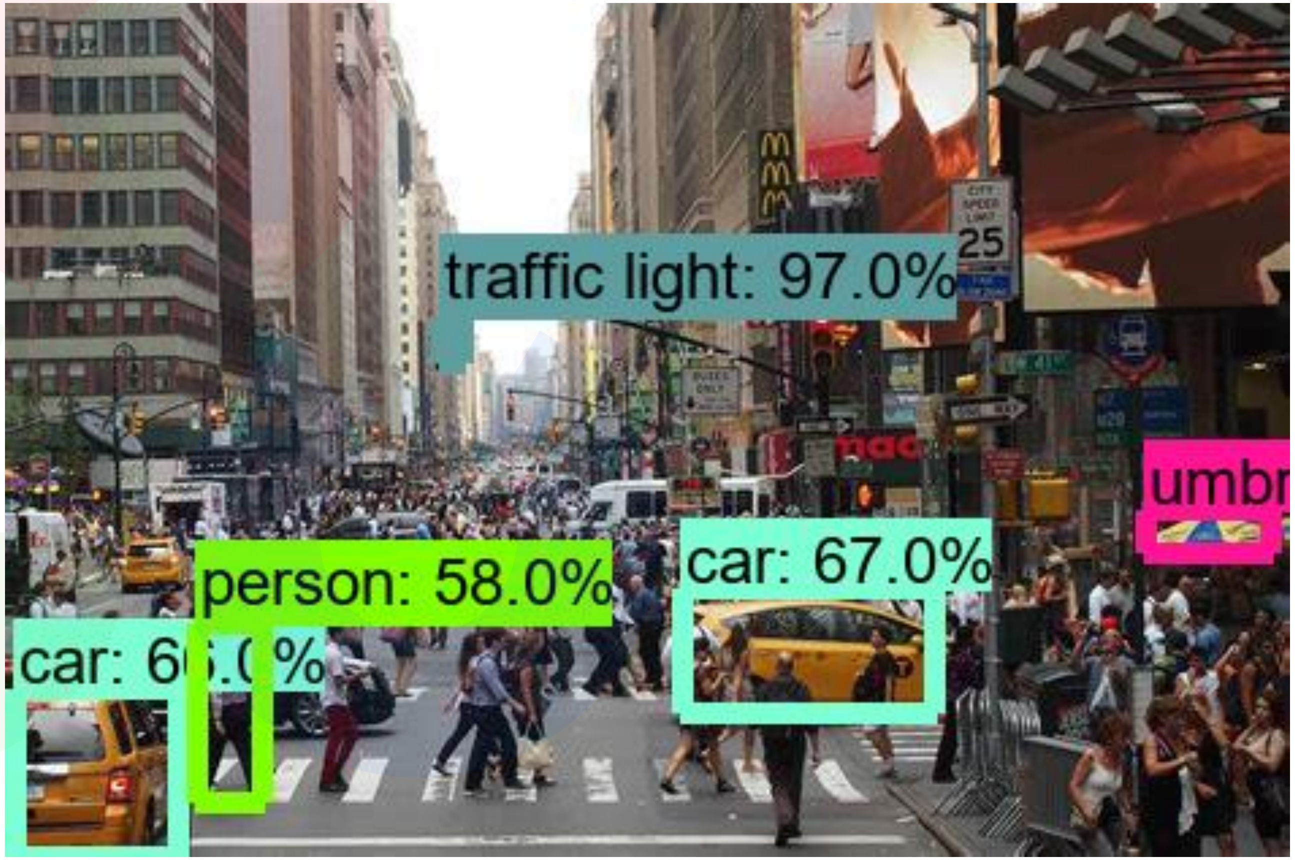
Developed by **Intel in 1999** for computer vision research.
Later, it became open-source and is now maintained by **OpenCV.org**.
Over the years, it has become widely used in **AI, robotics, automation, and deep learning.**



Face Detection through Deep learning using OpenCV Demo

Why Use OpenCV?

- ***Free & Open-Source*** – No cost, large community support.
- ***Fast & Optimized*** – Supports GPU acceleration (CUDA, OpenCL).
- ***Real-Time Processing*** – Works with live camera feeds & video.
- ***Deep Learning Integration*** – Compatible with TensorFlow & PyTorch.
- ***Cross-Platform & Multi-Language*** – Supports Python, C++, Java on Windows, Linux, macOS, and Android.



Object Detection from real world through OpenCV and deep learning

Installing OpenCV

There are two main ways to install OpenCV:

1. **Using pip (for standard Python)**
2. **Using Anaconda (for Conda environments)**

1. Installing OpenCV Using pip (Recommended for Most Users)

1 **Open Command Prompt (cmd) or Terminal**

2 Run the following command:

```
pip install opencv-python
```

3 To install with extra modules:

```
pip install opencv-contrib-python
```

4 Verify installation:

```
import cv2
print(cv2.__version__)
```

Installing OpenCV (continuous)

2. Installing OpenCV in Anaconda (Conda Environment)

1 **Open Anaconda Prompt**

2 Create a new environment (optional but recommended):

```
conda create --name myenv python=3.9
conda activate myenv
```

◆ Note: Replace "**myenv**" with any environment name you prefer.

3 Install OpenCV:

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

4 Verify installation:

```
import cv2
print(cv2.__version__)
```

Opening Jupyter Notebook

Jupyter Notebook is commonly used for writing and running OpenCV code in Python

☑ ***Method 1:*** **Opening Jupyter Notebook from Anaconda**

1. Open **Anaconda Navigator**.
2. Click on **Jupyter Notebook** and wait for it to launch in your browser.

☑ ***Method 2:*** **Opening Jupyter Notebook from Command Line**

1. Open **Anaconda Prompt** or **Command Prompt**.
2. Activate your Conda environment (replace "**myenv**" with your environment name):

```
conda activate myenv
```

3. Start Jupyter Notebook:

```
jupyter notebook
```

4. A new browser tab will open where you can create a Python notebook.

☑ ***Method 3:*** **Installing Jupyter Notebook (If Not Installed)**

```
pip install notebook
```

Reading and Displaying Images in OpenCV

OpenCV allows **reading** and **displaying** images easily using **cv2.imread()** and **cv2.imshow()** functions.

Steps to Read and Display an Image

1 **Import OpenCV:**

```
import cv2
```

2 **Read an image from a file**

```
# Load an image
img = cv2.imread("image.jpg")
```

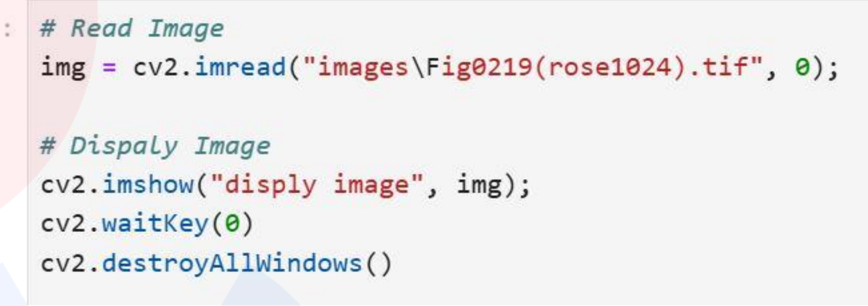
3 **Display the image:**

```
# Show the image
cv2.imshow("Displayed Image", img)
```

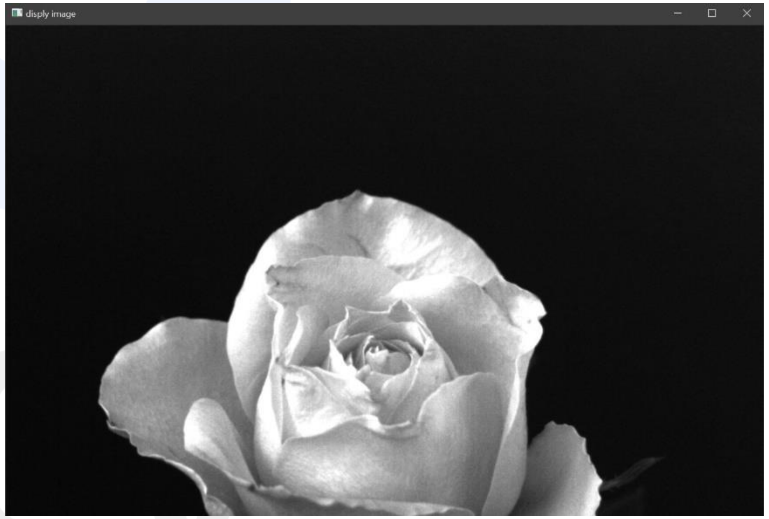
4 **Wait for a key press & close the window:**

```
 # Wait indefinitely until a key is pressed
cv2.waitKey(0)
 # Close all windows
cv2.destroyAllWindows()
```

Visualizing Image Pixels as table



```
# Read Image
img = cv2.imread("images\Fig0219(rose1024).tif", 0);

# Dispaly Image
cv2.imshow("disply image", img);
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Screenshot of an image being displayed using OpenCV.

Basic Image Processing in OpenCV

OpenCV provides various functions for basic **image processing** like **resizing**, **converting to grayscale**, and **applying filters**.

1. Convert to Grayscale

- Converts a color image to a **black & white** (grayscale) format.
- Useful for reducing complexity and improving processing speed.

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

cv2.imshow("Grayscale Image", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Colored Image

Input

Output

Grayscale Image

Basic Image Processing in OpenCV (Continuous)

2. Resize an Image

- Changes the dimensions of an image to a **specific width and height**.
- Helps in **reducing memory usage** and optimizing performance.

Original Image

Input

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Resize to 300x300
resized = cv2.resize(img, (64, 64))

cv2.imshow("Resized Image", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output

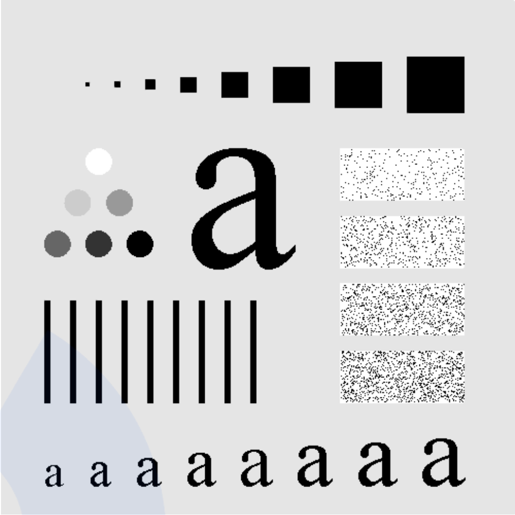
Resized image

Basic Image Processing in OpenCV *(Continuous)*

3. Apply Gaussian Blur

- Smoothens the image by reducing noise and detail.
- Used in **object detection and edge detection** tasks.

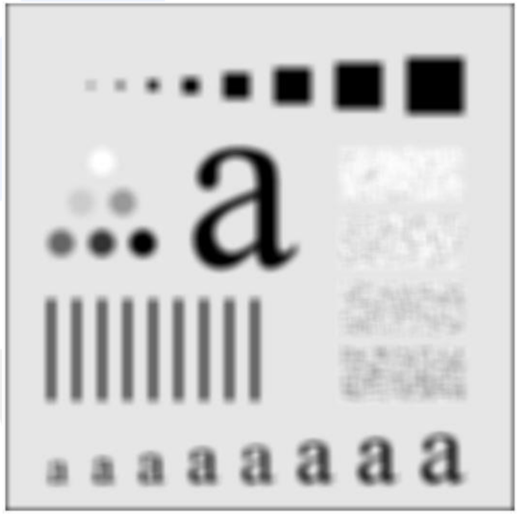
Colored Image



Input

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Apply Gaussian blur
blurred = cv2.GaussianBlur(img, (9, 9), 0)

cv2.imshow("Resized Image", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output



Blurred image

Edge Detection in OpenCV

Edge detection identifies object boundaries by detecting sharp intensity changes in an image. OpenCV provides multiple methods for this, including the Laplacian operator.

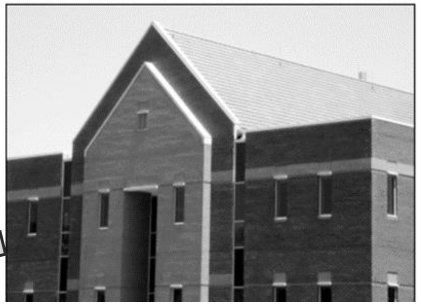
1. What is Edge Detection?

- Detects **significant changes in pixel intensity**.
- Used in **image processing, object detection, and pattern recognition**.

2. Laplacian Edge Detection

- Uses second-order derivatives to highlight edges in all directions.
- Enhances regions with **rapid intensity changes**.

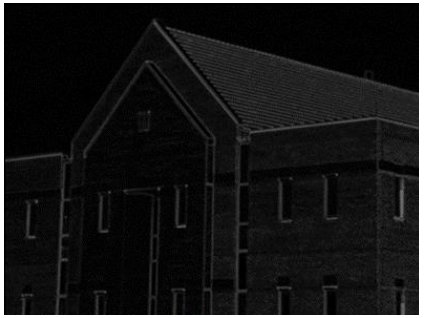
Original Image



Input

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Apply Laplacian edge detection
edges = cv2.Laplacian(img, cv2.CV_64F)

# Convert edges to 8-bit for display
edges = cv2.convertScaleAbs(edges)
```

```
# Show the edge-detected images
cv2.imshow("Laplacian Edge Detection",
edges)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



Edge Detected Image

Edge Detection in OpenCV *(continue)*

3. Other Edge Detection Methods:

- ✅ **Sobel Operator:**
  Detects edges separately in horizontal & vertical directions.
- ✅ **Canny Edge Detector:**
  More advanced, applies gradient thresholding & edge tracking.

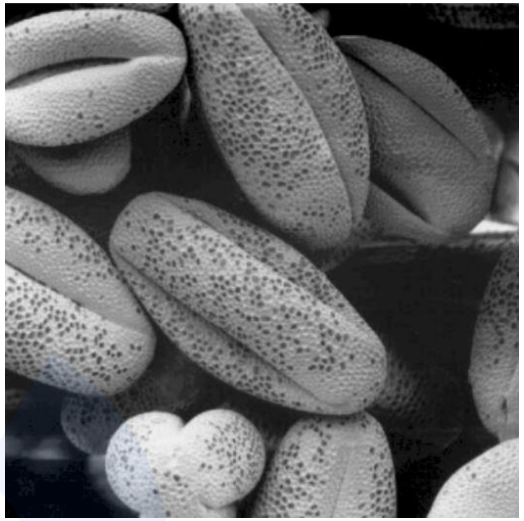
Image Thresholding in OpenCV

Thresholding is a technique used to convert **grayscale images into binary (black & white) images** based on a **threshold** value. It is useful in object segmentation, OCR, and image preprocessing.

1. Simple Thresholding

If a pixel's intensity is greater than the threshold, it's set to white; otherwise, it's set to black.

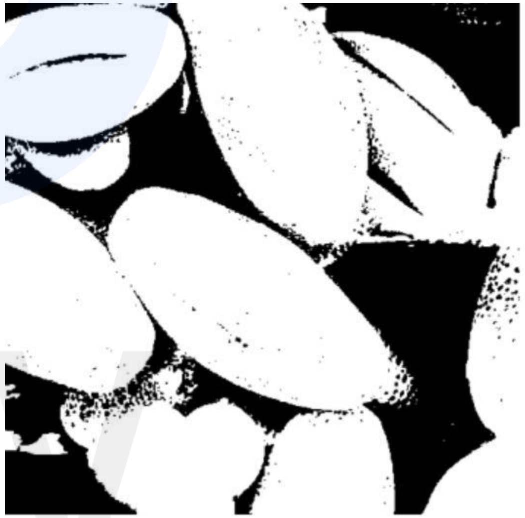
Original Image



Input

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Apply simple thresholding
ret, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
cv2.imshow("Grayscale Image", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output



Thrshold Image

Feature Detection in OpenCV

Feature detection identifies key points in an image, such as **corners**, **edges**, and **blobs**, which help in **object recognition** and **tracking**.

1. Harris Corner Detection

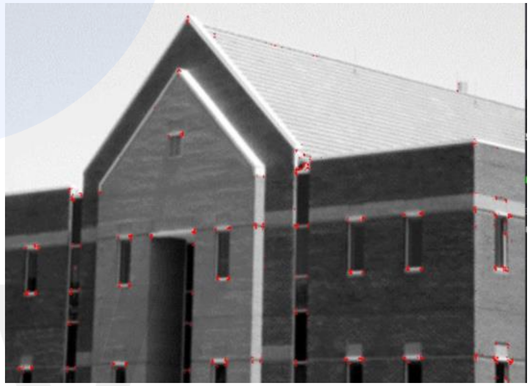
Detects corners by analyzing intensity variations

```
import cv2

# Load the imageimport cv2

img = cv2.imread("image.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

harris = cv2.cornerHarris(gray, 2, 3, 0.04)
img[harris > 0.01 * harris.max()] = [0, 0, 255]  # Highlight corners

cv2.imshow("Harris Corner Detection", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

img = cv2.imread("image.jpg")

# Apply simple thresholding
ret, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

cv2.imshow("Grayscale Image", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Original Image

Input

Output



detected corners image

Feature Detection in OpenCV *(continue)*

2. ORB Feature Detection

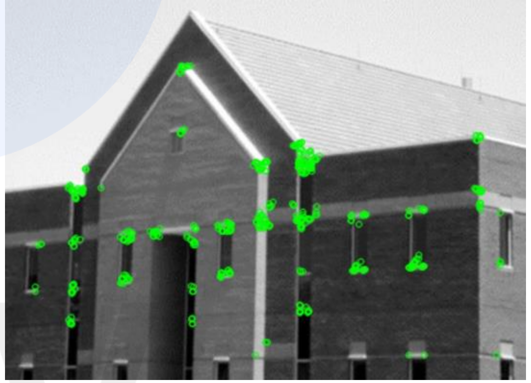
Detects key points efficiently for real-time applications.

```
import cv2
img = cv2.imread("images/house.JPG")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


orb = cv2.ORB_create()
keypoints = orb.detect(img, None)
img = cv2.drawKeypoints(img, keypoints, None, color=(0,255,0))


cv2.imshow("Harris Corner Detection", img)
cv2.waitKey(0
)cv2.destroyAllWindows()
```

Original Image

Input

Output



ORB Feature Detected Image

Face Detection with OpenCV

Face detection is the process of **identifying human faces** in an image or video using **machine learning models**. OpenCV provides **pre-trained models** for this task

1. Using Haar Cascade Classifier

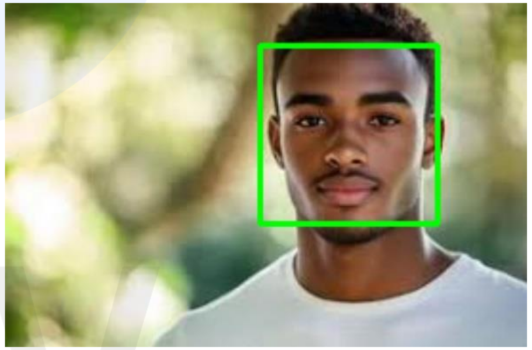
A pre-trained XML file is used to detect faces.

```
import cv2

# Load Haar cascade for face detection
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# Read image and convert to grayscale
img = cv2.imread("face.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect faces
faces = face_cascade.detectMultiScale(gray, 1.3, 5)

# Draw rectangles around detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow("Face Detection", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Original Image

Input

Output



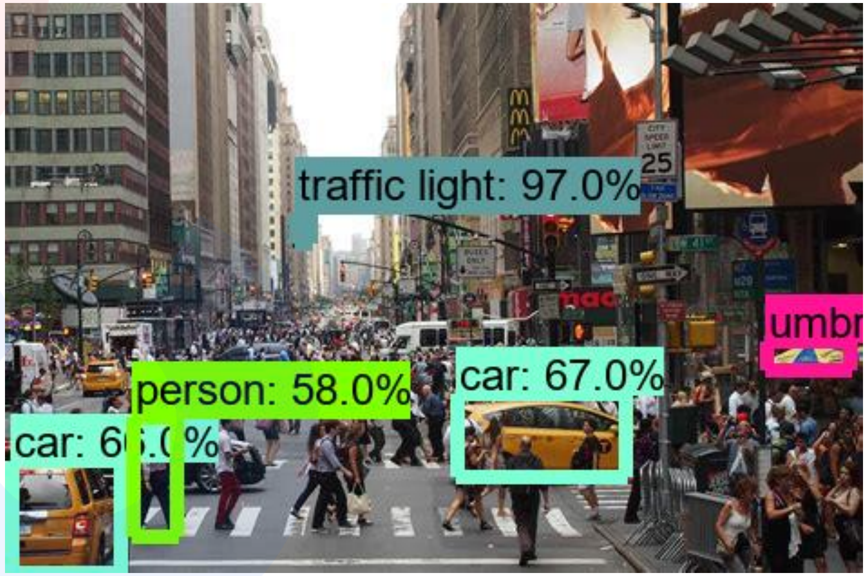
detected face in image

Object Detection in OpenCV

✅ Object Detection is a technique used to locate and classify objects within an image or video.

✅ It identifies objects and draws bounding boxes around them.

✅ OpenCV provides multiple techniques for object detection, such as:Deep Learning Models (YOLO, SSD, Faster R-CNN)



Object Detected Image

What is YOLO?

✅ YOLO (You Only Look Once) is a deep learning-based object detection algorithm.
✅ It is fast, accurate, and efficient, making it ideal for real-time object detection.
✅ YOLO processes an image in a single pass, unlike traditional sliding window approaches.

Any Question

Thank You