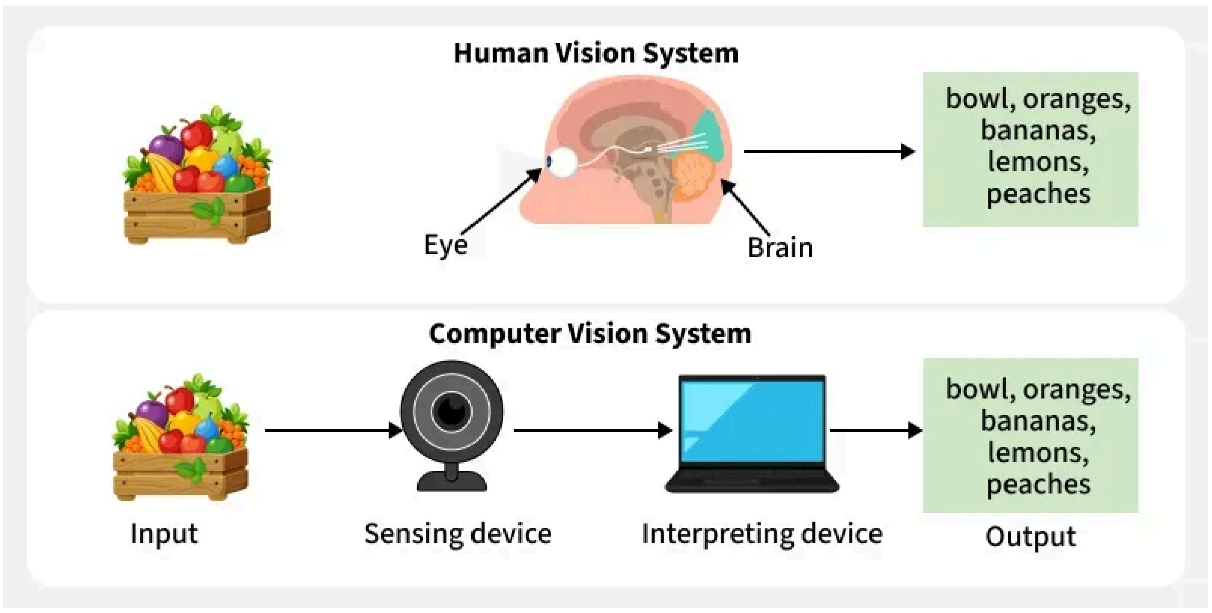
OPEN-CV

What is a Computer Vision

Computer Vision is a branch of computer science that helps computers understand and work with pictures and videos. It teaches machines to "see" like humans do, so they can recognize faces, objects, or actions in images.

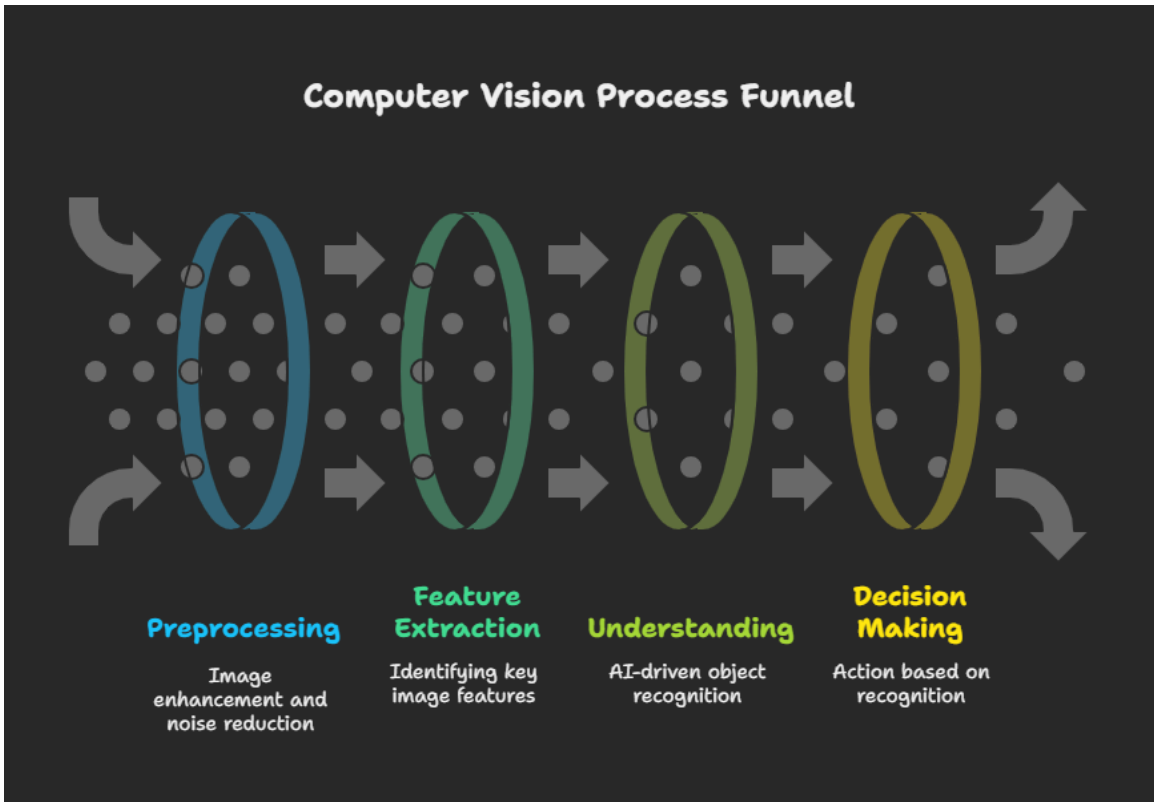


Computer Vision System – Tasks:

A **Computer Vision System** performs the following main tasks:

1. **Image Acquisition** – Capturing images using a camera or sensor.
2. **Image Preprocessing** – Improving image quality (e.g., removing noise, adjusting brightness).
3. **Object Detection** – Finding and locating objects in the image.
4. **Object Recognition** – Identifying what the object is (e.g., car, person, animal).
5. **Image Segmentation** – Dividing an image into parts or objects.
6. **Feature Extraction** – Picking out important details (like edges, shapes, colors).

7. **Classification** – Grouping or labeling the image (e.g., cat vs. dog).
8. **Decision Making** – Using the visual information to take action (e.g., unlock phone, stop a car).


Computer Vision Process Funnel
Preprocessing
Image enhancement and noise reduction
Feature Extraction
Identifying key image features
Understanding
AI-driven object recognition
Decision Making
Action based on recognition

What is OpenCV?

OpenCV stands for **Open Source Computer Vision Library**.
It is a **free and open-source software library** used to help computers understand and process **images and videos**.

It was **developed by Intel in 1999** and supports programming languages like **C++, Python, Java**, and more. OpenCV is widely used in tasks such as **face detection, object tracking, motion detection, and image processing**.

It works on **Windows, Linux, Mac**, and even **mobile devices**, making it a popular choice for building **real-time computer vision applications**.

Uses of OpenCV:

- Read and display images and videos

- Detect faces, eyes, and objects
- Apply filters and effects
- Track motion in videos
- Create real-time vision systems like surveillance or robotics

Tasks of OpenCV:

OpenCV is used to perform many tasks related to image and video processing. The main tasks include:

1. **Image Reading and Writing**
    - Load and save images in different formats (JPEG, PNG, etc.)
2. **Video Capture and Processing**
    - Capture real-time video from a camera or file
    - Process each frame of the video
3. **Image Processing**
    - Resize, crop, rotate, blur, sharpen images
    - Convert color (e.g., RGB to grayscale)
4. **Object Detection and Recognition**
    - Detect faces, eyes, hands, and other objects
    - Recognize known objects using trained models
5. **Feature Detection and Matching**
    - Find key points, edges, and corners in an image
    - Match features between two images
6. **Image Segmentation**

    - Divide an image into different parts or regions

7. **Motion Detection and Tracking**
    - Track movement of objects in a video
8. **Machine Learning Integration**
    - Use trained models for classification or prediction tasks

How OpenCV Works – Simple Explanation

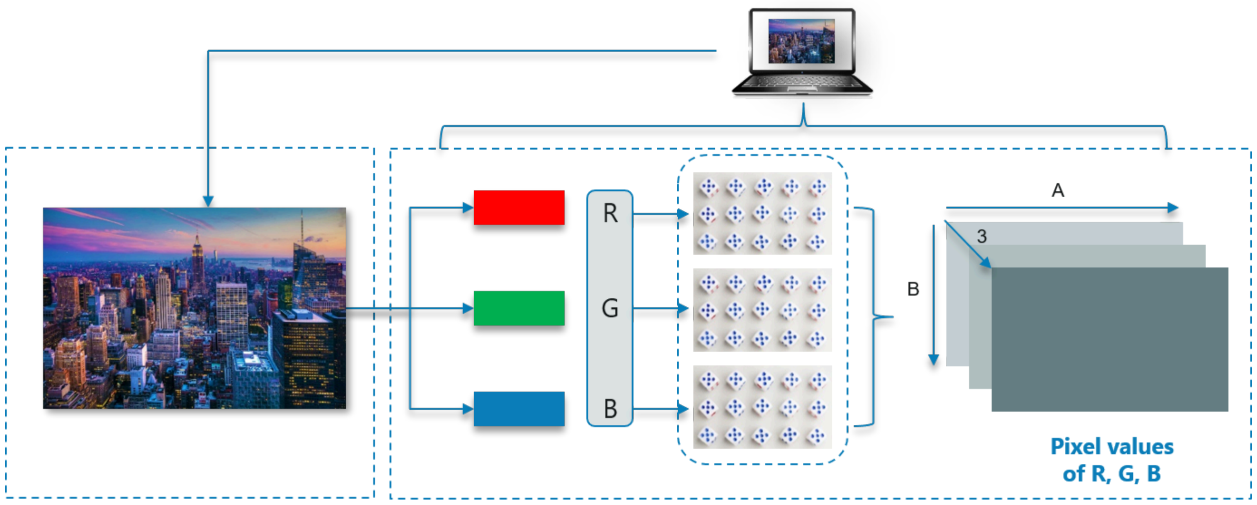
OpenCV works by using a set of pre-built functions and tools that help the computer process and understand images or videos. It follows these basic steps:

🧠 Steps in How OpenCV Works:

1. **Image or Video Input**
   OpenCV reads the image or video using a camera or file.
2. **Image Processing**
   It performs operations like resizing, converting to grayscale, blurring, etc.
3. **Feature Detection**
   It detects features like edges, corners, faces, or objects in the image.
4. **Analysis or Recognition**
   OpenCV uses algorithms or machine learning models to recognize or classify objects.
5. **Output or Action**
   Based on the result, it shows the output or performs an action like marking a face, saving an image, or controlling a device.

🧪 Example:

In face detection:
→ OpenCV reads the image → Converts it to grayscale → Applies face detection algorithm → Draws a rectangle on the face.



Why OpenCV is Used for Computer Vision?

OpenCV is used for **Computer Vision** because it provides **powerful tools** and **ready-made functions** that make it easy to build applications that can understand images and videos.

✅ Reasons Why OpenCV is Used:

1. **Free and Open Source**
   Anyone can use it without cost, and it is customizable.

2. **Supports Multiple Languages**
   Works with Python, C++, Java, etc., making it flexible.

3. **Large Collection of Functions**
   It has built-in tools for face detection, object tracking, image filtering, etc.

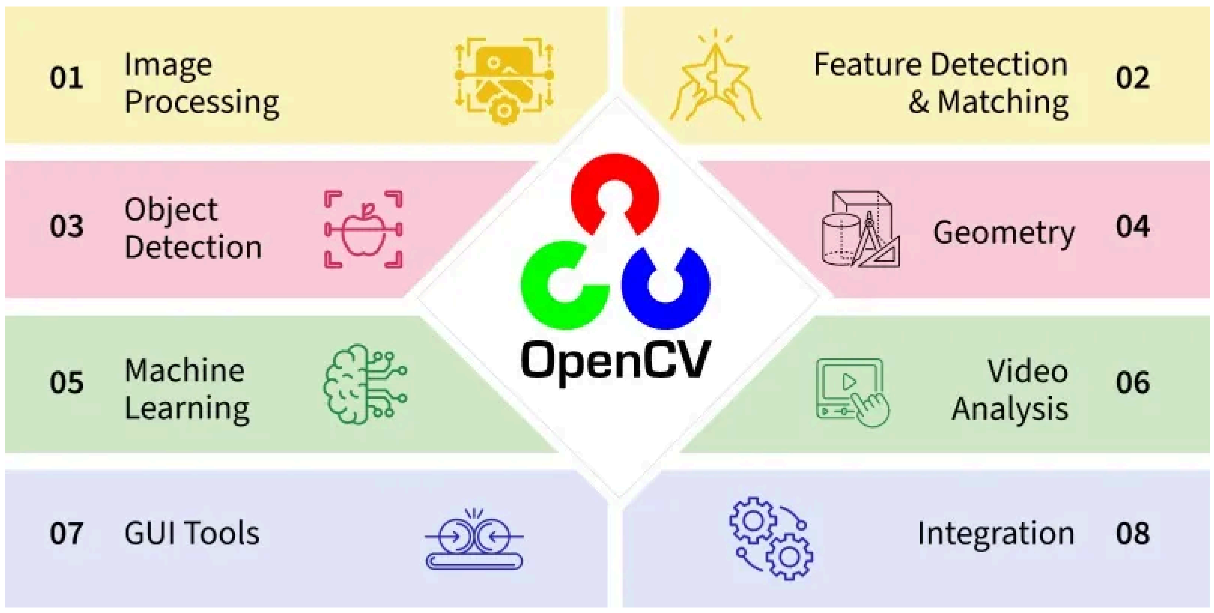
4. **Cross-Platform Support**
   Runs on Windows, Linux, Mac, and mobile devices.

5. **Fast and Efficient**
   Optimized for real-time processing of images and videos.

6. **Community Support**
   A large community provides help, tutorials, and updates.

🧠 In Simple Words:

OpenCV makes it **easy, fast, and efficient** to build **computer vision** applications like face recognition, surveillance, robots, and more.



📘 How to Read and Show Images in OpenCV (Python)

✅ 1. Import OpenCV Library

```
import cv2
```

✅ 2. Read the Image

```
img=cv2.imread(r"Images\img1.jpg")              # Read the Image
```

- 'image.jpg' is the file name or path.
- The image is stored in a variable as a matrix (array).

✅ 3. Display the Image

```
cv2.imshow('My Image', image)  # 'My Image' is the window name
cv2.waitKey(0)              # Waits for a key press to close the window
cv2.destroyAllWindows()       # Closes all OpenCV windows
```

📝 Summary:

Step	Function	Purpose
1	cv2.imread()	Reads the image
2	cv2.imshow()	Shows the image in a window
3	cv2.waitKey(0)	Waits for key press
4	cv2.destroyAllWindows()	Closes all windows

⚠️ Important Notes:

- The image file must exist in the given path.
- Use cv2.waitKey(0) to keep the image window open.
- You can read images in grayscale:

```
image = cv2.imread('image.jpg', 0)
```

📘 How to Resize an Image in OpenCV

✅ Step-by-step Code:

```
import cv2

# Read the original image

image = cv2.imread('image.jpg')



# Resize the image (width, height)

resized_image = cv2.resize(image, (400, 300))
```

```
# Display the resized image

cv2.imshow('Resized Image', resized_image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

📝 Explanation:

Function	Purpose
cv2.imread()	Reads the image
cv2.resize(image, (width, height))	Resizes the image to new dimensions
cv2.imshow()	Shows the resized image
cv2.waitKey(0)	Waits for a key press
cv2.destroyAllWindows()	Closes the window

⚠️ Important Notes:

- The size is given as **(width, height)**, not (height, width).
- You can also resize using a scale factor:

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

This resizes the image to **50% of its original size**.

How to Create a Slide Show in OpenCV (Python)

```
import cv2

import os

# Path to the image folder

path = r"C:\Users\Vikas\Desktop\OpenCV\Images"



# Get list of all image file names

list_name = os.listdir(path)



# Loop through each image

for name in list_name:

  img_path = os.path.join(path, name)  # Safer than manually joining

  img = cv2.imread(img_path)



  if img is not None:

    img = cv2.resize(img, (400, 400))  # Resize to 400x400

    cv2.imshow("Slide Show", img)

    cv2.waitKey(2000)  # Show image for 2 seconds

  else:

    print(f"Failed to load {img_path}")
```

```
cv2.destroyAllWindows()
```

📘 cv2.imread() – Read Image Function and Flags in OpenCV

✅ Syntax:

```
cv2.imread(filename, flags)
```

- filename: Path of the image file (e.g., 'image.jpg')
- flags: Specifies how the image should be read (optional)

🚩 Common flags used in cv2.imread():

Flag	Description
cv2.IMREAD_COLOR (or 1)	Loads a color image (default). Ignores alpha (transparency) channel.
cv2.IMREAD_GRAYSCALE (or 0)	Loads image in grayscale (black and white).
cv2.IMREAD_UNCHANGED (or -1)	Loads image as it is, including alpha channel (transparency).

🧪 Examples:

```
img1 = cv2.imread('photo.jpg', cv2.IMREAD_COLOR)      # Color image
img2 = cv2.imread('photo.jpg', cv2.IMREAD_GRAYSCALE)  # Grayscale image
```

```
img3 = cv2.imread('photo.png', cv2.IMREAD_UNCHANGED)  # Original image with transparency
```

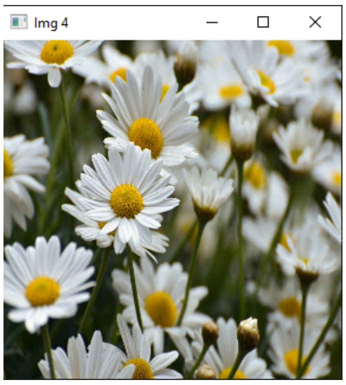
📌 Default Behavior:

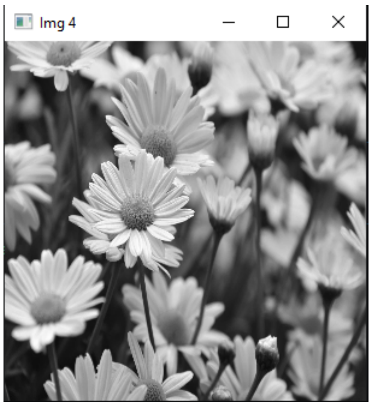
If you don’t give a flag, OpenCV uses cv2.IMREAD_COLOR by default.

Example

```
img = cv2.imread(r"C:\Users\Vikas\Desktop\OpenCV\Images\img3.jpg",0) #bgr
(0-255,0-255,0-255)

print(img.shape)

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

cv2.imshow("Img 3",img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Output





📘 How to Add Text Over Images Using OpenCV

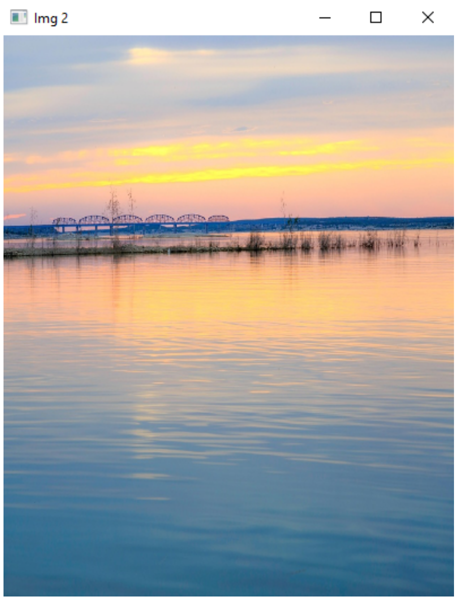
```
cv2.putText(img, text, org, fontFace, fontScale, color, thickness, lineType, bottomLeftOrigin)
```

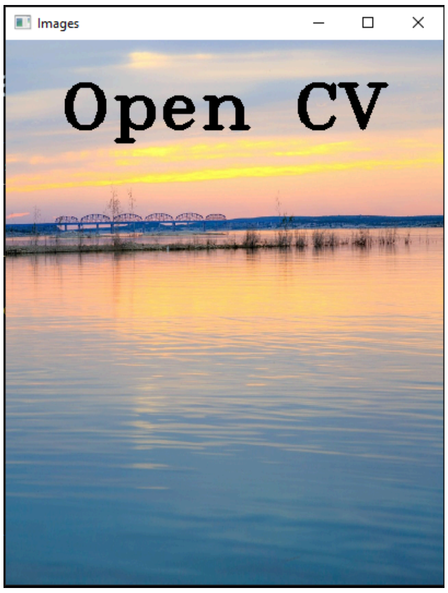
🧠 Parameters:

Parameter	Description
img	Image on which to write text
text	The string/text to be drawn
org	Bottom-left corner of the text in (x, y)
fontFace	Font style (e.g., cv2.FONT_HERSHEY_SIMPLEX)
fontScale	Size of the text
color	Text color in BGR format (e.g., (0,0,255) for red)
thickness	Thickness of the text
lineType	Line type (e.g., cv2.LINE_AA for anti-aliased text)
bottomLeftOrigin	If True, text origin is at the bottom-left (default False)

```
img_get = cv2.imread(r"C:\Users\Vikas\Desktop\OpenCV\Images\img2.jpg")

img_get = cv2.resize(img_get,(400,500))
```

```
txt = cv2.putText(img = img_get,

text = "Open VC",

org = (50,80),

fontFace=cv2.FONT_HERSHEY_COMPLEX_SMALL,

fontScale = 3,

color = (0,0,0),

thickness = 3,

lineType = cv2.LINE_8,

bottomLeftOrigin= False)



cv2.imshow("Images",img_get)

cv2.waitKey(0)

cv2.destroyAllWindows()
```





📘 How to Draw Rectangle and Line on Image Using OpenCV

OpenCV provides simple functions like cv2.rectangle() and cv2.line() to draw shapes on images.

✅ 1. Drawing a Rectangle

🔧 Syntax:

```
cv2.rectangle(img, pt1, pt2, color, thickness)
```

Parameter	Description
img	Image on which to draw
pt1	Top-left corner of rectangle (x1, y1)
pt2	Bottom-right corner (x2, y2)
color	Color in BGR (e.g., (0, 255, 0) = green)
thickness	Thickness of the border (use -1 to fill)

🧪 Example:

```
import cv2


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

img = cv2.resize(img, (400, 400))
```

```
cv2.rectangle(img, (50, 50), (300, 300), (255, 0, 0), 3)  # Blue rectangle


cv2.imshow("Rectangle", img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

✅ 2. Drawing a Line

🔧 Syntax:

```
cv2.line(img, pt1, pt2, color, thickness)
```

Parameter	Description
img	Image on which to draw
pt1	Starting point (x1, y1)
pt2	Ending point (x2, y2)
color	Color of the line
thickness	Line thickness

🧪 Example:

python

CopyEdit

```
cv2.line(img, (0, 0), (400, 400), (0, 255, 0), 4)  # Green line
```

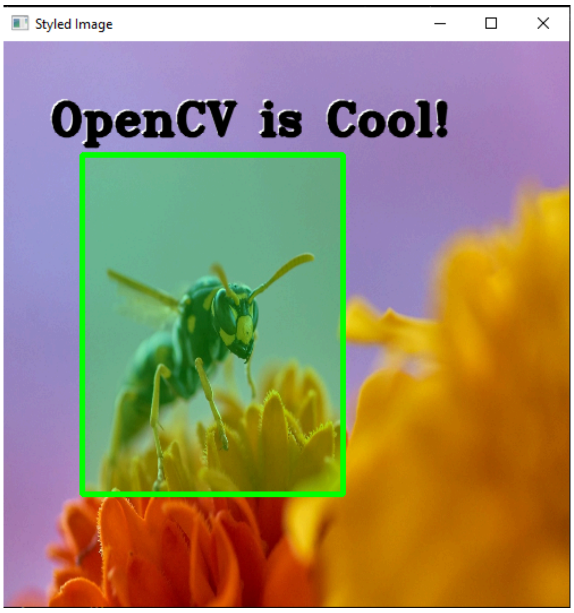
📝 Summary Table:

Function	Use
cv2.rectangle()	Draws a rectangle
cv2.line()	Draws a straight line

Example:-

```
import cv2

import numpy as np



# Read and resize the image

old_img = cv2.imread(r"C:\Users\Vikas\Desktop\OpenCV\Images\img7.jpg")

old_img = cv2.resize(old_img, (500, 500))



# Create overlay for transparency effect

overlay = old_img.copy()



# Draw a filled rectangle (cool semi-transparent box)

cv2.rectangle(overlay, pt1=(70, 100), pt2=(300, 400), color=(0, 255, 0), thickness=-1)
```

```
alpha = 0.3  # Transparency factor

new_img = cv2.addWeighted(overlay, alpha, old_img, 1 - alpha, 0)



# Draw border rectangle over it

cv2.rectangle(new_img, pt1=(70, 100), pt2=(300, 400), color=(0, 255, 0), thickness=3)



# Add cool glowing-style text

cv2.putText(new_img,

        text="OpenCV is Cool!",

        org=(40, 80),

        fontFace=cv2.FONT_HERSHEY_TRIPLEX,

        fontScale=1.3,

        color=(255, 255, 255),  # white for main text

        thickness=2,

        lineType=cv2.LINE_AA)



# Shadow effect (black offset text behind)

cv2.putText(new_img,

        text="OpenCV is Cool!",

        org=(42, 82),  # Slightly offset for shadow

        fontFace=cv2.FONT_HERSHEY_TRIPLEX,

        fontScale=1.3,

        color=(0, 0, 0),  # black shadow

        thickness=3,
        lineType=cv2.LINE_AA)



# Display the final image

cv2.imshow("Styled Image", new_img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```



📘 How to Draw Circles and Ellipses Using OpenCV

OpenCV provides built-in functions like cv2.circle() and cv2.ellipse() to draw shapes on images.

✅ 1. Drawing a Circle

🔧 Syntax:

```
cv2.circle(img, center, radius, color, thickness)
```

Parameter	Description
img	Image on which to draw
center	Center point of the circle (x, y)
radius	Radius of the circle
color	Color in BGR format (e.g., (0, 255, 0) = green)
thickness	Thickness of the outline (-1 to fill the circle)

🧪 Example:

```
import cv2

img = cv2.imread('image.jpg')
img = cv2.resize(img, (400, 400))
```

```
cv2.circle(img, center=(200, 200), radius=100, color=(0, 0, 255), thickness=3)  # Red circle


cv2.imshow("Circle", img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

✅ 2. Drawing an Ellipse

🔧 Syntax:

```
cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color, thickness)
```

Parameter	Description
img	Image on which to draw
center	Center of the ellipse (x, y)
axes	Length of major and minor axis (width, height)
angle	Rotation of the ellipse in degrees
startAngle	Starting angle of the arc

endAngle	Ending angle of the arc
color	Color in BGR format
thickness	Thickness of the border (-1 to fill)

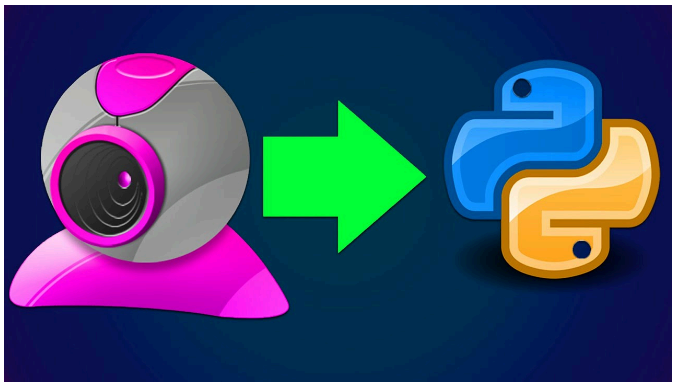
🧪 Example:

```
cv2.ellipse(img,

        center=(200, 200),

        axes=(100, 50),

        angle=0,

        startAngle=0,

        endAngle=360,

        color=(255, 0, 0),

        thickness=2)  # Blue ellipse
```

📝 Summary Table:

Function	Use
cv2.circle()	Draws a circle
cv2.ellipse()	Draws an ellipse or arc

Webcam & Video Saving



🟢 1. Opening Webcam in OpenCV

We use OpenCV to access and display the webcam feed in real-time.

✅ Code Explanation:

```
import cv2


cap = cv2.VideoCapture(0)  # 0 = default webcam


while True:

  ret, frame = cap.read()  # ret: True/False, frame: actual image


  if not ret:

    print("Could not read frame")

    break


  cv2.imshow("Webcam feed", frame)  # Show the webcam frame


  if cv2.waitKey(1) & 0xFF == ord("q"):  # Press 'q' to quit

    print("Quitting....")
    break


cap.release()          # Release camera

cv2.destroyAllWindows()  # Close all OpenCV windows
```

⚙️ Key Functions:

- cv2.VideoCapture(0): Opens default webcam.
- read(): Captures the current frame.
- imshow(): Displays the frame.
- waitKey(1): Waits for 1 millisecond for a key event.
- release(): Releases the camera resource.

🎥 2. Saving Webcam Video

You can record and save video from the webcam using VideoWriter.

✅ Code Explanation:

```
import cv2


camera = cv2.VideoCapture(0)  # Start webcam


# Get frame width and height

frame_height = int(camera.get(cv2.CAP_PROP_FRAME_HEIGHT))

frame_width = int(camera.get(cv2.CAP_PROP_FRAME_WIDTH))
```

```
# Define the codec and create VideoWriter object
codec = cv2.VideoWriter_fourcc(*'XVID')
recorder = cv2.VideoWriter("My_Video.avi", codec, 20, (frame_width, frame_height))

while True:
    success, image = camera.read()

    if not success:
        break

    recorder.write(image)  # Write the frame to the video file
    cv2.imshow("Recording live", image)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release everything
camera.release()
recorder.release()
cv2.destroyAllWindows()
```

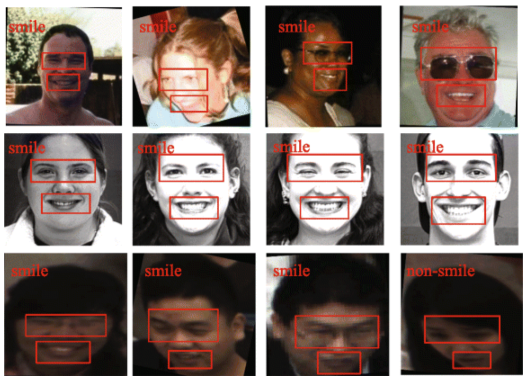
⚙️ Key Functions:

- cv2.CAP_PROP_FRAME_HEIGHT / WIDTH: Get video frame dimensions.
- cv2.VideoWriter_fourcc(*'XVID'): Codec for AVI format.
- cv2.VideoWriter(): Creates a video file writer.
- write(): Writes frames into the video file.

📁 Output:

- Live webcam shown in a window.
- Recorded video saved as **My_Video.avi** in the current directory.

Face, Eyes & Smile Detection – OpenCV Code



🔹 **Objective:**

To detect human **faces**, **eyes**, and **smiles** in real-time using a webcam and Haar Cascade classifiers provided by OpenCV.

🔹 **Key Components Used:**

- **OpenCV** library (cv2)
- **Haar Cascade XML files**:
    - haarcascade_frontalface_default.xml – for face detection
    - haarcascade_eye.xml – for eye detection
    - haarcascade_smile.xml – for smile detection
- **Webcam** for capturing live video feed

🔹 **Process Overview:**

1. **Import necessary libraries** (like OpenCV).
2. **Load Haar cascades** for face, eye, and smile detection.
3. **Initialize the webcam** to capture real-time video.
4. **Read frames** continuously from the webcam.
5. **Convert each frame to grayscale** (Haar cascades work better on grayscale images).
6. **Detect faces** in the grayscale image.
7. For each face:
    - Draw a **rectangle** around the detected face.
    - Define **Region of Interest (ROI)** to detect eyes and smiles **within** the face.
    - **Detect eyes** in the ROI.
    - **Detect smiles** in the ROI.
    - If eyes or smile are detected, display labels like "**Eyes Detected**" or "**Smile Detected**".
8. **Show the final frame** with annotations.
9. **Exit the loop** and close all windows when the user presses the **'q' key**.

🔹 **Detection Parameters:**

- **scaleFactor** – How much the image size is reduced at each image scale.
- **minNeighbors** – How many neighbors each candidate rectangle should have to retain it.

🔹 **Applications:**

- Real-time facial feature detection
- Emotion recognition (based on smile)

- Human-computer interaction systems
- Security and surveillance systems

For Example

Github link

```
# Import OpenCV library

import cv2



# ==========================

# Load Haar Cascade Classifiers

# ==========================

# Load the pre-trained Haar cascade files for detecting faces, eyes, and smiles.

face_cascade =
cv2.CascadeClassifier(r"C:\Users\Vikas\Desktop\Projects\OpenCV\Coding\Face and Object
Detection\haarcascade_frontalface_default.xml")

eye_cascade =
cv2.CascadeClassifier(r"C:\Users\Vikas\Desktop\Projects\OpenCV\Coding\Face and Object
Detection\haarcascade_eye.xml")

smile_cascade =
cv2.CascadeClassifier(r"C:\Users\Vikas\Desktop\Projects\OpenCV\Coding\Face and Object
Detection\haarcascade_smile.xml")



# ==========================

# Initialize Webcam

# ==========================

cap = cv2.VideoCapture(0)  # 0 is the default camera



# Check if the camera opened successfully

if not cap.isOpened():
    print("Error: Could not open webcam.")

    exit()



# ==========================

# Real-time Video Processing Loop

# ==========================

while True:

    ret, frame = cap.read()  # Read frame from the webcam

    if not ret:

        print("Error: Failed to capture frame.")

        break



    # Optional: Resize the frame for performance

    frame = cv2.resize(frame, (640, 480))



    # Convert the frame to grayscale (Haar cascades work on grayscale images)

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)



    # ==========================

    # Face Detection

    # ==========================

    faces = face_cascade.detectMultiScale(

        gray,

        scaleFactor=1.1,
        minNeighbors=5

    )


    # Loop through all detected faces

    for (x, y, w, h) in faces:

        # Draw a green rectangle around each face

        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)


        # Define region of interest (ROI) within the face

        roi_gray = gray[y:y + h, x:x + w]

        roi_color = frame[y:y + h, x:x + w]


        # ==========================

        # Eye Detection within Face ROI

        # ==========================

        eyes = eye_cascade.detectMultiScale(

            roi_gray,

            scaleFactor=1.1,

            minNeighbors=10

        )

        if len(eyes) > 0:

            # If eyes are detected, display label

            cv2.putText(frame, "Eyes Detected", (x, y - 30),

                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 0), 2)
        # ==========================
        # Smile Detection within Face ROI
        # ==========================
        smiles = smile_cascade.detectMultiScale(
            roi_gray,
            scaleFactor=1.7,
            minNeighbors=20
        )
        if len(smiles) > 0:
            # If smile is detected, display label
            cv2.putText(frame, "Smile Detected", (x, y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)

    # ==========================
    # Display the Processed Frame
    # ==========================
    cv2.imshow("Face, Eyes & Smile Detection", frame)

    # Break the loop when 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# ==========================
# Release Resources

# ===========================

cap.release()

cv2.destroyAllWindows()
```