CS-37 Machine Learning with **Python**

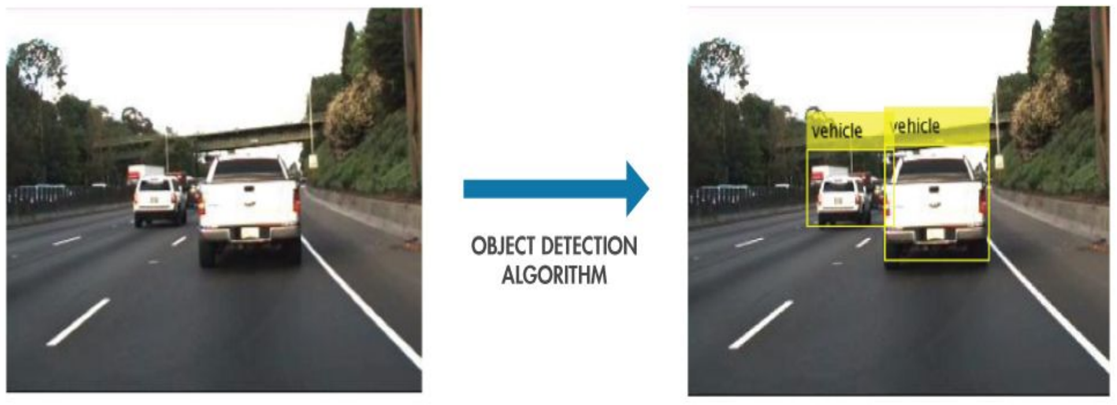
Unit– 05 Computer Vision with OpenCV

- Object Detection:
- Detecting and tracking objects using Haar cascades from images and videos
- Detecting face, eyes, mouth, nose, pupils

Object Detection

What Is Object Detection?

- ✓ Object detection is a computer vision technique for locating instances of objects in images or videos.
- ✓ Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results.
- ✓ When humans look at images or videos, we can recognize and locate objects of interest within a matter of moments.
- ✓ The goal of object detection is to replicate this intelligence using a computer.
- ✓ Object detection, a key technology used in advanced driver assistance systems (ADAS), enables cars to detect driving lanes and pedestrians to improve road safety.
- ✓ Object detection is also an essential component in applications such as visual inspection, robotics, medical imaging, video surveillance, and content-based image retrieval.



Using object detection to identify and locate vehicles.

How Object Detection Works

1. Object Detection Using Deep Learning

✓ You can use a variety of techniques to perform object detection. Popular deep learning–based approaches using convolutional neural networks (CNNs), such as YOLO, SSD, or R-CNN, automatically learn to detect objects within images.

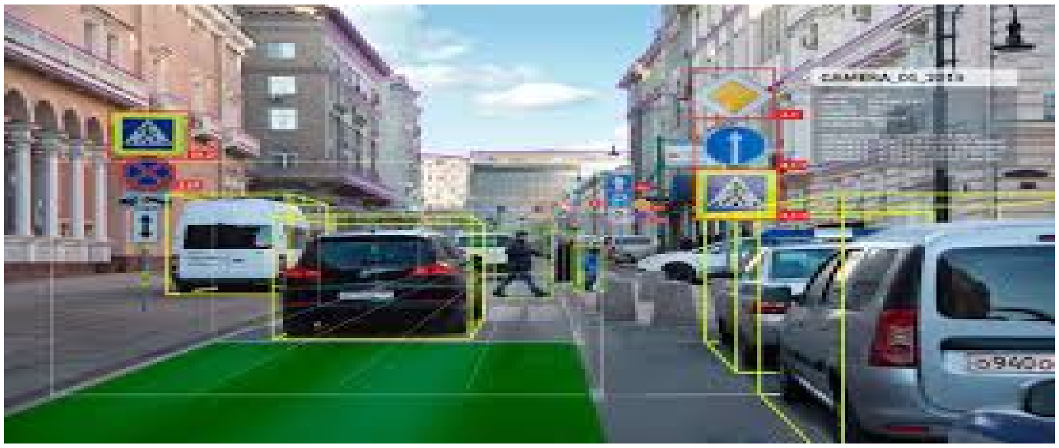
✓ You can choose from two key approaches to get started with object detection using deep learning:

- **Use pretrained object detectors**
- **Create and train a custom object detector**

2. Object Detection Using Machine Learning

✓ Machine learning techniques are also commonly used for object detection, and they offer different approaches than deep learning. Common machine learning techniques include:

- Aggregate channel features (ACFs)
- Support vector machine (SVM) classification using histograms of oriented gradient (HOG) features
- The Viola-Jones algorithm for human face or upper body detection



- ✓ As with deep learning–based approaches, you can choose to start with a pretrained object detector or create a custom object detector to suit your application.
- ✓ You will need to manually select the identifying features for an object when using machine learning, compared with automatic feature selection in a deep learning–based workflow.

Machine Learning vs. Deep Learning for Object Detection

- ✓ The best approach for object detection depends on your application and the problem you're trying to solve.
- ✓ When choosing between machine learning and deep learning, consider whether you have a powerful GPU and lots of labeled training images.
- ✓ If you don't have both, a machine learning approach might be the better choice.
- ✓ Deep learning techniques tend to work better when you have more images, and GPUs decrease the time needed to train the model.

Haar Cascades for Object Detection

- ✓ Object Detection is a computer technology related to computer vision, image processing and deep learning that deals with detecting instances of objects in images and videos. We will do object detection in this article using something known as haar cascades.

What are Haar Cascades?

- ✓ Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones in their paper Rapid Object Detection using a Boosted Cascade of Simple Features .
- ✓ Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

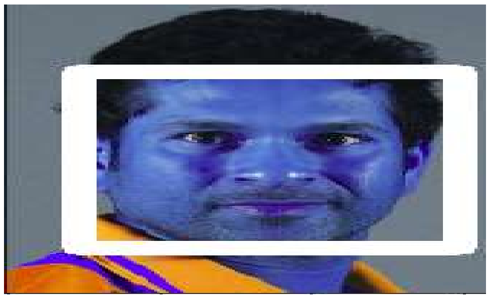
- **Positive images –** These images contain the images which we want our classifier to identify.
- **Negative Images –** Images of everything else, which do not contain the object we want to detect.

Requirements:

- ✓ Make sure you have python, Matplotlib and OpenCV installed on your pc (all the latest versions).
- ✓ The haar cascade files can be downloaded from the OpenCV Github repository.

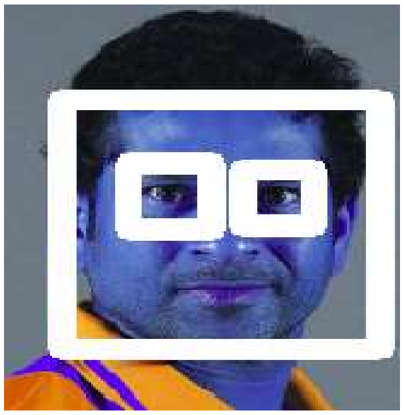
Implementation(For knowledge)

```
# Importing all required packages
import cv2
import numpy as np
import matplotlib.pyplot as plt % matplotlib inline



# Read in the cascade classifiers for face and eyes
face_cascade = cv2.CascadeClassifier('../DATA /
haarcascades / haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('../DATA /
haarcascades / haarcascade_eye.xml')




# create a function to detect face
def adjusted_detect_face(img):

    face_img = img.copy()

    face_rect = face_cascade.detectMultiScale(face_img,
                                              scaleFactor = 1.2,
                                              minNeighbors = 5)
    for (x, y, w, h) in face_rect:
        cv2.rectangle(face_img, (x, y),
                      (x + w, y + h), (255, 255, 255), 10)\

    return face_img

# create a function to detect eyes
def detect_eyes(img):

    eye_img = img.copy()
    eye_rect = eye_cascade.detectMultiScale(eye_img,
                                            scaleFactor = 1.2,
                                            minNeighbors = 5)
    for (x, y, w, h) in eye_rect:
        cv2.rectangle(eye_img, (x, y),
                      (x + w, y + h), (255, 255, 255), 10)
    return eye_img

# Reading in the image and creating copies
img = cv2.imread('../sachin.jpg')
img_copy1 = img.copy()
img_copy2 = img.copy()
img_copy3 = img.copy()

# Detecting the face
face = adjusted_detect_face(img_copy)
plt.imshow(face)
# Saving the image
cv2.imwrite('face.jpg', face)
```



Code : Detecting face and eyes

```
eyes_face = adjusted_detect_face(img_copy3)
eyes_face = detect_eyes(eyes_face)
plt.imshow(eyes_face)
cv2.imwrite('face+eyes.jpg', eyes_face)
```



- ✓ Haar Cascades can be used to detect any types of objects as long as you have the appropriate XML file for it. You can even create your own XML files from scratch to detect whatever type of object you want.

Detecting face, eyes, mouth, nose, pupils with OpenCV

What is OpenCV?

Open source Computer Vision is basically a library, used for real-time Computer Vision. It supports a wide range of programming languages like Python, C, C++, Java etc and also supports different platforms including Windows, Linux, MacOS. By using it, one can process images and videos to identify objects, faces or even handwriting of a human. When it integrated with various libraries, such as Numpy, Python is capable of processing the OpenCV array structure for analysis. To identify image pattern and its various features we use vector space and perform mathematical operations on these features.

How to code:(For Knowledge)

First, we have to install OpenCV library which can be easily installed by using the pip command: **pip install opencv-python** and then we will install the NumPy library by using the pip command: **pip install numpy**.

· To get started first we have to import the **NumPy** library named as **np** and **OpenCV** library named as **cv2**.

```
import numpy as np
import cv2

#load the xml files for face, eye and mouth detection into the
program
face_cascade=cv2.CascadeClassifier('haarcascade_frontalface_d
efault.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

```
mouth_cascade =
cv2.CascadeClassifier('haarcascade_mcs_mouth.xml')

#read the image for furthur editing
image = cv2.imread('big bang final.jpeg')

#show the original image
cv2.imshow('Original image', image)
cv2.waitKey(100)

#convert the RBG image to gray scale image
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#iteration through the faces array and draw a rectangle
for(x, y, w, h) in faces:
cv2.rectangle(image, (x, y), (x+w, y+h), (0, 0, 255), 2)
roi_gray = gray_image[y:y+h, x:x+w]
roi_color = image[y:y+h, x:x+w]

#identify the face using haar-based classifiers
faces = face_cascade.detectMultiScale(image, 1.4, 4)
```

```
#identify the eyes and mouth using haar-based classifiers
eyes = eye_cascade.detectMultiScale(gray_image, 1.3, 5)
mouth = mouth_cascade.detectMultiScale(gray_image, 1.5, 11)

#iteration through the eyes and mouth array and draw a rectangl
for(ex, ey, ew, eh) in eyes:
cv2.rectangle(image,(ex, ey), (ex+ew, ey+eh), (0, 255, 0),2)

for(mx, my, mw, mh) in mouth:
cv2.rectangle(image, (mx, my), (mx+mw, my+mh), (255, 0, 0), 2)

#show the final image after detection
cv2.imshow('face, eyes and mouth detected image', image)
cv2.waitKey()

#show a successful message to the user
print("Face, eye and mouth detection is successful")
```

Original Image:

The output after face, eyes and mouth detection:

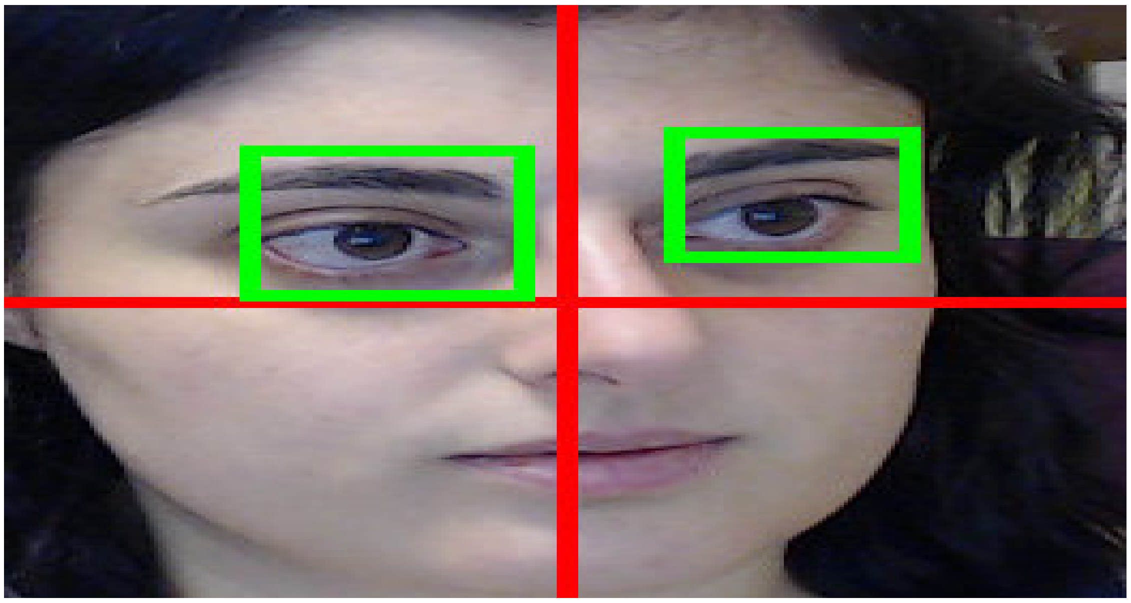
What is Face Detection?

Face detection is an advanced technology that focuses on identifying and locating human faces within digital images or videos. This innovative technology stands as the preliminary step in complex processes like face recognition, face analysis, and identity verification.

Utilizing artificial intelligence (AI) and machine learning, face detection algorithms discern faces from other elements in an image by analyzing key facial features, such as the distance between the eyes, the shape of the jawline, and the position of the nose.

What is Eyes Detection?

"Eyes detection" in machine learning refers to the process of using computer vision algorithms to identify and locate a person's eyes within an image or video frame, essentially allowing a machine to "see" where someone is looking by pinpointing the position of their eyes; this is often achieved using techniques like facial feature extraction and deep learning models, and is commonly applied in applications like driver fatigue monitoring, human-computer interaction, and gaze tracking systems.



What is Mouth, Nose, Pupils Detection?

"Mouth, nose, pupils detection" in machine learning refers to the process of using computer vision algorithms to identify and locate the precise positions of a person's mouth, nose, and pupils within an image, essentially pinpointing these facial features using machine learning techniques, often implemented through deep learning models like convolutional neural networks (CNNs).