**Object\_Detection**:

- `from imutils.video import FPS`

**Imports the FPS class from the imutils.video module. This class will be used to track the frames per second of the video stream.**

- `import numpy as np`

**Imports the NumPy library and gives it the alias "np". NumPy is a library for working with arrays and mathematical operations.**

- `import imutils`

**Imports the imutils library, which provides a set of convenience functions to make basic image processing functions easier.**

- `import cv2`

**Imports the OpenCV library, which provides computer vision and image processing functions.**

- `use\_gpu = True`

**A boolean variable that determines whether to use a GPU for processing if available.**

- `live\_video = False`

**A boolean variable that determines whether to use a live video stream or a pre-recorded video.**

- `confidence\_level = 0.5`

**A float variable that sets the minimum confidence level for object detection.**

- `fps = FPS().start()`

**Creates an instance of the FPS class and starts tracking the frames per second of the video stream.**

- `ret = True`

**A boolean variable that indicates whether there are still frames in the video stream to process.**

- `CLASSES`

**A list of string labels for the objects that can be detected using the MobileNetSSD model.**

- `COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))`

**Creates an array of random colors for each class label. The array has a shape of (num\_classes, 3), where each row represents a color in BGR format.**

- `net = cv2.dnn.readNetFromCaffe('ssd\_files/MobileNetSSD\_deploy.prototxt', 'ssd\_files/MobileNetSSD\_deploy.caffemodel')`

**Reads the pre-trained MobileNetSSD model from the Caffe framework.**

- `if use\_gpu:`

**Checks if the `use\_gpu` variable is set to True.**

- `print("[INFO] setting preferable backend and target to CUDA...")`

**Prints a message indicating that the program is setting the preferable backend and target to CUDA.**

- `net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_CUDA)`

**Sets the backend for the neural network to CUDA.**

- `net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CUDA)`

**Sets the target device for the neural network to CUDA.**

- `print("[INFO] accessing video stream...")`

**Prints a message indicating that the program is accessing the video stream.**

- `if live\_video:`

**Checks if the `live\_video` variable is set to True.**

- `vs = cv2.VideoCapture(0)`

**Creates a VideoCapture object that captures video from the default camera device (index 0).**

- `else:`

**If `live\_video` is False, the program will read a pre-recorded video.**

- `vs = cv2.VideoCapture('test2.mp4')`

**Creates a VideoCapture object that reads video frames from the file 'test2.mp4'. The program can be configured to read from other files by changing the filename.**

**The code uses the OpenCV library and the MobileNet-SSD deep neural network to perform real-time object detection on a video stream.**

- `while ret:`

**a loop that iterates until the video stream ends or the user presses the "ESC" key.**

- `ret, frame = vs.read()`

**reads the next frame of the video stream and stores it in the `frame` variable. The variable `ret` is a boolean indicating whether the frame was successfully read or not.**

- `if ret:`

**checks whether the frame was successfully read from the video stream.**

- `frame = imutils.resize(frame, width=400)`

**resizes the frame to a width of 400 pixels using the `imutils` library.**

- `(h, w) = frame.shape[:2]`

**retrieves the height and width of the frame.**

- `blob = cv2.dnn.blobFromImage(frame, 0.007843, (300, 300), 127.5)`

**creates a 4-dimensional blob from the resized frame. This blob is used as input to the deep neural network.**

- `net.setInput(blob)`

**sets the input blob as the input to the neural network.**

- `detections = net.forward()`

**runs the neural network to obtain the detections for the input frame.**

- `for i in np.arange(0, detections.shape[2]):`

**loops over all the detections returned by the neural network.**

- `confidence = detections[0, 0, i, 2]`

**retrieves the confidence score for the detection.**

- `if confidence > confidence\_level:`

**checks if the confidence score is higher than a threshold value specified by the `confidence\_level` variable.**

- `idx = int(detections[0, 0, i, 1])`

**retrieves the index of the detected object class.**

- `box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])`

**retrieves the bounding box coordinates for the detection and scales them to the original frame size.**

- `(startX, startY, endX, endY) = box.astype("int")`

**converts the bounding box coordinates to integers.**

- `label = "{}: {:.2f}%".format(CLASSES[idx], confidence \* 100)`

**creates a label for the detected object, including the object class and the confidence score.**

- `cv2.rectangle(frame, (startX, startY), (endX, endY), COLORS[idx], 2)`

**draws a rectangle around the detected object on the frame.**

- `y = startY - 15 if startY - 15 > 15 else startY + 15`

**computes the y-coordinate of the label text.**

- `cv2.putText(frame, label, (startX, y), cv2.FONT\_HERSHEY\_DUPLEX, 0.5, COLORS[idx], 1)`

**adds the label text to the frame.**

- `frame = imutils.resize(frame,height=400)`

**resizes the frame to a height of 400 pixels using the `imutils` library.**

- `cv2.imshow('Live detection',frame)`

**displays the frame with the detected objects.**

- `if cv2.waitKey(1)==27:`

**waits for a key press and checks if the key pressed was the "ESC" key (key code 27).**

- `fps.update()`

**updates the FPS counter for the current iteration.**

- `fps.stop()`

**stops the FPS counter.**

- `print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))`

**prints the elapsed time since the FPS counter was started.**

- `print("[INFO] approx. FPS: