ASSIGNMENT 6

Name : Mayank Yadav
Reg No.: 19BCY10146

```
#import the necessary packages

from imutils.video import VideoStream from imutils.video import FPS

import numpy as np

import argparse

import imutils

import time

import cv2

construct the argument parse and parse the arguments

apargparse.ArgumentParser()

ap.add_argument("-p". "--prototxt", required-True,help="path to Caffe 'deploy' prototxt file")

ap.add_argument("-m", -model", required=True,help-"path to Caffe pre-trained model")
ap.add_argument("c", "--confidence", type-float, default-0.2.help-"minimum probability to filter
weak detections")

args = vars(ap.parse_args())

# initialize the list of class labels MobileNet SSD was trained to

#detect, then generate a set of bounding box colors for each class CLASSES ["background",
"aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair". "cow", "diningtable",
"dog", "horse", "motorbike", "person". "pottedplant", "sheep","sofa", "train". "tvmonitor"]

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

#load our serialized model from disk
```

```
print("[INFO] loading model...") net - cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
#initialize the video stream, allow the camera sensor to warm up,

# and initialize the FPS counter

print("[INFO] starting video stream...")

vs- VideoStream(src-0).start() time.sleep(2.0)

fps FPS().start()

#loop over the frames from the video stream

while True:

#grab the frame from the threaded video stream and resize it

to have maximum width of 400 pixels

frame=vs.read()

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

#grab the frame dimensions and convert it to a blob

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

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

# predictions

net setInput(blob)
detections-net.forward()

#loop over the detections

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

# extract the confidence (i.e., probability) associated with

# the prediction

confidence detections[0, 0, 1, 2]
```

```
#filter out weak detections by ensuring the confidence is

#greater than the minimum confidence

if confidence> args["confidence"]: #extract the index of the class label from the

#'detections, then compute the (x, y)-coordinates of

#the bounding box for the object

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

box detections[0, 0, i, 3:7] np.array([w, h, w, h])

(startX, start Y, endX, endY) = box.astype("int") #draw the prediction on the frame

label": (:.2f%.format(CLASSES[idx),confidence 100)

cv2.rectangle(frame, (startX, startY), (endX, endY),COLORS[idx], 2) y = start Y - 15 if start Y-15
> 15 else startY+ 15 cv2.putText(frame, label, (startX, y),cv2.FONT HERSHEY SIMPLEX, 0.5,

COLORS[idx), 2)

#show the output frame

cv2.imshow("Frame", frame)

key-cv2.waitKey(1) & 0xFF

if the 'q' key was pressed, break from the loop

if key == ord("q"):
break

# update the FPS counter
# stop the timer and display FPS information

fps.stop()

# do a bit of cleanup

cv2.destroyAllWindows()
```

```
vs.stop()

fps.update()
```