# Person Counting Application

### SSD (Single Shot MultiBox Detector):

Single Shot MultiBox Detector (SSD) is a deep learning-based object detection algorithm that efficiently detects objects, including persons, in images. It operates by dividing the image into a grid of cells and predicting multiple bounding boxes and class probabilities for each cell. SSD uses a single convolutional neural network to predict the presence, location, and class of objects, making it faster and more accurate than previous methods. It employs a combination of feature maps at different scales to capture objects of various sizes. SSD is trained using a combination of localization and classification losses to optimize object detection performance. It is widely used for real-time object detection tasks due to its speed and effectiveness.

### Centroid Tracker:

The Centroid Tracker algorithm works by maintaining a list of the most recent centroids for each tracked object and computing the Euclidean distance between the centroids of the new detections and the previous ones. Centroid Tracker is a simple object tracking algorithm that uses the centroid of objects to track their motion. It is particularly useful for tracking objects in real-time video streams. The algorithm works by associating centroids in subsequent frames of the video stream, using a distance-based matching algorithm. In the context of object detection, the Centroid Tracker is typically used to track the motion of objects that have been detected by a detection algorithm such as MobileNet SSD. Once objects are detected in a video frame, the Centroid Tracker uses the coordinates of the bounding boxes around the objects to calculate their centroids. It then associates these centroids with objects detected in subsequent frames, based on their distance from the centroids in the previous frame. One of the advantages of the Centroid Tracker algorithm is that it is computationally efficient and can be implemented in real-time. It is also able to handle cases where objects move in and out of the frame, and where objects are partially occluded by other objects in the frame. To use the Centroid Tracker, we first need to initialize it by creating an instance of the class and specifying the maximum number of frames to keep track of. Next, we iterate over each frame of the video stream, perform object detection on each frame using MobileNet SSD, and pass the resulting bounding box coordinates to the Centroid Tracker.

**Step 1: Install required libraries**

**OpenCV (cv2):** OpenCV is a popular library for computer vision tasks, including image and video processing, object detection, and tracking.

```
$ pip3 install opencv-python
```

**NumPy (np):** NumPy is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

```
$ pip3 install numpy
```

**Imutils:** Imutils is a set of convenience functions to make basic image processing tasks (such as resizing, rotating, and displaying images) easier with OpenCV.

```
$pip install imutils
```

**CentroidTracker:** This appears to be a custom module or class for object tracking using centroid-based tracking. It's not a standard library and might be provided by the user or another source. Ensure that you have access to this module for the script to work correctly.

## Step 2: To create custom module of Centroid Tracker

- Create centroidtracker.py file
- Add the code which is given below

```
# import the necessary packages
from scipy.spatial import distance as dist
from collections import OrderedDict
import numpy as np


class CentroidTracker:
    def __init__(self, maxDisappeared=30, maxDistance=50):
 # initialize the next unique object ID along with two ordered dictionaries used to keep track of mapping a given object .ID to its centroid and number of consecutive frames it has been marked as "disappeared", respectively
        self.nextObjectID = 1
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()
        self.bbox = OrderedDict()  # CHANGE
 # store the number of maximum consecutive frames a given object is allowed to be marked as "disappeared" until we need to deregister the object from tracking
       self.maxDisappeared = maxDisappeared
# store the maximum distance between centroids to associate an object -- if the distance is larger than this maximum distance we'll start to mark the object as "disappeared"
        self.maxDistance = maxDistance


 def register(self, centroid, inputRect):
 # when registering an object we use the next available object ID to store the centroid
        self.objects[self.nextObjectID] = centroid
        self.bbox[self.nextObjectID] = inputRect  # CHANGE
        self.disappeared[self.nextObjectID] = 1
```

```
        self.nextObjectID += 1


    def deregister(self, objectID):
# to deregister an object ID we delete the object ID from both of our respective dictionaries
        del self.objects[objectID]
        del self.disappeared[objectID]
        del self.bbox[objectID] # CHANGE


    def update(self, rects):
# check to see if the list of input bounding box rectangles is empty
        if len(rects) == 0:
# loop over any existing tracked objects and mark them as disappeared
            for objectID in list(self.disappeared.keys()):
                self.disappeared[objectID] += 1
# if we have reached a maximum number of consecutive frames where a given object has been marked as missing, deregister it
                if self.disappeared[objectID] > 2:
                    self.deregister(objectID)
# return early as there are no centroids or tracking info to update return self.objects
            return self.bbox
# initialize an array of input centroids for the current frame
        inputCentroids = np.zeros((len(rects), 2), dtype="int")
        inputRects = []
# loop over the bounding box rectangles
        for (i, (startX, startY, endX, endY)) in enumerate(rects):
# use the bounding box coordinates to derive the centroid
            cX = int((startX + endX) / 2.0)
            cY = int((startY + endY) / 2.0)
```

**inputCentroids[i] = (cX, cY)**

**inputRects.append(rects[i])** # CHANGE

# if we are currently not tracking any objects take the input centroids and register each of them

**if len(self.objects) == 0:**

**for i in range(0, len(inputCentroids)):**

**self.register(inputCentroids[i], inputRects[i])** # CHANGE

# otherwise, are are currently tracking objects so we need to try to match the input centroids to existing object centroids

**else:**

# grab the set of object IDs and corresponding centroids

**objectIDs = list(self.objects.keys())**

**objectCentroids = list(self.objects.values())**

# compute the distance between each pair of object centroids and input centroids, respectively – our goal will be to match an input centroid to an existing object centroid

**D = dist.cdist(np.array(objectCentroids), inputCentroids)**

# in order to perform this matching we must (1) find the smallest value in each row and then (2) sort the row indexes based on their minimum values so that the row with the smallest value as at the *front* of the index list

**rows = D.min(axis=1).argsort()**

# next, we perform a similar process on the columns by finding the smallest value in each column and then sorting using the previously computed row index list

**cols = D.argmin(axis=1)[rows]**

# in order to determine if we need to update, register, or deregister an object we need to keep track of which of the rows and column indexes we have already examined

**usedRows = set()**

**usedCols = set()**

# loop over the combination of the (row, column) index tuples

**for (row, col) in zip(rows, cols):**

# if we have already examined either the row or column value before, ignore it

**if row in usedRows or col in usedCols:**

**continue**

# if the distance between centroids is greater than the maximum distance, do not associate the two centroids to the same object

**if D[row, col] > self.maxDistance:**

**continue**

# otherwise, grab the object ID for the current row, set its new centroid, and reset the disappeared counter

**objectID = objectIDs[row]**

**self.objects[objectID] = inputCentroids[col]**

**self.bbox[objectID] = inputRects[col]** # CHANGE

**self.disappeared[objectID] = 0**

# indicate that we have examined each of the row and column indexes, respectively

**usedRows.add(row)**

**usedCols.add(col)**

# compute both the row and column index we have NOT yet examined

**unusedRows = set(range(0, D.shape[0])).difference(usedRows)**

**unusedCols = set(range(0, D.shape[1])).difference(usedCols)**

# in the event that the number of object centroids is equal or greater than the number of input centroids we need to check and see if some of these objects have potentially disappeared

**if D.shape[0] >= D.shape[1]:**

# loop over the unused row indexes

**for row in unusedRows:**

# grab the object ID for the corresponding row index and increment the disappeared counter

**objectID = objectIDs[row]**

**self.disappeared[objectID] += 1**

# check to see if the number of consecutive frames the object has been marked "disappeared" for warrants deregistering the object

**if self.disappeared[objectID] > self.maxDisappeared:**

**self.deregister(objectID)**

# otherwise, if the number of input centroids is greater than the number of existing object centroids we need to register each new input centroid as a trackable object

**else:**

**for col in unusedCols:**

**self.register(inputCentroids[col], inputRects[col])**

# return the set of trackable object ,return self.objects

**return self.bbox**

## Step 3: Add models in models folder

Download this models from web and paste it into models/objectdetection/

**MobileNetSSD_deploy.prototxt**

**MobileNetSSD_deploy.caffemodel**

## Step 4: To create Person Counting script

- Create personcount.py file
- Add the code which is given below

```
import os
import cv2
import datetime
import imutils
import numpy as np
from centroidtracker import CentroidTracker

protopath = "model/object_detection/MobileNetSSD_deploy.prototxt"
modelpath = "model/object_detection/MobileNetSSD_deploy.caffemodel"
detector = cv2.dnn.readNetFromCaffe(prototxt=protopath, caffeModel=modelpath)

CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
        "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
        "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
        "sofa", "train", "tvmonitor"]

ct = CentroidTracker(60, 60)

def non_max_suppression_fast(boxes, overlapThresh):
   try:
      if len(boxes) == 0:
```

```
    return []

if boxes.dtype.kind == "i":
    boxes = boxes.astype("float")

pick = []

x1 = boxes[:, 0]
y1 = boxes[:, 1]
x2 = boxes[:, 2]
y2 = boxes[:, 3]

area = (x2 - x1 + 1) * (y2 - y1 + 1)
idxs = np.argsort(y2)

while len(idxs) > 0:
    last = len(idxs) - 1
    i = idxs[last]
    pick.append(i)

    xx1 = np.maximum(x1[i], x1[idxs[:last]])
    yy1 = np.maximum(y1[i], y1[idxs[:last]])
    xx2 = np.minimum(x2[i], x2[idxs[:last]])
    yy2 = np.minimum(y2[i], y2[idxs[:last]])

    w = np.maximum(0, xx2 - xx1 + 1)
    h = np.maximum(0, yy2 - yy1 + 1)

    overlap = (w * h) / area[idxs[:last]]

    idxs = np.delete(idxs, np.concatenate(([last],
                                           np.where(overlap > overlapThresh)[0])))
```

```
        return boxes[pick].astype("int")
    except Exception as e:
        print("Exception occurred in non_max_suppression : {}".format(e))



def main():
    # Redirect stderr to /dev/null to suppress GStreamer warning
    os.system("export GST_DEBUG='3'") # Set GST_DEBUG level if needed
    os.system("export GST_DEBUG_NO_COLOR=1") # Optionally, disable color in debug output
    os.system("export GST_DEBUG_FILE=/dev/null") # Redirect stderr to /dev/null

    # Manually specify the GStreamer pipeline for CSI camera inp
    gst_str = "nvarguscamerasrc sensor-id=0 ! video/x-raw(memory:NVMM), width=(int)640, height=(int)480, format=(string)NV12, framerate=30/1 ! nvvidconv ! video/x-raw, format=(string)BGRx ! videoconvert ! video/x-raw, format=(string)BGR ! appsink"

    cap = cv2.VideoCapture(gst_str)

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Flip the frame horizontally
        frame = cv2.flip(frame, 1)

        # Resize the frame
        frame = imutils.resize(frame, width=600)

        (H, W) = frame.shape[:2]

        blob = cv2.dnn.blobFromImage(frame, 0.007843, (W, H), 127.5)
```

```
        detector.setInput(blob)
        person_detections = detector.forward()
        rects = []
        for i in np.arange(0, person_detections.shape[2]):
            confidence = person_detections[0, 0, i, 2]
            if confidence > 0.5:
                idx = int(person_detections[0, 0, i, 1])

                if CLASSES[idx] != "person":
                    continue

                person_box = person_detections[0, 0, i, 3:7] * np.array([W, H, W, H])
                (startX, startY, endX, endY) = person_box.astype("int")
                rects.append(person_box)

        boundingboxes = np.array(rects)
        boundingboxes = boundingboxes.astype("int")
        rects = non_max_suppression_fast(boundingboxes, 0.3)

        objects = ct.update(rects)
        for (objectId, bbox) in objects.items():
            x1, y1, x2, y2 = bbox
            x1 = int(x1)
            y1 = int(y1)
            x2 = int(x2)
            y2 = int(y2)

            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
            text = "Count: {}".format(objectId)
            cv2.putText(frame, text, (x1, y1-5), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 0,
255), 1)
```

```
        cv2.imshow("Application", frame)

        # Add a delay to remove count after a certain time
        key = cv2.waitKey(1)  # Adjust this value as needed
        if key == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()



main()
```

**Step 5: To run the Application**

- Go to the terminal
- Go to the PersonCounting directory
- Run the command given below

$python3 personcount.py

Output:

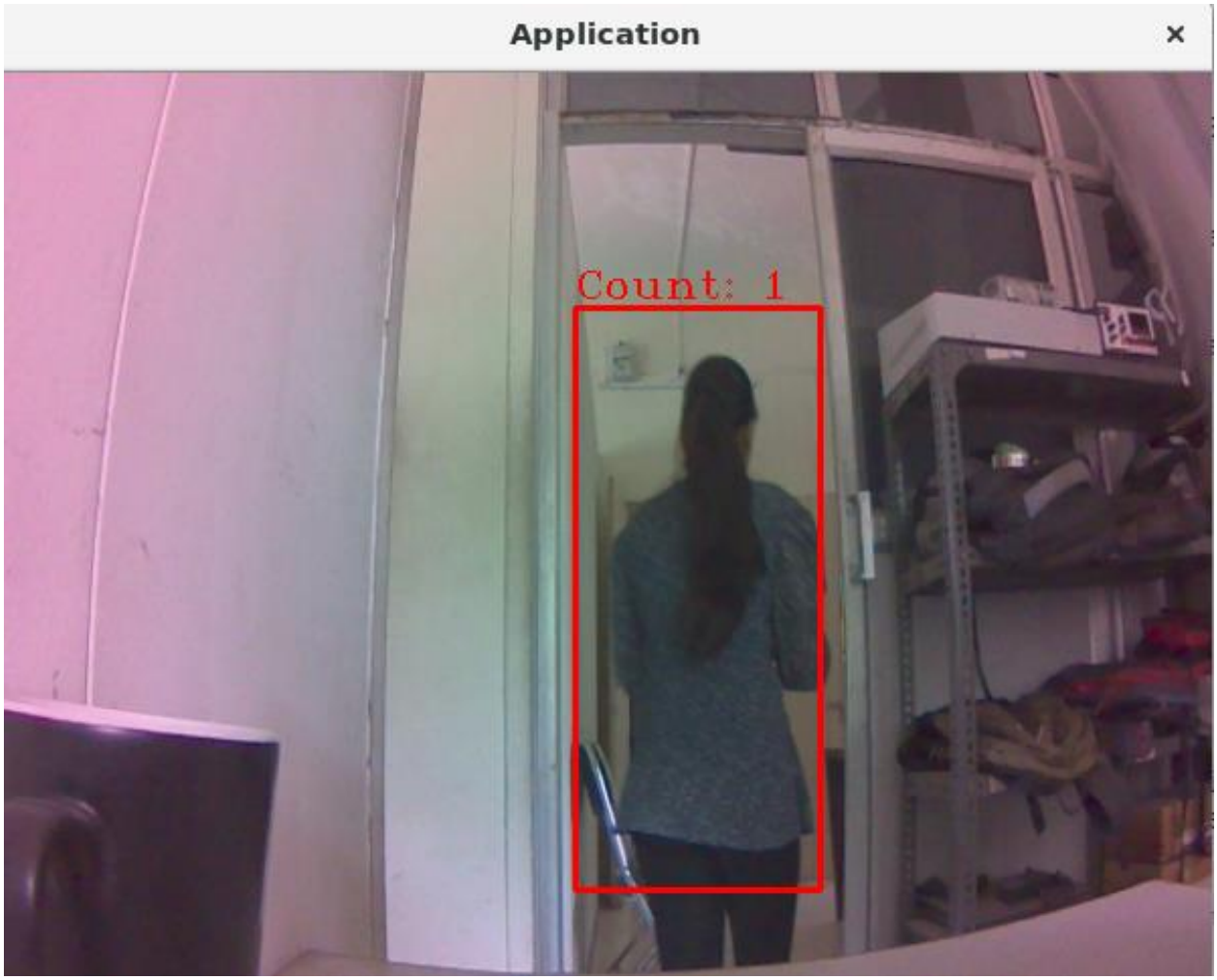


Fig: First person detected

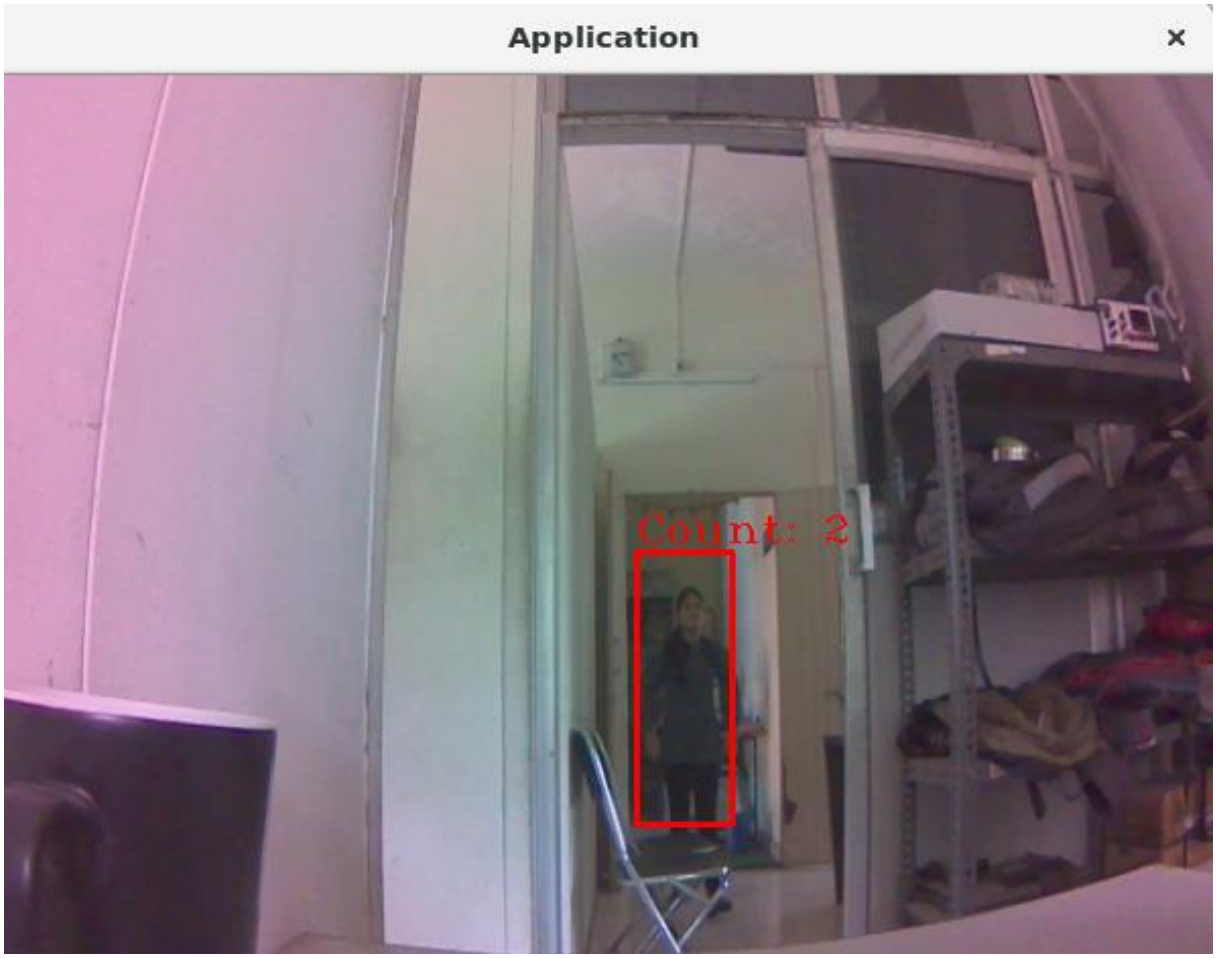


Fig: Second person detected so count is 2