\*\*\*\*\*VIDEO MUST BE PRESENT ALONG WITH OTHER CODE FILES IN SAME FOLDER

**This code integrates OpenCV for video capture and display, NumPy for array operations, PyTorch for YOLO object detection, and user input for counting. Adjust the region of interest (roi\_center\_x, roi\_center\_y, roi\_width, roi\_height) and the bounding box comparison values as needed for your application**.

This code uses the YOLOv5 model from the **ultralytics/yolov5** repository, which you can install using **torch.hub.load**. Adjust the region of interest (**roi\_center\_x**, **roi\_center\_y**, **roi\_width**, **roi\_height**) and the bounding box comparison values as needed for application.

To achieve this using PyTorch for YOLO object detection, you can utilize a pre-trained YOLO model and integrate it with OpenCV for tracking. Here's an example that incorporates user input for counting:

1. Install the required libraries:

**pip install torch torchvision numpy opencv-python**

**2.download weights**

**3.following is the code{Uddit}-**

**import cv2**

**import numpy as np**

**import torch**

**from torchvision import transforms**

**from PIL import Image**

**# Load YOLO model**

**model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)**

**# Initialize variables**

**people\_inside = []**

**people\_count = 0**

**# Capture video from the camera**

**cap = cv2.VideoCapture(0)**

**while True:**

**ret, frame = cap.read()**

**if not ret:**

**break**

**# Convert the frame to a PIL Image**

**frame\_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))**

**# Perform object detection**

**results = model(frame\_pil)**

**# Process the detected objects**

**for detection in results.xyxy[0]:**

**class\_id = int(detection[5])**

**if class\_id == 0: # Class ID 0 is for people**

**# Extract bounding box coordinates**

**center\_x = int((detection[0] + detection[2]) / 2)**

**center\_y = int((detection[1] + detection[3]) / 2)**

**# Check if the center of the bounding box is in the region of interest**

**# You can define your own region of interest (ROI) here**

**# For simplicity, let's assume the ROI is the center of the frame**

**roi\_center\_x = frame.shape[1] // 2**

**roi\_center\_y = frame.shape[0] // 2**

**roi\_width = 100**

**roi\_height = 100**

**if (roi\_center\_x - roi\_width // 2 <= center\_x <= roi\_center\_x + roi\_width // 2) and (**

**roi\_center\_y - roi\_height // 2 <= center\_y <= roi\_center\_y + roi\_height // 2):**

**# Check if this person is already inside the region**

**tracked = False**

**for person\_id, (prev\_x, prev\_y) in people\_inside:**

**if abs(prev\_x - center\_x) < 50 and abs(prev\_y - center\_y) < 50:**

**# Same person, update the count based on user's choice**

**tracked = True**

**count = 2 if input("Count same person multiple times? (y/n): ") == 'y' else 1**

**people\_count += count**

**people\_inside.remove((person\_id, (prev\_x, prev\_y)))**

**break**

**if not tracked:**

**# New person entering the region**

**people\_inside.append((len(people\_inside), (center\_x, center\_y)))**

**people\_count += 1**

**# Display the count**

**cv2.putText(frame, f"People count: {people\_count}", (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)**

**cv2.imshow("Frame", frame)**

**key = cv2.waitKey(1)**

**if key == 27: # Press Esc to exit**

**break**

**cap.release()**

**cv2.destroyAllWindows()**