Final Cars/People YOLO Video File

This code is using YOLO (You Only Look Once) from the Ultralytics library to perform object detection on a video stream. Let's go through each part of the code step by step:

1. Importing necessary libraries:

- `from ultralytics import YOLO`: Importing the YOLO object detection model from the Ultralytics library.

- `import cv2`: Importing the OpenCV library for image and video processing.

- `import cvzone`: Importing the cvzone library, which is used for drawing text on images.

- `import math`: Importing the math module for mathematical calculations.

- `from collections import OrderedDict`: Importing the OrderedDict class from the collections module to maintain the order of class counts.

2. Opening the video file:

- `cap = cv2.VideoCapture("D:\IIT B\Others\Season Of Codes 23\YOLO Cam Detection\pexels_videos_2034115 (1080p).mp4")`: Initializing a VideoCapture

object to open and read the video file. Replace the path with the correct location of your video file.

3. Loading the YOLO model:

- `model = YOLO('..\Yolo-Weights\yolov8s.pt')`: Loading the YOLO model with the specified weights. Replace the path with the correct location of your YOLO weights file.

4. Defining class names:

- `classnames`: A list containing the names of different object classes that the YOLO model can detect.

5. Initializing dictionaries for object tracking and class counting:

- `tracked_objects = {}`: An empty dictionary to store object IDs and their corresponding class names for tracking.

- `class_counts = OrderedDict()`: An empty ordered dictionary to store class names and their corresponding counts.

6. Main loop for object detection and tracking:

- The `while True` loop captures frames from the video stream, performs object detection using the YOLO model, and then tracks and counts the objects of interest.

7. Object detection and tracking:

- The code iterates through the results obtained from YOLO object detection. For each detected object, it extracts the bounding box coordinates, confidence score, and class index.

- If the detected object belongs to the classes 'car', 'person', or 'motorbike', and its confidence is greater than 0.3, it draws the bounding box and class name on the image. It also updates the `tracked_objects` dictionary with the current object's ID and class name.

8. Object counting:

- After detecting and tracking objects, the code updates the `class_counts` dictionary to count the number of objects for each class ('car', 'person', 'motorbike').

9. Displaying the object count:

- The code displays the object counts for each class on the image using cv2.putText() and cv2.imshow().

10. Exiting the loop:

- The loop continues indefinitely until you stop the execution manually (by pressing 'Ctrl+C' or closing the window).

This code essentially detects and tracks objects of the classes 'car', 'person', and 'motorbike' in the video stream while counting the number of objects of each class. It annotates the image with bounding boxes and class names and displays the class-wise object counts on the video window.

Yolo Webcam Detection.py

This code is using YOLO (You Only Look Once) from the Ultralytics library to perform real-time object detection on a video stream from the default camera (Webcam). Let's go through each part of the code step by step:

1. Importing necessary libraries:

- `from ultralytics import YOLO`: Importing the YOLO object detection model from the Ultralytics library.

- `import cv2`: Importing the OpenCV library for image and video processing.

- `import cvzone`: Importing the cvzone library, which is used for drawing text on images.

- `import math`: Importing the math module for mathematical calculations.

2. Setting up the webcam:

- `cap = cv2.VideoCapture(0)`: Initializing a VideoCapture object to access the default camera (Webcam).

- `cap.set(3,640)`: Setting the camera width to 640 pixels.

- `cap.set(4, 480)`: Setting the camera height to 480 pixels.

3. Loading the YOLO model:

- `model = YOLO('..\Yolo-Weights\yolov8n.pt')`: Loading the YOLO model with the specified weights. Replace the path with the correct location of your YOLO weights file.

4. Defining class names:

- `classnames`: A list containing the names of different object classes that the YOLO model can detect.

5. Main loop for real-time object detection:

- The `while True` loop captures frames from the webcam stream, performs object detection using the YOLO model, and then annotates the detected objects with bounding boxes and class names.

6. Object detection and annotation:

- The code iterates through the results obtained from YOLO object detection. For each detected object, it extracts the bounding box coordinates, confidence score, and class index.

- The bounding box is drawn on the image using `cv2.rectangle()` and annotated with the class name and confidence using `cvzone.putTextRect()`.

7. Displaying the annotated frame:

- The code displays the annotated frame in a window using `cv2.imshow()`.

- `cv2.waitKey(1)` is used to refresh the window and keep the video stream continuous.

8. Exiting the loop:

- The loop continues indefinitely until you stop the execution manually (by pressing 'Ctrl+C' or closing the window).

This code essentially performs real-time object detection on the video stream from the webcam. It annotates the image with bounding boxes and class names and displays the annotated video stream in a window. As the camera captures frames, the YOLO model detects objects in the frames and displays the results in real-time.

Yolo Basic.py

This code uses YOLO (You Only Look Once) from the Ultralytics library to perform object detection on a single image. Let's go through each part of the code step by step:

1. Importing necessary libraries:

- `from ultralytics import YOLO`: Importing the YOLO object detection model from the Ultralytics library.

- `import cv2`: Importing the OpenCV library for image processing.

2. Loading the YOLO model:

- `model = YOLO('..\Yolo-Weights\yolov8n.pt')`: Loading the YOLO model with the specified weights. Replace the path with the correct location of your YOLO weights file.

3. Performing object detection on the image:

- `results= model("D:\IIT B\Others\Season Of Codes 23\YOLO Cam Detection\\boy.jpg", show=True)`: Running the YOLO model on the input image "boy.jpg" located at the specified path.

- The `show=True` argument will display the annotated image with bounding boxes and class names.

4. Displaying the annotated image:

- `cv2.waitKey(0)`: Waits for a key event. In this case, the image with bounding boxes and class names will be displayed until any key is pressed.

- Once you press any key, the image window will close, and the program will end.

This code performs object detection on the "boy.jpg" image using YOLO and displays the annotated image with bounding boxes and class names. The image will be shown until you press any key to close the window.