import cv2

from ultralytics import YOLO

def traffic\_sign\_detection(video\_source=0): """

Real-time traffic sign detection using YOLOv8 and OpenCV. video\_source: 0 for webcam or path to video file.

"""

model = YOLO('yolov8n.pt')

cap = cv2.VideoCapture(video\_source)

print(f"Video capture opened: {cap.isOpened()}")

while True:

ret, frame = cap.read()

print(f"Frame read status: {ret}") if not ret:

break

# Perform detection

results = model(frame)

# Render results on frame

annotated\_frame = results[0].plot()

cv2.imshow('Traffic Sign Detection', annotated\_frame) if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

if name == " main ": import argparse

parser = argparse.ArgumentParser(description="YOLOv8-based traffic sign detection")

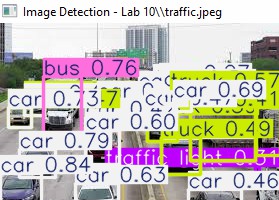
parser.add\_argument('--video', type=str, default='0',

help="Video source: 0 for webcam or path to video

file")

args = parser.parse\_args()

video\_source = int(args.video) if args.video.isdigit() else args.video traffic\_sign\_detection(video\_source)



import cv2

from ultralytics import YOLO

def traffic\_sign\_detection(video\_source=0): """

Real-time traffic sign detection using YOLOv8 and OpenCV. video\_source: 0 for webcam or path to video file.

"""

model = YOLO('yolov8n.pt')

cap = cv2.VideoCapture(video\_source)

print(f"Video capture opened: {cap.isOpened()}")

while True:

ret, frame = cap.read()

print(f"Frame read status: {ret}") if not ret:

break

# Perform detection

results = model(frame)

# Render results on frame

annotated\_frame = results[0].plot()

cv2.imshow('Traffic Sign Detection', annotated\_frame) if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows() import numpy as np

def football\_analysis(video\_source=0): """

Football analysis using YOLOv8 and OpenCV.

Detect players and detect when a goal occurs based on ball position. video\_source: 0 for webcam or path to video file.

"""

model = YOLO('yolov8n.pt')

cap = cv2.VideoCapture(video\_source)

print(f"Video capture opened: {cap.isOpened()}")

goal\_area = None # Define goal area as a rectangle (x1, y1, x2, y2) goal\_detected = False

while True:

ret, frame = cap.read()

print(f"Frame read status: {ret}") if not ret:

break

height)

height, width = frame.shape[:2] if goal\_area is None:

# Define goal area as bottom center rectangle (example)

goal\_area = (int(width\*0.4), int(height\*0.8), int(width\*0.6),

# Perform detection

results = model(frame)

annotated\_frame = frame.copy()

# Extract detections

detections = results[0].boxes.xyxy.cpu().numpy() # bounding boxes classes = results[0].boxes.cls.cpu().numpy() # class indices

needed)

player\_boxes = [] ball\_boxes = []

for i, cls in enumerate(classes):

if int(cls) == 0: # person class

player\_boxes.append(detections[i])

elif int(cls) == 32: # ball class (example class index, adjust as ball\_boxes.append(detections[i])

# Draw player boxes

for box in player\_boxes:

x1, y1, x2, y2 = map(int, box)

cv2.rectangle(annotated\_frame, (x1, y1), (x2, y2), (0,255,0), 2) cv2.putText(annotated\_frame, 'Player', (x1, y1-10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0,255,0), 2)

# Draw ball boxes and check for goal for box in ball\_boxes:

x1, y1, x2, y2 = map(int, box)

cv2.rectangle(annotated\_frame, (x1, y1), (x2, y2), (0,0,255), 2) cv2.putText(annotated\_frame, 'Ball', (x1, y1-10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0,0,255), 2)

# Check if ball is in goal area

ball\_center\_x = int((x1 + x2) / 2) ball\_center\_y = int((y1 + y2) / 2) gx1, gy1, gx2, gy2 = goal\_area

if gx1 <= ball\_center\_x <= gx2 and gy1 <= ball\_center\_y <= gy2: if not goal\_detected:

goal\_detected = True

print("Goal detected!")

cv2.putText(annotated\_frame, 'GOAL!', (int(width/2)-50,

int(height/2)),

else:

cv2.FONT\_HERSHEY\_SIMPLEX, 2, (0,0,255), 4)

goal\_detected = False

# Draw goal area

gx1, gy1, gx2, gy2 = goal\_area

cv2.rectangle(annotated\_frame, (gx1, gy1), (gx2, gy2), (255,0,0), 2) cv2.putText(annotated\_frame, 'Goal Area', (gx1, gy1-10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255,0,0), 2)

cv2.imshow('Football Analysis', annotated\_frame) if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

if name == " main ": import argparse

parser = argparse.ArgumentParser(description="YOLOv8-based detection systems")

parser.add\_argument('--task', type=str, choices=['traffic', 'football', 'image'], required=True,

help="Choose the task to run: 'traffic' for traffic sign detection, 'football' for football analysis, 'image' for image

detection")

parser.add\_argument('--video', type=str, default='0',

help="Video source: 0 for webcam or path to video

file")

parser.add\_argument('--image', type=str, default='',

help="Image filename to process (used with --task

image)")

args = parser.parse\_args()

video\_source = int(args.video) if args.video.isdigit() else args.video def image\_detection(image\_path):

model = YOLO('yolov8n.pt') import os

# Adjust path to include folder "Lab 10" if not absolute if not os.path.isabs(image\_path):

image\_path = os.path.join("Lab 10", image\_path) img = cv2.imread(image\_path)

if img is None:

print(f"Failed to load image: {image\_path}") return

results = model(img)

annotated\_img = results[0].plot()

cv2.imshow(f"Image Detection - {image\_path}", annotated\_img) cv2.waitKey(0)

cv2.destroyAllWindows()

if args.task == 'traffic':

traffic\_sign\_detection(video\_source) elif args.task == 'football':

football\_analysis(video\_source) elif args.task == 'image':

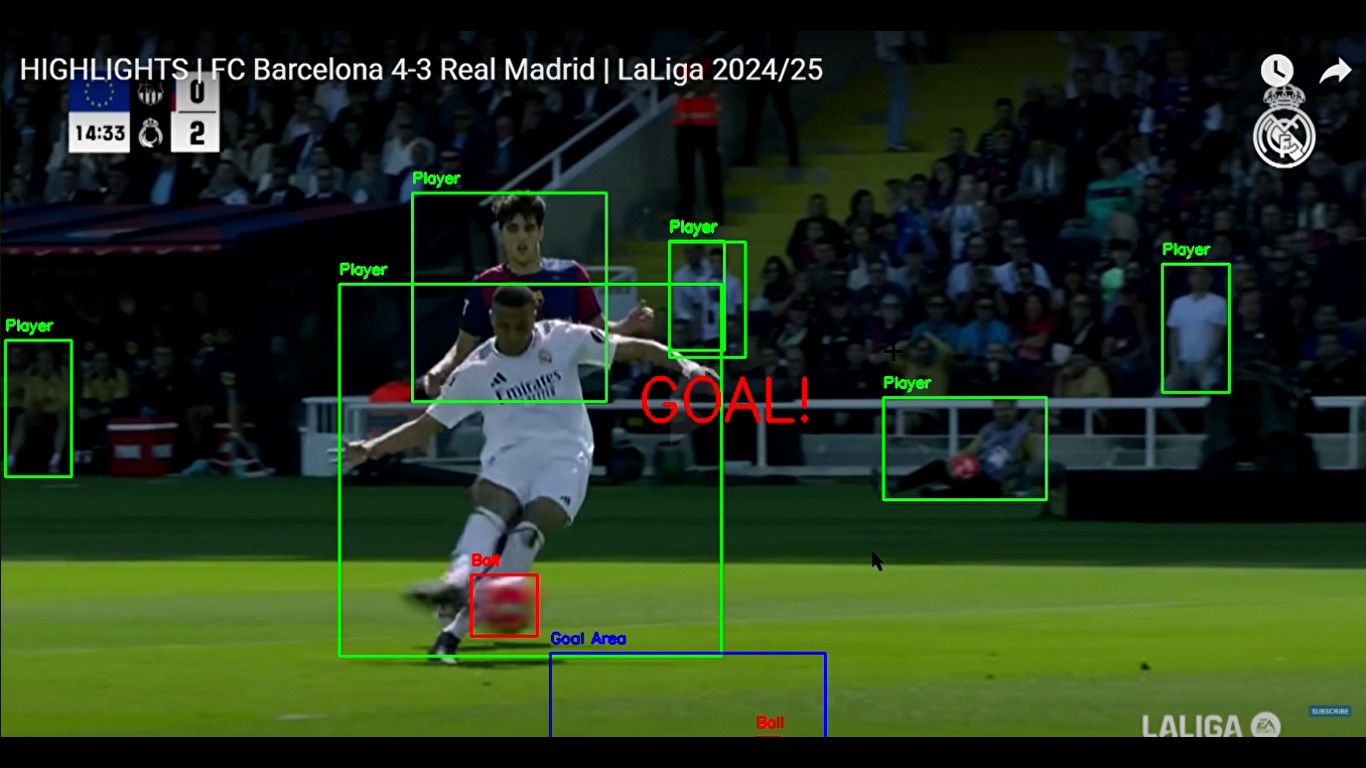
if args.image == '':

print("Please provide an image filename with --image when using --

task image")

else:

image\_detection(args.image)



1. **Object Localization:** Object Localization is the task of identifying the location of an

object within an image. It involves predicting the coordinates of a bounding box that tightly encloses the object. In object detection, localization is combined with

classification to both locate and identify objects in images.

1. **Non-Maximum Suppression (NMS) in YOLO**: NMS is a post-processing technique

used to eliminate redundant overlapping bounding boxes for the same object. YOLO predicts multiple bounding boxes with confidence scores; NMS selects the box with the highest confidence and suppresses others that have a high Intersection over

Union (IoU) overlap with it. This reduces duplicate detections and improves the precision of the model.

1. **Object Detection Performance Evaluation Metrics:**
   * Intersection over Union (IoU): Measures the overlap between predicted and ground truth bounding boxes.
   * Precision: The ratio of true positive detections to all positive detections.
   * Recall: The ratio of true positive detections to all actual objects.
   * Average Precision (AP): The area under the precision-recall curve for a specific class.
   * Mean Average Precision (mAP): The mean of APs across all classes, a common overall performance metric.
   * F1 Score: Harmonic mean of precision and recall.
   * Inference Time: Time taken to process an image, important for real-time applications.