import cv2

import numpy as np

import torch

import os

import time

import requests

from torchvision.models.detection import fasterrcnn\_resnet50\_fpn, FasterRCNN\_ResNet50\_FPN\_Weights

from torchvision.transforms import functional as F

# Download required YOLO files if they don't exist

def download\_yolo\_files():

    yolo\_files = {

        'yolov3.weights': 'https://pjreddie.com/media/files/yolov3.weights',

        'yolov3.cfg': 'https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg',

        'coco.names': 'https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names'

    }

    for filename, url in yolo\_files.items():

        if not os.path.exists(filename):

            print(f"Downloading {filename}...")

            response = requests.get(url, stream=True)

            with open(filename, 'wb') as f:

                for chunk in response.iter\_content(chunk\_size=8192):

                    f.write(chunk)

# Initialize models with proper weights

def load\_models():

    # Download YOLO files if missing

    download\_yolo\_files()

    # Faster R-CNN with updated weights API

    weights = FasterRCNN\_ResNet50\_FPN\_Weights.DEFAULT

    faster\_rcnn = fasterrcnn\_resnet50\_fpn(weights=weights)

    faster\_rcnn.eval()

    # YOLO (using OpenCV's DNN module)

    yolo\_net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

    with open("coco.names", "r") as f:

        yolo\_classes = [line.strip() for line in f.readlines()]

    return faster\_rcnn, yolo\_net, yolo\_classes, weights

# Process frame with Faster R-CNN

def process\_faster\_rcnn(frame, model, device, weights):

    # Preprocess image

    img = F.to\_tensor(frame).to(device)

    img = img.unsqueeze(0)

    with torch.no\_grad():

        predictions = model(img)

    # Get class labels

    labels = [weights.meta["categories"][i] for i in predictions[0]['labels'].cpu().numpy()]

    return {

        'boxes': predictions[0]['boxes'].cpu().numpy(),

        'labels': labels,

        'scores': predictions[0]['scores'].cpu().numpy()

    }

# Process frame with YOLO

def process\_yolo(frame, net, classes):

    height, width = frame.shape[:2]

    # Prepare input blob

    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)

    net.setInput(blob)

    # Get output layers

    output\_layers = net.getUnconnectedOutLayersNames()

    layer\_outputs = net.forward(output\_layers)

    # Process detections

    boxes = []

    confidences = []

    class\_ids = []

    for output in layer\_outputs:

        for detection in output:

            scores = detection[5:]

            class\_id = np.argmax(scores)

            confidence = scores[class\_id]

            if confidence > 0.5:  # Confidence threshold

                center\_x = int(detection[0] \* width)

                center\_y = int(detection[1] \* height)

                w = int(detection[2] \* width)

                h = int(detection[3] \* height)

                x = int(center\_x - w / 2)

                y = int(center\_y - h / 2)

                boxes.append([x, y, w, h])

                confidences.append(float(confidence))

                class\_ids.append(class\_id)

    # Apply non-max suppression

    indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    results = []

    if len(indices) > 0:

        for i in indices.flatten():

            results.append({

                "class": classes[class\_ids[i]],

                "confidence": confidences[i],

                "box": boxes[i]

            })

    return results

# Anomaly detection logic

def detect\_anomalies(faster\_rcnn\_results, yolo\_results):

    anomalies = []

    # Example anomaly: person with weapon

    weapons = ['knife', 'gun', 'pistol', 'rifle', 'scissors']

    # Check YOLO results for weapons

    weapon\_detected = any(res['class'] in weapons for res in yolo\_results)

    # Check Faster R-CNN for people

    people\_detected = 'person' in faster\_rcnn\_results['labels']

    if weapon\_detected and people\_detected:

        anomalies.append("Weapon detected with person")

    # Add more anomaly detection logic here

    # Example: unattended baggage, crowd gathering, etc.

    return anomalies

# Main function

def main():

    # Check for CUDA

    device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

    print(f"Using device: {device}")

    # Load models

    faster\_rcnn, yolo\_net, yolo\_classes, weights = load\_models()

    faster\_rcnn = faster\_rcnn.to(device)

    # Open camera

    cap = cv2.VideoCapture(0)  # Use 0 for default camera

    if not cap.isOpened():

        print("Error opening camera")

        return

    try:

        while True:

            start\_time = time.time()

            # Read frame

            ret, frame = cap.read()

            if not ret:

                break

            # Process with Faster R-CNN

            faster\_rcnn\_results = process\_faster\_rcnn(frame, faster\_rcnn, device, weights)

            # Process with YOLO

            yolo\_results = process\_yolo(frame, yolo\_net, yolo\_classes)

            # Detect anomalies

            anomalies = detect\_anomalies(faster\_rcnn\_results, yolo\_results)

            # Display results

            display\_frame = frame.copy()

            # Draw Faster R-CNN results

            for box, label, score in zip(faster\_rcnn\_results['boxes'], faster\_rcnn\_results['labels'], faster\_rcnn\_results['scores']):

                if score > 0.7:  # Confidence threshold

                    box = box.astype(int)

                    cv2.rectangle(display\_frame, (box[0], box[1]), (box[2], box[3]), (0, 255, 0), 2)

                    cv2.putText(display\_frame, f"{label} {score:.2f}", (box[0], box[1]-10),

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

            # Draw YOLO results

            for res in yolo\_results:

                x, y, w, h = res['box']

                cv2.rectangle(display\_frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

                cv2.putText(display\_frame, f"{res['class']} {res['confidence']:.2f}",

                            (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

            # Display anomalies

            if anomalies:

                for i, anomaly in enumerate(anomalies):

                    cv2.putText(display\_frame, f"ALERT: {anomaly}", (10, 30 + i\*30),

                                cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

            # Calculate FPS

            fps = 1.0 / (time.time() - start\_time)

            cv2.putText(display\_frame, f"FPS: {fps:.2f}", (10, display\_frame.shape[0] - 10),

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

            # Show frame

            cv2.imshow("AI Surveillance", display\_frame)

            # Exit on 'q'

            if cv2.waitKey(1) & 0xFF == ord('q'):

                break

    finally:

        cap.release()

        cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    main()