```
import cv2

def extract_frames(video_path, output_folder):
    cap = cv2.VideoCapture(video_path)
    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        cv2.imwrite(f"{output_folder}/frame_{frame_count:04d}.jpg", frame)
        frame_count += 1
    cap.release()
import torch
from torch.utils.data import DataLoader
from yolov3 import YOLOv3, YOLOv3Loss
from custom_dataset import CustomDataset  # Assume a custom dataset class is defined

# Load dataset
train_dataset = CustomDataset('path/to/train/data')
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)

# Initialize model and optimizer
model = YOLOv3(num_classes=80)  # Adjust number of classes as needed
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training loop
for epoch in range(num_epochs):
    model.train()
    for images, targets in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = YOLOv3Loss(outputs, targets)
        loss.backward()
        optimizer.step()
import cv2
import torch
from yolov3 import YOLOv3

model = YOLOv3(num_classes=80)
model.load_state_dict(torch.load('path/to/weights.pth'))
model.eval()

cap = cv2.VideoCapture(0)  # Use 0 for webcam or provide a video path
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    # Preprocess frame
    input_tensor = preprocess_frame(frame)
    with torch.no_grad():
        detections = model(input_tensor)
    # Draw detections on frame
    frame = draw_detections(frame, detections)
    cv2.imshow('Object Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
import cv2
import numpy as np
```

```
# Initialize Kalman Filter
kalman = cv2.KalmanFilter(4, 2)
kalman.measurementMatrix = np.array([[1, 0, 0, 0], [0, 1, 0, 0]], np.float32)
kalman.transitionMatrix = np.array([[1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1]], np.float32)
kalman.processNoiseCov = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]], np.float32) *
0.03

def track_objects(detections):
    for det in detections:
        # Assume det is (x, y, w, h)
        center = np.array([[np.float32(det[0] + det[2] / 2)], [np.float32(det[1] + det[3] / 2)]])
        kalman.correct(center)
        prediction = kalman.predict()
        # Draw prediction
        cv2.rectangle(frame, (int(prediction[0] - det[2] / 2), int(prediction[1] - det[3] / 2)),
(int(prediction[0] + det[2] / 2), int(prediction[1] + det[3] / 2)), (0, 255, 0), 2)
def draw_detections(frame, detections):
    for det in detections:
        x, y, w, h = det
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
    return frame
```