Object Detection and Tracking Report

**Flowchart:** Approach, implementation details, and evaluation.

# A diagram of a software development Description automatically generated

# Step 1: Dataset Preprocessing

Directory Setup: Created a directory to store YOLOv8-related data.

Data Extraction: Extracted dataset from a .tar.gz file for model training.

# Step 2: YOLO Labeling Format

Bounding Box Extraction: Developed a function to convert segmentation masks to YOLO format annotations.

Classes Considered: Focused on the "Person" class (ID=2) for object detection.

Bounding Boxes: Used contours to extract bounding boxes from segmentation masks.

Label File Generation: Saved YOLO-formatted annotations in .txt files corresponding to each image.

# Step 3: Model Training Setup

Installation of Libraries: Cloned and installed the ultralytics YOLO library and other dependencies like opencv, nvidia-pyindex, and nvidia-tensorrt.

Data Configuration: Created a YAML file defining training and validation paths for images and specifying object classes and their corresponding IDs.

Training Configuration: Trained the YOLOv8 model (yolov8n.pt) for 50 epochs with a batch size of 16 and image size of 640.

# Step 4: Model Optimization Using TensorRT

Export to ONNX: Converted the trained YOLOv8 model to the ONNX format.

TensorRT Conversion: Used TensorRT to optimize the model for faster inference, including support for FP16.

Model Saving: Saved the optimized model as a .trt file.

# Step 5: Object Detection with Webcam

Real-Time Capture: Used JavaScript to capture an image from the webcam and saved it as photo.jpg.

Detection: Ran YOLOv8 on the captured image for object detection.

Display: Displayed detected objects and their confidence scores in real-time.

# Step 6: Object Detection from Video

Real-Time Detection: Applied YOLOv8 for real-time object detection on video frames.

Focus on People: Focused on detecting people (Class ID = 2).

Display: Drew bounding boxes around detected people on each video frame.

# Step 7: Object Tracking Using Kalman Filters

Kalman Tracker: Implemented Kalman filters to track detected objects (people) across video frames.

Real-Time Tracking: Performed both tracking and detection simultaneously in a video file.

# Step 8: Evaluation and Metrics

Metrics: Used Precision, Recall, and IoU to evaluate the model's performance.

Performance Evaluation: Evaluated FPS (Frames Per Second) and latency for the object detection pipeline.

Scoring: Calculated precision, recall, and mean IoU scores based on ground truth boxes.

## Evaluation Results:

Precision: 0.0017

Recall: 0.5000

Mean IoU: 0.0086

FPS: 47.69

Average Latency: 0.0210 seconds