YOLOv8 Bottle Detection Report

Click here to see the video.

Overall Process

This project focused on training a YOLOv8 model to detect plastic bottles using a custom dataset. The dataset was compiled from two distinct folders: one containing training images and the other containing validation images. These images were organized and labeled properly, and a corresponding `data.yaml` file was prepared to define paths and class information.

The `data.yaml` file included the following configuration:
train: E:/Task 3A/dataset/bottle/train/images
val: E:/Task 3A/dataset/Plastic Bottle Image Dataset/valid/images

nc: 1
names: ['bottle']

Training was carried out using the Ultralytics YOLOv8n architecture. The model was trained for 50 epochs, a number chosen based on dataset size and complexity. This ensured the model had enough iterations to learn meaningful features without risking overfitting or excessive computation.

Once training completed, the best weights—determined via validation metrics—were used for inference. The model was tested on an unseen video, and predictions were saved automatically. The outputs included bounding boxes and confidence scores, which were further visualized and analyzed.

Directory Structure

```
TASK 3A/
├── dataset/
│   ├── bottle/train/
│   └── Plastic Bottle Image Dataset/valid/
├── predicted_photos/
├── predicted_videos/
├── runs/              # YOLO training outputs
├── bottle_vid.mp4     # Test video for inference
├── data.yaml          # Dataset configuration
├── yolov8n.pt         # Pretrained YOLOv8n model
├── main.py            # Training script
├── test_model_1.py    # Alternate inference script
```

└── test_model_2.py   # Primary inference script

Code and Logic

Training Logic (main.py)

The training logic is implemented in `main.py`. It begins by loading the YOLOv8n model from the Ultralytics library. The `.train()` method is used, referencing the `data.yaml` file to locate training and validation sets. The training process runs for 50 epochs, during which key metrics such as loss and mAP are monitored. The best-performing model weights are saved under the `runs/detect/train` directory.

Inference Logic (test_model_2.py)

`test_model_2.py` handles inference. It loads the best weights obtained during training and applies the `.predict()` method to analyze the test video. A confidence threshold of 0.2 is set, meaning only predictions with at least 20% confidence are considered. The script generates annotated outputs (bounding boxes and labels), which are stored in the appropriate output folders for further inspection.