Explanation video link: Click here

Overall Process

This project aimed to train a YOLOv8 model for bottle detection using a dataset accumulated from multiple sources. The process began with downloading two Kaggle datasets containing bottle images. The datasets were cleaned and merged, and a `data.yaml` file was created to define the classes and paths.

Using the Ultralytics YOLOv8 framework, the model was trained for 50 epochs on this data. The number of epochs was chosen because the dataset contained over 8,000 images — a moderately large volume where sufficient epochs were necessary to ensure the model could generalize well without overfitting. Fewer epochs might have led to underfitting, while significantly more would risk unnecessary computation or overfitting.

After training, the best model (based on validation performance) was used to run inference on an unseen test video. The predictions (bounding boxes and confidence scores) were saved automatically, and the results were later visualized and analyzed.

Directory Structure

```
Task 3a/
├── main.py              # YOLOv8 training script
├── test_yolo.py         # YOLOv8 inference script
├── data.yaml            # Dataset configuration file
├── yolov8n.pt           # Pretrained YOLOv8n weights (downloaded during training)
├── runs/
└── detect/          # YOLO training outputs (results, weights, plots)
```

Code and Logic

Training Logic (main.py)

The training script begins by loading a pretrained YOLOv8n model provided by the Ultralytics library. It then uses the `.train()` method with the specified `data.yaml` file and sets the number of training epochs to 50. During training, the model evaluates performance metrics such as loss, mAP, and accuracy to improve detection accuracy over time. The best weights based on validation results are automatically saved in the `runs/detect/train` directory.

Inference Logic (test_yolo.py)

The inference script loads the best-performing model weights from the training process. It then runs detection on a pre-recorded video using the `.predict()` function, specifying a confidence threshold of 0.2. This means only detections with a confidence score of 20% or higher are considered valid. The output includes annotated frames showing detected objects, which are automatically saved for review.