Project Overview: End-to-End Pallet Detection and Ground Segmentation Pipeline

This document explains the full process followed to develop, train, convert, deploy, and test the pallet detection and ground segmentation pipeline using both Jupyter notebooks and ROS 2 with Docker.

1. Dataset Preparation

A. Dataset Cleaning (Notebook: DATASET CLEANING.ipynb)

- **Removed Duplicates**: Identified and deleted duplicate image entries.
- **File Filtering**: Removed blurred, irrelevant, or improperly formatted images.
- **Standardization**: Renamed files uniformly for consistency in training and testing.

B. Ground Mask Generation with SAM (Notebook: GROUND SEGMENTATION MASKS PIPELINE - SAM.ipynb)

- **Tool Used**: Meta AI's Segment Anything Model (SAM).
- **Process**:
    - Manually drew bounding boxes.
    - Inspected segmentation previews.
    - Saved binary ground masks with proper overlay previews.
- **Output**: A high-quality dataset of ground segmentation masks aligned with original images.
- **Other**: Also implemented LabelMe to manually create/refine masks for around 2000 images

2. Model Training

A. Pallet Detection with YOLOv8

Notebook: PALLET_DETECTION_YOLOv8.ipynb

- **Model**: YOLOv8 (Ultralytics)
- **Input**: Cleaned and annotated dataset of pallet bounding boxes.
- **Augmentations**: Used basic transformations (flips, color shifts).
- **Training**: Conducted over multiple epochs with batch evaluation.

Notebook: TWO_PALLET_DETECTION_YOLOv8_with_AUGMENTATION.ipynb

- **Enhancement**: Included multi-pallet scenarios and richer augmentations for better generalization.

B. Ground Segmentation with U-Net + EfficientNetB3

Notebook: GROUND_SEGMENTATION_EFFICIENTNETB3_UNET.ipynb

- **Architecture**: U-Net decoder with EfficientNetB3 encoder (via segmentation_models)
- **Preprocessing**: Resized, normalized images and binary masks.
- **Training**: Trained on the SAM-generated ground mask dataset.
- **Evaluation**: Measured accuracy and IoU per epoch.

C. Other

Considered implementing **RTDETR** for pallet detection. However, was unable to do it due to lack of processing power. In addition, keeping in mind the efficiency and constraints of the assessment, I decided to stick with YOLOv8.

3. Model Conversion to ONNX

Notebook: TF2ONNX.ipynb

- **Purpose**: Convert TensorFlow .h5 U-Net model to ONNX format.
- **Tool**: tf2onnx.convert for compatibility with ONNXRuntime.
- **Verification**: Ensured correct output shapes, input format [1, 256, 256, 3]

4. Model Testing

Notebook: EUROPALLET - TEST MULTIPLE.ipynb

- **YOLO Evaluation**: Loaded ONNX YOLO model and applied to batch test images.
- **U-Net Evaluation**: Ran ONNX U-Net model on corresponding test images.
- **Visualization**: Overlaid segmentation + bounding boxes using cv2 and matplotlib.

5. ROS 2 Inference Node Development

A. ROS 2 Package Setup

- **Package**: pallet_infer_node
- **Two Nodes**:
    - pallet_infer_node.py: Inference logic using ONNX YOLO and U-Net models.
    - static_image_publisher.py: Publishes static images from folder to simulate real-time feed.
- **Topics**:
    - Input: /robot1/zed2i/left/image_rect_color
    - Output:
        - /pallet_detection (bounding boxes)
        - /ground_segmentation (binary masks)

B. Inference Logic

- **Image Format**: Subscribed to BGRA images, converted to BGR → RGB.
- **YOLOv8**:
    - Resized to [640x640] and normalized.
    - Parsed predictions, applied confidence threshold.
- **U-Net**:
    - Resized to [256x256], ran inference.

    - Mask thresholded and resized back.
- **Overlay**: Used PIL for transparent segmentation and confidence-labeled bounding boxes.
- **Output**: Saved annotated images to overlay_output/ and published them.

NOTE: I faced multiple issues while trying to read the .db3 bag file in ROS 2. To simulate a real camera input for model inference and visualization, I chose to publish a sequence of static test images instead. This method was easier to debug, platform-independent, and enabled rapid prototyping inside a Dockerized ROS 2 environment.

6. Dockerization

A. Docker Setup

- Dockerfile included in ros2_ws/.
- Dependencies:
    - ROS 2 Humble + rclpy
    - OpenCV, ONNXRuntime, cv_bridge, Pillow
- Local Development: Built image locally as ros2-humble-workspace
- **Export:**

docker save -o ros2_workspace_image.tar ros2-humble-workspace

B. Usage

- Shareable .tar file lets anyone replicate the ROS 2 environment instantly.
- All outputs visible in overlay_output/ on host system.

7. Final Outcome

- A complete, portable deep learning + robotics inference pipeline.
- Seamless transition from training to real-time deployment.

- Tested inside Docker, ready to be run on any Ubuntu ROS 2 system.
- All dependencies encapsulated and reproducible.