KITTI Dataset Preprocessing — Full English Documentation

Milestone 1: Data Collection, Exploration, and Preprocessing

This document provides a full, detailed explanation of the complete preprocessing pipeline you

implemented for the KITTI dataset in YOLO format. It explains every function, the logic behind each

step, and how the entire system fits together.

Overview

The code performs the following tasks:

- Read KITTI images and YOLO-formatted labels
- Match each image with its corresponding label
- Split the dataset into training and validation subsets
- Normalize images
- Apply data augmentation (flip, rotate, crop)
- Save the processed dataset in a structured format

The code is divided into four main functional modules:

1. preprocess_kitti_dataset() — Manages the entire preprocessing workflow
2. process_dataset_split() — Processes each subset (train/val)
3. normalize_image() — Normalizes and standardizes images
4. augment_image_and_labels() — Applies augmentation to images

---

1. preprocess_kitti_dataset()

Purpose:

This is the main function that orchestrates the entire preprocessing pipeline.

Workflow:

1) Set Random Seed

2) Create Output Directories

3) Discover Image Files

4) Discover YOLO Label Files

5) Match Images with Labels

6) Split Dataset into Train / Validation

7) Process Train & Validation Sets

---

2. process_dataset_split()

Purpose:

Processes each dataset split independently.

Main Tasks:

- Load image

- Load label

- Save normalized image

- Save YOLO label file

- Apply augmentation to training images

---

3. normalize_image()

Purpose:

Convert images into a normalized, standardized format suitable for training.

Steps:

- Convert pixel values to 0–1

- Compute mean & std

- Standardize

- Convert back to uint8 for saving

---

4. augment_image_and_labels()

Purpose:

Apply augmentation techniques to increase dataset diversity.

Augmentation Types:

- Horizontal flip (correctly adjusts YOLO x_center)

- Rotation (label update needed)

- Crop (label update needed)

Summary

This preprocessing pipeline successfully:

- Reads KITTI images and YOLO labels

- Matches files correctly

- Splits data into train/validation

- Normalizes images

- Applies augmentation

- Saves dataset in a clean structure

Milestone 2: Object Detection Model Development – Documentation

1. Introduction

This milestone focuses on developing and training a real-time object detection model for autonomous driving environments using the KITTI dataset. YOLOv8s was selected due to its balance between detection accuracy and real-time inference speed.

2. Model Selection

Three common real-time detection architectures were considered: YOLO, SSD, and Faster R-CNN.

YOLOv8s was selected because it offers high FPS, strong accuracy, and is suitable for embedded automotive systems.

3. Dataset Configuration

A custom dataset YAML file was created containing:

- Train and validation dataset paths
- Number of classes
- Explicit class name mapping

4. Transfer Learning

YOLOv8s pretrained on the COCO dataset was used to accelerate training and improve generalization. Transfer learning reduces training time and increases accuracy on KITTI's custom classes.

5. Model Training

The model was trained for 50 epochs using AdamW optimizer. Training included:

- Automatic mixed precision (AMP)

- Extensive augmentations (HSV, scale, shear, mosaic, mixup)

- Early stopping and checkpoint saving

- Batch size = 32, image size = 576

6. Evaluation

The model was evaluated on:

- mAP50

- mAP50–95

- Precision and recall

- IoU scores

- FPS during inference

7. Inference

Inference was performed on:

- External test images

- KITTI test dataset

Outputs include bounding boxes, class labels, and confidence scores.

8. Checkpoints and Saving

The training pipeline saves:

- best.pt (best performance)

- last.pt (final epoch)

- periodic checkpoints every 10 epochs

9. Conclusion

This milestone successfully develops a real-time object detection model based on YOLOv8s, optimized for autonomous driving tasks. The trained model demonstrates strong accuracy and speed, making it suitable for deployment in real-world environments.

Milestone 3: Deployment and Real-Time Testing
1. Introduction

Milestone 3 focuses on deploying the trained object detection model using Streamlit to create an interactive web-based interface capable of real-time video processing. This phase ensures that the model can operate with low latency, stable performance, and user-friendly monitoring when deployed for autonomous vehicle environments.
2. Deployment Strategy:

- Inference pipeline built using OpenCV and Streamlit

- Real-time video feed processed on-vehicle

3. Deployment Setup

3.1 Hardware Environment:

-Camera: USB camera / laptop camera (webcam)

-Streaming environment: Localhost Streamlit server

3.2 Software Environment:

-Python
-ONNX Runtime
-Streamlit
-PyTorch
-OpenCV
-NumPy

Testing Scenarios:

- Urban roads, highways, night time, foggy environment, rainy environment.

Performance Observations:

- Stable detection in most environments

Deliverables:

- Deployed Model Container (Streamlit)

- Real-time testing videos and logs

10. Conclusion:

Using Streamlit, the team successfully built a user-friendly deployment interface that supports real-time object detection. The model was tested in both controlled and real environments, validating its performance and readiness for use in autonomous vehicle contexts.