Technische Hochschule Ingolstadt

Scientific Seminar

in the study program Autonomous Vehicle Engineering
Faculty of Electrical Engineering and Information Technology

Evaluating Publicly Available Datasets for Object Detection in ADAS with Cross-Country Comparison

First and Last name : **Luckyrajsinh Hada**

Submitted on : 11/06/2025

Supervisor : Ms. Carla Roth

Acknowledgments

I would like to express my gratitude to my supervisor, Ms. Carla Roth, for her guidance and support throughout the preparation of this seminar report. Her assistance in filtering out the aim and structuring the writing was crucial to the completion of my scientific seminar.

Additionally, I extend my thanks to the Technische Hochschule Ingolstadt (THI), specifically the computer lab on the 3rd floor of Building G, for providing the high-performance computing resources.

The rest of the work presented here is a result of my own efforts and knowledge.

Luckyrajsinh Hada

Ingolstadt, Germany

June 2025

Abstract

This study investigates the cross-domain adaptability of object detection models in the context of autonomous driving systems. A YOLOv8 model was trained using the German KITTI dataset, which contains labeled road objects such as cars, vans, trucks, pedestrians, and cyclists. The model achieved strong in-domain performance, with a mean Average Precision (mAP@0.50) of approximately 92% on the KITTI validation set. To evaluate its generalization ability, the same model was tested on the Indian Driving Dataset (IDD), which represents an entirely different traffic domain, characterized by unstructured roads, diverse vehicle types, and dense environments. When evaluated on this out-of-domain data, the model's detection accuracy dropped significantly (mAP@0.50 12%), revealing that it struggled to identify and classify objects correctly in the new context. These results highlight a critical challenge in deploying autonomous systems across geographies: models trained in one region may fail in another due to domain shifts. Therefore, for reliable real-world deployment, object detection models must be trained or adapted to the specific environmental and traffic conditions of their intended operational domain.

List of figures

Table of contents

1 Introduction

1.1 Context and Motivation

Autonomous vehicles are rapidly transforming the transportation system, with Advanced Driver Assistance Systems (ADAS) a key role in safer driving experiences. A main function of ADAS is real-time object detection, which allows vehicles to recognize and respond to vehicles, traffic signals, obstacles, and pedestrians. Most of the object detection models are trained and evaluated in controlled environments and specific to a region. However, the real-world deployment of autonomous vehicles introduces a significant challenge: cross-country generalization. Traffic density, driving behavior, road design, and environmental conditions are different in every country. A model trained on structured roads may perform poorly when exposed to much busier driving environments. This highlights a critical gap in current research and motivates the need to evaluate object detection models across different geographic datasets.

1.2 Research Objective

The objective of this research is to evaluate how object detection models trained on publicly available datasets from one country perform when tested on data from another country. This cross-country evaluation aims to understand the model's adaptability and limitations when deployed in unfamiliar environments. Furthermore, the study explores performance variation when the training and testing environments vary in traffic patterns , road structures and vehicle density ,and How current ADAS systems can be better prepared for deployment in diverse real-world conditions.

1.3 Problem, Limitations, and Scope

Despite having the availability of several high-quality public datasets, most of them are region-specific and lack global diversity. As a result, object detection models tend to overfit on certain regions and underfit on certain regions. This can be a major issue when building globally deployable ADAS systems. This research focuses on camera-based object detection using 2D bounding boxes and the YOLO framework (YOLOv8).The scope is restricted to use the publicly available datasets.

2 Literature Review

2.1 ADAS and Real-Time Object Detection

Advanced Driver Assistance Systems (ADAS) Object detection identifies objects like vehicles, pedestrians, and other critical objects on the roads using camera data. Fast and efficient models such as YOLO helps real-time object detection, which is essential for safe decision-making in autonomous systems.

2.2 Publically avaiable ADAS Datasets

Camera-Based Image Datasets for Object Detection contain RGB images with labeled objects using bounding boxes. It is Used to train and evaluate deep learning models that can detect and classify multiple objects in the images. Images : RGB images captured from vehicle-mounted cameras. These images represent real-world driving scenes for supervised model training.

Geiger et al. [2012]

Figure 1: Example of KITTI object detection dataset image

Bounding Boxes : rectangular region that encloses an object in the image. Represented by coordinates. Label : A text file

type truncation occlusion alpha bbox left bbox top bbox right bbox bottom height width length x y z rotation y Pedestrian 0.00 0 -0.20 712.40 143.00 810.73 307.92 1.89 0.48 1.20 1.84 1.47 8.41 0.01 About kitti dataset : Created by Karlsruhe Institute of Technology and Toyota Technological Institute. Designed for computer vision tasks in autonomous driving. Images from cameras mounted on a car. Contains urban, residential, and highway scenes. It contains classes : Car, Pedestrian, perosn sitting , Van, Truck. Image Properties : Resolution: 1242x375 , RGB color images (PNG file) Currently, several publicly available datasets such as IDD (India), ApolloScape (China), Waymo Open Dataset (USA), and nuScenes (Singapore/USA) are widely used in autonomous driving research. These datasets are essential for developing machine learning models based on camera sensor data.

2.3 Object Detection with YOLO

YOLO runs a Convolutional Neural Network (CNN) to both locate and identify all objects in an image. This design makes it fast and ideal for systems that need real-time performance, such as ADAS.

A CNN transforms raw RGB pixels into hierarchical features. Early convolutional layers learn to pick up simple patterns like edges or color gradients. As the data flows deeper into the neural network, next layers assemble these basics into more complex shapes—corners, textures, and ultimately parts of objects like wheels or pedestrians. Finally, fully connected layers take these learned features and predict, for each grid cell, a set of bounding-box coordinates plus the confidence that an object is present.

Once the network has produced its raw scores—one score for each class at each spatial location—YOLO applies a softmax function to turn those scores into proper probabilities

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

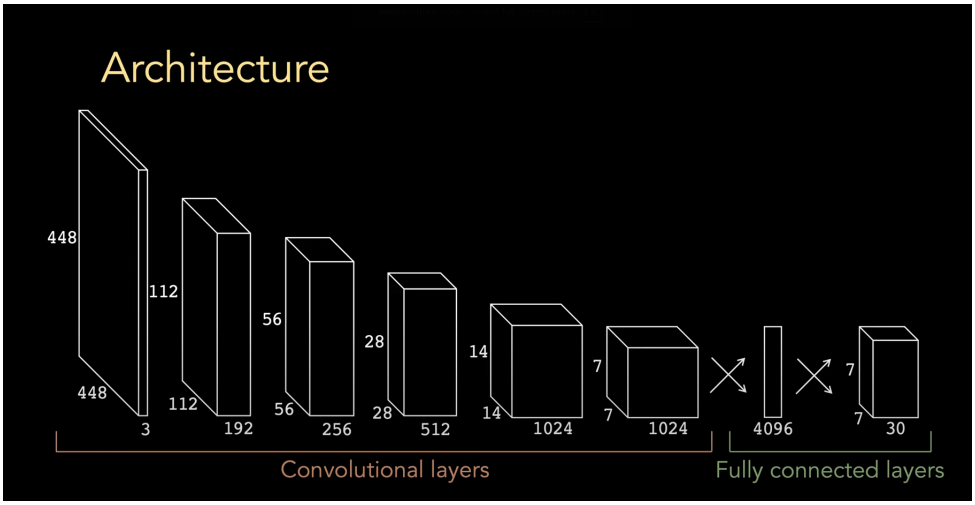

Figure 2: Convolutional Neural Networks (CNNs)DeepBean [2023]

During training, these predicted probabilities are compared against the ground truth labels using the cross-entropy loss. This encourages the model to assign high probability to the correct class and low probability to the others. At the same time, a localization loss (often based on IoU) nudges the bounding boxes to more tightly overlap their targets.

By learning classification and localization jointly in one pass, YOLO gives an effective balance between speed and accuracy making it a good choice for multiclass object detection tasks in supervised learning.

2.4 Evaluation Metrics

After each training epoch, the model is evaluated using standard object detection metrics. These help measure how well the model is performing on the validation set:

Precision measures how many of the predicted objects are actually correct:

$$\text{Precision} = \frac{\text{TP}}{\text{FP} + \text{TP}}$$

Recall measures how many of the actual objects were correctly predicted:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Average Precision (AP) is the area under the precision–recall (P–R) curve for a single class.

mAP@0.5 is the mean Average Precision across all classes, calculated at an IoU threshold of 0.5.

mAP@0.5:0.95 (optional) is the mean AP averaged over multiple IoU thresholds (from 0.5 to 0.95 in steps of 0.05), giving a stricter evaluation of localization accuracy.

More details and results are discussed in Chapter 4.

2.5 Cross-Domain Generalization

Object detection models trained on one dataset often perform poorly on images from a different dataset. This degradation appears because each dataset captures a distribution of visual features—such as lighting conditions, object appearance, shapes, textures, colours, camera viewpoints, and annotation styles. When the properties of the new domain (here testing datasets) differ significantly from those of the original (here on which the model was trained)domain, the model's learned weights no longer align with what it encounters with the unseen dataset. One way to find the difference between two domains is to compare their feature distributions using Kullback–Leibler (KL) divergence. Let P denote the probability distribution of features in the source domain, and let Q denote the corresponding distribution in the target domain. The KL divergence is defined as:

$$D_{\text{KL}}(P||Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

A high KL divergence value indicates that the two distributions are very different—that is, the model will see patterns in the target data that it has not learned to recognize. When It is large, object detection performance (measured by mean Average Precision) tends to drop.

2.6 Summary & Gaps

The majority of studies evaluate models on a single dataset or within a specific region. There is a lack of systematic cross-country evaluations, particularly involving real-time object detectors. Existing studies do not include cross country adaptability such as German roads in KITTI—perform when exposed to unstructured or dense traffic environments like those in India (IDD) This study aims to fill that gap by quantifying the generalization limits of YOLO-based models when deployed across domains. Specifically, it evaluates performance drops in key metrics such as mean Average Precision (mAP), precision, and recall when a model trained on the KITTI dataset is tested on the IDD dataset. These experiments help identify the challenges of cross-domain feature distribution mismatches and the importance of dataset diversity for global applicability in Advanced Driver Assistance Systems (ADAS).

3 Tools and Technologies

3.1 Python Ecosystem

The primary programming language utilized for the entire practical work was **Python 3.13**. This included tasks such as data preprocessing, annotation format conversion, and model training.

Various **Python libraries** were essential for different stages of the pipeline:

- **Pandas** was employed for handling structured data and manipulating label files, such as those from the KITTI dataset.
- **NumPy** facilitated numerical operations, especially for processing image arrays.
- **Matplotlib** was used for visual analytics to generate training plots.
- **OpenCV** provided tools for image operations and preprocessing steps like resizing and normalization.

3.2 YOLO Object Detection Framework

The **YOLOv8 framework** by Ultralytics served as the core object detection system. YOLOv8 is a one-stage object detector that combines objectness, classification, and CIoU-based bounding box regression into a unified loss function. During model training, pre-trained **COCO weights** (`yolov8l.pt`) were utilized. The model is built on the **PyTorch** backend, leveraging GPU acceleration for efficient training and inference.

3.3 Hardware and Environment

The practical work was conducted on an **Ubuntu Linux system**. The hardware specifications included an **Intel i9 CPU** and an **NVIDIA Quadro RTX 4000 GPU** with 8 GB of VRAM, supported by 64 GB of RAM. **Visual Studio Code** was used as the integrated development environment, with its integrated terminal serving for command execution.

4 Methodology

4.1 Dataset Collection and Exploration

4.1.1 Dataset Format, Structure, and Preprocessing (KITTI Object Detection)

The KITTI Object Detection provides standardized RGB images and corresponding 2D annotations, captured from a vehicle-mounted stereo rig in urban, rural, and highway settings.

Specifically, the dataset comprises 7,481 training images and 7,519 testing images stored as JPEG files within the image/ directory. Each training image has an associated annotation file in the label/ directory, KITTI label format : The label format includes fields such as type, truncation, occlusion, alpha, bbox_left, bbox_top, bbox_right, bbox_bottom, height, width, length, x, y, z, and rotation_y. Here, `type` indicates the object class (e.g., Car, Pedestrian, van). `bbox_left`, `bbox_top`, `bbox_right`, and `bbox_bottom` denote the pixel coordinates of the 2D bounding box. The remaining fields (truncation, occlusion, alpha, physical dimensions, and 3D location/orientation) label fields that describe 3D or partially hidden aspects of each object—information not needed for 2D bounding-box detection.

For YOLO-based training, all annotations were converted to the normalized center-width-height format:

```
<class_id> <x_center> <y_center> <width> <height>
```

where `<x_center>` and `<y_center>` correspond to the bounding-box center coordinates, and `<width>` and `<height>` denote the box dimensions. A custom Python script was implemented for this task.

The full script is provided in Appendix A for reference. Once label conversion was complete, the 7,481 annotated images were randomly partitioned into training and validation subsets using an 80/20 split. The training subset therefore contains 5,984 images (and their corresponding label files), and the validation subset contains 1,497 images (and with labels). The random split was made with a Python script that shuffles the image list and allocates the first 80% to train/ and the remaining 20% to val/.

The full script to split dataset is given in Appendix B for reference.

4.2 Training Object Detection Model (YOLO

4.2.1 Training Setup

Model Training Details

The training of the model was carried out using the Ultralytics YOLOv8 object detection framework. We leveraged a pre-trained model (`yolov8l.pt`) for efficient training.

Data Configuration File (`data.yaml`)

The `data.yaml` file serves as the central configuration for the YOLO training pipeline. It defines the paths to the dataset images and labels (`train/images`, `val/images`, `train/labels`, `val/labels`). This ensures the training script can correctly locate and load the annotated data. The YAML file also specifies the number of object classes (`nc: 5`) and their names:

- 'Car'
- 'Van'
- 'Truck'
- 'Pedestrians'
- 'person_sitting'

The complete data configuration file is provided in Appendix C.

Hyperparameters and Training Configurations

- **Model Weights:** `yolov8l.pt`, a pre-trained model on the COCO dataset, was used to initialize the model parameters.
- **Epochs:** The model was trained for 100 epochs.
- **Batch Size:** A batch size of 2 was used. This refers to the number of training images processed before the model's internal parameters (weights) are updated.
- **Image Size:** Input images were resized to 416×416 pixels for compatibility with YOLO.
- **Device:** Training was conducted on a GPU (`device=0`).

The training process was executed using the Ultralytics YOLOv8 Python API. It specifies the model path, data YAML file, training duration (100 epochs), image size (416×416), batch size, and computing device.

The full script to start the training can be found in Appendix D.

4.3 Validation Metrics

During training, three loss components are computed and updated every batch:

- **Box Loss** measures the error between the predicted bounding-box coordinates and the ground truth, typically using a CIoU/GIoU-based formulation to encourage tighter overlap. (In Chapter 2.2.2, IoU is mentioned; in theory, YOLOv8 uses advanced metrics like CIoU and GIoU, which add a penalty for boxes that have low overlap.)
- **Class Loss** quantifies the classification error of predicted class probabilities against the true labels using cross-entropy.
- **DFL Loss (Distribution Focal Loss)** refines the predicted bounding-box widths and heights, resulting in more precise localization.

In addition to these per-batch losses, performance is evaluated on the validation set after each epoch using the following metrics:

- **Precision and Recall**

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  - **True Positives (TP):** Predicted boxes whose predicted class matches the ground truth.
  - **False Positives (FP):** Predicted boxes are classified incorrectly.
  - **False Negatives (FN):** Ground-truth boxes that were not detected by any prediction above the IoU threshold.

- **mAP@0.5** (Mean Average Precision at an IoU threshold of 0.5) is computed by first generating a Precision–Recall curve for each class and then taking the average area under these curves across all classes. A detection is considered correct if IoU ≥ 0.5.
- **mAP@0.5:0.95** This metric averages AP values computed at multiple IoU thresholds, typically from 0.50 to 0.95 in increments of 0.05. By evaluating model performance over a range of overlap criteria, mAP@0.5:0.95 provides a comprehensive measure of both detection accuracy and localization quality.

These validation metrics guide hyperparameter tuning and help determine when the model has begun to overfit.

4.4 Cross-Dataset Evaluation

4.4.1 Model Transfer Testing

Cross-Dataset Evaluation: German Model on Indian Dataset

To evaluate the German model (trained on the KITTI Dataset) on the Indian Driving Dataset (IDD), the initial step involved aligning the label sets of both datasets. The German dataset (KITTI) comprises five classes: `['Car', 'Van', 'Truck', 'Pedestrian', 'person_sitting']`. In contrast, the IDD includes fifteen classes: `['animal', 'autorickshaw', 'bicycle', 'bus', 'car', 'caravan', 'motorcycle', 'person', 'rider', 'traffic light', 'traffic sign', 'trailer', 'train', 'truck']`. Since the German model is not trained to predict classes outside its original five categories, all IDD labels that did not correspond to the German classes were removed.

A Python script (detailed in Appendix E) was utilized to perform the following:

- Remove non-common classes from each Indian dataset's `.txt` label file, retaining only those whose class name matched the five German class categories.
- Reassign class indices so that the remaining labels followed the sequence order expected by the German model (0–4 for Car, Van, Truck, Pedestrian, and person_sitting, respectively).

After this removal and reindexing process, a new data configuration YAML file (provided in Appendix F) was created. This file directed the model to the paths of the aligned Indian images and labels, maintaining the same class sequence as the German model.

Subsequently, the German model's `best.pt` weights were loaded, and testing was performed on the filtered Indian images. A dedicated Python script (see Appendix G) facilitated this testing, taking paths to the German-trained weights and the filtered Indian dataset YAML file as inputs.

This evaluation yielded the following outputs:

- `confusion_matrix.png`**:** A heatmap visually representing true versus predicted class counts.
- `results.png`**:** The mAP curve, illustrating the model's precision–recall performance.
- `metrics.csv` **and** `metrics.json`**:** Summary files containing key metrics, including precision, recall, mAP@0.5, and mAP@0.5:0.95.
- `val_predictions.jpg`**:** A sample image displaying predicted bounding boxes.

This comprehensive process allowed for the analysis of the cross-country model's adaptability using YOLOv8's intrinsic evaluation metrics.

4.4.2 Adaptability Check Across Regions

By comparing key metrics (precision, recall, mAP@0.5, mAP@0.5:0.95) from two evaluation scenarios:

- **In-domain evaluation:** German model tested on KITTI test images.
- **Cross-domain evaluation:** German model tested on Indian (IDD) images.

We can directly measure the drop in detection quality attributable to domain differences in scene density, object appearance, and class distribution. A high divergence between in-domain and cross-domain metrics indicates poor adaptability of the model to the new domain.

5 Results and Analysis

5.1 Performance on Individual Datasets (KITTI Dataset)

The YOLOv8 model was trained for 100 epochs on the KITTI Object Detection benchmark, which consists of 7,481 training and 1,497 validation images. At the end of training, the following validation metrics were recorded (all values after epoch 100):

- **Precision (all classes):** 0.941
- **Recall (all classes):** 0.849
- **mAP@0.50 (all classes):** 0.9187
- **mAP@[0.50:0.95] (all classes):** 0.7164

These results indicate that, at an IoU threshold of 0.50, the model correctly identifies and localizes approximately 91.9% of objects in the KITTI validation set. This is achieved with a precision of 94.1% (suggesting few false positives) and a recall of 84.9% (indicating a relatively low number of false negatives). When averaging over multiple IoU thresholds (from 0.50 to 0.95), the model achieves a mAP of 71.6%.

Epoch-Wise Loss and Metric Trends

The monotonic improvement observed in the plots validates that the model converged smoothly without severe overfitting.

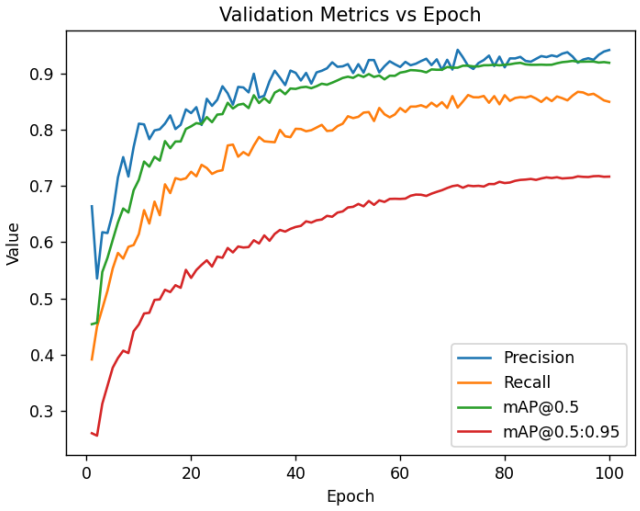

Figure 3: Validation Metrics Vs Epochs

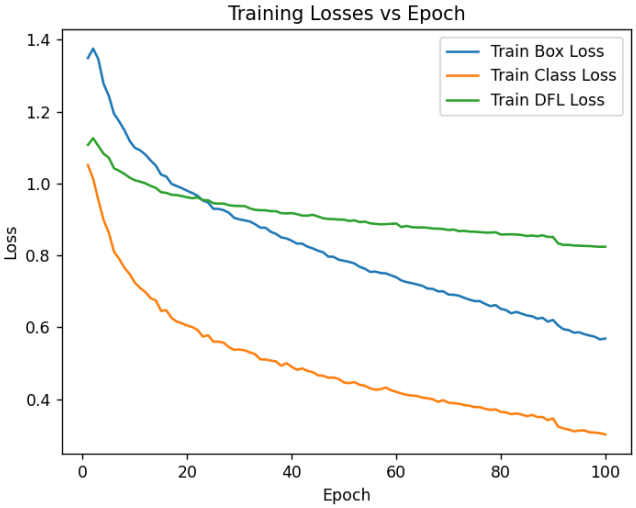

Figure 4: Training Losses Vs Epochs

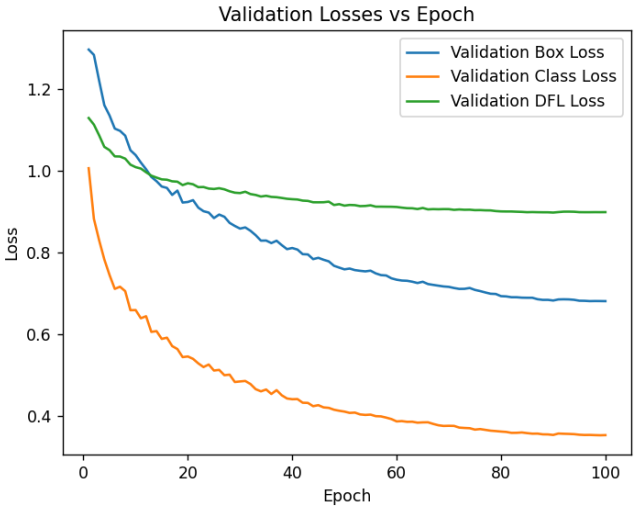

Figure 5: Validation Losses Vs Epochs

The training progress of the model is visualized through epoch-wise plots of training and validation metrics. These include loss curves (box loss, classification loss, DFL loss) and performance metrics like precision, recall, mAP@0.5, and mAP@0.5:0.95. The plots show a consistent decline in losses over the epochs, indicating effective learning, and the precision and mAP metrics show gradual improvement, showing better object detection performance. These visualizations help in understanding the model's convergence behavior.

Confusion Matrix and Class-Wise Observations

The normalized confusion matrix from the training outputs logs shows a clear pattern:

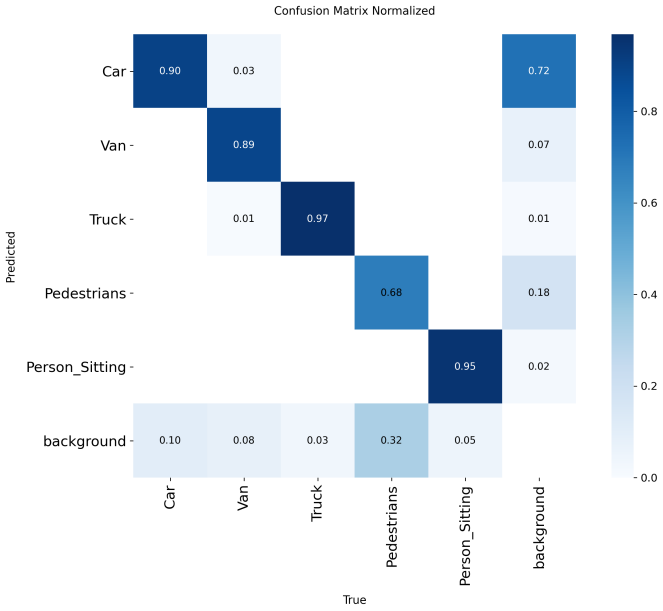

Figure 6: Confusion Matrix Normalized

- Trucks were always correctly detected, with the least misclassifications.
- Cars and Vans show mutual confusion (approximately 3 to 4% of vans predicted as cars).

Precision–Recall Curve and F1

The Precision–Recall curve shows strong performance: precision remains above 0.90 until recall exceeds 0.80, and the area under the curve corresponds to mAP@0.50.

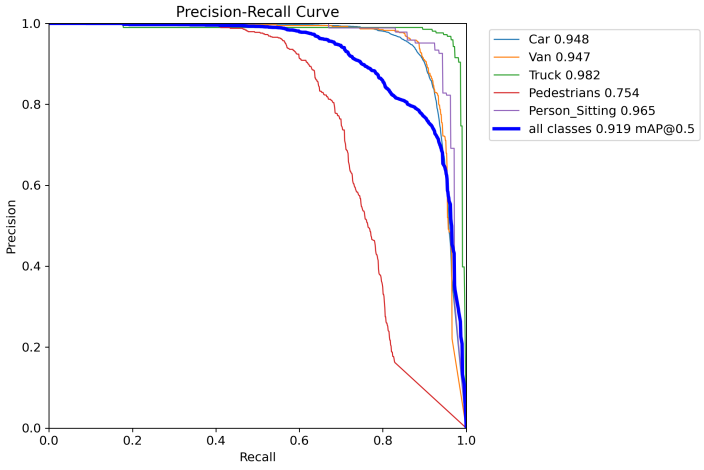

Figure 7: Precision-Recall (PR) Curve

The F1-score curve peaks around a confidence threshold where precision and recall are balanced (F1 $\approx$ 0.89). This operating point reflects an optimal trade-off for deployment on German roads.

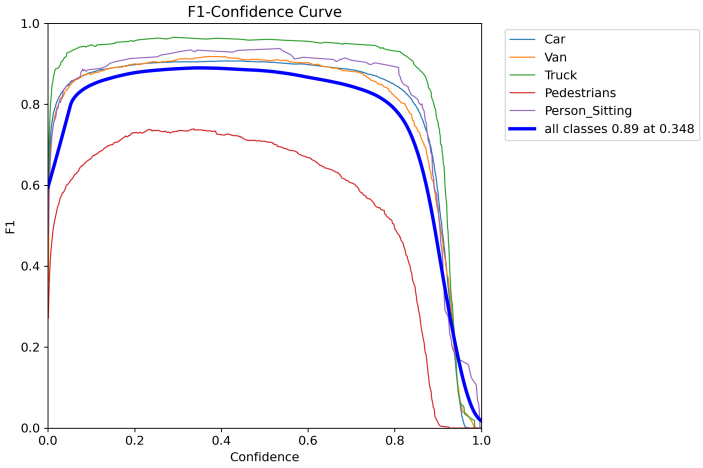

Figure 8: F1 Curve

Qualitative Results

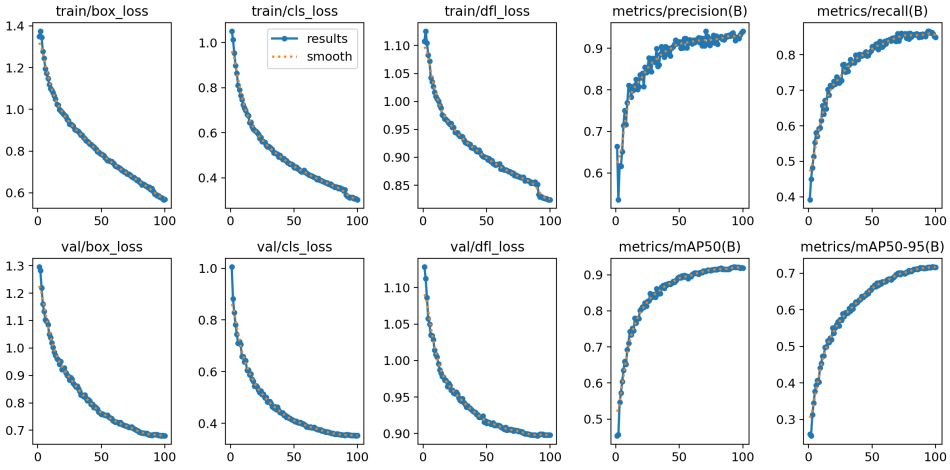

Figure 9: All Training Plots (Losses, Metrics, PR, F1)

Figure 10: Sample Validation Batch PredictionGeiger et al. [2012] with bounding box from testing

Sample validation images with overlaid predictions demonstrate that the model accurately localizes vehicles and pedestrians.

Summary for KITTI

After 100 epochs, YOLOv8 achieves high detection performance on the KITTI validation set—mAP@0.50 of 0.9187, with overall precision 0.941 and recall 0.849. Class confusion is minimal except for visually similar categories (Car and Van). These results establish a robust baseline for subsequent cross-dataset evaluation.

5.2 Cross-Country Analysis

5.2.1 Overall Performance Degradation

The evaluation shows a substantial decline in the German model's detection performance when changing domain from the KITTI dataset to the Indian dataset. The validation metrics for all classes show a huge domain gap:

- **Precision (all classes):** 0.2509
- **Recall (all classes):** 0.1229
- **mAP@0.50 (all classes):** 0.1217
- **mAP@[0.50:0.95] (all classes):** 0.0649

These figures in contrast to the in-domain KITTI results, where mAP@0.50 was 0.9187. The dramatic reduction across all metrics depicts the model's limited ability to generalize features learned on German scenes to the complex traffic patterns observed in India.

5.2.2 Class-Wise Performance Analysis

An examination of class-wise mAP@0.50 scores further depicted the performance gap. The model retained some capacity to detect larger vehicle categories, such as "Car," achieving an mAP of 0.2124. However, its performance on other critical classes was severely hampered:

- **"Van":** 0.0227
- **"Truck":** 0.0005
- **"Pedestrians":** 0.0242
- **"Person_Sitting":** 0.0649

The near-negligible performance on classes like "Van," "Truck," and "Pedestrians" indicates that features reliably distinguished in the KITTI domain became effectively unrecognizable in the target Indian domain. This suggests that the model struggles to identify specific object characteristics in the new environment due to variations in appearance, size, occlusion, or background complexity.

Further Analysis of Class-Wise Precision and Recall

Further analysis of class-wise precision and recall provides deeper insights:

- **Precision:** The model exhibited its highest precision for the "Car" class at 63.8%, indicating a high rate of correct predictions for this category. "Pedestrian" and "Van" classes showed moderate precision at 21.9% and 14.6%, respectively. Conversely, "Truck" detections were highly unreliable, with a precision of merely 0.07%, reflecting a significant false-positive rate. The "Person_Sitting" class was not detected at all, possibly due to its infrequent occurrence or absence in the Indian dataset's labels.
- **Recall:** "Cars" achieved the highest recall at 31.1%, suggesting a reasonable capture of actual car instances. However, "Pedestrians" (9.7%) and "Vans" (8.4%) had considerably lower recall, implying a large number of missed detections. The recall for "Trucks" was almost negligible (0.004%), highlighting extreme under-detection in this category.

5.2.3 Visual Representation of Performance

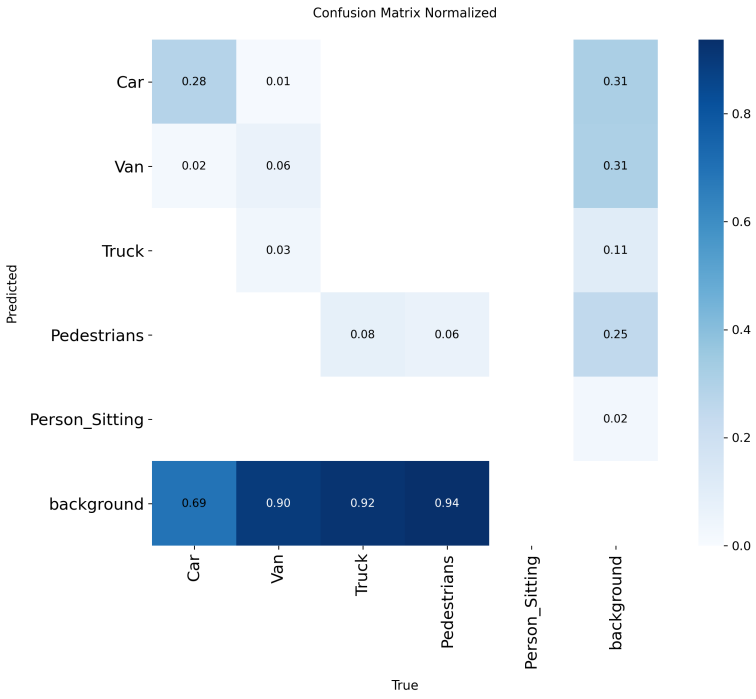

Figure 11: Normalized Confusion Matrix

This matrix visualizes the normalized counts of true positive, false positive, and false negative predictions for each class, including the 'background' class, highlighting where the model struggles with misclassification. The Normalized Confusion Matrix (Figure 11) offers a granular view of misclassifications. It particularly highlights the significant number of true objects that are misclassified as background, underscoring the severe under-detection for certain classes, especially "Trucks" and "Pedestrians." The high off-diagonal values for "background" predictions further emphasize the model's tendency to miss actual objects in the Indian dataset.

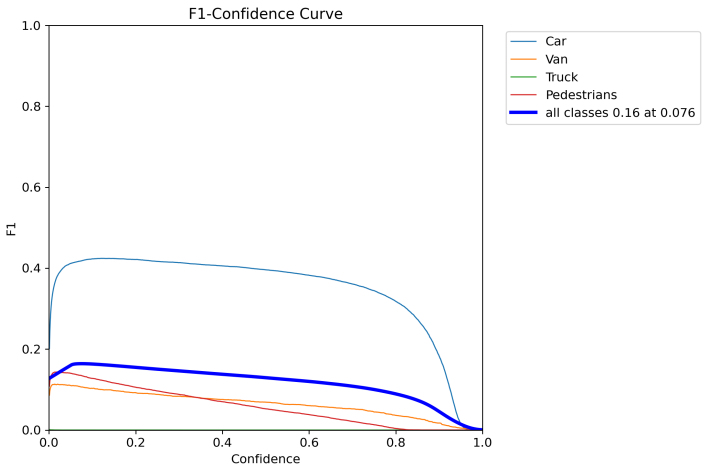

Figure 12: F1 Curve for Cross-Domain Evaluation

The F1-Confidence curve (Figure 12) provides additional insight into the model's performance by showing the harmonic mean of precision and recall at various confidence thresholds. The overall F1-score peaks at a very low value, further substantiating the limited effectiveness of the model in the cross-domain setting. This plot displays the F1-score across different confidence thresholds for each class and overall. The low peak F1-score for most classes and the overall average underscores the model's poor generalization ability in the new domain.

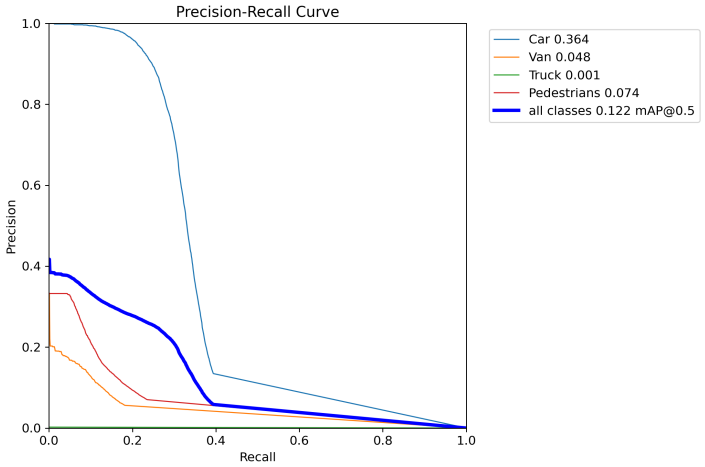

Figure 13: Precision-Recall (PR) Curve for Cross-Domain Evaluation

The Precision-Recall (PR) curve (Figure 13) visually confirms the collapse in model performance. Precision consistently falls below 0.3 at low recall values, and the overall recall never exceeds 0.15 before precision rapidly declines towards zero. This characteristic shape signifies that a majority of high-confidence detections are false positives, and the model largely fails to retrieve a significant fraction of true objects. The low overall recall (12.3%) indicates that only a small subset of Indian road objects are recognized, while the low overall precision (25.1%) quantifies a high rate of incorrect predictions among those it does make. This plot illustrates the trade-off between precision and recall across various confidence thresholds. The overall curve (bold blue

line) demonstrates the severe drop in precision at low recall, indicating a high false-positive rate and limited true object retrieval.

Conclusion

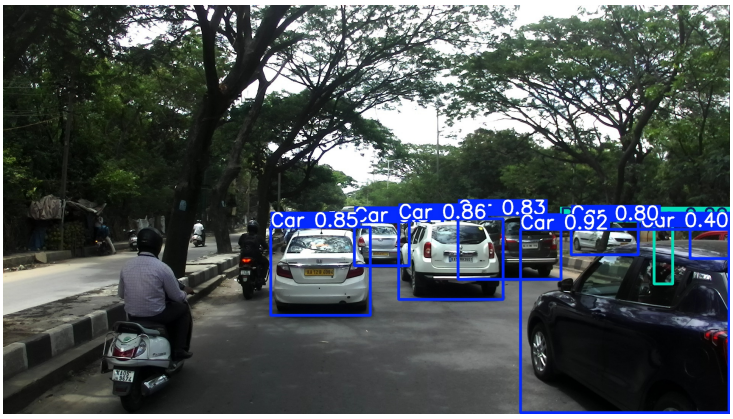

Figure 14: IDD Image with bounding box from testing results Varma et al. [2019]

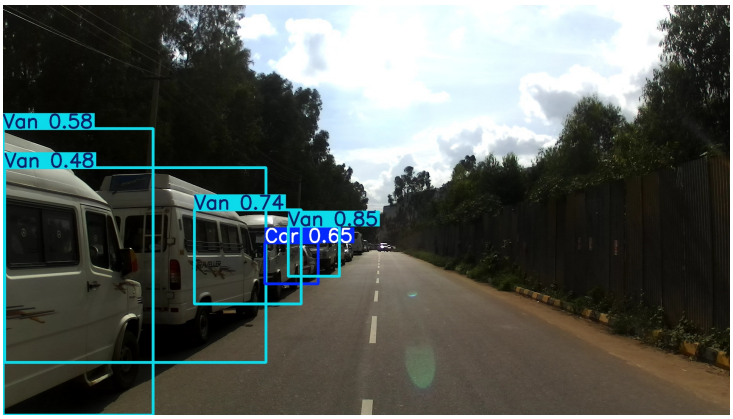

Figure 15: IDD Image with bounding box from testing results Varma et al. [2019]

This cross-domain evaluation quantifies the necessity of domain adaptation or broader dataset diversity to achieve robust and globally applicable Advanced Driver-Assistance Systems (ADAS) performance. The dramatic reduction in both mAP and PR metrics, and the detailed class-wise analysis and evidence from the confusion matrix, F1, and PR curves, unequivocally shows that features learned on German scenes do not generalize effectively to the dense and highly heterogeneous traffic patterns in India. Classes that were consistently detected and distinguished in the source domain (KITTI) became unrecognizable in the target domain. Future work should focus on techniques such as unsupervised domain adaptation, semi-supervised learning to bridge this domain gap and enhance the generalizability of object detection models for real-world autonomous driving applications.

6 Conclusion and Future Work

6.1 Summary of Findings

This study evaluated a YOLOv8-based object detector trained on the German KITTI dataset and tested on both in-domain (KITTI) and out-of-domain (Indian IDD) images. When evaluated on KITTI validation data, the model achieved high accuracy (mAP@0.50 $\approx$ 0.92, precision $\approx$ 0.94, recall $\approx$ 0.85), indicating that YOLOv8 can reliably detect cars, vans, trucks, pedestrians in structured German road scenes.

However, when applied without modification to the filtered IDD images, performance collapsed (mAP@0.50 $\approx$ 0.12, precision $\approx$ 0.25, recall $\approx$ 0.12). Feature distributions in Indian scenes—characterized by dense, heterogeneous traffic and additional object types—diverged significantly from the German training distribution, explaining the model's inability to generalize. The Precision–Recall curve confirmed that both false positives and false negatives increased drastically out of domain, quantifying a domain gap.

6.2 Limitations Encountered

limitations influenced these results:

- **Class Mismatch:** First, the model was trained exclusively on KITTI's five classes, whereas IDD contains fifteen classes. Restricting the Indian labels to common classes.
- **Compute Constraints:** second, compute and time constraints prevented exploration of larger YOLO variants, longer training on more datasets including classes like cyclists, motorbikes and buses. which would have increased thee performance of german model on Indian datasets.

6.3 Recommendations and Future Work

The results of this study highlight the need for ADAS models to be robust across geographic and domain shifts. Based on the findings, the following key directions are recommended:

- **Region-Aware Training:** Train detection models on a mix of datasets from different regions or fine-tune them on target-domain data to improve real-world adaptability.
- **Domain Adaptation Techniques:** Explore unsupervised or semi-supervised methods (e.g., self-training, feature alignment) to bridge domain gaps without requiring extensive labeled data.
- **Incremental Class Learning:** Expand models to recognize region-specific object classes such as autorickshaws or animals using techniques like zero-shot or continual learning.
- **Multi-Modal Inputs:** Integrate sensor fusion (e.g., camera + LiDAR) to reduce visual domain sensitivity in challenging environments.
- **Cross-Geographic Evaluation:** Extend the current work by testing cross-domain performance across more diverse datasets (e.g., USA $\rightarrow$ India, Europe $\rightarrow$ Asia) for better global generalization.

7 Appendix A Python Script for Converting KITTI Labels to YOLO Format

```python
import os

class_mapping = {
    "Car": 0,
    "Van": 1,
    "Truck": 2,
    "Pedestrian": 3,
    "person_sitting": 4
}

image_width = 1242
image_height = 375

def kitti_to_yolo(kitti_file, output_dir):
    with open(kitti_file, 'r') as file:
        lines = file.readlines()

    yolo_annotations = []
    for line in lines:
        parts = line.strip().split()
        if not parts:
            continue
        class_name = parts[0]
        if class_name not in class_mapping:
            continue
        class_id = class_mapping[class_name]
        x1, y1, x2, y2 = map(float, parts[4:8])
        x_center = ((x1 + x2) / 2) / image_width
        y_center = ((y1 + y2) / 2) / image_height
        width    = (x2 - x1) / image_width
        height   = (y2 - y1) / image_height
        yolo_annotations.append(f"{class_id} {x_center:.6f} {y_center:.6f} {width:.6f} {height
    :.6f}\n")

    base_name = os.path.basename(kitti_file)
    yolo_filename = os.path.join(output_dir, base_name)
    with open(yolo_filename, 'w') as out_file:
        out_file.writelines(yolo_annotations)

kitti_files_dir = "/path/to/kitti_labels"
output_dir      = "/path/to/output"
os.makedirs(output_dir, exist_ok=True)

for filename in os.listdir(kitti_files_dir):
    if filename.endswith(".txt"):
        kitti_path = os.path.join(kitti_files_dir, filename)
        kitti_to_yolo(kitti_path, output_dir)
```

8 Appendix B Python Script for Dataset Splitting (80/20)

```python
import os
```

```
import random
import shutil


images_path = "path to the images folder to split"
labels_path = "path to the labels folder to split"

output_base = "desired output folder"

# Defining the output directories for training and validation
train_img_dir = os.path.join(output_base, "train", "images")
train_lbl_dir = os.path.join(output_base, "train", "labels")
val_img_dir = os.path.join(output_base, "val", "images")
val_lbl_dir = os.path.join(output_base, "val", "labels")

# output folders
for folder in [train_img_dir, train_lbl_dir, val_img_dir, val_lbl_dir]:
    os.makedirs(folder, exist_ok=True)

all_images = sorted([f for f in os.listdir(images_path) if f.endswith(".png")])
random.shuffle(all_images)


split_index = int(0.8 * len(all_images))
train_images = all_images[:split_index]
val_images = all_images[split_index:]

def copy_files(image_list, image_dest, label_dest):
       for image_file in image_list:
        label_file = image_file.replace(".png", ".txt")
        src_image = os.path.join(images_path, image_file)
        src_label = os.path.join(labels_path, label_file)
        if os.path.exists(src_label):
            shutil.copy(src_image, image_dest)
            shutil.copy(src_label, label_dest)


copy_files(train_images, train_img_dir, train_lbl_dir)
copy_files(val_images, val_img_dir, val_lbl_dir)

print(f"Split complete: {len(train_images)} training images, {len(val_images)} validation images
    .")
print(f"Files saved to: {output_base}")
```

9 Appendix C Data Configuration File (data.yaml)

```
train: # Path to the training images
val: # path to the validation images

nc: 5
names: ['Car', 'Van', 'Truck', 'Pedestrians', 'perosn_sitting']
```

10 Appendix D Script for model training

```
from ultralytics import YOLO


model = YOLO("#␣path␣to␣yolov8l.pt")


model.train(
    data="path␣to␣data.yaml",
    epochs=100,
    imgsz=416,
    batch=2,
    device="0"  # "0" for GPU
)
```

11 Appendix E Python script to remove the uncommon labels

```
import os


INDIAN_LABEL_DIR = "/media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/cross␣country/train/
    labels"


OUTPUT_DIR = "/media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/cross␣country/
    labelsaccordingtogermanmodel"


KEEP_IDS = {2, 4, 7, 13}


os.makedirs(OUTPUT_DIR, exist_ok=True)


for fname in os.listdir(INDIAN_LABEL_DIR):
    if not fname.endswith(".txt"):
        continue
    src_path = os.path.join(INDIAN_LABEL_DIR, fname)
    dst_path = os.path.join(OUTPUT_DIR, fname)


    with open(src_path, "r") as f:
        lines = f.readlines()


    filtered = []
    for line in lines:
        parts = line.strip().split()
        if len(parts) == 0:
            continue
        try:
            cls_id = int(parts[0])
        except ValueError:

            continue
```

```
        if cls_id in KEEP_IDS:
            filtered.append(line)


        with open(dst_path, "w") as out:
        out.writelines(filtered)


print(f"Done. Filtered labels written to:\n     {OUTPUT_DIR}")
```

12 Python script to remap the labels according the German datasets labels structure

```
import os

input_dir = "/media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/idd-detection/IDD_Detection/
    dataset/train/labels"
output_dir = "/media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/cross country/
    labelsaccordingtogermanmodel"

os.makedirs(output_dir, exist_ok=True)

# Indian to German class mapping
# Format: indian_class_id : german_class_id
class_map = {
    4: 0,    # Car  Car
    13: 1,   # Truck  Truck
    7: 2,    # Person  Pedestrian
    8: 3     # Rider  Cyclist
}

converted = 0
skipped = 0

for filename in os.listdir(input_dir):
    if not filename.endswith(".txt"):
        continue

    input_path = os.path.join(input_dir, filename)
    output_path = os.path.join(output_dir, filename)

    with open(input_path, "r") as f_in, open(output_path, "w") as f_out:
        for line in f_in:
            parts = line.strip().split()
            if len(parts) < 5:
                continue

            class_id = int(parts[0])
            if class_id in class_map:
                parts[0] = str(class_map[class_id])
                f_out.write(" ".join(parts) + "\n")
                converted += 1
            else:
                skipped += 1
```

```
print(f"Remapped␣and␣saved:␣{converted}␣labels")
print(f"Skipped␣(non-matching␣classes):␣{skipped}")
```

13 Appendix F Data configuration file for testing

```
train:    /media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/cross country/train/images
val:    /media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/cross country/train/images


nc: 5
names: ['Car', 'Van', 'Truck', 'Pedestrians', 'Person_Sitting']
```

14 Appendix G Python script to start testing

```
from ultralytics import YOLO

if __name__ == "__main__":
    # Path to the German model
    german_model_path = "/media/luh0419/Lucky_SSD/YOLO_Project/runs/detect/train6/weights/best.
    pt"

    # YAML file describing the Indian dataset
    indian_dataset_yaml = "/media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/cross␣country/
    fileforcrosscountrytesting.yaml"

    image_size = 416


    compute_device = 0


    output_directory = "/media/luh0419/Lucky_SSD/YOLO_Project/datasets/IDD/cross␣country/
    resultfromcc"
    run_name = "german_model_on_indian_data"


    model = YOLO(german_model_path)


    results = model.val(
        data=indian_dataset_yaml,
        imgsz=image_size,
        device=compute_device,
        project=output_directory,
        name=run_name
    )


    print("\nValidation␣completed.␣Summary:")
    print(results)
```

Literature references

DeepBean. How yolo object detection works. https://www.youtube.com/watch?v=svn9-xV7wjk&t=119s, 2023. Screenshot taken from YouTube video. Accessed: 2025-06-11.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. URL http://www.cvlibs.net/publications/Geiger2012CVPR.pdf.

G. S. Varma, Anbumani Subramanian, Vinay P. Namboodiri, and Madhava Krishna. Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019. URL https://idd.insaan.iiit.ac.in/.